21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS 2021, September 9–10, 2021, Lisbon, Portugal (Virtual Conference)

Edited by

Matthias Müller-Hannemann

Federico Perea

OASICS

Editors

Matthias Müller-Hannemann ⓘ
Martin Luther University of Halle-Wittenberg, Germany
muellerh@informatik.uni-halle.de

Federico Perea ⓘ
University of Seville, Spain
perea@us.es

ACM Classification 2012
Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete mathematics; Mathematics of computing → Combinatorics; Mathematics of computing → Mathematical optimization; Mathematics of computing → Graph theory; Applied computing → Transportation

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Regular Papers

Short Papers

Preface

Transportation is one of the key aspects in the development of a society. There are countless examples of cities, regions, or even countries which have witnessed enormous increases in their well-being after investments in their transportation systems. This improvement in the quality of life might be observed in several ways: less traveling times for people, moving goods more efficiently, access to a larger variety of products, among many others. The constant evolvement of transportation systems gives rise to new and more complex optimization problems, which in turn require the development of more efficient algorithms for solving them. The ever increasing volume of goods and people being transported imply more and more complexity in the problems to be solved, and therefore ask for new procedures for finding solutions to them, or even the need for new approaches. Although these new challenges have always been present, the COVID-19 pandemic has made the world reconsider many aspects of their "normal" functioning, among them of course: transportation. Will commerce go more local? Will telecommuting become the norm, and therefore the equilibria found in the transportation of people will change? These are only two examples of the questions that should be answered in the near future. Researchers and practitioners have the opportunity (or even the obligation) to answer these questions. The Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS) symposia, which have been running since 2000, are an excellent opportunity to show the latest advances in the approaches to model and solve the different problems arising in any transportation system.

Sadly, another consequence of the COVID-19 pandemic, is the fact that traditional conferences (like ATMOS) have gone online. Although ATMOS 2021 was meant to happen in Lisbon (Portugal), the mobility restrictions and the minimization of contagion risks, made the chairs of ALGO (the conference where ATMOS is included) take the difficult decision of running this conference online. This format has made the organization of ATMOS 2021 an even greater challenge. However, thanks to the help of the ALGO chairs, the ATMOS steering committee, and the outstanding ATMOS 2021 Program Committee (PC), we are confident that the quality and reputation of the ATMOS symposium has been maintained. In ATMOS 2021, the new category of short papers has been inroduced, presenting preliminary results or work-in-progress on a specific topic.

We received in total 29 submissions from all over the world, 24 of them were regular submissions, the other 5 being short paper submissions. All submissions were reviewed by at least three PC members, and the unanimous impression was the excellent quality of the submissions that we finally accepted. The time limitations of a two-day symposium forced us to accept only 19 submissions (16 regular and 3 short papers).

The ATMOS 2021 best paper award was given to *Carlo S. Sartori, Pieter Smet and Greet Vanden Berghe*, for their paper *Efficient duration-based workload balancing for interdependent vehicle routes*. Special thanks go to the sponsor of this prize: TRUCKSTERS, a young and dynamic company that offers express international transport services with maximum safety, efficiency, and sustainability (`https://www.trucksters.io/`).

ATMOS 2021 had *Anita Schöbel* (University of Kaiserslautern and Fraunhofer Institute for Industrial Mathematics (ITWM), Germany) as a plenary ALGO 2021 speaker who gave a talk on *Approaches for integrated planning: The case of public transport optimization*.

We would like to thank the members of the Steering Committee of ATMOS for giving us the opportunity to serve as Program Chairs of ATMOS 2021, all authors who submitted papers, Anita Schöbel for accepting our invitation to be a plenary speaker, the members of

the Program Committee and the additional reviewers for their valuable work in selecting the papers appearing in this volume, as well as Arlindo Oliveira (Chair of the ALGO 2021 Organizing Committee) and his team for hosting the symposium as part of ALGO 2021. We also acknowledge the use of the EasyChair system for the great help in managing the submission and review processes, and Schloss Dagstuhl for publishing the proceedings of ATMOS 2021 in its OASIcs series.

August 2021

Matthias Müller-Hannemann and Federico Perea

◼ Committees

Program Committee

Claudia Archetti	ESSEC, France
Valentina Cacchiani	University of Bologna, Italy
Luis Cadarso	University Rey Juan Carlos, Spain
Claudio Contardo	IBM, Canada
Francesco Corman	ETH Zurich, Switzerland
Daniel Delling	Apple, USA
Jesper Larsen	DTU Copenhagen, Denmark
Janny Leung	University of Macao, China
Christian Liebchen	TH Wildau, Germany
Matthias Müller-Hannemann (co-chair)	Martin Luther University Halle-Wittenberg, Germany
José Fernando Oliveira	University of Porto, Portugal
Fernando Ordoñez	University of Chile, Chile
Federico Perea (co-chair)	University of Seville, Spain

Steering Committee

Alberto Marchetti-Spaccamela	Sapienza University of Rome, Italy
Marie Schmidt	Erasmus University Rotterdam, the Netherlands
Anita Schöbel	Technical University of Kaiserslautern & Fraunhofer ITWM, Germany
Christos Zaroliagis (Chair)	CTI & University of Patras, Greece

Organizing Committee

Arlindo Oliveira	INESC-ID, Instituto Superior Técnico, University of Lisbon
Alexandre Francisco	INESC-ID, Instituto Superior Técnico, University of Lisbon
Susana Vinga	INESC-ID, Instituto Superior Técnico, University of Lisbon
Luís Russo	INESC-ID, Instituto Superior Técnico, University of Lisbon
Ana Teresa Freitas	INESC-ID, Instituto Superior Técnico, University of Lisbon
Ana Sofia Correia	INESC-ID, Instituto Superior Técnico, University of Lisbon
Tatiana Rocher	INESC-ID, Instituto Superior Técnico, University of Lisbon

List of Subreviewers

Diego Delle Donne
Niels Lindner
Berenike Masing
Valentina Morandi
Bart van Rossum
Yanlu Zhao

Efficient Duration-Based Workload Balancing for Interdependent Vehicle Routes

Carlo S. Sartori[1] ✉ 🆔
Department of Computer Science, KU Leuven, Belgium

Pieter Smet ✉ 🆔
Department of Computer Science, KU Leuven, Belgium

Greet Vanden Berghe ✉ 🆔
Department of Computer Science, KU Leuven, Belgium

— Abstract —

Vehicle routing and scheduling problems with interdependent routes arise when some services must be performed by at least two vehicles and temporal synchronization is thus required between the starting times of these services. These problems are often coupled with time window constraints in order to model various real-world applications such as pickup and delivery with transfers, cross-docking and home care scheduling. Interdependent routes in these applications can lead to large idle times for some drivers, unnecessarily lengthening their working hours. To remedy this unfairness, it is necessary to balance the duration of the drivers' routes. However, quickly evaluating duration-based equity functions for interdependent vehicle routes with time windows poses a significant computational challenge, particularly when the departure time of routes is flexible. This paper introduces models and algorithms to compute two well-known equity functions in flexible departure time settings: min-max and range minimization. We explore the challenges and algorithmic complexities of evaluating these functions both from a theoretical and an experimental viewpoint. The results of this paper enable the development of new heuristic methods to balance the workload of interdependent vehicle routes with time windows.

1 Introduction

Concerns regarding workload balancing in Vehicle Routing Problems (VRPs) have recently received attention in the literature [14, 15]. Most of this research addresses the VRP with route balancing [10], where the routes of multiple drivers are balanced according to some *equity function* in order to fairly distribute the workload between all workers. Route balancing is a challenging problem since it typically takes place in the context of a bi-objective VRP, where conflicting objectives such as total cost and workload imbalance must both be minimized. The difficulty of the problem increases when time windows are incorporated [16], in which case the workload is typically measured in terms of the route duration: from the departure time of the route until its completion, which includes possible idle periods of the driver.

---

[1] Corresponding author.

Existing literature has mainly focused on balancing workload between *independent* vehicle routes. By contrast, problems in which *interdependent* routes are considered have rarely been explored in terms of workload balancing. Interdependent routes occur when the start time of one driver's service depends on the completion time of another driver's service. In other words, there are temporal precedence constraints between tasks in different vehicle routes which therefore requires those routes to be synchronized somehow. Many real-world applications contain such interdependencies: pickup and delivery with transfers [17], equipment delivery and installation [1, 9] and home health care [8]. In all these applications, complications arise from the combination of time windows and interdependent routes. These difficulties are further compounded by the fact that we consider departure times of routes to be flexible. The combination of these three characteristics means that the computation of duration-based equity functions for workload balancing represents a nontrivial question and one which has not been previously addressed in the literature.

In this paper, we will consider two duration-based equity functions: min-max and range. These functions are often applied within decision support tools because they are very intuitive for decision makers [14]:

(1) Min-max: minimization of the longest route duration;

(2) Range: minimization of the difference between the longest and shortest route durations.

Evaluating these equity functions requires computing the minimum duration for all routes in a VRP solution. For VRPs with independent routes, such as the VRP with time windows, evaluating these durations can be performed in constant time after a preprocessing step [18]. In contrast, when routes are interdependent then these techniques for independent routes fail to correctly optimize functions (1) or (2). Indeed, [7] has noted that they are unaware of any constant-time method to update these duration-based equity functions that accommodate interdependent vehicle routes. When departure times are fixed, we can compute (1) and (2) with a linear time algorithm as detailed in Section 2. However, we have been unable to find studies concerning specialized algorithms with any complexity to correctly evaluate these functions when departure times are flexible.

The contributions of this paper are twofold. First, we describe how computing duration and corresponding equity functions of interdependent routes is challenging. Second, we introduce algorithms based on established methods in the literature to compute the duration-based workload balance of these routes along with their algorithmic complexity. A series of computational experiments provides additional understanding concerning the algorithmic performance in practice. The introduced algorithms can be incorporated within heuristic methods in which new solutions must be quickly evaluated with respect to workload balance. Hence, our contributions also open new research avenues for other researchers who would like to heuristically address vehicle routing problems which feature interdependent routes, time windows and workload balancing.

2   The interdependent route scheduling problem

This section defines the *Interdependent Route Scheduling Problem* (IRSP). The IRSP is defined over a graph $G = (V, A)$, where V is the set of nodes and A is the set of arcs that define temporal precedence constraints between pairs of nodes. Additionally, a set of fixed vehicle routes R is defined in G. A route $r_k \in R$ is a sequence of nodes $r_k = (\lambda_1, \ldots, \lambda_{|r_k|})$ where $\lambda_i \in V$. All nodes in V belong to exactly one route. For a route $r_k \in R$, its first and last nodes are the origin and destination locations and denoted o_k and d_k, respectively.

Every node $i \in V$ has an associated time window $[e_i, l_i]$ which indicates the earliest time e_i and latest time l_i that service is allowed to begin at node i. A vehicle is allowed to arrive at i before e_i and wait for service to start, but it may never arrive later than l_i. The service duration at i is w_i units of time. Furthermore, there is a time horizon H so that all services, including departure and completion time of the routes, must lie within $[0, H]$.

Arcs are subdivided into two sets $A = A_R \cup A_P$. Arc set A_R contains route arcs (i, j, t_{ij}) which connect nodes i and j belonging to the same route. They represent trips of duration t_{ij}. Meanwhile, set A_P contains *interdependency arcs* (u, v, δ_{uv}) which connect nodes u and v belonging to two different vehicle routes. The start time of service at u and v is captured by means of Equation 1, where variable h_i denotes the start time of service at node $i \in V$ and where δ_{uv} is a parameter.

$$h_u + \delta_{uv} \leq h_v \tag{1}$$

Correctly defining δ_{uv} enables us to model the five most common interdependency constraints encountered in practice [6]. For example, setting $\delta_{uv} = w_u$ (the service duration at u) creates the *minimum difference* interdependency found in VRPs with transfers [13, 17]. Meanwhile, creating two arcs (u, v, δ_{uv}) and (v, u, δ_{vu}) models *general synchronization* constraints occurring in some delivery and installation problems [9]. When $\delta_{uv} = 0$ and $\delta_{vu} = -2$h service at u and v may start simultaneously or with a difference of at most 2h. Similarly, if $\delta_{uv} = \delta_{vu} = 0$ then *strict synchronization* of the services at u and v is required, which is encountered in home health care problems [8]. In this paper, we present examples using the minimum difference interdependency, but the algorithms and models are valid for any constraint so long as it can be represented with the interdependency arcs in A_P. Interested readers are referred to Appendix A for more information on parameter δ_{uv}.

The goal of the IRSP is to produce a schedule where the starting time of service h_i complies with all of the time window constraints for every node $i \in V$. This includes deciding the departure and completion times of the routes at their origin and destination locations. Furthermore, we introduce three variants of the IRSP in this paper, which induce additional constraints to the decision of the starting times of service. The variants are:

(1) Feasibility: All routes $r_k \in R$ must comply with a maximum duration M.
(2) Min-max: produce a schedule that minimizes the longest duration $x_{\max}$ across all routes;
(3) Range minimization: produce a schedule that minimizes the difference between the longest duration $x_{\max}$ and the shortest duration $x_{\min}$.

When departure times are fixed, these three variants reduce to computing the completion time of all routes and this can be trivially solved in $O(|V|)$ by assigning a start time of service to each node in a topological ordering of G (see Appendix B). However, we consider departure times to be additional decision variables in the IRSP, thereby increasing the search space and the complexity of solving the problem. Despite substantially complicating the evaluation of route duration, flexible departure times are encountered in many real-world applications [18] and are of significant importance for ensuring the best use of all resources.

Figure 1(a) illustrates an instance of the IRSP with two routes: $r_1 = (1, 2, 3, 4, 5)$ and $r_2 = (6, 7, 8, 9)$. The service duration is $w_i = 0$h30, $\forall i \in V$ and the departure and completion times of the routes must lie within [0:00, 23:59] (time horizon $H = 24$h). Only nodes 4 and 7 have associated time windows. There is one minimum difference interdependency $(3, 8, 0$h30$) \in A_P$ which indicates that service at node 8 can only begin 0h30 after the start of service at node 3. Figures 1(b)–(e) depict four different solutions for the instance outlined in Figure 1(a). In these solutions, grey rectangles are service periods, blue rectangles (D) are driving periods and white rectangles (I) are idle (or waiting) periods.

Figure 1(b) presents a solution to the IRSP in which all drivers depart at time $t = 0:00$. Due to the time window at node 4, route r_1 has 6h of idle time and a total duration of 16h. Note that removing the idle time in r_1 requires delaying the start of service at node 3 which consequently delays the start time of service at node 8, thereby lengthening the duration of route r_2. Indeed, if all of the idle time in r_1 is removed, then the duration of route r_2 is increased to 17h, as illustrated by Figure 1(c). This effectively increases both the Min-max and the Range equity functions compared to 1(b). Furthermore, to comply with a maximum duration of $M = 15$h, route r_1 must be postponed by an hour, which delays start of service at node 8 by an hour as well. This lengthens the duration of r_2 to 12h, as shown in Figure 1(d). The optimal schedule for both Min-max and Range is depicted in Figure 1(e), where a balance is achieved between the durations of routes r_1 and r_2. In this schedule, any further reduction concerning the duration of route r_1 would increase the duration of r_2, leading to suboptimal solutions. The optimal schedule is obtained by postponing the departure time of route r_1 by 2:30, which is not an intuitive solution.

(a) Instance with two routes. Solid arcs represent direct trips where the weight is the trip's duration. Meanwhile, the dotted arc represents an interdependency constraint between the two routes.

(b) Schedule obtained after computing earliest service times. Longest duration $x_{\max} = 16$h. Range $x_{\max} - x_{\min} = 5$h. This schedule is optimal with respect to both the min-max and range equity functions if departure times are fixed at $t = 0:00$.

(c) Schedule obtained by removing all idle time from route r_1. This reduces the duration of r_1 to 10h, but also increases the duration of r_2 to 17h. The longest duration is $x_{\max} = 17$h and the range is $x_{\max} - x_{\min} = 7$h.

(d) Schedule in which all routes comply with maximum duration $M = 15$h. This is only possible if departure times are flexible since route r_1 must start at $t = 1:00$. Note that this delay increases the duration of r_2 from 11h to 12h.

(e) Schedule with optimal $x_{\max} = 13$h30 and minimum difference $x_{\max} - x_{\min} = 0$ when departure times are flexible. This solution is obtained by delaying the departure time of route r_1 and the completion time of r_2 by 2h30.

Figure 1 An IRSP instance and four possible solutions.

Note that when considering the VRP with time windows, minimizing route duration is equivalent to minimizing total waiting time [18], however this is not the case for the IRSP. Indeed, the total waiting time in the four solutions outlined in Figure 1 is the same: 5h30. The key difference is in how this total waiting time is distributed across all the routes. Therefore, simply minimizing total waiting time could lead to any of the four solutions in Figures 1(b)–(e), which is not the desirable outcome.

Finally, as the number of interdependent routes increases, the complex interactions between routes become more difficult to manage. This motivates us to examine whether it is possible to design efficient algorithms to effectively schedule interdependent vehicle routes.

3   The feasibility problem

The feasibility problem is the decision-version of IRSP for which an algorithm must provide
an answer to the following question: can all routes comply with a given maximum duration
M? This section introduces a Mathematical Programming (MP) formulation to precisely
describe the feasibility problem along with two special-purpose algorithms to solve it.

3.1   Mathematical formulation

A Linear Program (LP) for the feasibility IRSP is:

$$h_i - h_j \leq -w_i - t_{ij}, \quad \forall \, (i,j,t_{ij}) \in A_R \tag{2}$$

$$h_u - h_v \leq -\delta_{uv}, \quad \forall \, (u,v,\delta_{uv}) \in A_P \tag{3}$$

$$h_i \geq e_i, \quad \forall \, i \in V \tag{4}$$

$$h_i \leq l_i, \quad \forall \, i \in V \tag{5}$$

$$(h_{d_k} - h_{o_k}) \leq M, \quad \forall r_k \in R \tag{6}$$

Several general-purpose methods can be employed to solve this LP. For example, the
Simplex algorithm, Karmarkar's algorithm [11] or more recent approaches whose worst-case
time complexity make them more efficient in theory [3]. However, special-purpose algorithms
exist which are capable of solving the LP much quicker.

3.2   Simple temporal networks

A *Simple Temporal Network* (STN) is a graph which comprises of nodes that are events and
arcs between these nodes enable us to capture temporal relations between them. STNs have
been used in the past to check feasibility of VRP solutions with interdependent routes such
as the the dial-a-ride problem with transfers [13]. The formulation defined by Constraints
(2)–(6) can be represented as an STN. In order to do so, we define a special node α as the
beginning of time $t = 0$ and we replace Constraints (4) and (5) with:

$$h_\alpha - h_i \leq -e_i, \quad \forall \, i \in V \tag{7}$$

$$h_i - h_\alpha \leq l_i, \quad \forall \, i \in V \tag{8}$$

Constraints (2),(3),(6),(7) and (8) define a *Simple Temporal Problem* (STP) [5], which has
an associated STN. This network is a distance graph $G_D = (V_D, A_D)$, where $V_D = V \cup \{\alpha\}$
is the set of vertices and where A_D is the set of arcs that represent the constraints of the
STP formulation. Note that all constraints are binary, meaning they all contain exactly two
variables. A constraint of the form $h_i - h_j \leq \omega_{ij}$ induces an arc from node j to i with weight
ω_{ij} in G_D. Figure 2 depicts the STN associated with the instance illustrated in Figure 1(a).

Let $\tau_{i\alpha}$ denote the shortest path distance from i to α in G_D. Then, setting $h_i = -\tau_{i\alpha}$
provides the earliest feasible schedule for the routes of the corresponding IRSP instance. In
other words, the LP can be solved by computing shortest paths in G_D [5]. Note, however,
that the graph contains cycles and arcs of negative weight. Therefore, one must use methods
that can detect negative cycles in graphs, such as the Bellman-Ford algorithm. If G_D has a
negative-cost cycle then the STP is inconsistent, implying that the IRSP instance has no
feasible solution.

The asymptotic time complexity of the Bellman-Ford algorithm over G_D is $O(|V_D||A_D|)$.
The number of arcs $|A_D|$ is $O(|V|)$ given that all nodes $i \in V$ have no more than three
outgoing arcs and node α has no more than $|V|$ outgoing arcs. This means that the complexity
of determining feasibility of an IRSP instance is $O(|V|^2)$.
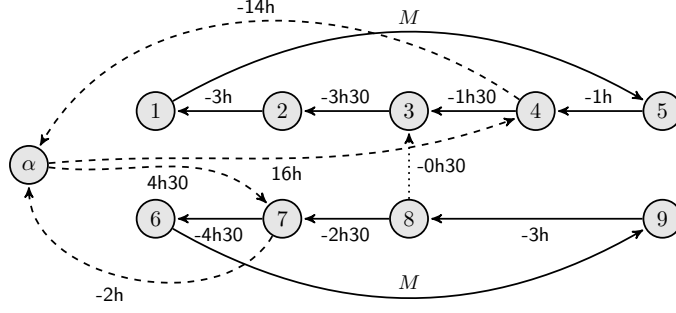
■ **Figure 2** STN created from the instance in Figure 1(a). Dashed arcs denote time window constraints. Service durations have been included in the travel times between nodes.

3.3 Surrogate graph

The graph depicted in Figure 1(a) is a directed acyclic graph (DAG). Similar to STNs, the introduction of maximum duration constraints in this DAG creates cycles, as illustrated in Figure 3(a). Exactly $|R|$ maximum duration arcs must be included: one per route.

Computing shortest paths in a DAG, or in the IRSP the earliest feasible start times of service, is straightforward and can be efficiently performed in $O(|V|)$ time (see Appendix B). We are therefore interested in removing the $|R|$ maximum duration arcs that were introduced in order to remove the cycles induced by them while still ensuring compliance with the maximum duration M. To remove these arcs, we employ the strategy introduced by [19] for almost acyclic graphs. We define an associated *surrogate graph* G_S where a new source node α is created. Then, each maximum duration arc of the form $(d_k, o_k, -M)$ is replaced with an arc $(\alpha, o_k, 0)$. In doing so, G_S becomes a DAG. This is illustrated in Figure 3(b).

Once G_S has been defined, we can solve the LP (2)–(6) by means of shortest paths employing the *Surrogate Algorithm* [19] outlined in Algorithm 1. This procedure needs to perform no more than $|R| + 1$ iterations of the for-loop (lines 2–9). In each iteration, the start time of service is computed in $O(|V|)$ via the procedure in line 3 (Appendix B), which returns `true` if no time window has been violated and `false` otherwise. At the end of each iteration, the departure time of each route $r_k \in R$ is updated using the current completion time at the destination node d_k and the maximum route duration M (line 5). Updating the departure times corresponds to dynamically updating the weights $\omega_{\alpha o_k}$ of the surrogate arcs $(\alpha, o_k, \omega_{\alpha o_k})$ in G_S. Since every iteration of the for-loop takes $O(|V|)$, the total complexity of the Surrogate Algorithm is $O(|V||R|)$.

■ **Algorithm 1** Surrogate Algorithm.

Input: An instance of the IRSP and maximum duration M.
Output: Returns `true` if all routes comply with M, and `false` otherwise.
1: $p \leftarrow$ `true`
2: **for** i $= 0$ **until** $|R|$ **do**
3:     $p \leftarrow$ `ComputeStartTimeOfService`(G_S)
4:     **if** $p =$ `true` **then**
5:         $\omega_{\alpha o_k} \leftarrow \max\{0, h_{d_k} - M\}, \ \forall \ r_k \in R$
6:     **else**
7:         **goto** 10
8:     **end if**
9: **end for**
10: **return** p
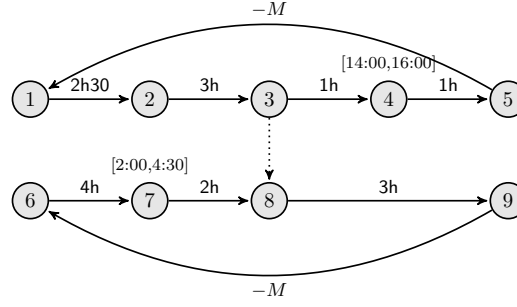
(a) Instance modified by introducing maximum duration arcs for each route.
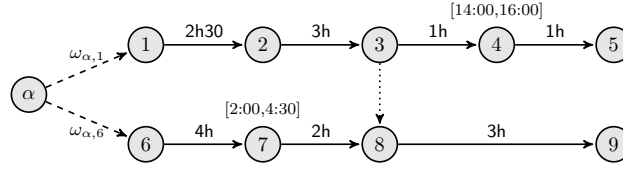
(b) Maximum duration arcs replaced with surrogate arcs from a dummy source node α. Surrogate arcs have variable weights $\omega_{\alpha i}$, which will be updated during the execution of the Surrogate Algorithm.

■ **Figure 3** The surrogate graph of the instance in Figure 1(a).

The correctness and complexity of Algorithm 1 follow directly from [19]. Note that computing the earliest feasible start time of a service corresponds to computing the longest path from α to any node in G_S, which can be accomplished in linear time over a DAG [4].

4 The min-max problem

In the Min-max problem, we seek to minimize the longest duration so as to alleviate the working hours of the drivers who work the most. This is performed even though the duration of some shorter routes is increased in the process. The methods presented to solve Min-max build upon those of the feasibility problem (Section 3).

4.1 Mathematical formulation

The Min-max IRSP can be formulated as an LP by defining a continuous variable $x_{\max}$ to represent the longest duration. The model is:

$$\min \quad x_{\max} \tag{9}$$

$$\text{constraints (2)–(5)}$$

$$x_{\max} \geq (h_{d_k} - h_{o_k}), \quad \forall r_k \in R \tag{10}$$

The current best general-purpose LP algorithm that can solve Min-max is not asymptotically faster than $O^*(|V|^{2.37} \log(|V|/\gamma))$, for a given precision $0 < \gamma \leq 1$ [3][2]. Therefore, we are interested in determining whether it is possible to solve Min-max more efficiently.

---

[2] Complexity O^* is based on the notation by [3] to hide extra factors (for example, $n^{o(1)}$).

4.2   A special-purpose algorithm

Algorithm 2 outlines a simple procedure to solve Min-max. This algorithm is based on the research introduced by [12] and performs a binary search over the space of route durations in the range $[a, b]$, which is initially $[0, H]$. For every mid-point m in this range, feasibility with respect to maximum duration m is checked using procedure `DurationFeasibility` (line 5). This test can be implemented using any of the feasibility algorithms outlined in Section 3. Limits a and b are subsequently updated according to the feasibility of m (line 6). These steps are repeated as long as $b - a > \epsilon$ for a given precision value $\epsilon > 0$.

In practice, m, a, b and ϵ are floating-point variables and therefore permit only a finite value representation. This implies that Algorithm 2 is guaranteed to finish executing in a finite number of steps. The number of iterations performed in the algorithm is $O(\log H)$. The complexity of each iteration depends on the algorithm employed at line 5. If the STN method is employed, then Algorithm 2 has complexity $O(|V|^2 \log H)$. However, if the Surrogate Graph is used, the complexity is reduced to $O(|V||R| \log H)$ because $|V| > |R|$.

▨ **Algorithm 2** Duration minimization.

Input: An instance of the IRSP.
Output: Minimum longest duration $x_{\max}$.
 1: $a \leftarrow 0$
 2: $b \leftarrow H$
 3: **while** $(b - a) > \epsilon$ **do**
 4:     $m \leftarrow (b + a) \cdot 0.5$
 5:     $p \leftarrow \texttt{DurationFeasibility}(m)$
 6:     **if** $p = \texttt{true}$ **then** $b \leftarrow m$ **else** $a \leftarrow m$
 7: **end while**
 8: **return** b

---

5   The range minimization problem

The minimization of range is a complicated problem to formulate using an MP when time windows are incorporated [16]. This is because routes may be artificially lengthened by increasing the waiting time at service locations, thereby decreasing the difference between the longest and shortest routes. To avoid unnecessary waiting times, a formulation that forces the start time of all services to be as early as possible was proposed by [16]. However, their scheduling problem was much simpler than the IRSP because (i) routes were independent and (ii) departure times were fixed at $t = 0$. The same modeling ideas thus cannot be applied to the IRSP due to the combination of flexible departure times and interdependent routes.

5.1   Mathematical formulation

A naive MP formulation to minimize Range uses the following objective function:

$$\min \quad x_{\max} - x_{\min} \tag{11}$$

However, this function minimizes $x_{\max}$ at the same time that it *maximizes* $x_{\min}$. This, in turn, leads to the situation depicted in Figure 4. The instance in Figure 4(a) has two routes: $r_1 = (1, 2, 3, 4)$ and $r_2 = (5, 6, 7, 8)$. Only node 6 has an associated time window. Service durations are $w_i = 0\text{h}30$, $\forall i \in V$ and the length of the time horizon is $H = 24\text{h}$. There is one minimum difference interdependency $(2, 7, 0\text{h}30) \in A_P$.

Assuming a departure time for both routes at $t = 0{:}00$, we can produce the solution in Figure 4(b) where the range is 6h. By contrast, the naive MP formulation would produce the solution depicted in Figure 4(c) in which the range is optimal: 0h. Note that such an optimal

solution is only possible by incurring 6h of idle time in r_2, even though this is unnecessary. Although mathematically optimal, the solution in 4(c) is very unlikely to be accepted in practice given that it delays several services and it forces the second driver to be in route for a much longer period even though almost half of their working time is idle. Furthermore, this naive formulation could guide a VRP solver to produce solutions with a mathematically perfect balance by creating a lot of idle time in some routes, while other routes would be completely exhausted with working time. From the perspective of the workers, this would be seen as a great *imbalance* in workloads, thereby negatively impacting their morale.

(a) Instance considered in the example.   **(b)** Solution where $x_{\max} - x_{\min} = 13h - 7h = 6h$.

(c) Solution where $x_{\max} - x_{\min} = 13h - 13h = 0h$.
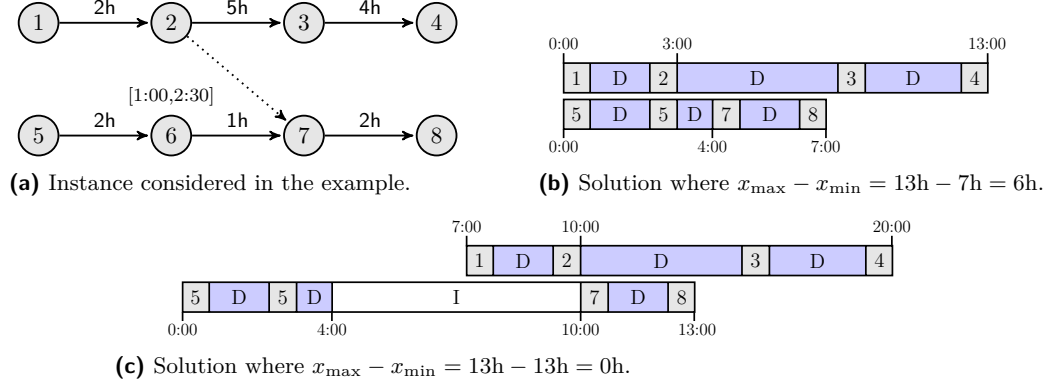
Figure 4 Instance for which a naive MP fails to correctly minimize the range.

The modeling approach proposed by [16] cannot be applied to the IRSP because the earliest start time of service at interdependent nodes depends on the departure time of the routes, which is flexible. For example, the start time of service at node 2 (and subsequently node 7) depends on the departure time of route r_1. In the IRSP, it does not appear to be possible to force values for the start times of services without sacrificing optimality.

To correctly minimize the range by means of an MP, we propose a two-stage approach. First, we solve the LP from Section 4 to obtain $x_{\max}$. Then, we obtain $x_{\min}$ by solving the following Mixed-Integer Linear Programming (MILP) formulation:

$$\min \quad x_{\min} \tag{12}$$

$$\text{constraints } (2)\text{--}(5)$$

$$X_{\max} \geq (h_{d_k} - h_{o_k}), \quad \forall\, r_k \in R \tag{13}$$

$$x_{\min} \geq (h_{d_k} - h_{o_k}) + H(y_k - 1), \quad \forall\, r_k \in R \tag{14}$$

$$\sum\nolimits_{k=1}^{|R|} y_k \geq 1 \tag{15}$$

Here $X_{\max}$ refers to a constant value equal to the min-max duration $x_{\max}$. Meanwhile, for each route $r_k \in R$, a binary variable $y_k = 1$ if route r_k has the shortest duration among all in R, otherwise $y_k = 0$. This effectively (de)activates Constraints (14) which set the value of variable $x_{\min}$. Unfortunately, solving MILP (12)–(15) in addition to LP (9)–(10) can create a significant computational overhead. Therefore, we are interested in determining whether a special-purpose algorithm can be defined to minimize range.

5.2 A special-purpose algorithm

Range minimization can also be achieved by Algorithm 3. `DurationMinimizer` is any method capable of solving Min-max, such as those detailed in Section 4. Here, this procedure takes three values as input: a set of routes $R' \subseteq R$ for which the longest duration is to be minimized in addition to the lower and upper bounds (a and b) for the duration of each route.

Algorithm 3 begins by computing $x_{\max}$ (line 1): the min-max duration considering all routes in R. The loop spanning lines 3–6 then attempts to minimize the duration of each route $r_k \in R$ independently in order to produce the minimum duration $x_{\min}$ considering all routes in R. In line 4, `DurationMinimizer` receives as input $R' = \{r_k\}$, $a = 0$ and $b = x_{\max}$, and computes the minimum duration x_k for route r_k. However, the computation of x_k may modify the duration of other routes in R because of the interdependencies, which can subsequently increase the longest duration $x_{\max}$. To avoid this, the duration of all routes $r_z \in R : r_z \neq r_k$ is constrained to be at most $x_{\max}$ when computing x_k. Moreover, the minimization taking place in line 4 is performed without considering the results of previous iterations so as to not interfere with the computation of x_k. The result is then used to update variable $x_{\min}$ (line 5). Finally, the minimum range $x_{\max} - x_{\min}$ is returned at line 7.

■ **Algorithm 3** Range minimization.

Input: An instance of the IRSP.
Output: Minimum value for range $x_{\max} - x_{\min}$.
1: $x_{\max} \leftarrow$ `DurationMinimizer`$(R, 0, H)$
2: $x_{\min} \leftarrow +\infty$
3: **for each** $r_k \in R$ **do**
4:     $x_k \leftarrow$ `DurationMinimizer`$(\{r_k\}, 0, x_{\max})$
5:     $x_{\min} \leftarrow \min\{x_{\min}, x_k\}$
6: **end for**
7: **return** $(x_{\max} - x_{\min})$

The complexity of Algorithm 3 depends on that of `DurationMinimizer`. If STNs are employed, then the algorithm's complexity is $O(|V|^2 |R| \log H)$. However, when using Surrogate Graphs it is $O(|V||R|^2 \log H)$. Alternatively, one could employ a general-purpose LP solver as `DurationMinimizer` by trivially modifying the formulation in Section 4. This would result in a complexity of $O^*(|V|^{2.37}|R| \log(|V|/\gamma))$. However, solving $|R|$ LPs is likely to incur a prohibitive computational overhead despite the polynomial time complexity. In all of these algorithmic variants, the additional $|R|$ derives from the for-loop spanning lines 3–6.

6   Computational experiments

Table 1 summarizes the worst-case asymptotic time complexities when solving the IRSP variants by employing each of the algorithms described in this paper. These complexities indicate that the Surrogate approach represents the fastest method of all the options because the relation $|R| < |V|$ is always valid.

■ **Table 1** Worst-case asymptotic time complexity for the algorithms. Recall that V is the set of nodes and R the set of routes in the IRSP instance, while H denotes the length of the time horizon. Value γ is the desired precision for the LP solver [3].

Problem	STN	Surrogate	MP
Feasibility	$O(|V|^2)$	$O(|V||R|)$	$O^*(|V|^{2.37} \log(|V|/\gamma))$
Min-max	$O(|V|^2 \log H)$	$O(|V||R| \log H)$	$O^*(|V|^{2.37} \log(|V|/\gamma))$
Range min.	$O(|V|^2|R| \log H)$	$O(|V||R|^2 \log H)$	$O^*(|V|^{2.37}|R| \log(|V|/\gamma))$

In addition to these theoretical results, we have also performed a computational study of the algorithms to examine their processing times for real-sized instances. We implemented all algorithms in `C++` and compiled them using `g++` 7.5 with optimization flag `-O3`. The MP

components were implemented using the Gurobi 9 API for `C++` which is a state-of-the-art solver, even though it does not necessarily implement the LP algorithm introduced by [3]. All executions were restricted to a single thread on a computer equipped with an Intel i7-8850H processor at 2.6 GHz, 32 GB of RAM and Ubuntu 18.04 LTS operating system.

IRSP instances were obtained by solving the VRP with multiple synchronization constraints [9]. To produce solutions for the VRP, we employed the Slack Induction by String Removals heuristic [2]. For each new solution, the Min-max and Range equity functions were evaluated using the three algorithms. The IRSP instances that were generated had characteristics with the following ranges: $|V| \leq 300$, $|R| \leq 35$ and $|A_P| \leq 100$. Note that these are already large scale instances for most real-world purposes.

Let us begin the analysis by considering the worst-case performance observed during the experiments. This deserves focus because the algorithms must run as fast as possible even in their worst-case to be safely employed in practice. The graphs in Figure 5 report the maximum recorded execution time in microseconds (μs)[3] according to the number of nodes $|V|$ in the IRSP instance. Due to the significant differences across the algorithms, the graphs are presented in logarithmic scale. The raw data points are plotted directly, while the curves were produced by polynomial interpolation in order to more easily analyze the results.

The MP approach is 20–30 times slower than the other two methods. This is not surprising because Gurobi is a general-purpose solver which incurs significant overhead when addressing structurally simple problems such as the IRSP. Meanwhile, STN solves Min-max 50% quicker and is almost twice as fast when minimizing Range compared to the Surrogate Algorithm. These results clearly contradict the theoretical worst-case time complexity. The reason for this is that STN can detect infeasible maximum durations much faster than Surrogate. This is reflected in the processing times of the two algorithms since, particularly for Range, many feasibility tests must be performed to obtain the optimal solution.
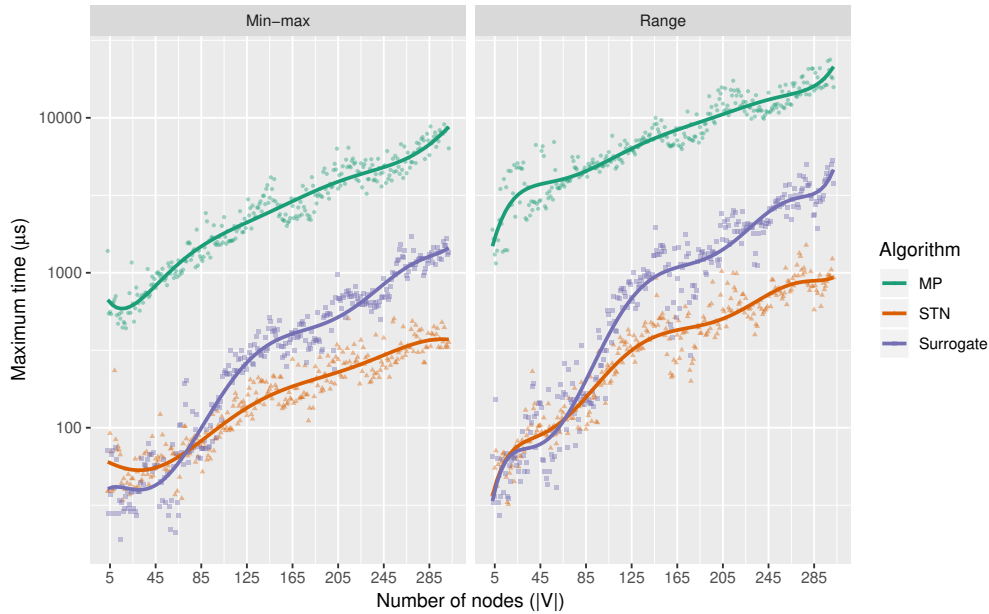
Figure 5 Maximum processing times in microseconds of the three algorithms (logarithmic scale).

---

[3] Processing times were measured using the `C++` library `std::chrono::high_resolution_clock`.

Alternatively, let us now consider the average processing times of the algorithms. The graphs in Figure 6 report the raw data points for the average processing times as well as an interpolation of the data according to the number of nodes in the IRSP instance, similar to the graphs in Figure 5.
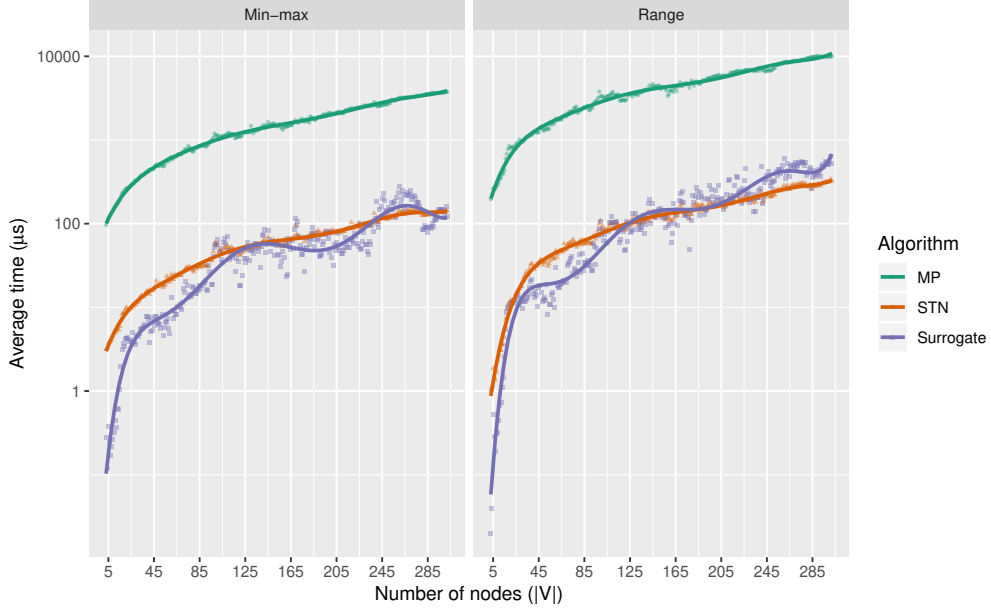
Figure 6 Average processing times in microseconds (μs) of the algorithms (logarithmic scale).

On the one hand, the MP is again 20–25 times slower than the other two methods, on average. On the other hand, comparison of STN and Surrogate is more subtle this time around. Surrogate is 16% faster than the STN when solving Min-max, whereas when minimizing Range the STN is 17% faster. These results are significantly different from the worst-case because Surrogate has more variability in its processing times, while both MP and STN are consistent. Once again, these observations are explained by the fact that Surrogate sometimes requires many iterations to prove infeasibility of a maximum duration M. Meanwhile, in some other instances, Surrogate benefits from its reduced complexity and quickly provides the optimal solution. All of these reasons help explain why, on average, the differences between STN and Surrogate are reduced.

Finally, the experiments indicate that minimizing Range is 2–3 times slower than solving Min-max, which is what one would expect given the time complexities outlined in Table 1. Hence, it may be worth exploring the differences of employing Min-max and Range when balancing workloads, similar to the study conducted by [15] for independent vehicle routes.

7   Conclusion

Interdependent route scheduling is a nontrivial problem when both time windows and flexible departure times must be taken into account. The problem becomes even more challenging when duration-based workload balance between these interdependent routes is desired given how the decisions made for one route can have unforeseen impacts on others, potentially leading to unfair schedules for the drivers. To overcome these challenges, this paper introduced complementary optimization models for balancing duration-based workload among drivers in addition to algorithms for the efficient evaluation of the corresponding equity functions.

The resulting evaluation methods may be employed within, for example, local-search heuristics to produce balanced vehicle routes. Balanced routes help improve the working conditions and morale of the drivers. There are many real-world applications that can benefit from these methods: logistics, transportation, home health care and workforce scheduling.

In spite of our results, many questions remain open. Are there more efficient algorithms to evaluate the Min-max and Range equity functions? Are there alternative approximations that can be employed to compute them faster? Can we extend the methods to address multiple time windows per customer node? More broadly, how can we model an entire VRP with interdependent routes such that the range is minimized? Is it possible to use only one MILP? All of these exciting research opportunities are open for researchers to explore in future studies.

 ── **References** ──

1 Ousmane Ali, Jean-François Côté, and Leandro C. Coelho. Models and algorithms for the delivery and installation routing problem. *European Journal of Operational Research*, 291(1):162–177, 2021. `doi:10.1016/j.ejor.2020.09.011`.

2 Jan Christiaens and Greet Vanden Berghe. Slack induction by string removals for vehicle routing problems. *Transportation Science*, 54(2):417–433, 2020. `doi:10.1287/trsc.2019.0914`.

3 Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM*, 68(1):3:1–3:39, 2021. `doi:10.1145/3424305`.

4 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2009.

5 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. `doi:10.1016/0004-3702(91)90006-6`.

6 Anders Dohn, Matias Sevel Rasmussen, and Jesper Larsen. The vehicle routing problem with time windows and temporal dependencies. *Networks*, 58(4):273–289, 2011. `doi:10.1002/net.20472`.

7 Michael Drexl. A generic heuristic for vehicle routing problems with multiple synchronization constraints. *Gutenberg School of Management and Economics–Discussion Paper Series: Mainz, Germany*, 1412:43, 2014.

8 Christian Fikar and Patrick Hirsch. Home health care routing and scheduling: A review. *Computers & Operations Research*, 77:86–95, 2017. `doi:10.1016/j.cor.2016.07.019`.

9 Hossein Hojabri, Michel Gendreau, Jean-Yves Potvin, and Louis-Martin Rousseau. Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. *Computers & Operations Research*, 92:87–97, 2018. `doi:10.1016/j.cor.2017.11.011`.

10 Nicolas Jozefowiez, Frédéric Semet, and El-Ghazali Talbi. Parallel and hybrid models for multi-objective optimization: Application to the vehicle routing problem. In Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, José Luis Fernández-Villacañas Martín, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VII, 7th International Conference, Granada, Spain, September 7-11, 2002, Proceedings*, volume 2439 of *Lecture Notes in Computer Science*, pages 271–280. Springer, 2002. `doi:10.1007/3-540-45712-7_26`.

11 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984. `doi:10.1007/BF02579150`.

12 Lina Khatib, Paul H. Morris, Robert A. Morris, and Francesca Rossi. Temporal constraint reasoning with preferences. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 322–327. Morgan Kaufmann, 2001.

13 Renaud Masson, Fabien Lehuédé, and Olivier Péton. The dial-a-ride problem with transfers. *Computers & Operations Research*, 41:12–23, 2014. `doi:10.1016/j.cor.2013.07.020`.

14   Piotr Matl, Richard F. Hartl, and Thibaut Vidal. Workload equity in vehicle routing problems: A survey and analysis. *Transportation Science*, 52(2):239–260, 2018. `doi:10.1287/trsc.2017.0744`.

15   Piotr Matl, Richard F. Hartl, and Thibaut Vidal. Workload equity in vehicle routing: The impact of alternative workload resources. *Computers & Operations Research*, 110:116–129, 2019. `doi:10.1016/j.cor.2019.05.016`.

16   Belén Melián-Batista, Alondra De Santiago, Francisco Ángel-Bello, and Ada M. Alvarez. A bi-objective vehicle routing problem with time windows: A real case in tenerife. *Applied Soft Computing*, 17:140–152, 2014. `doi:10.1016/j.asoc.2013.12.012`.

17   Snežana Mitrović-Minić and Gilbert Laporte. The pickup and delivery problem with time windows and transshipment. *INFOR: Information Systems and Operational Research*, 44:217–227, 2006.

18   Martin W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *INFORMS Journal on Computing*, 4(2):146–154, 1992. `doi:10.1287/ijoc.4.2.146`.

19   Donald K. Wagner. Shortest paths in almost acyclic graphs. *Operations Research Letters*, 27(4):143–147, 2000. `doi:10.1016/S0167-6377(00)00054-7`.

A   Types of temporal interdependency

The five most common types of temporal interdependencies encountered in practice as described by [6] can be represented with Equation 1 by correctly parameterizing δ values. Table 2 details the setting of these parameters for each type of constraint when relating two nodes u and v which belong to two different vehicle routes.

Table 2 Definition of δ weights for temporal interdependencies. N/A denotes that no value is assigned (no relation or arc is defined). Here, $\alpha_{\min}$ and $\alpha_{\max}$ are parameters defining the desired minimum and maximum time differences. Table adapted from [6].

Interdependency	δ_{uv}	δ_{vu}
strict synchronization	0	0
overlap	$-w_v$	$-w_u$
minimum difference	$\alpha_{\min}$	N/A
maximum difference	N/A	$-\alpha_{\max}$
general synchronization	$\alpha_{\min}$	$-\alpha_{\max}$

Appendix B describes how these interdependencies can be modeled in a precedence graph and the impact they can have on the computation of service start times. Particularly, note that *minimum* and *maximum difference* are both unidirectional constraints, while the other three are all bidirectional constraints.

B   Computing service start times

Given an instance of the IRSP with a graph $G = (V, A)$, we can compute earliest feasible start time of service h_i at every node $i \in V$ by following a topological ordering of G [4]. In doing so, the computation is guaranteed to be performed in $O(|V|)$ time.

Before going into details about the procedure, we must note that in order to obtain a topological ordering, G must be a DAG. However, Table 2 shows that some interdependencies are bidirectional and therefore incur cycles when represented as a graph. These *cyclic interdependencies* arise whenever two interdependency arcs are required to represent them in a graph. In other words, whenever for two nodes u and v there are arcs (u, v, δ_{uv}) and (v, u, δ_{vu}) in set A_P. Figure 7 illustrates the cycles and how we can trivially eliminate them

to obtain a DAG. Figure 7(a) depicts one of the three interdependencies on a graph with a cycle of size two. Fortunately, these cycles can be removed by duplicating nodes as per Figure 7(b), where $w_{u'} = w_{v'} = 0$.

(a) Cyclic interdependency in graph form.

(b) An equivalent directed acyclic graph.
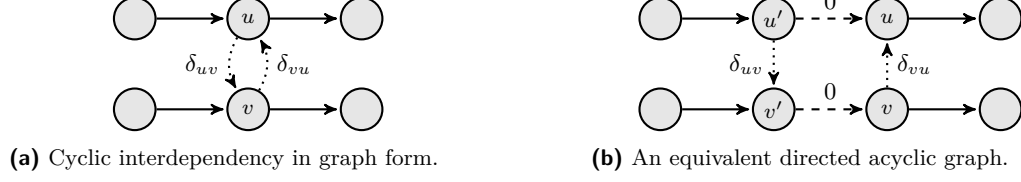
■ **Figure 7** Cyclic interdependencies on a graph.

Once a topological order of G is obtained, we can compute start time of service h_i at every node $i \in V$. First, we set the departure times at origins $o_k, \forall r_k \in R$. For simplicity purposes we assume $h_{o_k} = 0$ for all routes, but in practice any departure time can be set if known or previously computed (for example after each iteration of the Surrogate Algorithm). Then, for each node j in the topological ordering (and such that j is not an origin location), the start time of service h_j is computed by:

$$h_j = \max\{e_j, h_i + w_i + t_{ij}\}, \quad (i, j, t_{ij}) \in A_R$$

However, if j is part of an interdependency constraint, that is, $(u, j, \delta_{uj}) \in A_P$, then we must also take into account the relation captured by Equation 1:

$$h_j = \max\{h_j, h_u + \delta_{uj}\}, \quad \text{if } (u, j, \delta_{uj}) \in A_P \tag{16}$$

Value h_u is always known when computing h_j in Equation 16 thanks to the topological order. In this way, every value $h_i, \forall i \in V$ is computed exactly once and all interdependency relations are respected. If, however, $h_j > l_j$ for any node $j \in V$ then there is an infeasibility and the procedure terminates.

Forward Cycle Bases and Periodic Timetabling

Niels Lindner[1] ✉ 🄳
Zuse Institute Berlin, Germany

Christian Liebchen ✉ 🄳
Technical University of Applied Sciences Wildau, Germany

Berenike Masing ✉ 🄳
Zuse Institute Berlin, Germany

Abstract

Periodic timetable optimization problems in public transport can be modeled as mixed-integer linear programs by means of the Periodic Event Scheduling Problem (PESP). In order to keep the branch-and-bound tree small, minimum integral cycle bases have been proven successful. We examine forward cycle bases, where no cycle is allowed to contain a backward arc. After reviewing the theory of these bases, we describe the construction of an integral forward cycle basis on a line-based event-activity network. Adding turnarounds to the instance `R1L1` of the benchmark library PESPlib, we computationally evaluate three types of forward cycle bases in the Pareto sense, and come up with significant improvements concerning dual bounds.

1 Introduction

As any mathematical optimization problem, periodic timetable optimization is about detecting a feasible solution with the best possible objective value. To solve the mixed integer programs that arise from the *Periodic Event Scheduling Problem* (PESP) [25], a standard model for periodic timetable optimization, to optimality, it is of interest to focus on those parts of the feasible region that allow for good objective values in particular. Hence, quite an amount of research has been dedicated to identifying valid inequalities for the PESP polytope, and making use of them as cutting planes in a branch-and-cut context [2, 15, 16, 19, 20, 21, 23]. Following the general goal of describing the convex hull of the integer points as sharp as possible, these approaches focus on the entire feasible region. Yet, as we deal with an optimization problem, it is of particular interest to examine those parts of the feasible region that allow for good objective values.

Most of the known successful cutting planes for PESP can be derived from oriented cycles in the underlying event-activity network. However, as these cycles contain in general both forward and backward arcs, it is hard to establish a link between constraints from cycles, typically containing coefficients with different signs, and the objective function, whose coefficients are all non-negative.

---

[1] corresponding author

We therefore propose to focus on *forward cycles*, i.e., oriented cycles that contain exclusively forward arcs. Then the classical cycle inequalities by Odijk [23] have entirely non-negative coefficients for the travel time variables, so that tight constraints directly relate to the objective function.

In the area of periodic timetabling, there is one collection of reference instances publicly available, the so-called PESPlib [6]. To date, for all of its instances their best known solutions still show relatively large duality gaps. Having a closer look to the railway-motivated instances `RxLy`, providing some feasible solution is trivial, because when removing all free arcs, i.e., arcs for which any travel time modulo the period time is feasible, these instances essentially decompose into paths [7, 17]. Hence, the main challenge for the `RxLy` instances of the PESPlib is about the actual optimization process rather than feasibility. We want to support this process better by seeking in particular those valid inequalities which have a significant impact on the objective function.

We propose to modify the instances `RxLy` slightly: We match the paths mentioned above to lines, and add turnaround arcs at the line ends. It then becomes possible to find a cycle basis consisting of forward cycles only. Moreover, turnarounds allow for an investigation of vehicle rotations, so that we can discuss optimality in the Pareto sense with respect to both the minimum passenger travel time and the minimum number of vehicles. Varying minimum turnaround times and turnaround weights, it turns out that the forward cycles approach is fruitful, and we can compute a new dual incumbent for the instance `R1L1`.

We formally describe the Periodic Event Scheduling Problem in Section 2. In Section 3, we motivate the use of forward cycles in detail, review the theory of forward cycles bases, and provide a construction for an integral cycle basis for timetabling networks with a specific structure. Analyzing the PESPlib benchmark instance `R1L1`, we describe in Section 4 how to add turnaround arcs in a meaningful way, which allows to compute forward cycle bases on a modified instance `R1L1v`. Section 5 presents computational results, evaluating passenger travel time slack and the number of required vehicles for different choices of cycle bases, minimum turnaround times, and turnaround weights. We conclude the paper in Section 6.

2   Periodic Event Scheduling

2.1   Problem Definition

The *Periodic Event Scheduling Problem* (PESP) has been introduced by Serafini and Ukovich [25]. A PESP instance consists of a 5-tuple (G, T, ℓ, u, w), where

- $G = (V, A)$ is a directed graph, often called *event-activity network*,
- $T \in \mathbb{N}$ is a *period time*,
- $\ell \in \mathbb{R}_{\geq 0}^A$ is a vector of *lower bounds* with $0 \leq \ell < T$,
- $u \in \mathbb{R}_{\geq 0}^A$ is a vector of *upper bounds* such that $0 \leq u - \ell < T$,
- $w \in \mathbb{R}_{\geq 0}^A$ is a vector of *weights*.

For an instance (G, T, ℓ, u, w), a vector $\pi \in [0, T)^V$ is called a *periodic timetable* if there is a *periodic tension* $x \in \mathbb{R}^A$ such that

$$\forall ij \in A : \quad \ell_{ij} \leq x_{ij} \leq u_{ij} \quad \text{and} \quad x_{ij} \equiv \pi_j - \pi_i \mod T.$$

If x is a periodic tension, then $y := x - \ell$ is called *periodic slack*.

Interpreting the vertices in V as events and the arcs in A as activities, a periodic timetable π fixes event timings, and a periodic tension x is an assignment of activity durations, both modulo the period time T. In the context of periodic timetabling, events are typically arrivals or departures of vehicles at stations, and activities model driving between stations, dwelling or transferring at a station, turnarounds, etc. [13].

▶ **Definition 1** ([25]). *Given (G, T, ℓ, u, w), the* Periodic Event Scheduling Problem *(PESP) is to find a periodic timetable π along with a periodic tension x such that the weighted slack $\sum_{a \in A} w_a y_a$ is minimum or to decide that no periodic timetable exists.*

As the arc weights w often reflect the number of passengers using a specific activity, and periodic tensions correspond to activity durations, the PESP objective then amounts to minimize the total passenger travel time. It is also possible to shift the focus onto the number of vehicles by introducing large weights on turnaround activities [13]. We will evaluate both weighted slack and number of vehicles in our computational Section 5.

2.2 Cycle-Based Mixed-Integer Program

We will use the following cycle-based mixed-integer programming formulation for PESP throughout this paper:

$$
\begin{aligned}
\text{Minimize} \quad & w^\top y \\
\text{s.t.} \quad & \Gamma(y + \ell) = Tz, \\
& 0 \le y \le u - \ell, \\
& z \in \mathbb{Z}^B.
\end{aligned}
\tag{1}
$$

In (1), B is an integral cycle basis of G with cycle matrix Γ. We refer to Section 3.2 or [12] for details. The integer z-variables model the modulo T conditions, we will call them *cycle offset variables*. Periodic timetables are only implicit in this formulation, for a feasible periodic slack y, a timetable can be recovered by a graph traversal.

We conclude this section by recalling a class of valid inequalities. An *oriented cycle* in G is a vector $\gamma \in \{-1, 0, 1\}^A$ such that $\{a \in A \mid \gamma_a \neq 0\}$ constitutes a cycle in the undirected graph arising from G by forgetting the arc orientations. We can decompose $\gamma = \gamma_+ - \gamma_-$ into its positive (forward) and negative (backward) part $\gamma_+ \in \{0, 1\}^A$ and $\gamma_- \in \{0, 1\}^A$, respectively.

▶ **Theorem 2** ([23]). *Let (G, T, ℓ, u, w) be a PESP instance. Let $\gamma \in \{-1, 0, 1\}^A$ be an oriented cycle in G. Then the* cycle inequalities

$$
\left\lceil \frac{\gamma_+^\top \ell - \gamma_-^\top u}{T} \right\rceil \le \frac{\gamma^\top(y + \ell)}{T} \le \left\lfloor \frac{\gamma_+^\top u - \gamma_-^\top \ell}{T} \right\rfloor
\tag{2}
$$

are valid for all feasible periodic slacks y.

Since the rows of Γ are composed of oriented cycles, the cycle inequalities (2) may also be used to determine bounds on the cycle offset variables z in (1).

3 Forward Cycle Bases

3.1 Motivation

Let us first highlight a few observations about the mixed-integer program (1) and the cycle inequalities (2):

- The overall objective function in (1) is composed of the objective values that arise on the individual arcs.
- The contribution of one individual arc a to the objective function is a monotonic increasing function of the slack y_a.
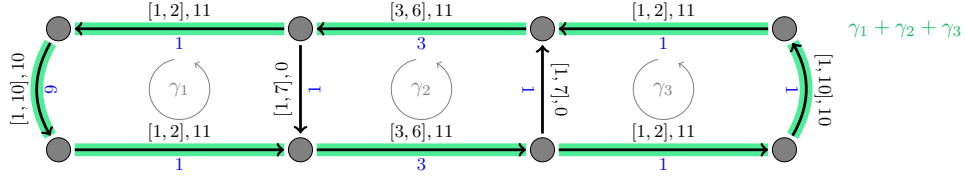
Figure 1 Example event-activity network with period time $T = 10$ and labels $\xrightarrow[x_a]{[\ell_a, u_a], w_a}$.

- The cycle inequalities (2) are defined for oriented cycles and involve the slack variables of their forward arcs with coefficient $+1$, and the slack variables of their backward arcs with coefficient -1.
- As a consequence, along a forward arc, the rounding benefit in the lower bound in (2) can force the slack y_a to increase and hereby immediately contribute to increasing the objective value. This is the right direction when the dual bound is supposed to increase.
- In contrast, along a backward arc, increasing its slack is relative to the upper bound u_a: If the value $u_a - (\ell_a + y_a)$ increases, then y_a decreases – and so does the objective function, which is *not* the actual intention.
- Hence, if a lower bound cycle inequality (2) enforces slack to be added, this risks to stay without any effect on the dual bound, whenever slack values along forward and backward arcs cancel out due to their opposite signs.

Thus, in order to identify strong lower bounds for the objective function – hence impose lower bounds on the slack values of groups of "expensive" arcs – we propose to also strive in particular for such cycle inequalities that prevent large slack values to just cancel out, hereby not contributing to the objective values. This is the case if the slack values of *all* arcs of the cycle γ appear in the valid inequality with the same sign, i.e., if the arcs of the cycle γ are forward arcs, i.e., if γ is a directed circuit.

We illustrate the possible benefit of considering the cycle inequalities for directed circuits in the example in Figure 1. Here, we set the period time $T = 10$. The graph G is planar and we consider the three oriented circuits γ_1, γ_2, and γ_3 that constitute the three finite faces of G (thus, a 2-basis or planar basis [11]), where all of them are oriented counter-clockwise. Moreover, we are going to deal with linear combinations of them, e.g., $\gamma_1 + \gamma_2 + \gamma_3$ is the outer oriented circuit, which is marked in green in Figure 1. This directed circuit represents the circulation of the trains of one line: One direction consists of the three bottom arcs, the opposite direction of the three top arcs. The two arcs in the terminus stations having the feasible interval $[1, 10]$ model the turnaround activities of the trains.

The two remaining arcs in the center region could, for instance, model single-track requirements [13]. Notice that such a configuration of single tracks close to both endpoints of a line could, e.g., be found at Line S2 of the Berlin S-Bahn network.

In order to ensure efficient operations, a major goal for a timetable in this example network is to operate the line with as few train units as possible. Equivalently, we minimize each slack time of any arc that models an activity of a vehicle. This applies to the arcs of $\gamma_1 + \gamma_2 + \gamma_3$ and we put weight $w_a = 10$ to these vehicle arcs. Moreover, slack time on any of the six non-turnaround vehicle arcs also involves extra waiting time for the passengers on board the train. Hence, for these six arcs, we even slightly increase their weights to $w_a = 11$. The two single-track arcs do not show any penalty weight, thus $w_a = 0$.

Now, let us consider the mixed-integer program (1) as well as its LP relaxation. On the one hand, in the integer optimum solution in particular the tension values $x = y + \ell$ along the outer cycle $\gamma_1 + \gamma_2 + \gamma_3$ have to sum up to an *integer* multiple of the period time. Indeed, one optimal solution reads

◼ **Table 1** Overview of the bounds of the cycle inequalities of the PESP instance in Figure 1.

Oriented circuits	lower bound	upper bound	integer values	forward	minimum arc weight
γ_1, γ_3	$\left\lceil \frac{1+1+1-7}{10} \right\rceil = 0$	$1 = \left\lfloor \frac{2+10+2-1}{10} \right\rfloor$	2	no	0
γ_2	$\left\lceil \frac{3+1+3+1}{10} \right\rceil = 1$	$2 = \left\lfloor \frac{6+7+6+7}{10} \right\rfloor$	2	yes	0
$\gamma_1 + \gamma_2$, $\gamma_2 + \gamma_3$	$\left\lceil \frac{1+1+3+1+3+1}{10} \right\rceil = 1$	$3 = \left\lfloor \frac{10+2+6+7+6+2}{10} \right\rfloor$	3	yes	0
$\gamma_1 + \gamma_2 + \gamma_3$	$\left\lceil \frac{12}{10} \right\rceil = 2$	$4 = \left\lfloor \frac{40}{10} \right\rfloor$	3	yes	10

$$x_a = \begin{cases} \ell_a + 8 = 9 & \text{for the westernmost arc } a, \\ \ell_a & \text{otherwise.} \end{cases}$$

On the other hand, for the LP relaxation it is well-known that its trivial optimal solution is just $x = \ell$ and $z = \frac{1}{T} \cdot (\Gamma x)$. Hence, we want to add valid inequalities in order to cut off this trivial fractional solution – and get as close as possible to the integer optimal solution.

One immediate way of doing so is to carefully select the cycle basis B in (1), and then to add the cycle inequalities (2) for the basic cycles. Traditionally, the choice of the cycle basis has been mainly motivated by the goal to keep as few as possible values for the integer variables z [14].

On this particular instance, it turns out that the three oriented cycles γ_1, γ_2, and γ_3 are the only ones that constrain their corresponding integer variables to only two values, see Table 1. Hence, these are the most attractive cycles in the well-established approach of allowing only few integer values for the integer variables. But when adding their cycle inequalities to the LP relaxation, its trivial optimum solution value persists. The same holds for five further valid inequalities. Only when adding the lower bound cycle inequality (2) for the directed circuit $\gamma_1 + \gamma_2 + \gamma_3$, the trivial optimum solution is cut off – and the dual bound is even pushed immediately to the integer optimum value. However, well-established separation heuristics [2, 16] are not guaranteed to consider this specific cycle, but we want to profit from its impact on the dual bound.

Hence, we propose to consider in particular the following simple oriented cycles for the separation of cycle inequalities, and for finding an integral cycle basis for (1):

1. heavy cycles, whose smallest weight is maximum,

2. forward cycles, i.e., without any backward arcs.

Let us shortly discuss why these two properties seem to be promising. Imagine that a lower bound of a cycle inequality implies a certain amount of slack to be distributed among the arcs of the cycle. Then there might be only little effect on the dual bound for the objective value if the slack can be concentrated on arcs that have only very small weight – there, the unavoidable slack somehow escapes[2] the objective function. Similarly, in the presence of backward arcs, on these, slack is relative to the *upper* bounds u_a of these arcs. Rounding effects in the cycle inequalities (2) thus tend to push the slack away from its upper bound, hence increase $u_a - (\ell_a + y_a)$, decrease y_a, and finally do *not* immediately support the dual bound to improve.

---

[2] Also during manual planning, it is a common saying that the *art* of periodic timetabling is to locate any unavoidable slack on those activities which are least important.

We will discuss theoretical properties of cycle bases exclusively composed of forward cycles in Section 3.2, and we will give a recipe to construct an integral forward cycle basis on a common type of event-activity networks in Section 3.3.

3.2   Theory of Forward Cycle Bases

Let $G = (V, A)$ be a digraph. The elements $\gamma \in \mathbb{Z}^A$ that satisfy flow conservation at every vertex, i.e.,

$$\forall v \in V : \quad \sum_{a \in \delta^+(v)} \gamma_a = \sum_{a \in \delta^-(v)} \gamma_a,$$

form an abelian group, the *cycle space* $\mathcal{C}$ of G. In this language, an *oriented cycle* is an element $\gamma \in \mathcal{C}$ with $\gamma_a \in \{-1, 0, 1\}$ for all $a \in A$; we write $a \in \gamma$ iff $\gamma_a \neq 0$. Arcs $a \in \gamma$ with $\gamma_a = 1$ and $\gamma_a = -1$ are called *forward* and *backward*, respectively. A *forward cycle* is an oriented cycle γ containing only forward arcs, i.e., $\gamma \in \mathcal{C} \cap \{0, 1\}^A$.

The rank of $\mathcal{C}$ is the *cyclomatic number* μ. A set B of μ oriented cycles is called

(1) a *cycle basis* of G if B is a basis of the $\mathbb{R}$-vector space $\mathcal{C} \otimes \mathbb{R}$,

(2) an *undirected cycle basis* of G if B is a basis of the $\mathbb{F}_2$-vector space $\mathcal{C} \otimes \mathbb{F}_2$,

(3) an *integral cycle basis* of G if B is a basis of the abelian group $\mathcal{C}$,

(4) a *weakly fundamental cycle basis* of G if the cycles in B can be ordered in such a way that for $i \in \{2, \ldots, \mu\}$, there is an arc $a \in \gamma_i$ with $a \notin \gamma_1 \cup \cdots \cup \gamma_{i-1}$,

(5) a *strictly fundamental cycle basis* of G if B is the set of fundamental cycles of some spanning forest F of G.

In this hierarchy of cycle bases, the implications $(5) \Rightarrow (4) \Rightarrow (3) \Rightarrow (2) \Rightarrow (1)$ hold [11].

▶ **Definition 3.** *A* forward cycle basis *is a cycle basis consisting exclusively of forward cycles.*

We prefer the term *forward cycles* over *directed cycles*, as the notion of *directed cycle basis* is already taken for a cycle basis consisting of arbitrary oriented cycles [11]. The existence of forward cycle bases is related to strong connectedness:

▶ **Theorem 4** ([26, 5]). *A digraph G has a forward cycle basis if and only if each 2-edge-connected component is strongly connected.*

In contrast to the general situation, even a strongly connected digraph does not necessarily admit a forward strictly fundamental cycle basis, see Example 10 in the appendix.

We now turn to the *minimum weight cycle basis problem*: Given arc weights $c \in \mathbb{R}_{\geq 0}^A$, find a cycle basis B such that its weight $c(B) := \sum_{\gamma \in B} \sum_{a \in \gamma} c_a$ is minimum. As mentioned in Section 3.1, finding minimum weight cycle bases has been proven useful for accelerating the branch-and-cut process in MIP solvers for (1). As $\mathcal{C} \otimes \mathbb{R}$ and $\mathcal{C} \otimes \mathbb{F}_2$ are vector spaces, the minimum weight cycle basis and minimum weight undirected cycle bases problems can be solved by the greedy algorithm on vector matroids. A polynomial-time algorithm can be constructed by restricting to a polynomially bounded set of cycles that are guaranteed to contain a minimum weight cycle basis, this is Horton's algorithm [10]. Recall that there are more efficient algorithms known [1]. For integral cycle bases, the complexity is unclear, whereas APX-hardness is known for the minimum weight weakly [24] resp. strictly [4] fundamental cycle basis problem .

▶ **Definition 5.** *Given $c \in \mathbb{R}_{\geq 0}^A$, the* minimum forward (undirected/integral/. . .) cycle basis problem *is to find a* forward *(undirected/integral/. . .) cycle basis B of minimum weight.*

Forward cycles in G together with sets of linearly independent forward cycles in $\mathcal{C} \otimes \mathbb{R}$ and $\mathcal{C} \otimes \mathbb{F}_2$ form a vector matroid, so that the greedy algorithm applies. Horton's algorithm can be adapted in such a way that it computes a minimum forward cycle basis or minimum forward undirected cycle basis in polynomial time [5].

However, we require *integral* cycle bases for the purpose of periodic timetabling. In practice, we observed that minimum (forward) undirected cycle bases are almost always integral, so that applying Horton's algorithm already solves the minimum weight integral cycle basis problem. We will also use this approach in Section 4.

3.3 ILTY cycles

We indicate the existence of forward integral cycle bases for event-activity networks with a special structure typical for periodic timetabling instances; we will call these networks *line-based*. A *line network* (N, L) is an undirected graph $N = (S, E)$ together with a set L of pairwise edge-disjoint simple paths whose union is E. We call S the set of *stations*, and L the set of *lines*.

▶ **Construction 6.** Given a line network (N, L), we construct a digraph G as follows:
(1) For each line $l \in L$ with sequence of stations $(s_1, \ldots, s_k)$, add
  (a) *driving arcs* $((s_i, \overrightarrow{l}, \text{dep}), (s_{i+1}, \overrightarrow{l}, \text{arr}))$ and $((s_{i+1}, \overleftarrow{l}, \text{dep}), (s_i, \overleftarrow{l}, \text{arr}))$
     for $i \in \{1, \ldots, k-1\}$,
  (b) *dwell arcs* $((s_i, \overrightarrow{l}, \text{arr}), (s_i, \overrightarrow{l}, \text{dep}))$ and $((s_i, \overleftarrow{l}, \text{arr}), (s_i, \overleftarrow{l}, \text{dep}))$
     for $i \in \{2, \ldots, k-1\}$,
  (c) *turnaround arcs* $((s_k, \overrightarrow{l}, \text{arr}), (s_k, \overleftarrow{l}, \text{dep}))$ and $(s_1, \overleftarrow{l}, \text{arr})), ((s_1, \overrightarrow{l}, \text{dep})$.
(2) For each station $s \in S$ and each pair (l, l') of distinct lines containing s, add *transfer arcs* $((s, \overrightarrow{l}, \text{arr}), (s, \overrightarrow{l'}, \text{dep}))$, $((s, \overrightarrow{l}, \text{arr}), (s, \overleftarrow{l'}, \text{dep}))$, $((s, \overleftarrow{l}, \text{arr}), (s, \overrightarrow{l'}, \text{dep}))$, and $((s, \overleftarrow{l}, \text{arr}), (s, \overleftarrow{l'}, \text{dep}))$.

(a) Line network (N, L).　　　　**(b)** Line-based event-activity network G.
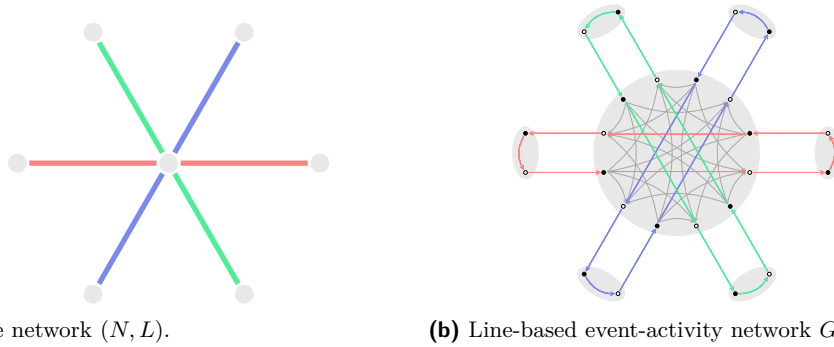
Figure 2 Example of Construction 6.

Construction 6 implicitly defines the vertices of G, each vertex is a triple consisting of a station in S, a line in L together with one of the direction markers $\leftarrow$ or $\rightarrow$, and the departure/arrival flag dep or arr. In particular, we can speak of the station or the line of a vertex, and conversely of vertices of a station or line. An example for Construction 6 on a star-shaped line network is depicted in Figure 2.

▶ **Definition 7.** *A digraph G is called* line-based *if it arises from a line network (N, L) via Construction 6.*

We will now define particularly simple types of a cycles in line-based digraphs.

▶ **Construction 8.** Let G be line-based network arising from $(N = (S, E), L)$, let $s \in S$ be a station. Construct the following forward cycles in G:

(1) *I-cycles* at s: forward cycles that contains no transfer arcs and at least one vertex of s,

(2) *L-cycles* at s: exactly two transfer arcs, both at s, and no dwell arcs at s,

(3) *T-cycles* at s: exactly two transfer arcs, both at s, and exactly one dwell arc at s,

(4) *Y-cycles* at s: exactly three transfer arcs, all of them at s, and no dwell arcs at s.

Intuitively, an I-cycle contains all driving, dwell and turnaround arcs associated to a single line. L- and T-cycles connect two lines, and Y-cycles connect three lines, all of them make use of transfers only at a single station. This family of cycles is depicted in Figure 3.

(a) *I*-cycle: two dwell arcs.

(b) *L*-cycle: two transfer arcs.

(c) *T*-cycle: one dwell arc, two transfer arcs.

(d) *Y*-cycle: three transfer arcs.
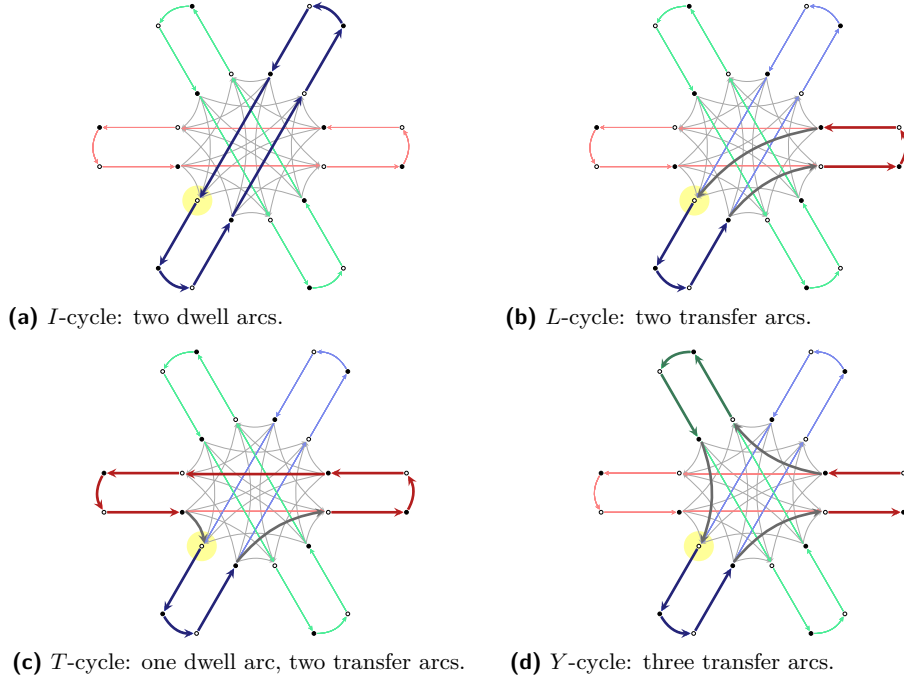
Figure 3 $ILTY$ cycles for the event-activity network in Figure 2 passing through a fixed vertex (marked in yellow).

The following is our main theorem about $ILTY$ cycles, due to length restrictions, we refer to [18] for a proof.

▶ **Theorem 9.** *Let $(N = (S, E), L)$ be a line network, defining G via Construction 6. At each station $s \in S$, fix a vertex v of G at s, and let B_s denote the set of $ILTY$ cycles at s passing through v. Then there is a set B' of forward cycles in G such that:*

(1) *B' projects to a strictly fundamental cycle basis of N,*

(2) *$\bigcup_{s \in S} B_s$ is a weakly fundamental basis of the subspace of the cycle space of G generated by all $ITLY$ cycles,*

(3) *$B' \cup \bigcup_{s \in S} B_s$ is an integral cycle basis of G.*

4 Turnarounds in `R1L1`

The railway instances `RxLy` of the PESPlib are similarly structured [7, 17]. We will analyze the smallest instance `R1L1`, and to some extent reverse-engineer an underlying line network. The outcome is a modified instance `R1L1v`, where certain turnaround arcs have been added.

After removing four arcs with lower and upper bound 0, the network of `R1L1` becomes bipartite, vertices can hence be partitioned into arrival and departure events. This vertex labeling can be done in such a way that free arcs, i.e., arcs a with $u_a - \ell_a \geq T - 1 = 59$, always originate at an arrival and end at a departure. We interpret these arcs as transfer arcs. The remaining arcs a going from arrivals to departures have all $[\ell_a, u_a] = [1, 5]$, we take them as dwell arcs. We view the arcs from departures to arrivals as driving arcs. In fact, the network can be seen as a subnetwork of a line-based network as defined in Section 3.3.

If we now remove all transfer arcs, then the network decomposes into 110 directed simple paths, alternatingly consisting of driving and dwell arcs. We observe that for each such path, we find a path using exactly the same sequence of lower and upper bounds for the driving arcs in reverse order. Actually, this complies with the ordering of the activities as given in the text file describing the instance. This leads to a perfect matching of the directed paths, which we use to create 55 bidirectional lines. In the spirit of Construction 6, we do this by adding, in total, 110 turnaround arcs at both ends for each line. We call the resulting network `R1L1v`, where v stands for "vehicle", as vehicle turnarounds at the line ends have been modeled. `R1L1v` satisfies the hypothesis of Theorem 4, so that it admits a forward cycle basis. We will computationally evaluate several combinations of bounds and weights for the turnaround arcs and compare four cycle bases for `R1L1v` in Section 5.

5 Computational Results

For the network `R1L1v` described in Section 4, we compare four minimum turnaround times, seven weights (see Section 5.1.1), and four cycle bases, three of which are forward (see Section 5.1.2). Combining these, we obtain $4 \cdot 7 \cdot 4 = 112$ scenarios in total. We attack these scenarios with five primal strategies and one dual strategy (see Section 5.1.3) within the `ConcurrentPESP` solver [3] that computed the currently best primal and dual bounds for all PESPlib instances [6]. `ConcurrentPESP` invokes Gurobi 9.1 [9] as MIP solver. The computations were performed on an Intel Xeon E3-1270 CPU running at 3.80 GHz with 32 GB RAM, using up to 8 threads, with a wall time limit of 1 hour for each scenario.

5.1 Detailed Set-up

5.1.1 Bounds and Weights

Table 2 Parameters for turnaround arcs. The maximum weight in the original `R1L1` instance is 72 523, the arithmetic mean weight is 7 388.

Parameter	Values
minimum turnaround times ℓ_a	0, 5, 10, 15
turnaround weights w_a	0, 2500, 5000, 10000, 20000, 40000, 80000

The arc set of `R1L1v` consists of the arc set of `R1L1` plus 110 new turnaround arcs. For the turnaround arcs, we choose the same lower bound and the same weight from the values in Table 2, creating $4 \cdot 7 = 28$ PESP instances. All turnaround arcs are introduced as free arcs, so that we set $u_a := \ell_a + 59$. For the other arcs, we keep the bounds and weights as in `R1L1`. However, we also add the turnaround weight to all vehicle-related arcs, i.e., all driving and dwell arcs, in order to reflect the full vehicle rotation in the objective function.

5.1.2   Cycle Bases

We will consider the following four cycle bases on `R1L1v`:

(B1) `span`: a minimum integral cycle basis w.r.t. the cost $c_a := u_a - \ell_a$ for $a \in A$,

(B2) `forward span`: a minimum forward integral cycle basis w.r.t. $c_a := u_a - \ell_a$ for $a \in A$,

(B3) `forward bottleneck`: a forward integral cycle basis B maximizing $\sum_{\gamma \in B} \min_{a \in \gamma} w_a$,

(B4) `ILTY`: a forward integral cycle basis consisting of as many *ILTY* cycles as possible.

The minimum `span` basis (B1) is the classical approach to minimize the number of values the integer variables in (1) can attain. We motivated (B2) and (B3) in Section 3.1, (B4) comes from the construction in Section 3.3. Cycle bases (B1)-(B3) can be computed by (a modification of) Horton's algorithm [10], the result always turns out to be not only a minimum undirected cycle basis, but also integral. Note that unlike the other bases, (B3) needs to be recomputed for every choice of weight for the turnaround arcs, and we need to consider bottleneck shortest paths for Horton's algorithm. For (B4), as `R1L1v` is only a subnetwork of a line-based network, we follow an analytic approach instead of the synthetic one pursued in Construction 8 to construct *ILTY* cycles: We seek for single lines, pairs and triples of lines, and construct all possible *ILTY*-shaped cycles whenever they exist in the network. This produces a subspace of the cycle space of dimension 2823, however, its codimension is only 9. We therefore complete the *ILTY* cycles with 9 cycles from the `forward span` basis.

5.1.3   Solution Strategies

■ **Table 3** Solution strategies for the computational experiments.

Strategy	MIP	Initial solution	Ignore light arcs	Other heuristics
`complete`	✓		✓	✓
`mip`	✓			
`mip-start`	✓	✓		
`mip-ignore`	✓		✓	
`mip-ignore-start`	✓	✓	✓	
`dual`	✓	✓		

As `ConcurrentPESP` contains a variety of PESP algorithms, we singled out 6 solution methods, summarized in Table 3. As we want to compare cycle bases, we concentrate on the MIP component of the solver. For `mip-start`, `mip-ignore-start`, and `dual`, the current PESPlib incumbent for `R1L1` with weighted slack 29 894 745 [6, 17] is given the solver as an initial solution. Ignoring light arcs means to solve PESP on a subnetwork (with the same strategy) as described in [7, 3]. Other heuristics include, e.g., the modulo network simplex method [22, 8]. The emphasis for `mip-start` lies on the primal bound, whereas `dual` focuses on improving the dual bound, and additionally invokes flip cutting planes [16].

5.2   Results

Figure 4 displays the results of our computations in the Pareto sense for the weighted slack and the number of vehicles as objectives. The weighted slack is computed on the original instance `R1L1`, i.e., omitting turnaround arcs, and hence is the passenger-oriented component. The number of vehicles is obtained as the sum of the driving, dwell and turnaround times (periodic tensions) for each line divided by the period time $T = 60$. Clearly, higher turnaround times require more vehicles.
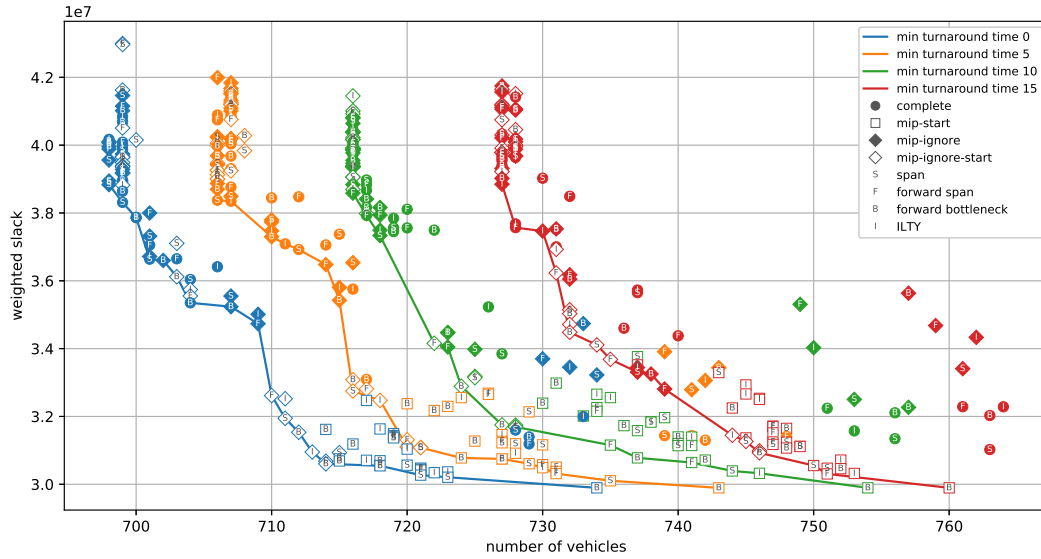
Figure 4 Overview of all results for the strategies `complete`, `mip-start`, `mip-ignore`, and `mip-ignore-start`. The solid lines depict the Pareto front for the four minimum turnaround times. The marker shape indicates the strategy, the inscribed letter stands for the cycle basis. A higher resolution version of the chart is available at `https://www.zib.de/lindner/R1L1v-hi-res.pdf`.

For the `mip` strategy, which runs Gurobi without an initial solution and no further help from other heuristics, *all* solutions have a weighted slack of more than 48 000 000, so that the solution quality is much worse than for the other strategies. For the other strategies, the picture is quite diffuse: Each of the other four primal strategies and each of the cycle bases produces at least once a non-dominated solution, and this holds even for each minimum turnaround time individually. However, there is a tendency that starting with the PESPlib incumbent maintains the small weighted slack, while starting without initial solution brings the number of vehicles down.

To assess the impact of the cycle bases, we compare these for each strategy with a hypothetical cycle basis that attains the best weighted slack (or number of vehicles) among all four cycle bases for each instance with that strategy. Table 4 reports the relative gap in percent, averaged over all 28 combinations of lower bounds and weights.

Table 4 Average relative gaps in percent. The first column per basis is for the weighted slack, the second column is for the number of vehicles. E.g., 2.00 for `complete` in the leftmost `span` column means that the `span` basis produce a weighted slack that is in average 2% worse than a hypothetical cycle basis that takes the best slack among all four cycle bases with the `complete` strategy.

strategy	span		fwd bottleneck		forward span		ILTY	
`complete`	2.00	0.13	2.16	0.12	2.59	0.14	2.32	0.17
`mip`	0.12	0.07	5.47	0.44	3.42	0.24	4.19	1.05
`mip-start`	1.16	0.57	1.73	0.12	0.90	0.43	1.89	0.48
`mip-ignore`	1.26	0.12	1.49	0.12	2.15	0.05	1.49	0.09
`mip-ignore-start`	1.32	0.08	1.05	0.06	1.53	0.06	1.27	0.05
`dual`	17.66		4.81		3.35		1.44	

It turns out that the traditional minimum span basis, which is allowed to contain backward arcs, performs best in terms of weighted slack for the strategies that were not given a high quality timetable as input. For the other strategies, a forward basis was better. However, with the exception of the `mip` strategy, the quality differences are mostly only minor, and the same holds for the number of vehicles. We can conclude that on the primal side, forward cycle perform similar to the minimum span basis.

What is however striking is the dual side: After the time limit of one hour, the minimum span basis is far worse than all forward cycle bases, and the `ILTY` basis is the clear winner.

Note that the best lower bound on the number of vehicles is quickly obtained by summing up the lower bound of each cycle inequality (2) for the *I*-cycles of each line.

5.3   A New Dual Incumbent for `R1L1`

Motivated by the dual performance of the forward cycle basis, we try to compute a new dual bound for the original PESPlib instance `R1L1`. When the minimum turnaround time is 0 and the turnaround weights are 0, every timetable for `R1L1` is feasible for `R1L1v` and vice versa, and the weighted slacks do not change. In this setting, we can hence use the forward cycle bases on `R1L1v` to compute dual bounds for `R1L1`. One can check that forward cycle bases do not exist on `R1L1`, as it fails to satisfy the hypothesis of Theorem 4.

■ **Table 5** A new dual incumbent for `R1L1` through `R1L1v`.

instance	cycle basis	dual bound
`R1L1v`	span	20 638 013
`R1L1v`	forward span	20 609 801
`R1L1v`	forward bottleneck	20 591 564
`R1L1v`	ILTY	20 901 883
`R1L1`	span	20 693 118

Table 5 compares the dual bounds after 24 hours wall time on up to 6 threads obtained by the four cycle bases on `R1L1v`, and additionally by the `span` basis on `R1L1`. As bounds basically stop moving after one hour in Gurobi, we switched to CPLEX 12.10 for this particular computational experiment, which performs better in the long run. We again use flip inequalities as source for cutting planes. Although the `span` basis becomes better, `ILTY` on `R1L1v` produces the best dual bound, although `R1L1v` is a larger instance and the cycle space dimension has increased by 110.

6   Conclusion and Outlook

Minimum forward cycle bases are competitive when seeking high quality solutions to PESP instances, and superior when computing of dual bounds. The *ILTY* cycles, that can be constructed on line-based event-activity networks, are particularly strong: They allow for better dual bounds on the PESPlib instance `R1L1`, although the computation is carried out on the larger `R1L1v`. The natural question is whether this strength extends to other PESP instances. So far the PESPlib does not contain instances with specified turnarounds. We submitted a realization of `R1L1v` with minimum turnaround time 10 and turnaround weight 5000, and this instance is now part of the PESPlib. A similar procedure is possible for all `RxLy` PESPlib instances.

References

1   E. Amaldi, C. Iuliano, T. Jurkiewicz, K. Mehlhorn, and R. Rizzi. Breaking the $o(m^2n)$ barrier for minimum cycle bases. In A. Fiat and P. Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2009. `doi:10.1007/978-3-642-04128-0_28`.

2   R. Borndörfer, H. Hoppmann, M. Karbstein, and N. Lindner. Separation of cycle inequalities in periodic timetabling. *Discrete Optimization*, 35:100552, 2020. `doi:10.1016/j.disopt.2019.100552`.

3   R. Borndörfer, N. Lindner, and S. Roth. A concurrent approach to the Periodic Event Scheduling Problem. *Journal of Rail Transport Planning & Management*, 15:100175, 2020. Best Papers of RailNorrköping 2019. `doi:10.1016/j.jrtpm.2019.100175`.

4   G. Galbiati, R. Rizzi, and E. Amaldi. On the approximability of the minimum strictly fundamental cycle basis problem. *Discrete Applied Mathematics*, 159(4):187–200, 2011. `doi:10.1016/j.dam.2010.10.014`.

5   P. M. Gleiss, J. Leydold, and P. F. Stadler. Circuit bases of strongly connected digraphs. *Discussiones Mathematicae Graph Theory*, 23(2):241, 2003. `doi:10.7151/dmgt.1200`.

6   M. Goerigk. PESPlib – A benchmark library for periodic event scheduling, 2012. URL: `http://num.math.uni-goettingen.de/~m.goerigk/pesplib/`.

7   M. Goerigk and C. Liebchen. An improved algorithm for the periodic timetabling problem. In *ATMOS*, volume 59 of *OASICS*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

8   M. Goerigk and A. Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers & Operations Research*, 40(5):1363–1370, 2013.

9   Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL: `https://www.gurobi.com`.

10   J. D. Horton. A polynomial-time algorithm to find the shortest cycle basis of a graph. *SIAM J. Comput.*, 16(2):358–366, 1987. `doi:10.1137/0216026`.

11   T. Kavitha, C. Liebchen, K. Mehlhorn, D. Michail, R. Rizzi, T. Ueckerdt, and K. A. Zweig. Cycle bases in graphs characterization, algorithms, complexity, and applications. *Computer Science Review*, 3(4):199–243, 2009. `doi:10.1016/j.cosrev.2009.08.001`.

12   C. Liebchen. *Periodic timetable optimization in public transport*. PhD thesis, Technische Universität Berlin, 2006.

13   C. Liebchen and R. H. Möhring. The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables — and Beyond. In F. Geraets, L. Kroon, A. Schöbel, D. Wagner, and C. D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, Lecture Notes in Computer Science, pages 3–40, Berlin, Heidelberg, 2007. Springer. `doi:10.1007/978-3-540-74247-0_1`.

14   C. Liebchen and L. Peeters. Integral cycle bases for cyclic timetabling. *Discrete Optimization*, 6(1):98–109, 2009. `doi:10.1016/j.disopt.2008.09.003`.

15   C. Liebchen and E. Swarat. The Second Chvatal Closure Can Yield Better Railway Timetables. In M. Fischetti and P. Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, volume 9 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 2190-6807. `doi:10.4230/OASIcs.ATMOS.2008.1580`.

16   N. Lindner and C. Liebchen. Determining All Integer Vertices of the PESP Polytope by Flipping Arcs. In D. Huisman and C. D. Zaroliagis, editors, *20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020)*, volume 85 of *OpenAccess Series in Informatics (OASIcs)*, pages 5:1–5:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 2190-6807. `doi:10.4230/OASIcs.ATMOS.2020.5`.

17   N. Lindner and C. Liebchen. Timetable Merging for the Periodic Event Scheduling Problem. Technical Report 21-06, Zuse Institute Berlin, 2021. URL: `https://nbn-resolving.org/urn:nbn:de:0297-zib-81587`.

18   N. Lindner, C. Liebchen, and B. Masing. Constructing forward integral cycle bases for periodic timetabling, in preparation.

19   T. Lindner. *Train Scheduling in Public Rail Transport*. PhD thesis, Technische Universität Braunschweig, 2000.

20   K. Nachtigall. Cutting Planes for a Polyhedron Associated with a Periodic Network. Technical Report 112-96/17, Deutsches Zentrum für Luft- und Raumfahrt, 1996.

21   K. Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables*. Habilitation thesis, Universität Hildesheim, 1998.

22   K. Nachtigall and J. Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In Matteo Fischetti and Peter Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, volume 9 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

23   M. A. Odijk. Construction of periodic timetables, part 1: A cutting plane algorithm. Technical Report 94-61, TU Delft, 1994.

24   R. Rizzi. Minimum Weakly Fundamental Cycle Bases Are Hard To Find. *Algorithmica*, 53(3):402–424, 2009. `doi:10.1007/s00453-007-9112-8`.

25   P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.

26   P. Seymour and C. Thomassen. Characterization of even directed graphs. *Journal of Combinatorial Theory, Series B*, 42(1):36–45, 1987. `doi:10.1016/0095-8956(87)90061-X`.

A   Non-Existence of Forward Strictly Fundamental Cycle Bases

(a) Graph.   **(b)** γ_1.   **(c)** γ_2.   **(d)** γ_3.   **(e)** γ_4.   **(f)** γ_5
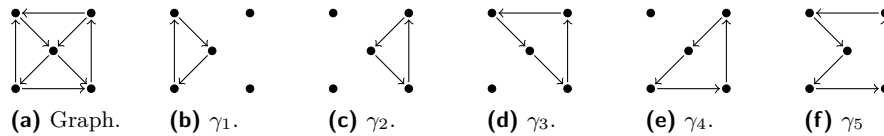
Figure 5 Strongly connected graph without a forward strictly fundamental cycle basis.

▶ **Example 10.** Consider the strongly connected digraph G in Figure 5a. There are five forward cycles $\gamma_1, \ldots, \gamma_5$ (cf. Figure 5), note that $\gamma_2 + \gamma_5 = \gamma_3 + \gamma_4$. The cyclomatic number of G is 4, and the combinations of three of the cycles $\gamma_2, \ldots, \gamma_5$ with γ_1 form all forward cycle bases. A spanning tree F of G has four co-tree arcs, in a strictly fundamental cycle basis for F, each co-tree arc must appear in exactly one cycle. However, in any forward cycle basis, we find at most three exclusive arcs, so that no forward cycle basis is strictly fundamental.

Towards Improved Robustness of Public Transport by a Machine-Learned Oracle

Matthias Müller-Hannemann ✉ 🆔
Martin-Luther-Universität Halle-Wittenberg, Germany

Ralf Rückert ✉
Martin-Luther-Universität Halle-Wittenberg, Germany

Alexander Schiewe ✉ 🆔
TU Kaiserslautern, Germany

Anita Schöbel ✉ 🆔
Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM, Kaiserslautern, Germany
TU Kaiserslautern, Germany

—— Abstract ——

The design and optimization of public transport systems is a highly complex and challenging process. Here, we focus on the trade-off between two criteria which shall make the transport system attractive for passengers: their travel time and the robustness of the system. The latter is time-consuming to evaluate. A passenger-based evaluation of robustness requires a performance simulation with respect to a large number of possible delay scenarios, making this step computationally very expensive.

For optimizing the robustness, we hence apply a machine-learned oracle from previous work which approximates the robustness of a public transport system. We apply this oracle to bi-criteria optimization of integrated public transport planning (timetabling and vehicle scheduling) in two ways: First, we explore a local search based framework studying several variants of neighborhoods. Second, we evaluate a genetic algorithm. Computational experiments with artificial and close to real-word benchmark datasets yield promising results. In all cases, an existing pool of solutions (i.e., public transport plans) can be significantly improved by finding a number of new non-dominated solutions, providing better and different trade-offs between robustness and travel time.

1 Introduction

The design and planning of public transport systems is a challenging, multi-faceted optimization problem. Given an infrastructure network (stations and direct connections), a line concept (a set of lines with corresponding frequencies), and a passenger demand (origin-destination pairs between which passengers wish to travel), we here focus on the integrated optimization of a timetable with a corresponding vehicle schedule. The resulting public transport plan shall be cost-efficient, attractive to passengers, and robust against different types of disturbances. As usual in multi-criteria optimization, we are especially interested in finding non-dominated solutions, i.e. solutions for which no other solution exists which is at least as good in all criteria and strictly better in at least one.

In recent years, many different robustness concepts have been proposed, for recent surveys see [12, 16]. All these concepts compute the robustness of public transport systems differently, but as was, e.g., stressed in [21], considering the passengers when evaluating the robustness

is very important. To do that, there are several approaches, where especially the effect of disruptions on passengers are examined, see e.g., [4, 5, 8, 9, 15]. But such an assessment requires extensive performance simulations with respect to a large number of scenarios, each measuring the effect of specific delays on the passengers. Already a single robustness simulation is computationally very expensive, and therefore such an approach is hard to use within an iterative optimization framework. This motivates the design of a much faster, scenario-based robustness approximation by using methods from machine learning. In a recent paper, we developed an efficient oracle for the estimation of the robustness of a public transport plan by training an artificial neural network [19]. Based on only a few key features of a public transport plan, the trained neural network can be used as a black box to instantly predict the robustness of a previously unseen public transport plan with high accuracy.

Contribution.   The long-term goal of this work is to provide an improved methodology for the planning of robust, but still efficient and attractive public transport systems. Here, we develop two algorithmic approaches, a local search and a genetic algorithm, both using the robustness oracle to iteratively compute competitive solutions w.r.t. passenger quality and robustness. First, we apply this oracle as a black box for increasing the robustness of a given timetable and a corresponding vehicle schedule by local search. We propose several alternative definitions of neighborhoods for this local search. Second, we develop a genetic approach that also uses the oracle to speed up the computations. Our experimental results are encouraging:

1. Local search, trying to improve robustness succeeds in most cases without worsening the average perceived travel time by too much.
2. More specifically, local search applied to a pool of non-dominated instances generates many new non-dominated solutions, thereby improving our approximation of the Pareto front significantly.
3. Similarly, our genetic approach shows clear improvements from a given starting population of solutions within few rounds.

Related work.   Public transport planning consists of several stages that are traditionally solved sequentially. For this paper, we are considering *public transport systems*, namely a line plan, a timetable and a vehicle schedule. For an overview of line planning, see Schöbel [26], for an overview of timetabling, see Lusby et al. [17], and for an overview of vehicle scheduling, see Bunte and Kliewer [3].

There are several robustness concepts in literature. Due to the increase in complexity, methods for finding timetables based on these concepts often use heuristics, see e.g., Polinder et al. [23] or Pätzold [22] for recent approaches. For an overview of robust timetabling, see Lusby et al. [16]. Related to robust timetable creation is delay management, where trains are rescheduled in specific delay scenarios. For an overview of delay management, see Dollevoet et al. [6] and König [14]. Here, we consider the delay management strategy to be fixed and implicitly learnt by the robustness oracle. Hence, our used robustness evaluations are applicable to any given delay management strategy.

Both local search and genetic algorithms are used extensively in public transport research. See e.g., [10, 11, 13, 28] for local search applications and [1, 20, 27] for usages of a genetic algorithm approach. To our knowledge this paper is the first to use both approaches in the context of passenger flow-based robustness of timetables. There are other approaches of using machine learning (ML) in the optimization of public transport systems. For example, Matos et al. [18] use reinforcement learning for the optimization of periodic timetables and

Bauer and Schöbel [2] develop an approach to learn the quality of a connection in delay management. As far as we know, ML has not been used for robustness optimization in public transport before.

Overview.   The remainder of this paper is structured as follows. In Section 2, we provide background information and basic notions used in this paper about public transport systems, sketch the evaluation of robustness, and briefly describe the machine learning approach by which we create an oracle for fast robustness evaluation. Then, Section 3 describes our local search framework and the different sets of neighborhoods used for optimization. In Section 4, we introduce a second approach based on a genetic algorithm. Experimental results are presented and discussed in Section 5. Finally, we conclude with an outlook.

2   Background: Public Transport, Robustness and Machine Learning

2.1   Public Transport Systems

To present our algorithmic approaches, we first need to clearly define the structures we are working with. For all our algorithms mentioned below, we will assume that an infrastructure network, a line concept and a passenger demand are given and cannot be changed. Here, a line concept is a set of paths through the infrastructure network, each with a frequency, i.e., a number of times the line should be served per planning period. We will call the infrastructure network with a given line concept and passenger demand a *dataset*. Overall, we want to determine the robustness of *instances*, i.e., a dataset combined with a timetable, a vehicle schedule and a corresponding set of passenger routes. As a basic underlying model, we use an event-activity network $(\mathcal{E}, \mathcal{A})$ with events $\mathcal{E}$, representing the departures and arrivals of vehicles at stops, and activities $\mathcal{A}$ between them. For activities, we are considering *drive*, *wait* and *turnaround* activities to model the vehicle behavior and *change* activities for transferring of passengers. Additionally, we assume that for every activity $a \in \mathcal{A}$ a lower bound l_a and an upper bound u_a are given, determining the feasibility of the timetable. Depending on the context, different sets of change activities will be considered. For timetable construction, we consider as change activities a small set of important transfer possibilities which shall be guaranteed. Afterwards, i.e., for evaluating a timetable, we allow passengers to use all possible transfers as change activities. A feasible *timetable* now assigns a time π_e to each event $e \in \mathcal{E}$ such that the duration d_a of every activity $a \in \mathcal{A}$ stays within the given bounds. We are considering both periodic and aperiodic timetables, depending on the algorithm used.

A *trip* is a path of drive and wait activities in the event-activity-network that needs to be operated by a single vehicle. The *vehicle schedule* is a collection of vehicle tours, each covering a set of trips. To be feasible, each trip in the event-activity network needs to be covered exactly once and the corresponding turnaround activities are feasible w.r.t. their bounds, i.e., the last event of a trip and the first event of the consecutive trip by the same vehicle have enough time between them, e.g., to drive from one station to the other.

The last objects to consider are the *passenger routes*. We use the given passenger demand data, with corresponding earliest departure times for each passenger, to determine a realistic passenger routing. To achieve this, scarce vehicle capacities are also very important. Each passenger chooses a route that optimizes a utility function. For this, we use a model where the passengers are searching for their shortest paths w.r.t. the perceived travel time, i.e., a weighted sum of travel time and the number of transfers (e.g. by counting 5 minutes per transfer), while respecting the capacity of the vehicles. Conflicts are resolved using seat-reservation in a first come first serve order, i.e., once a passenger chooses her path the

■ **Table 1** Robustness tests RT-1-RT-4 with a description and a motivation, as well as the parameters used in our experiments.

name	description	motivation	parameter for paper
RT-1	initial delay of a single vehicle	emulates problems at the beginning of a trip	source delays of 5 minutes
RT-2	slow-down of single network sections	emulates problems like road work	increase of travel time of section by 2 minutes
RT-3	temporary blocking of single station	emulates a gridlock at a station	blocking of 15 minutes
RT-4	random delay simulation	emulates multiple common independent delays	empirical distribution of delays based on [9]

corresponding capacity in the vehicle is guaranteed. In our experiments, this model is much faster than a more complex simulation involving capacity checks when boarding a vehicle but provides nearly the same results. See [19] for a more detailed discussion of these models.

2.2   Robustness

The evaluation of the robustness of public transport networks we use is based on Friedrich et al. [8, 9]. With a simulation framework, we conduct several robustness tests simulating certain aspects of common disturbances during daily operation. During this process, we measure the arrival times of all passengers compared to their initially planned arrival time. The sum of these differences for all passengers provides the *robustness value* of the simulation. This value serves as an orientation for comparing public transport plans using the same passenger demand. For better interpretability and comparability, however, we normalize the robustness values for a set of known instances to a scale from 0 to 100 where 100 is the worst instance. Hence, smaller robustness values are better.

In [9], we give a very detailed description of all aspects of the four tests we use here. To better illustrate this method we now give a detailed explanation of the first robustness test RT-1. The task of RT-1 is to simulate the total effect that starting delays have on the schedule. The delay caused when the first departure of a vehicle is not on time is a common occurrence in daily operation. To evaluate this metric, RT-1 creates a separate scenario where each vehicle has a delay of x minutes and all other vehicles are on time. The sum of all passenger delays at their destination is the final result of RT-1. The result of this test is deterministic but highly dependent on the parameters specifying the passenger and the delay management models. So if, for example, each vehicle waits for transferring passengers or passengers neglect maximum vehicle capacities this produces another specific robustness value. In Table 1, we provide a brief description of the considered robustness tests.

2.3   Robustness Estimation by Machine Learning

Conducting the four robustness tests mentioned in the last section is computationally expensive. In an optimization scenario where parts of an instance are altered we want to know the effect on the robustness value as quickly as possible. To this end, we want to approximate the real robustness by using an oracle as a predictor based on machine learning. In [19] we first introduced such an oracle and evaluated its approximation performance. In this section, we will briefly explain how this oracle is created and how we use it for robustness approximation. First, we give a short overview of how the process works, explaining the most important steps. The creation of the oracle can be done in four steps:
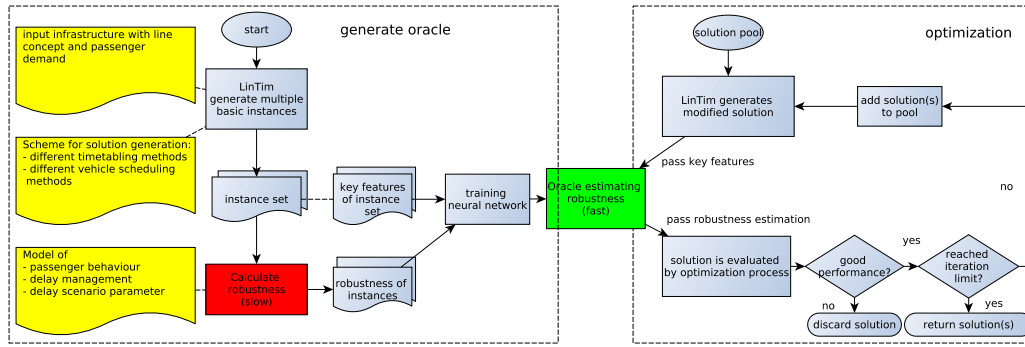
Figure 1 Left box: Workflow of the creation of the oracle for estimating robustness of a public transport system by training a neural network. Right box: Optimization as an exemplary application of the oracle. Yellow fields denote input or choices of models and methods which are specific for each application (but can easily be adapted).

Table 2 Our selection for key features and their length, which is specific for each infrastructure network. In our networks the maximal values were 240 minutes for the maximal travel time $\text{traveltime}^{\max}$, 10 for the maximal number of transfers $\#\text{transfers}^{\max}$ and 30 minutes for the maximal turnaround time $\text{turnaround}^{\max}$. m is the number of infrastructure edges, n the number of stations.

#	description	# elements
1	the avg. occupancy rate of the corr. vehicle in percent for each drive activity	m
2	the number of passenger groups with a perceived travel time of i minutes	$\text{traveltime}^{\max}$
3	the share of passengers with i transfers	$\#\text{transfers}^{\max}$
4	the average slack on wait activities per station	n
5	the average slack on transfer activities per station	n
6	the share of transfers per station	n
7	the average sum of line frequencies per station	n
8	the share of events per station	n
9	the number of trips with an outgoing turnaround slack of i minutes	$\text{turnaround}^{\max}$

1. defining a set of key features representing an instance,
2. generating a large number of training instances,
3. calculating the robustness of the instances with the original robustness test,
4. using ML to estimate robustness only by knowing the key features.

Figure 1, adapted from [19], illustrates how the creation and usage of the oracle is linked to the optimization process. The definition of a set of key features is essential for several reasons. We want a compact way of representing an instance with a fixed number of elements so ML-algorithms can easily use this as input. If this is achieved the creation of the key features during the optimization can be done fast without transferring large data sets to the oracle. Characteristic features include slack values on activities, occupancy rates on vehicles, and the number of passengers using a transfer. For a detailed list of the key features we selected for our model see Table 2.

In the second step, we need to create a large number of instances as the training set for the ML-algorithm. To do so, we use several different timetabling and vehicle scheduling methods as well as buffering strategies, provided by the open-source library LinTim [24, 25]. These instances should ideally be diverse and cover most robustness values. If there are gaps where we have no instances associated with a certain interval of robustness values (as can
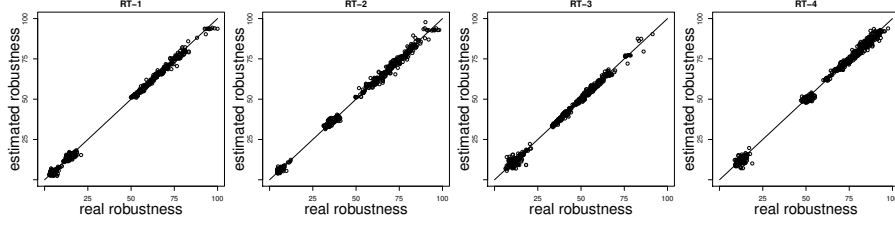
Figure 2 Predictions of all four robustness tests for all instances of the `grid` network [19].

be observed in Figure 2), new instances may not accurately be predicted near this range, see [19] for details. In the third step, the robustness values for the training set need to be calculated. This is done by the framework mentioned in Section 2.2. In the last step, the training of the oracle is done. We selected an artificial neural network (ANN) for the task of machine learning. More specifically, we use a neural network with five hidden layers and an output layer with four neurons predicting each of the four robustness tests separately. The network was trained using a training-, test- and validation set. For all our four infrastructure networks we achieved an average error below 1% in terms of the combined robustness value of all four robustness tests [19] (see Figure 2). However, the oracle performed worse when tested with instances created with a different method and robustness values outside the clusters in the training set. In spite of these limitations, we will see that the oracle is powerful enough to guide local search and the genetic algorithm into the desired direction.

3 Improving Robustness by Local Search

Our first approach in utilizing the robustness oracle presented in Section 2.3 is a generic local search approach. It was first described in [19] and is stated in Algorithm 2, Appendix A. The main idea is to determine a local neighborhood of the current solution in each step, evaluating all solutions in the neighborhood using the robustness oracle.

The algorithm has a given instance as a starting solution, i.e., a fixed dataset, consisting of an infrastructure network, a line concept and a passengers' demand, in combination with a timetable, a vehicle schedule and corresponding passenger routes. Since the dataset is fixed, we want to improve the robustness of the starting solution by changing the timetable and the vehicle schedule. To do so, we compute a local neighborhood of timetables in each step, introducing possible changes to the timetable. Note that this may include changes of the duration of turnaround activities, therefore requiring to adapt the vehicle schedule as well.

For every neighborhood, we consider several different activities for which an increase of its slack (i.e. the difference of planned duration and lower bound) could benefit the robustness of the instance: We sort the wait, drive and change activities each based on their current slack, divided by the number of passengers using the activity in the current passenger routes, and the turnaround activities by their current slack. Obtaining the N (here: $N = 20$) activities with the smallest weight from every sorted list, we get a candidate set of $4N$ activities. For each candidate $a = (i, j) \in \mathcal{A}$, $i, j \in \mathcal{E}$ with a lower bound l_a and a current duration d_a, we then increase the slack $d_a - l_a$ of the activity, resulting in a later time for the target event j.

This resulting timetable may be infeasible, since the lower bound on some activities $(j, k) \in \mathcal{A}$, $k \in \mathcal{E}$ may not be respected anymore. Therefore, we need a propagation strategy to reconstruct a feasible timetable. We considered the following four strategies here:

Strategy 1: Use all original slack. For every infeasible activity $a = (i, j) \in \mathcal{A}$, we increase the target event time exactly as much such that the lower bound of the activity is fulfilled again, i.e., $\pi_j = \pi_i + l_a$. Thus, the slack of this activity is reduced to zero.

Strategy 2: Reuse no slack. We maintain the original slack on each activity $a = (i, j) \in \mathcal{A}$, therefore shifting the complete timetable after the considered candidate, i.e., $\pi_j = \pi_i + d_a$.

Strategy 3: Reuse $J\%$ (here: $J = 50$) of the original slack, shifting the target event time of an infeasible activity $a = (i, j) \in \mathcal{A}$ according to $\pi_j = \pi_i + l_a + \frac{J}{100} \cdot (d_a - l_a)$.

Strategy 4: Reuse some slack, maintaining a minimal slack of K seconds (here: $K = 300$) (or the original slack, if it was less) on $a = (i, j) \in \mathcal{A}$, i.e., $\pi_j = \pi_i + l_a + \min(K, d_a - l_a)$

Note that we are considering all possibly infeasible activities here, including all change activities that are used by some passenger. The latter implies that passenger routes remain feasible. For ease of presentation, we do not check the upper bounds of the activities, assuming that we can postpone all events arbitrarily. If the upper bounds u_a should be considered as well, the above equations can be easily adapted to include the corresponding constraints. It may be the case that this leads to infeasibilities that can not be handled by our propagations strategies. In these cases we simply remove the candidate from the neighborhood. The different strategies lead to different trade-offs between robustness and travel time of the passengers, as can be seen in the computational experiments in Section 5.

Another aspect of the propagation strategy is whether to consider aperiodic or periodic timetables: The formulas given above can be extended to the periodic case, allowing us to maintain a feasible periodic timetable where we need to shift all corresponding periodic events at once. This leads to much larger changes in the resulting timetable and an additional travel time increase for the passengers. Furthermore, the reconstruction of feasible solution may not be possible due to the additional periodicity constraint. In such cases, we again remove the candidate from the neighborhood. Mathematically, the case where we allow an aperiodic timetable is a relaxation of the periodic case, therefore allowing better solutions w.r.t. robustness. We provide computational evaluations for both cases in Section 5.

After every candidate has a restored feasible timetable, we can use the oracle to predict the robustness value of the corresponding instance. Additionally, we evaluate the current passenger routes, rejecting candidates where the travel time of the passengers increases too much. The resulting set of instances serves as the neighborhood set for the local search and we can choose the best solution in terms of estimated robustness as the new current solution.

To improve the runtime of the local search, we do not update the passenger routes in every step. Since we maintain a feasible timetable, all passenger routes remain feasible as well but may be suboptimal for single passengers. The idea is that for every single iteration, the changes in the corresponding timetable are not too big, therefore not changing the optimal passenger routes too drastically. To maintain an accurate robustness prediction, we introduce additional rerouting steps, where we recompute all passenger routes every few iterations and therefore improve the accuracy of the robustness oracle.

4 Genetic Algorithm

Our second approach applying the robustness oracle is a genetic algorithm. Feasible solutions, i.e., instances as defined above, are mutated and breed to create new and hopefully better solutions w.r.t. robustness. The general procedure is described in Algorithm 1.

To allow an easy mutation and breeding of different instances without losing feasibility, we choose a specific data model to represent solutions in a compact way as genes of equal length. Every current solution in our algorithm is determined by a vector s of slacks for each possible activity in the event-activity-network, as well as a set of passenger routes. This

◾ **Algorithm 1** Genetic algorithm using machine learning.

Data: the starting solution set `currentSolutions`
currentSolutions = currentSolutions $\cup$ mutate(`currentSolutions`)
while *iteration limit not reached* **do**
   if *Rerouting step?* **then**
     | Reroute passengers in all solutions and update `currentSolutions`
   end
   currentSolutions = breed(`currentSolutions`)
   currentSolutions = predictAndSelect(`currentSolutions`)
end
Result: `currentSolutions`

---

allows for easy mutation and breeding, since every non-negative slack vector can be converted into a feasible timetable, by propagating the slack from a given start event. Note that we again do not consider upper bounds on the activities here and that we deliberately omit the vehicle schedule. Since optimal vehicle schedules can be computed very fast, these are calculated ad hoc when needed. To mutate an instance, a given number of l (here: $l = 100$) entries in the vector s are randomly selected and the corresponding slack value is changed by a random value in $[-m, m]$ for a given m (here: $m = 120$ seconds), provided that the updated slack remains non-negative. For breeding, we choose two parents randomly from the previous generation and combine their slack vectors, i.e., for each entry in the slack vector we randomly decide which of the possible values to use. This is done n times where n is given beforehand, i.e., we gain n new individuals for each generation. We choose a rather low number of 10 as the generation size and number of breedings per iteration due to the relatively large amount of memory needed to store the different entities. Additionally, the child is mutated as described above and directly inherits the passenger routes of one parent.

After creating the next generation in the breeding step, we introduce a selection process, reducing the number of candidates to a given g. In our implementation, both parents and children enter the selection process. We tested different variants here: The *quality* strategy selects solutions solely based on their estimated robustness, ignoring the travel time for the passenger. The *Pareto* strategy on the other hand chooses the non-dominant solutions in the current solution pool, i.e., solutions with a worse estimated robustness may remain in the population if their travel time is good enough. If less than g solutions are non-dominated, we choose the best non-selected solutions w.r.t. the robustness estimation to fill up the next generation. The difference between the two strategies is discussed in Section 5.

Note that since we only store the timetable for each solution, we need to compute a new vehicle schedule for each evaluation. To do so, a flow-based integer programming formulation of the open-source software library LinTim [24, 25] was used. Additionally, to maintain realistic passenger routes, we add a rerouting step that is executed every few iterations, computing new optimal passenger routes for every instance in the current generation.

5 Experiments

Algorithms 1 and 2 and their beforehand discussed variants were implemented and tested on several datasets: two artificial benchmark datasets, `grid` and `ring`, see [7], and two close-to real world datasets, the bus system in Göttingen, Germany (`goevb`) and the regional train network in southern Lower Saxony, Germany (`lowersaxony`). All datasets are available as part of the open-source library LinTim, see [24, 25]. Their key features are given in Table 3, for a visualization of the infrastructure networks see Appendix B.

Table 3 Sizes of the used datasets.

Name	# Stations	# Edges	# Passengers	# Lines	# Events
`grid`	80	145	1676	30	728
`ring`	161	320	2022	37	1376
`goevb`	257	548	1943	22	2348
`lowersaxony`	35	36	11967	7	508

Table 4 Average improvement of the different propagation strategies using 120 seconds of slack on all datasets and starting instances.

strategy	avg. robustness change	avg. perceived travel time change
Strategy 1	-7.00%	+0.27%
Strategy 2	-0.97%	+2.17%
Strategy 3	-10.66%	+0.28%
Strategy 4	-5.44%	+1.25%

5.1 Local Search

First, we discuss our evaluations of the local search algorithm, presented in Section 3. We study the different neighborhoods resulting from the propagation strategies, as well as the potential to improve the pool of existing solutions. The following experiments were run for several different starting instances per dataset. For the presentation, we selected one instance with small, medium and high initial robustness values, respectively.

Propagation Strategy

To evaluate the local search, we first discuss the different propagation strategies proposed in Section 3. In Table 4, the average changes to the two objective functions are given for all considered starting instances on all datasets. On average, Strategy 3 provides the best trade-off, by significantly improving the robustness at a small expense of increasing travel time. Figure 3 shows all different strategies used on three starting instances for `grid`. We can see that the expected performance, i.e., an improvement in robustness and an increase in passenger travel time can be observed for all cases. Furthermore, we see that the different strategies provide different trade-offs, e.g., Strategy 3 being a non-dominated solution (w.r.t. the other propagation strategies) for the starting solution with high robustness but not for the starting solution with the middle robustness. Note that although the improvement in robustness does not look significant for the starting solution with good robustness, the relative robustness improvement is still high, e.g., 31% for Strategy 4 compared to around 50% for Strategy 2 on the other two starting instances. Additionally, we see that choosing a smaller slack increase does not significantly alter the results obtained by the local search.

Using non-dominated start instances

Next, we want to consider the overall quality of the solutions found by the local search. Since the quality of the solution is dependent on the starting instance, Figure 4 shows the effect of using the local search on every non-dominated original instance, i.e., on every original instance that is not dominated by another one. For this, we chose an initial slack increase of 120s per iteration and propagation Strategy 4. We can see that we find a huge
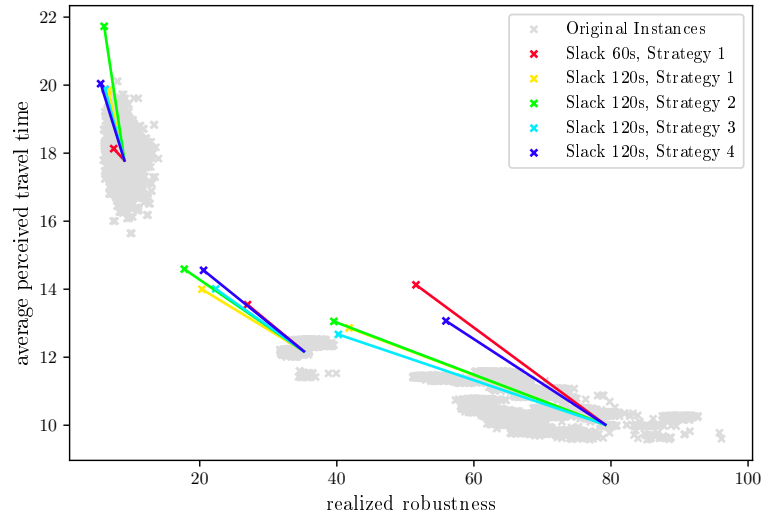
Figure 3 Different propagation strategies for the local search, evaluated on `grid`. The performance is depicted by a line from the starting instance to the end result, where the end result is additionally marked by an "x". The strategies are given by their number from Section 3 and the slack increase in each iteration.

(a) `lowersaxony`.

(b) `grid`.

(c) `ring`.

(d) `goevb`.
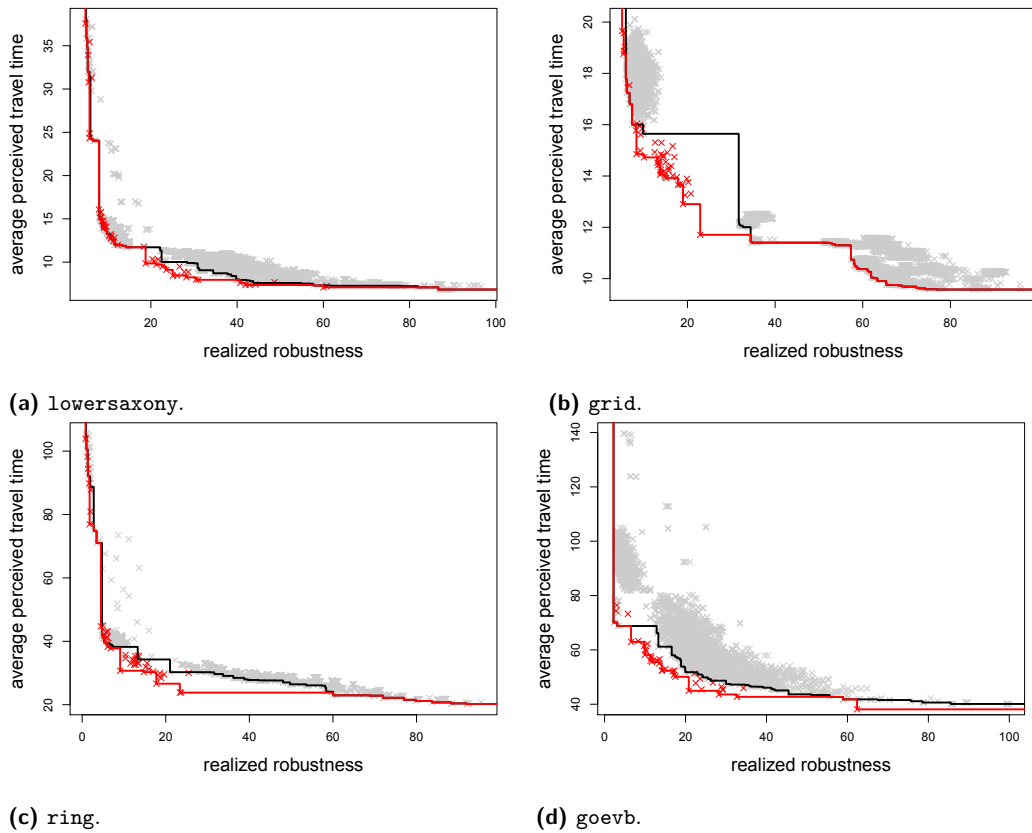
Figure 4 Aperiodic case: Using the local search on all non-dominated original instances. Old instances are grey, local search solutions are marked in red.

Table 5 Aperiodic case: Sizes of the approximated Pareto sets using local search.

dataset	originally non-dominated	together non-dominated	of those # new
grid	51	58	14
ring	44	29	12
lowersaxony	50	50	21
goevb	38	18	14

Table 6 Periodic case: Average improvement of the different periodic propagation strategies using 120 seconds of slack on all datasets and starting instance.

strategy	avg. robustness change	avg. perceived travel time change
Strategy 1	-2.87%	+0.14%
Strategy 2	-0.09%	+0.01%
Strategy 3	-2.2%	+0.07%
Strategy 4	-0.7%	+0.04%

amount of solutions with structures not used beforehand, i.e., that have objective values that are very different from the original instances. This is especially true for `grid`, Figure 4b, where we have a large number of solutions in between the original clusters. But we find competitive results for all instances, now dominating multiple beforehand non-dominated solutions. An extreme example is `goevb`, where almost all originally non-dominated instances are dominated by local search solutions, namely 34 of 38 instances. An overview of the number of non-dominated solutions in the different solution sets can be found in Table 5.

Periodic Timetabling and Local Search

If we restrict the local search to finding periodic timetables, the algorithm can still improve the robustness of the start instances. In Table 6, the average changes to the two objective functions are given for all considered starting instances on all datasets. The number of new solutions found that are non-dominated can be seen in Figure 5. Table 7 is a visualization of the new Pareto fronts. Contrary to the aperiodic case, compare Table 4, there is no dominant solution on average, i.e., on average all strategies provide different trade-offs. But the amount of change in the two objective functions is smaller when compared to the aperiodic case, due to the additional periodic restrictions. As can e.g. be seen in Figure 6, using the periodic local search on a starting instance with middle robustness improves the only robustness by 40% instead of the 50% of the aperiodic case. Still, the periodic local search is able to improve the robustness of every given starting instance on dataset `grid`.

Table 7 Periodic case: Sizes of the approximated Pareto sets using periodic local search with 120s slack on the different datasets.

dataset	originally non-dominated	together non-dominated	of those # new
grid	51	40	12
ring	44	47	14
lowersaxony	50	63	28
goevb	38	39	3

(a) `lowersaxony`.

(b) `grid`.

(c) `ring`.

(d) `goevb`.
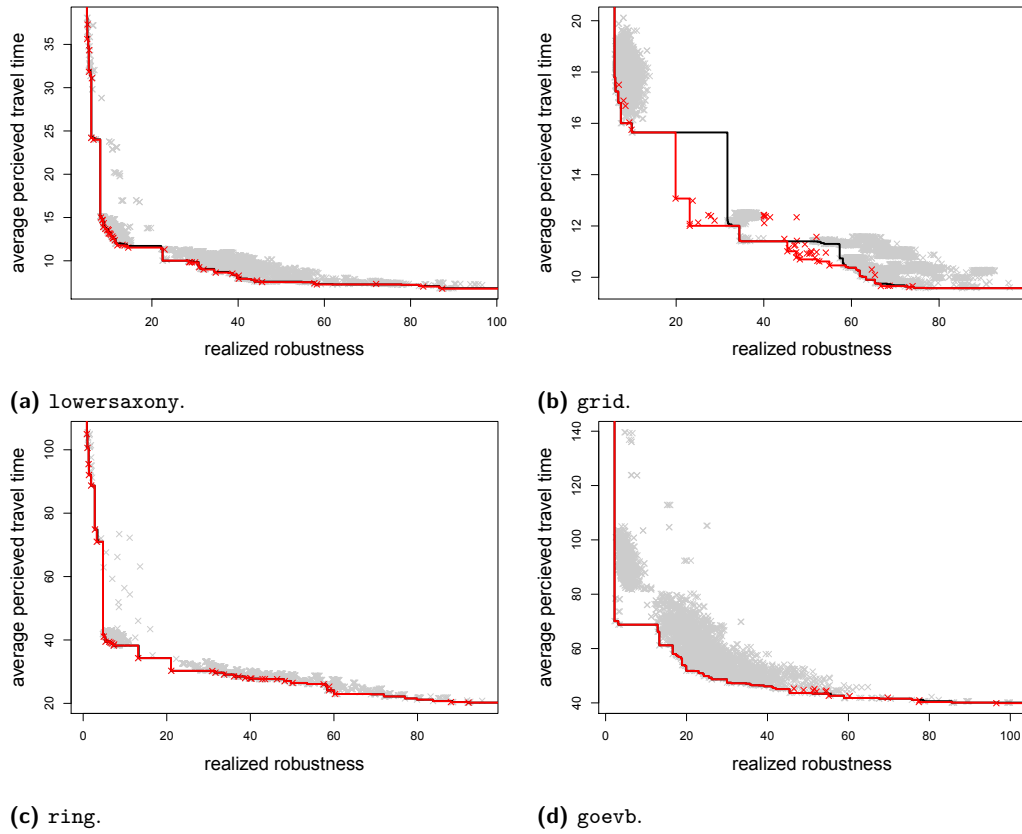
Figure 5 Periodic case: Approximated Pareto fronts for the solutions computed by the periodic local search with 120s slack on the different datasets. Old instances are grey, new solutions are marked in red.
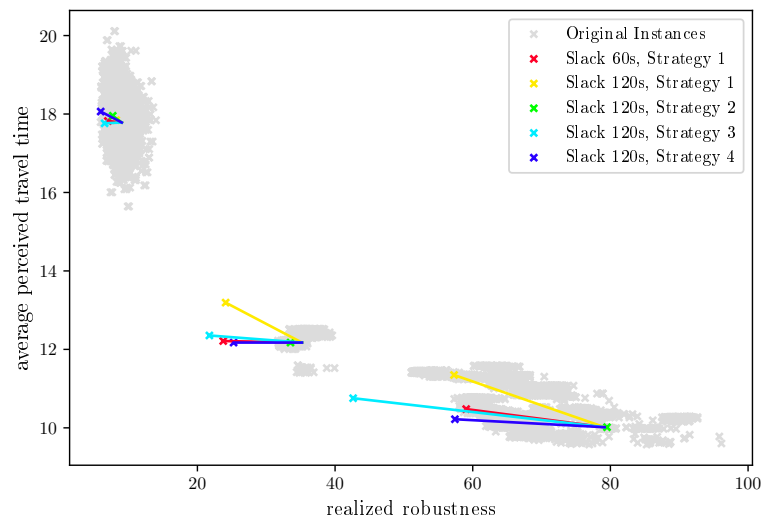
Figure 6 Using periodic planning for the local search, evaluated on `grid`. The performance is depicted by a line from the starting instance to the end result, where the end result is additionally marked by an "x". The strategies are given by their number from Section 3 and the slack increase in each iteration.
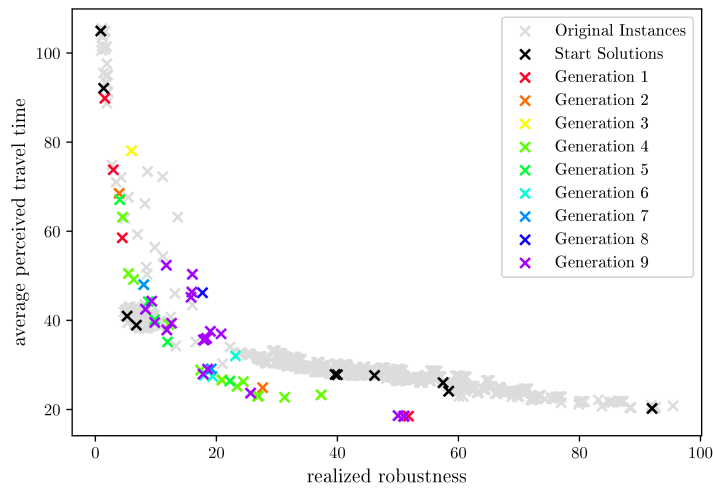
Figure 7 Using the Pareto selection for the genetic algorithm, evaluated on `ring`.
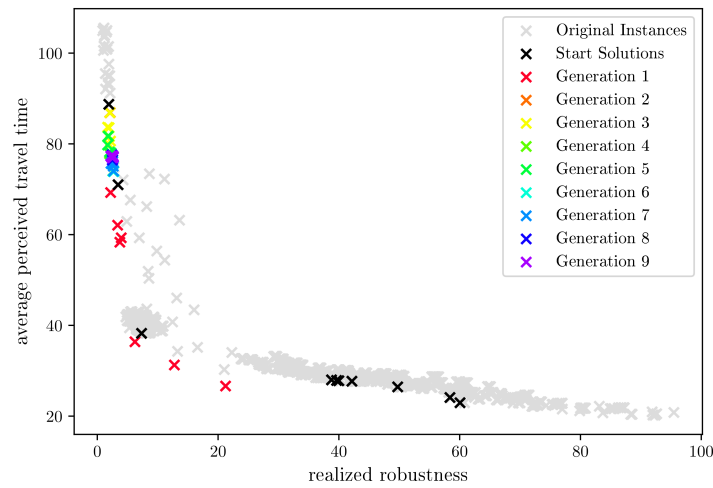
Figure 8 Using the quality selection for the genetic algorithm, evaluated on `ring`.

5.2 Genetic Algorithm

The genetic algorithm was evaluated on the described datasets as well. Note that all experiments discussed here were repeated multiple times, due to their randomness. But since the behavior of different runs were similar, only one run is presented for each experiment.

Choice of the selection process

The choice of the selection process shows the different qualities of the genetic algorithm. When using the Pareto selection, shown in Figure 7, the genetic algorithm produces multiple solutions dominating the original instances, exploring a large area of the previously empty part of the solution space. This produces several new competitive solutions for a decision maker to choose from. On the other hand, using the quality selection, shown in Figure 8, allows the genetic algorithm to focus on the estimated robustness value of the solution, producing more robust solutions with a higher travel time. Therefore both selection strategies have their advantages, the best strategy is dependent on the desired outcome of the algorithm.

Table 8 Sizes of the approximated Pareto sets using the genetic algorithm.

dataset	originally non-dominated	together non-dominated	of those # new
grid	51	37	34
ring	44	48	43
lowersaxony	50	76	68
goevb	38	35	33

But we also see a disadvantage of the genetic algorithm: Since the algorithm only optimizes the estimated robustness value, it is dependent on the quality of the robustness oracle used. As was already discussed in [19], using the oracle in unexplored solution space potentially increases the error, complicating the computation of robust solutions. This can, e.g., be seen in the variance of the last generation in Figure 7. But nevertheless, the overall quality, i.e., the real simulated robustness, of the computed solutions is very high.

Comparison to local search

To compare the genetic algorithm results with the local search results, we choose a depiction similar to Figure 4. In Figure 9, we collect the different solutions computed for the genetic algorithm experiments. With this, we can compare the approximated Pareto front of the different sets, namely the original instances, the local search solutions in Figure 4 and the genetic algorithm solutions. The genetic algorithm is able to compute a large set of competitive solutions, dominating even more original instances than the local search. For an overview of the number of non-dominated solutions, see Table 8. Especially in the area with worse robustness, the Pareto selection strategy combined with the randomness of the genetic algorithm results in a higher density of solutions. The genetic algorithm is therefore not only able to compute solutions with a good robustness but with very different trade-offs between robustness and passenger quality. Overall, both algorithms presented here are competitive and serve different means: While the local search can improve a single given starting solution w.r.t. the robustness value, the genetic algorithm is able to compute competitive solutions with different trade-offs from a set of given starting solutions.

Operating costs

Up until this point, we did not mention the operating cost of solutions since they are not in the focus of this work and we do not try to optimize them. But clearly robust and fast solutions still need to have competitive operating costs to be chosen by any public transport planner. Here, we only calculate and evaluate operational cost a posteriori.

LinTim includes operating costs based on the number of vehicles used, driven kilometers and an additional cost per hour for every vehicle in use. In our experiments the corresponding parameters were set to 100000 € per vehicle, 1.5 € per kilometer and 25 € per hour. Figure 10 shows the Pareto fronts concerning cost and robustness for the aperiodic local search with 120s slack. The networks `grid` and `goevb` show several clusters of solutions where the costs are dominated by solutions inside clusters near the Pareto front. We can observe that several of the new solutions have costs that are competitive and belong to the Pareto-front.

(a) `lowersaxony`.

(b) `grid`.

(c) `ring`.

(d) `goevb`.
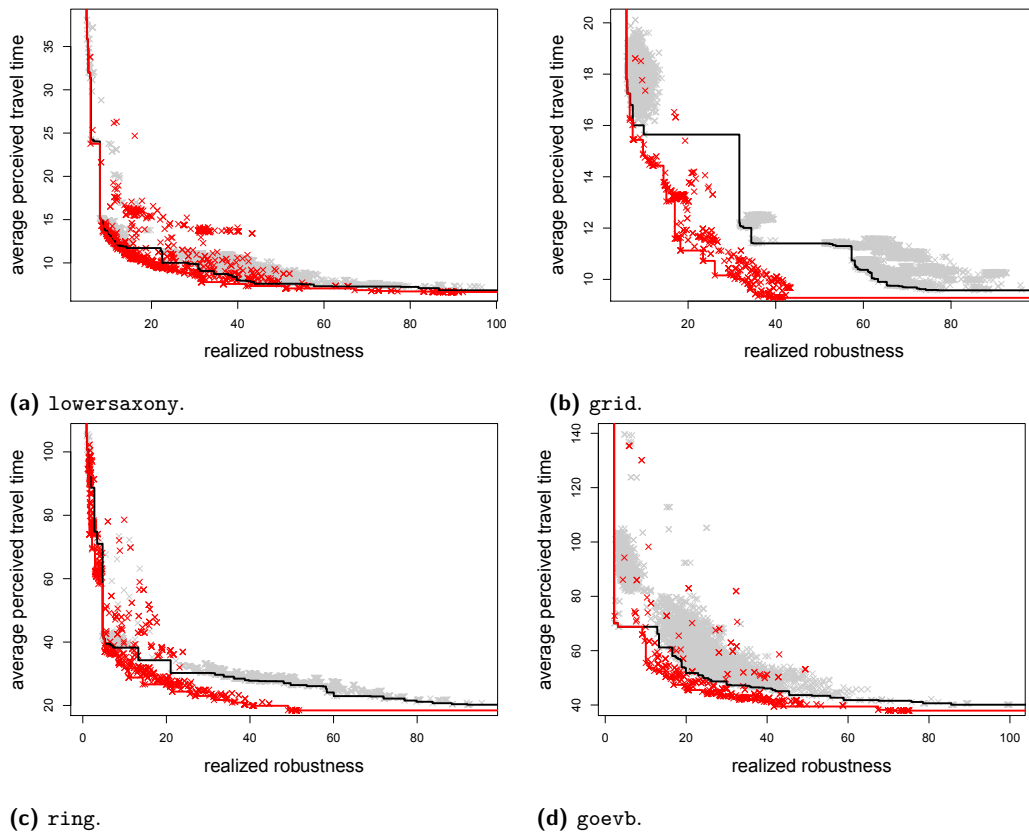
Figure 9 Approximated Pareto fronts for the solutions computed by the genetic search. Old instances are grey, new solutions are marked in red.

6 Outlook

In this paper, we have focused on improving the robustness of public transport systems from a passenger-oriented point of view. Our computational tests with local search and genetic algorithms demonstrated the ability of both methods to generate many new non-dominated solutions. However, there are still several improvements to consider. With respect to local search, we may further extend the definition of the used neighborhood and may consider combinations of several ones. For the genetic algorithm, next to changes in the selection process and the choice of starting instances, different mutation strategies would be possible to consider as well. This may further improve the exploration of the solution space, leading to more competitive solutions.

Improvement of the oracle and retraining is also of high importance. We need to eliminate gaps in the codomain, which are the robustness values. During the optimization using the genetic algorithm, we discovered many such solutions inside these gaps. This can be observed in Figure 9(b) where the space between 20 and 35 in the realized robustness objective is now populated. New solutions could now be added to the training process of the oracle, potentially allowing a better robustness estimation for future runs of the algorithms.

We plan to continue this line of work to see if similar results are possible when we modify the line concept, which is currently assumed to be fixed. Changing it would lead to different solution structures to learn for the oracle, extending the covered area in the solution space. Future work may also include further metaheuristics and stochastic local search methods.

(a) `lowersaxony`.

(b) `grid`.

(c) `ring`.

(d) `goevb`.
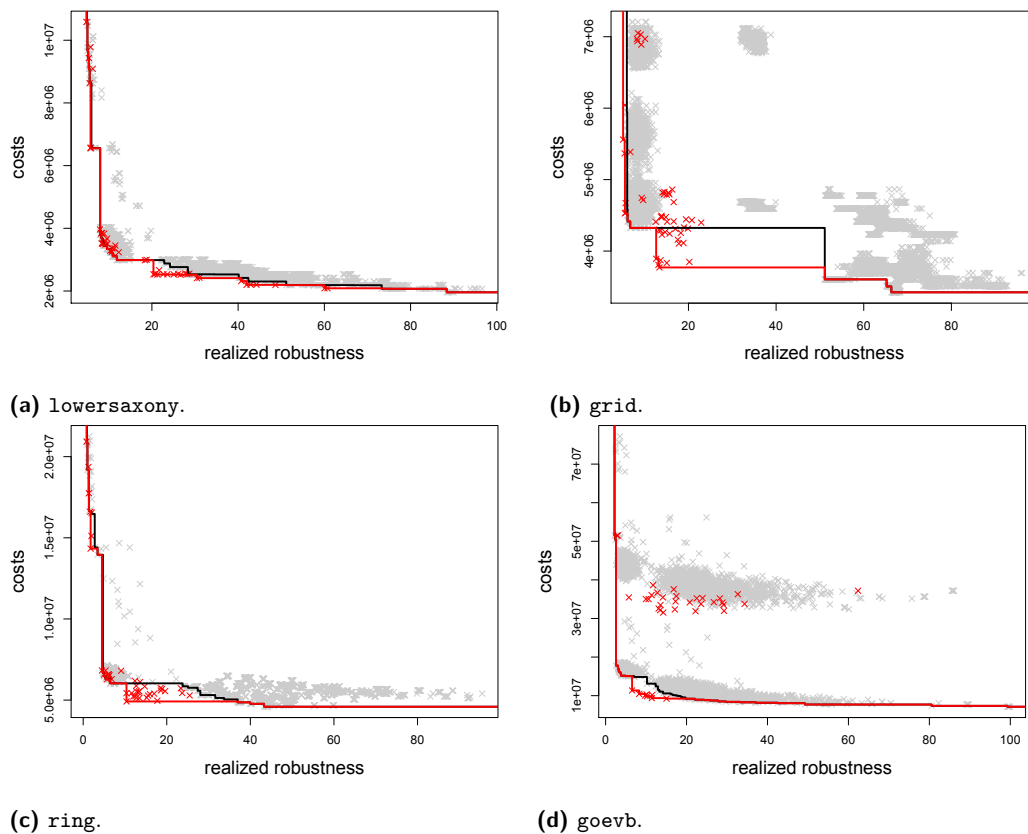
Figure 10 Approximated Pareto fronts (robustness vs. operational costs) for the solutions computed by the aperiodic local search with 120s slack and strategy 4 on the different datasets. Old instances are grey, new solutions are marked in red.

References

1 D. Arenas, R. Chevrier, S. Hanafi, and J. Rodriguez. Solving the train timetabling problem, a mathematical model and a genetic algorithm solution approach. In *6th international conference on railway operations modelling and analysis (RailTokyo2015)*, 2015.

2 R. Bauer and A. Schöbel. Rules of thumb — practical online strategies for delay management. *Public Transport*, 6(1):85–105, 2014.

3 S. Bunte and N. Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4):299–317, 2009.

4 O. Cats. The robustness value of public transport development plans. *Journal of Transport Geography*, 51:236–246, 2016.

5 A. De-Los-Santos, G. Laporte, J. A. Mesa, and F. Perea. Evaluating passenger robustness in a rail transit network. *Transportation Research Part C: Emerging Technologies*, 20(1):34–46, 2012. Special issue on Optimization in Public Transport+ISTT2011. `doi:10.1016/j.trc.2010.09.002`.

6 T. Dollevoet, D. Huisman, M. Schmidt, and A. Schöbel. Delay propagation and delay management in transportation networks. In *Handbook of Optimization in the Railway Industry*, pages 285–317. Springer, 2018.

7 Collection of open source public transport networks by DFG Research Unit "FOR 2083: Integrated Planning For Public Transportation", 2018. URL: `https://github.com/FOR2083/PublicTransportNetworks`.

8 M. Friedrich, M. Müller-Hannemann, R. Rückert, A. Schiewe, and A. Schöbel. Robustness Tests for Public Transport Planning. In G. D'Angelo and T. Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 6:1–6:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.ATMOS.2017.6`.

9 M. Friedrich, M. Müller-Hannemann, R. Rückert, A. Schiewe, and A. Schöbel. Robustness as a Third Dimension for Evaluating Public Transport Plans. In R. Borndörfer and S. Storandt, editors, *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, volume 65 of *OpenAccess Series in Informatics (OASIcs)*, pages 4:1–4:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/OASIcs.ATMOS.2018.4`.

10 M. Goerigk. Exact and heuristic approaches to the robust periodic event scheduling problem. *Public Transport*, 7(1):101–119, 2015.

11 M. Goerigk and A. Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers & Operations Research*, 40(5):1363–1370, 2013.

12 M. Goerigk and A. Schöbel. Algorithm engineering in robust optimization. In L. Kliemann and P. Sanders, editors, *Algorithm Engineering: Selected Results and Surveys*, volume 9220 of *LNCS State of the Art*, pages 245–279. Springer, 2016.

13 O. Ibarra-Rojas, F. López-Irarragorri, and Y. Rios-Solis. Multiperiod bus timetabling. *Transportation Science*, 50(3):805–822, 2016.

14 E. König. A review on railway delay management. *Public Transport*, 12(2):335–361, 2020.

15 Q.-C. Lu. Modeling network resilience of rail transit under operational incidents. *Transportation Research Part A: Policy and Practice*, 117:227–237, 2018. `doi:10.1016/j.tra.2018.08.015`.

16 R. Lusby, J. Larsen, and S. Bull. A survey on robustness in railway planning. *European Journal of Operational Research*, 266(1):1–15, 2018.

17 R. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. Railway track allocation: models and methods. *OR spectrum*, 33(4):843–883, 2011.

18 G. Matos, L. Albino, R. Saldanha, and E. Morgado. Solving periodic timetabling problems with SAT and machine learning. *Public Transport*, 2020. `doi:10.1007/s12469-020-00244-y`.

19 M. Müller-Hannemann, R. Rückert, A. Schiewe, and A. Schöbel. Estimating the robustness of public transport systems using machine learning, 2021. `arXiv:2106.08967`.

20 K. Nachtigall and S. Voget. A genetic algorithm approach to periodic railway synchronization. *Computers & Operations Research*, 23(5):453–463, 1996.

21 J. Parbo, O. Nielsen, and C. Prato. Passenger perspectives in railway timetabling: a literature review. *Transport Reviews*, 36(4):500–526, 2016.

22 J. Pätzold. Finding robust periodic timetables by integrating delay management. *Public Transport*, 2021. `doi:10.1007/s12469-020-00260-y`.

23 G. Polinder, V. Cacchiani, M. Schmidt, and D. Huisman. An iterative heuristic for passenger-centric train timetabling with integrated adaption times. ERIM Report Series Research in Management ERS-2020-006-LIS, Erasmus Research Institute of Management (ERIM), ERIM is the joint research institute of the Rotterdam School of Management, Erasmus University and the Erasmus School of Economics (ESE) at Erasmus University Rotterdam, 2020. URL: `https://ideas.repec.org/p/ems/eureri/127816.html`.

24 A. Schiewe, S. Albert, P. Schiewe, A. Schöbel, and F. Spühler. LinTim - Integrated Optimization in Public Transportation. Homepage. `https://lintim.net`, 2020.

25 A. Schiewe, S. Albert, P. Schiewe, A. Schöbel, and F. Spühler. LinTim: An integrated environment for mathematical public transport optimization. Documentation for version 2020.12, 2020. URL: `https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-62025`.

26 A. Schöbel. Line planning in public transportation: models and methods. *OR spectrum*, 34(3):491–510, 2012.

27 P. Tormos, A. Lova, F. Barber, L. Ingolotti, M. Abril, and M. Salido. A genetic algorithm for railway scheduling problems. In *Metaheuristics for scheduling in industrial and manufacturing applications*, pages 255–276. Springer, 2008.

28 A. van den Heuvel, J. van den Akker, and M. van Kooten. Integrating timetabling and vehicle scheduling in public bus transportation. Technical report, Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, 2008.

A Local Search

The local search algorithm used here, first described in [19], can be found in Algorithm 2.

Algorithm 2 Local search using machine learning, as stated in [19].

Data: the starting solution `currentSolution`
currentValue = evaluateByOracle(`currentSolution`)
while *true* **do**
  bestImprovement = ∅
  bestValue = ∞
  foundImprovement = False
  Compute local neighborhood of `currentSolution`
  if *Rerouting step?* **then**
    Reroute all passengers and update `currentSolution`
    currentValue = evaluateByOracle(`currentSolution`)
  end
  for *newSolution in local neighborhood* **do**
    introduceAdditionalSlack(`newSolution`)
    value = evaluateByOracle(`newSolution`)
    if *passengerUtility(newSolution) too bad* **then**
      continue
    end
    if *value < bestValue* **then**
      bestValue = value
      bestImprovement = newSolution
    end
  end
  if *currentValue > bestValue* **then**
    currentValue = bestValue
    currentSolution = bestImprovement
    foundImprovement = true
  end
  if *not foundImprovement* **then**
    break
  end
end

B Dataset information

Figures 11– 14 provide a visualization of the datasets used in this paper.
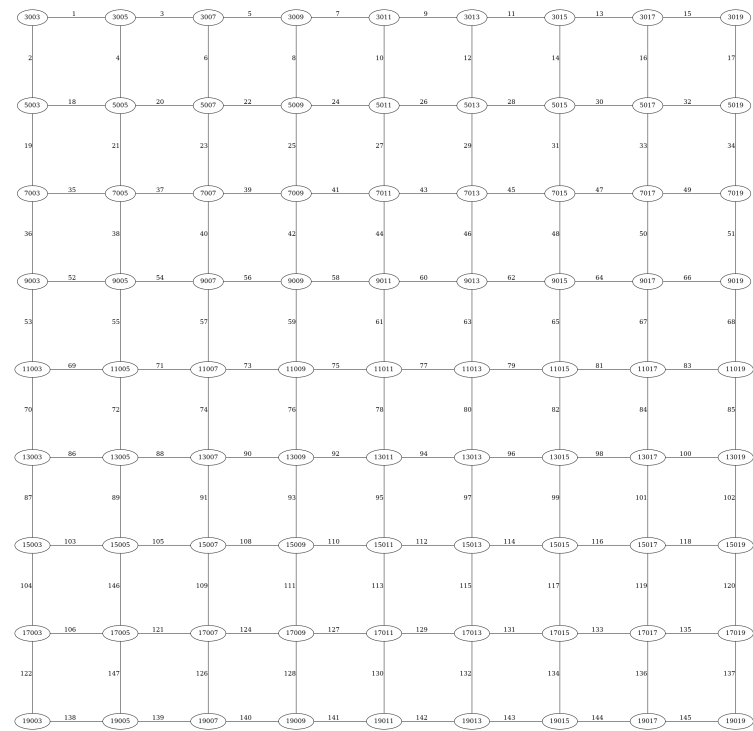
Figure 11 Infrastructure network of `grid`.
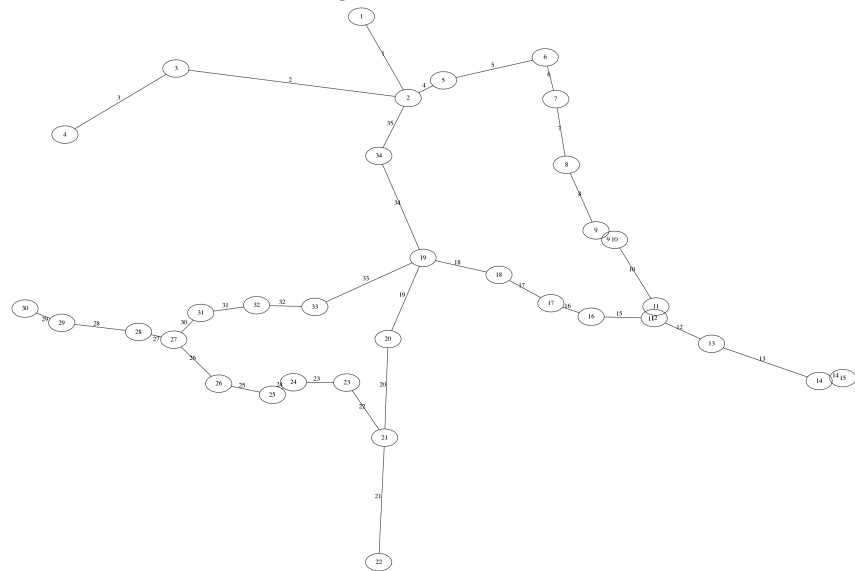
Figure 12 Infrastructure network of `lowersaxony`.
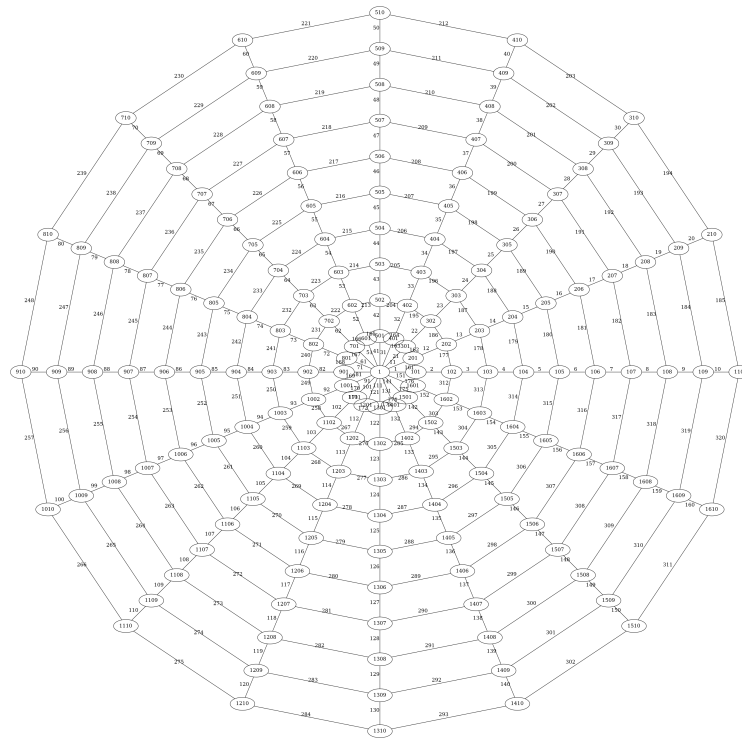
Figure 13 Infrastructure network of `ring`.
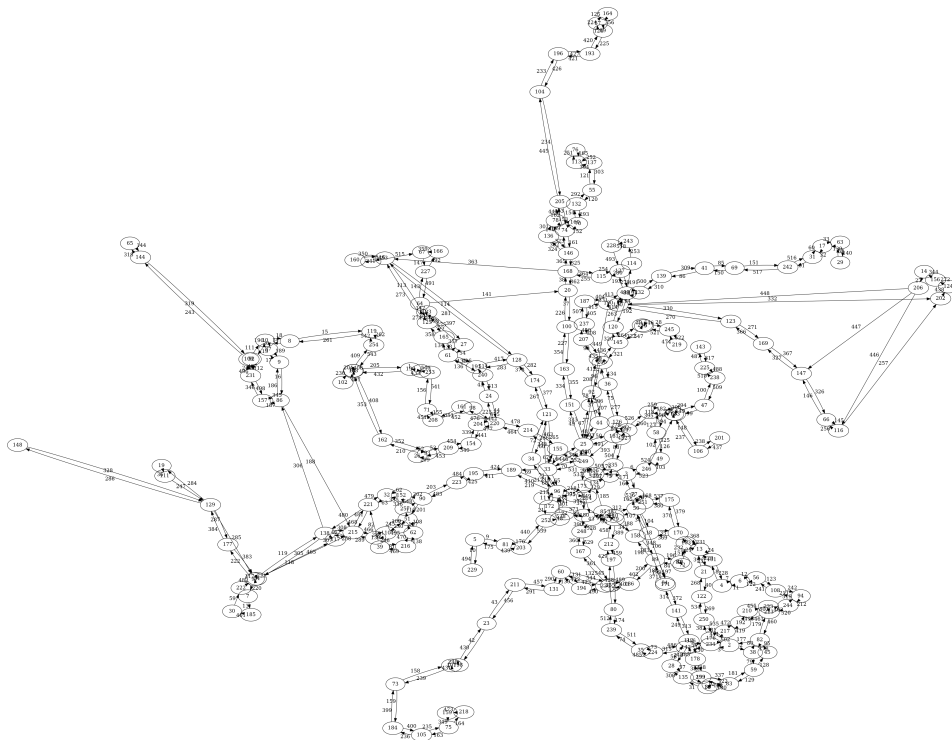
Figure 14 Infrastructure network of `goevb`.

Solving the Home Service Assignment, Routing, and Appointment Scheduling (H-SARA) Problem with Uncertainties

Syu-Ning Johnn ✉ 📧
School of Mathematics, The University of Edinburgh, UK

Yiran Zhu ✉ 📧
School of Mathematics, The University of Edinburgh, UK

Andrés Miniguano-Trujillo ✉ 📧
The University of Edinburgh, UK
Heriot-Watt University, Edinburgh, UK
Maxwell Institute for Mathematical Sciences, Edinburgh, UK

Akshay Gupte ✉ 📧
School of Mathematics, The University of Edinburgh, UK

── **Abstract** ──────────────────────────────

The Home Service Assignment, Routing, and Appointment scheduling (H-SARA) problem integrates the strategic fleet-sizing, tactical assignment, operational vehicle routing and scheduling problems at different decision levels, with a single period planning horizon and uncertainty (stochasticity) from the service duration, travel time, and customer cancellation rate. We propose a stochastic *mixed-integer linear programming* model for the H-SARA problem. Additionally, a reduced deterministic version is introduced which allows to solve small-scale instances to optimality with two acceleration approaches. For larger instances, we develop a tailored two-stage decision support system that provides high-quality and in-time solutions based on information revealed at different stages. Our solution method aims to reduce various costs under stochasticity, to create reasonable routes with balanced workload and team-based customer service zones, and to increase customer satisfaction by introducing a two-stage appointment notification system updated at different time stages before the actual service. Our two-stage heuristic is competitive to CPLEX's exact solution methods in providing time and cost-effective decisions and can update previously-made decisions based on an increased level of information. Results show that our two-stage heuristic is able to tackle reasonable-size instances and provides good-quality solutions using less time compared to the deterministic and stochastic models on the same set of simulated instances.

1 Background

Model Introduction. The home service industry constitutes businesses whose primary purpose is to provide services to people in their homes. Home services cover various sectors, including home healthcare, banking service, home beauty care, appliance repairs, home maintenance, and more. The typical requirements for the business providers are to decide

on the number of professional service teams to deliver services to geographically distributed customers, the assignment of the service teams to customers, the sequences of customer visits, and the scheduling of appointment time-slots to all customers with service demand. These specific decisions form the Home Service Assignment, Routing, and Appointment scheduling (H-SARA) problem, which is related to a set of widely studied problems in both academia and industry and was presented for the 13th AIMMS-MOPTA Optimization Modeling Competition [34]. The first is the *Vehicle Routing Problem* (VRP) which is a generalisation of the well-known *Travelling Salesman Problem* (TSP). For a typical VRP, the main aim is to determine a set of minimum-distanced tours visiting all the locations starting and ending at a depot, meanwhile satisfying a list of general limitations including space and time capacity, time windows, maximum vehicle travel time, or traversal distance. With numerous applications in logistics, transportation and general distribution management, the VRP has been studied widely in the past few decades and has been extended with several variants and applications [4, 23, 24]. Whereas a VRP minimises the total routing costs using a predetermined number of vehicles or service teams (typically of homogeneous type), a *fleet-sizing problem* (FSP) minimises both the total routing costs and the economical fleet size [13], addressing a trade-off between fixed vehicle costs and variable routing costs.

Scheduling usually refers to the chronological allocation of tasks to workers such that the list of tasks (components) are accomplished within the shortest amount of time and with the minimal time clashes. In the H-SARA problem, an appointment time slot is assigned to all customers with service demand. Equivalently, from a service provider's perspective, each customer visit is scheduled as part of a service team's timetable in sequential order. The *Vehicle Routing Problem with Time Windows* (VRP-TW) is a VRP-variant stressing that the vehicle arrival and/or departure times must satisfy additional customer availability requirements. We identify the difference of *scheduling* from the VRP-TW as the pro-activeness from the decision-makers: the visiting sequence is the result of initial routing criteria instead of the customer-imposed time requirements. Some related problems are the *Appointment Scheduling Problem* (ASP) in the context of healthcare [14] and the *Home Health Care Routing and Scheduling Problem* (HHC-RSP) [6, 11].

Model Uncertainties.   In reality, one or multiple elements of the classical VRP is often expected to be uncertain due to the limited availability of information. Common uncertainties include customer presence, traversal times, and service duration. These can be modelled as stochastic random variables, giving rise to the *Stochastic Vehicle Routing Problem* (SVRP) and its variants [27]. The SVRP is usually solved by applying (two-stage) stochastic programming techniques [21, 32]. *A priori* optimisation [3] works on real-world applications in which randomness is a major concern. It applies the two-stage strategy: an initial solution is first created before the parameters are revealed in the second stage. It means that first-stage decisions should possess sufficient flexibility for the second-stage recourse actions.

The idea of *a priori* optimisation can be easily found in reality. Many international shippers (e.g., DPD [9], Royal Mail [8]) have now adapted to similar concepts in their last-mile deliveries: they first assign an estimated time slot to all customers based on the pre-collected information, then re-assign a narrower time slot on the actual day of service when more information is known (e.g., customer delivery sequence, cancellations). This multi-stage approach also suits the real-life circumstances in the home healthcare service industry, where last-minute service cancellations, i.e. customers cancelling their requests after being given an appointment time, are allowed. Home service statistics show that the average daily visits per service team in the U.S is around 6 [12], and that the driving time typically accounts

for 18% to 26% of the total working time [20], which indicate how a single cancellation can considerably change the timescale for the following visits and the necessity of a robust service planning system. Several works on home service-related research implement this multi-stage approach [10, 26, 29, 30, 35].

There are existing works in the literature that deal with uncertainties in travel times, service times, or customer presence in the context of *Home Health Care* (HHC). Readers are referred to [15] for reviews of relevant models and methods in HHC. An excessive studies in VRP with stochastic travel times can be found in [27]. Particularly, [22, 37, 38, 39] consider randomness in service times. [25, 35] consider travel and service times uncertainties. [5] considers customers who request service cancellation, and [17] considers random customer behaviours in attended home delivery. Yet, to our best knowledge, there is no research that integrates all three types of uncertainties with the four decisions in the context of HHC.

Our Contributions. The main contribution of this work is the treatment of an H-SARA problem integrating the four decisions levels: strategic fleet-sizing, tactical assignment, operational routing, and operational scheduling. Travel times, service duration, and cancellation rates are considered jointly as uncertain quantities, which to our best knowledge, has not been investigated in the literature before. We developed a two-stage heuristic approach that takes the evolution of information into account, thus allowing decision-making based on imperfect information before the actual customer demands are revealed, and updating existing solutions with an increased level of information. This paper is based on the authors' submission to the AIMMS-MOPTA Modeling Competition [34] at which they were awarded the First Prize.

2 Problem Statement

Let a service area be represented by a directed graph $G := (V, A)$. Here the node set V encloses the customer set $[\![1, n]\!] := \{1, \ldots, n\}$, a single depot $\{0\}$, and its duplicate $\{n + 1\}$. The arc set consists of all the arcs linking each pair of customers, as well as a single link from the depot to each customer and another from each customer back to the duplicated depot, all with the shortest distance computed using the Euclidean metric; namely $A := \{(i, j) : i \neq j, \forall i, j \in [\![1, n]\!]\} \cup \{(0, j) : \forall j \in [\![1, n]\!]\} \cup \{(i, n + 1) : \forall i \in [\![1, n]\!]\}$. The service for all n customers of known geographical location is provided by a group of no more than m homogeneous service teams, each of which makes a single trip starting from and returning to the depot. We aim to partition the set of customers into the minimum number of groups, each visited exactly once by an individual service team in an explicit visiting sequence, and to determine customer appointment time-slots prior to the actual visits. The solution should satisfy time and capacity constraints given by the customers and the service teams. Customers must be informed of their appointment times (or time slots) on the service day before the cut off time (8 am) or the departure of the assigned service teams from the depot, whichever is earlier. Lastly, the probability distributions associated with travel and service times are known and assumed to be independent.

3 Mixed Integer Programming Model

3.1 Uncertainties inside the model

We apply *a priori* optimisation, where a set of *a priori* vehicle routes is first planned in the presence of estimated expected travel and service times. The precise duration of each tour becomes available only after the actual travel and service times are revealed in the second

stage. Consequently, there is always an inevitable chance of a solution "failing" under the stochasticity setting, forcing the decision-makers to develop relevant recourse policies to repair a failed (infeasible) solution.

A extension beyond the consideration of stochastic travel and service times is the stochastic customer behaviour (customer presence). An option provided by Sörensen and Sevaux [36] is to first include all customers in the routes, then remove customer set $\mathbb{I} \in I$ who cancel their service requirements on short notice. This gives a conservative or risk-averse approach for the decision-makers since the routes are feasible for any customer set realisations, provided that the traversal and service times are feasible. Base on this assumption, if the customer in position i is removed, the service team will travel directly from customer $i-1$ to customer $i+1$.

3.2 Stochastic MIP model

Parameters.   We introduce the MIP formulation for the *H-SARA* and derive a set Ξ of different scenarios ξ, each associated with a different realisation of the travel and service durations with a certain probability q_ξ. We impose a stochastic traversal duration matrix $T = \tau_{i,j}^\xi$ under scenario ξ for any arc $(i,j) \in A$, and a stochastic service duration vector $S = s_i^\xi$ for customer i under scenario ξ. The Euclidean distance from i to j is labelled $d_{i,j}$. In this formulation, symmetry of τ and d is not required, capturing possible discrepancies in the underlying road network; i.e., city topography and street layout. Let $p : [\![1,n]\!] \to \mathbb{R}_{\geq 0}$ be a probability mass function defined over the set of customers, such that for each customer $i \in [\![1,n]\!]$ the probability of last-minute service cancellation of i is given by p_i. The cost of hiring a homogeneous team $i \in [\![1,m]\!]$ is taken as a constant f_m. The maximum allowed working time is given by $L \geq 0$. Working times are expected to be allocated in the interval $[0, L]$, yet we anticipate possible overtime occurring in the interval $(L, L+\theta]$ with $\theta > 0$. Any additional time beyond the maximum working time L and within $L + \theta$ results in overtime cost. Finally, let c_{wait}, c_{idle}, and c_{over} be fixed non-negative unit waiting, idling, and overtime costs, respectively.

Decision Variables.   For the decision variables, we let $x_{i,j}$ be a binary variable which takes the value of one if the arc $(i,j) \in A$ is traversed by a service team, otherwise it takes the value of zero. We use a continuous variable $0 \leq a_i \leq L$ for the team's arrival time at customer $i \in [\![1,n]\!]$. Likewise, w_i and h_i are non-negative real-valued variables for the customer's waiting time, and service team's idling time at customer $i \in [\![1,n]\!]$, respectively. g_i is a real-valued variable measuring the overtime of a service team, registered at their arrival at the depot when returning from customer $i \in [\![1,n]\!]$. Finally, since an actual arrival time under the stochastic setting could be different from a customer's initial appointment time, we have differentiated an appointment time (scheduled service start time) variable t_i for each customer $i \in [\![1,n]\!]$. We assume the appointment time window is $[t_i - W, t_i + W]$ with a fixed width $2W$. The arrival of a service team before the appointment time window leads to team idling, whereas an arrival after the time window leads to the customer waiting.

We have the traversal variables $x_{i,j}$ (also fleetsize, if we treat the total number of edges linking customers with the depot as twice the fleet) and the appointment time variables t_i as our first-stage decisions. In contrast, the team idling time h_i, overtime g_i, and customer waiting time w_i are our second-stage decisions dependent on the different scenarios. The first and second stage formulations for the stochastic *H-SARA* model are as follows:

(1st Stage)

$$\min_{x} \quad f_m \sum_{i \in [\![1,n]\!]} x_{i,n+1} + \sum_{(i,j) \in A} d_{i,j} x_{i,j} + \mathbb{E}\left[Q(x,\xi)\right] \tag{1a}$$

subject to

$$\sum_{i \in [\![1,n]\!]} x_{0,i} \leq \hat{m}, \tag{1b}$$

$$\sum_{i \in [\![0,n]\!]} x_{i,j} = 1 \qquad\qquad \forall j \in [\![1,n]\!] \tag{1c}$$

$$\sum_{i \in [\![0,n]\!]} x_{i,j} = \sum_{i \in [\![1,n+1]\!]} x_{j,i} \qquad\qquad \forall j \in [\![1,n]\!] \tag{1d}$$

$$\sum_{i \in [\![1,n]\!]} x_{0,i} = \sum_{i \in [\![1,n]\!]} x_{i,n+1}, \tag{1e}$$

$$x_{i,j} \in \{0,1\} \qquad\qquad \forall (i,j) \in A. \tag{1f}$$

where $\mathbb{E}\left[Q(x,\xi)\right] = \sum_{\xi \in \Xi} q^\xi \cdot Q(x,\xi)$ for any x satisfying the above equations, and any $\xi \in \Xi$ associated with probability q^ξ. The objective function (1a) minimises the total traversal costs, team hiring costs, and expected idling, waiting, overtime costs under all scenarios. Constraint (1b) states that there are no more than $\hat{m}$ homogeneous service teams departing from the depot $\{0\}$. (1c) require that each customer must be visited once and only once by a service team. The flow conservation constraints (1d) require that a team travelling to any customer node must leave the node afterwards. This is complemented with (1e), which stresses that the number of teams leaving the depot must equal the number that returns. (1f) are the domain constraints.

(2nd Stage)

$$Q(x,\xi) = \min_{w,h,g} \quad c_{\text{wait}} \sum_{i \in [\![1,n]\!]} w_i^\xi + c_{\text{idle}} \sum_{i \in [\![1,n]\!]} h_i^\xi + c_{\text{over}} \sum_{i \in [\![1,n]\!]} g_i^\xi \tag{1g}$$

subject to

$$a_i^\xi + h_i^\xi + s_i^\xi + \tau_{i,j}^\xi \leq a_j^\xi + M(1 - x_{i,j}) \qquad\qquad \forall(i,j) \in A, \tag{1h}$$
$$a_i^\xi + h_i^\xi + s_i^\xi + \tau_{i,j}^\xi \geq a_j^\xi - M(1 - x_{i,j}) \qquad\qquad \forall(i,j) \in A, \tag{1i}$$
$$a_i^\xi + s_i^\xi + \tau_{i,n+1}^\xi - L \leq g_i^\xi + \theta(1 - x_{i,n+1}) \qquad\qquad \forall i \in [\![1,n]\!], \tag{1j}$$
$$h_i^\xi \geq (t_i - W) - a_i^\xi \qquad\qquad \forall i \in [\![1,n]\!], \tag{1k}$$
$$w_i^\xi \geq a_i^\xi - (t_i + W) \qquad\qquad \forall i \in [\![1,n]\!], \tag{1l}$$
$$t_i \leq L, \;\; g_i^\xi \leq \theta \qquad\qquad \forall i \in [\![1,n]\!], \tag{1m}$$
$$a_i^\xi, h_i^\xi, w_i^\xi, g_i^\xi \geq 0 \qquad\qquad \forall i \in [\![1,n]\!]. \tag{1n}$$

Scenario-based objective function (1g) minimises the idling, waiting and overtime costs. The functionality of (1h) is two-fold. First they join (1i) to link together the arrival time to the first customer, its service time, and the traversal time to the next customer given that the two customer visits are consecutive. Second it forbids the formation of subtours, which are circles formed only by a group of customers without the depot. (1j) determine the incurred overtime when returning to the depot from the last customer. Constraints (1k) and (1l) specify the idling and waiting times, respectively. Constraints (1m) give the upper bounds, and (1n) provide lower bounds for the relevant variables.

4 Exact Solution Method

4.1 Bounding the number of service teams

This section presents the upper and lower bounds of a feasible number of service teams to hire. For notation simplicity, the travel and service times involved in the following models are the expected values for each arc and customer node, namely $\hat{\tau}$ and $\hat{s}$. Following the steps given in [16], we can find an upper bound on the number of teams required to visit all clients by solving the following linear problem:

$$\min \ell_u \tag{2a}$$

subject to

$$\sum_{i \in [\![1,n]\!]} \hat{s}_i + \sum_{i \in V} \big(\max\{\hat{\tau}_{i,j} : (i,j) \in A\} + \max\{\hat{\tau}_{j,i} : (j,i) \in A\} \big) \leq \ell_u(L+\theta), \tag{2b}$$

$$1 \leq \ell_u \leq \hat{m}, \qquad \text{and} \qquad \ell_u \in \mathbb{Z}, \tag{2c}$$

where ℓ_u is a decision variable representing the maximum number of needed teams to satisfy, in a mean-worst-case scenario, all the transportation and services requirements. Here, $\hat{m}$ is an upper limit on the number of teams, which can be as large as the number of customers n, and an optimal solution of (2) determines a choice over m. Observe that if we divide constraint (2b) by ℓ_u, the resulting expression distributes the routing task in two parts: There is a term averaging service time, and another term averaging the time required, taking time-consuming paths, to travel between customers. Notice that the optimal solution can be obtained using exhaustive enumeration in $\mathcal{O}(\hat{m})$ time.

Likewise, service times can provide a lower bound on the amount of time that all service teams spend on the road. To do so, we define ℓ_l as the minimum number of teams required to distribute the aggregated service time and minimum transportation time. Thus we need to solve the following nonlinear program

$$\max_{\ell_l} F(\ell_l) = \sum_{i \in [\![1,n]\!]} \frac{\hat{s}_i}{\ell_l} + \sum_{i \in V} \left[\frac{\min\{\hat{\tau}_{i,j} : (i,j) \in A \wedge i \neq j\}}{\ell_l} + \frac{\min\{\hat{\tau}_{j,i} : (j,i) \in A \wedge i \neq j\}}{\ell_l} \right] \tag{3a}$$

subject to

$$F(\ell_l) \leq L+\theta, \qquad 1 \leq \ell_l \leq \hat{m}, \quad \text{and} \quad \ell_l \in \mathbb{Z}. \tag{3b}$$

Notice that if this problem is infeasible, then there are not enough teams to solve the *H-SARA* with mean values for service and transportation times. As a result, we have an infeasibility certificate. Again, this problem can be solved in $\mathcal{O}(\hat{m})$ time.

4.2 Deterministic Exact Solution Method

The deterministic model can be considered as a single-scenario stochastic model, where appointment time t_i is the same as the arrival time a_i with zero service team idling time $h_i = t_i - a_i = 0$ at customer $i \in [\![1,n]\!]$. Besides, the model has a pre-specified set of customer nodes with known coordinates, since we assume all cancelled customers are already removed. The instances are generated using a scenario-based approach specified in Section 6.1. We first attempted to solve the deterministic *H-SARA* model to optimality. The model was inputted with a pre-specified number of customer nodes. The first deterministic model (first

Table 1 Results for deterministic version of the model with acceleration approaches.

Type		Number of customers n				
		10	15	20	25	30
Deterministic	CPU-time	0.54	1.14	4.97	18.81	1800*
	Gap	0%	0%	0%	0%	0.05%
Deterministic[1] (fixed m)	CPU-time	0.65	1.34	4.41	16.65	1438.16
	Gap	0%	0%	0%	0%	0%
Deterministic[2] (fixed $m+$ gap)	CPU-time	0.63	1.31	2.07	3.62	6.33
	Gap	0.1%	1.7%	1.6%	0.9%	1.3%

two rows) in Table 1 shows the average CPU times and the average gap solved using CPLEX 20.1.0 for 10 iterations with time limit 1800 seconds. The gap indicates the solution's quality and is defined as the difference in percentage between the upper and the lower bounds.

Although solving a smaller-scaled deterministic problem is computationally manageable, the solver fails to find feasible solutions for large or even moderate-sized instances within 30 minutes on average, as shown in Table 1. As a result, we have proposed two acceleration approaches to reduce the computing time for the deterministic *H-SARA* model. The approaches are implemented inside our solution framework and are described below.

First, we apply a root node solution method to address a trade-off between fixed vehicle costs and variable routing costs, aiming for an "economical fleet size" [13]. We pre-define the number of service teams m in constraint (1b) and change its sense to strict equality so that the solver is no longer required to optimise the fleet size but treats it as an input parameter. For each fixed fleet size m in $\{\ell_l, \ell_l + 1, \ldots, \ell_u\}$ computed in Section 4.1, we used CPLEX to callback the first feasible (integer) solution we receive at the root node. After all associated root node values are computed, we instruct CPLEX to identify the smallest root node value and return its associated fleetsize m, which will be used as the final fleet size to optimise the routing and scheduling decisions. Using this method, we observe a considerable improvement in computation speed without a significant loss of solution quality, as shown in the third and fourth rows of Table 1.
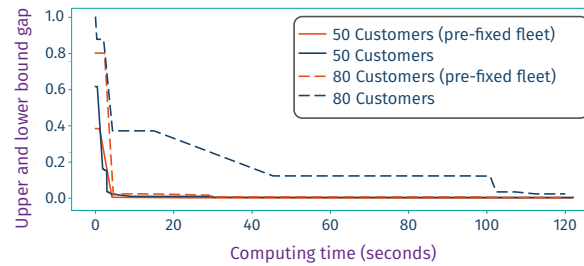
Figure 1 Deterministic model gap versus computing time.

Secondly, we observe from experimental testings that CPLEX's default heuristic solution method can reach an integer solution at the root node with reasonably good quality and within a concise computing time (less than 1 minute). Nevertheless, reaching a global optimum is difficult due to the time-consuming nature of the branch-and-bound process encoded in the solver. This trend is shown in Figure 1 and can be observed visually during the solution process that the solver spends an awfully long time improving the visiting

sequence of customers. For an 80-customer instance, the optimal root node solution has the hiring costs outweigh routing and scheduling costs, allowing the algorithm to terminate when the idling, waiting and overtime penalties are small. Based on experimental results, we fixed a 2% gap for the total staffing, routing and scheduling costs, assessing the terminating speed (how fast to reach 2%) and solution quality (routing and scheduling decision quality). The solution time and relative MIP gap reported by CPLEX for different customer sizes are presented in the last two rows of Table 1.

4.3   Stochastic Exact Solution Method

The multi-scenario stochastic model is noticeably more challenging to tackle than its deterministic counterpart, which can be considered as a single-scenario stochastic model.

We realise the natural partitioning of our stochastic model, where the first stage is a mixed-integer linear programming problem and the second-stage recourse model is linear. Furthermore, the second-stage problem is scenario-dependent, and therefore its structure suggests the application of *Benders' Decomposition* [2], taking the first stage as the master problem that decides which set of paths to take, and treating each scenario inside the second-stage recourse model as a subproblem. Each subproblem provides a scenario of the travel and service times for the arc traversal decisions made during the first stage.

We use CPLEX built-in Benders algorithm to solve a full model. The first and second rows in Table 2 list the numerical results of solving the complete stochastic model as a whole, incorporating the fleet size pre-solving procedure described in the deterministic model, and limiting the gap to 2%. The third and fourth rows are with Benders' algorithm. The empirical results show that *Benders Decomposition* is not suitable for our models as it consumes much longer computing time to provide worse results. Moreover, we notice that due to specific parameter scale settings, we have the fleetsize cost dominating the other costs. For a 15-customer instance, we notice that only two teams were hired, which results in seriously high idling, waiting, and overtime penalty costs. This is the reason behind the long solution process before termination, since an additional team hire brings up the total costs but is the only way to bring down the penalty costs.

Table 2 Results for stochastic model with 10 scenarios.

Type		Number of customers n				
		5	10	15	20	30
Stochastic	CPU-time	0.19	2.05	1421.18	25.38	183.31
	Gap	1.99%	1.97%	2.12%	1.99%	1.91%
Stochastic	CPU-time	0.52	11.28	1800*	1800*	1800*
(Benders)	Gap	1.99%	2.00%	2.93%	2.12%	3.66%

We have observed from Table 1 and 2 that even with efficient accelerating approaches, solving a large-scale H-SARA problem jointly to optimality is still not practical due to the time-consuming nature of exact methods. On top of that, the problem involves a range of uncertainties in real-life traversal times, service duration, and customer presence rates, all of which require a flexible solution method that focuses more on adapting to fast-changing information and a large number of scenarios, meanwhile achieving in-time solution with good quality. These results drove us to explore and develop a simple and flexible heuristic as an alternative.

5   Two-Stage Solution Strategy

5.1   Solution Framework

For our tailored two-stage heuristic, we have first decomposed the problem into different stages with an embedded chronological structure, allowing us to make dynamical decisions at each stage with an increased level of information. At each stage, we have also partitioned the decision set into its fleet-sizing, districting, routing, and scheduling components and introduced an "inter-feedback process" among different decisions, which avoids the deficiency of a hierarchical decision process that may lead to sub-optimal solutions.

Our two-stage heuristic resembles a typical home service rundown: previous-day initial plannings (Section 5.2), service day tour refinements (Section 5.3), and post-service performance evaluation (Section 5.4). The heuristic showcases the "inter-feedback process" that the previously-made decisions can be re-optimised and updated at a later stage with an increased level of information. Figure 2 shows an example for our two-stage heuristic timeline, and Figure 3 displays an example for the two-stage heuristic outputs.
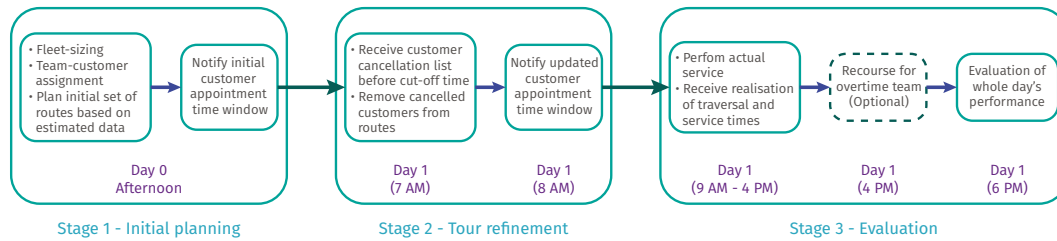
Figure 2 Heuristic rundown with chronological timeline.

During the initial planning stage, the decision-makers need to make pre-arrangements with limited information to guarantee a smooth rundown on the actual service day. The tour refinement stage resembles the actual service day, with the visiting sequences re-optimised based on last-minute cancellation outcomes. For the post-service evaluation stage, complete information about travel and service durations are revealed after the actual service, allowing decision-makers to evaluate the service teams' performance. One crucial requirement for the first-stage decisions is robustness, which allows the second-stage decisions to refine the previous ones without much modification.

5.2   Initial Planning Stage

Before we formally introduce the two-stage heuristic, we first provide an estimation on the activity measure, which is the expected amount of time required to include a specific customer in a tour. This helps us to determine the size of a customer cluster serveable by an individual team. In our application, the customer cancellation rate is known probabilistically, which means that the actual sequencing of customers or the computation of route lengths is pointless without knowing the actual cancellation list. Yet, we can estimate the travel and service times without explicit routing as in [1]. The estimated total time required for a group of customers can be divided into (i) *stem time*: estimated travel time from the depot to the nearest customer inside the group; (ii) *intermediate transit*: estimated travel time between customers of the same group; (iii) *service time*: estimated stopping time at each customer. Parts (i) and (iii) are self-explanatory and can be estimated by the relevant travel
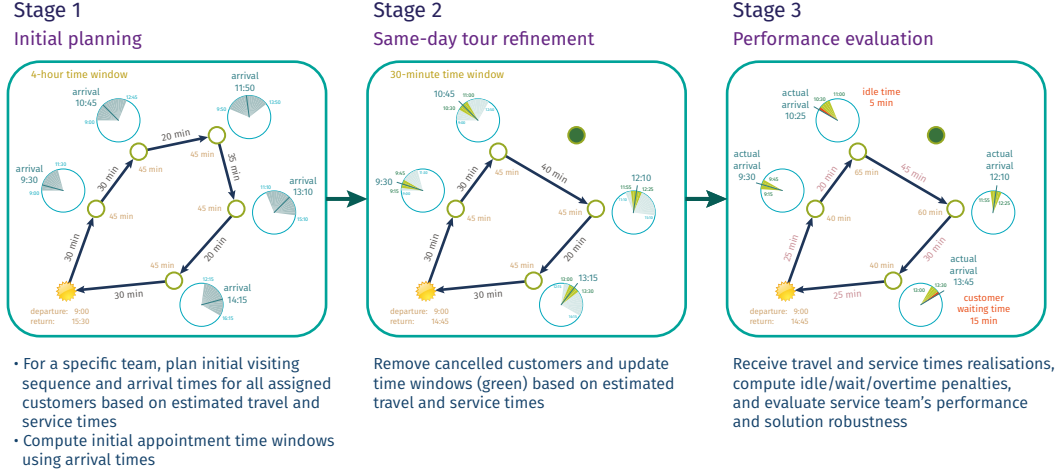
Figure 3 Heuristic framework: initial planning, tour refinement, and post-service performance evaluation stages.

and service times distributions. For (ii), we can estimate e_i, which is the expected travel time from customer i to any same-group customer j with probabilistic customer presence rate, using the formula given in [1]:

$$e_i = \sum_{j=1}^{b_i} p_{i,j}^{(*)} \cdot \frac{d_{i,j}}{v_{i,j}} = \sum_{j=1}^{b_i} \frac{(1-p_j)(b_i - R_{i,j} + 1)}{\sum_{l=1}^{b_i}(1-p_l)(b_i - R_{i,l} + 1)} \cdot \frac{d_{i,j}}{v_{i,j}} \tag{4}$$

where p_j is customer j's probabilistic cancellation rate, b_i is the number of closest customers to customer i, $R_{i,j}$ is the rank of the jth closest customer to i, with $j \in [\![1, b_i]\!]$, and $p_{i,j}^{(*)}$ can be interpreted as the likelihood of customer j following i on a route. $d_{i,j}$ is the Euclidean metric and $v_{i,j}$ is the travel velocity from node i to j.

As a result, the activity measure ω_i for customer i can be estimated by the expected service time $\hat{s}_i$ plus the estimated travel time from i to the district centre j using (4). Here we use the expected travel velocity $\hat{v}$. We estimate the number of nearest customers to be the average number of customers inside a district $b_i = \lceil \frac{n}{m} \rceil$. This way we have for a specific customer i:

$$\omega_i = \hat{s}_i + e_i = \hat{s}_i + \sum_{i=1}^{b_i} p_{i,j}^{(*)} \cdot \frac{d_{i,j}}{\hat{v_{i,j}}} \tag{5}$$

At the beginning of the initial planning stage, we apply a cluster-first-route-second construction heuristic to come up with an initial set of routes. A feasible fleet size m can be pre-determined using the root-node solution method we described in Section 4.2. We adapt the districting formulation proposed by Hess et al. [19] and solve the MIP model to optimality to receive our initial customer-team assignment decisions. The specific MIP formulation can be found in Appendix A.1. Mathematically, we first aggregate customers into m compact and balanced districts that are each manageable by an individual service team. After clustering the customers, we form a single cycle inside each district containing all its customers and the depot. This is equivalent to solving the TSP for m times. We adapt the DFJ formulation for TSP [7] to receive our initial routing decisions. A comprehensive review on the TSP heuristics methodologies and implementations can be found in [28]. However, considering the size of our problem, an exact solution can be obtained using existing solvers.

To improve upon these routes, we employ the *adaptive large neighbourhood search* (ALNS) meta-heuristic. ALNS was first introduced by Ropke and Pisinger [31] as an extension of the *large neighbourhood search* (LNS) proposed by Shaw [33] with the general principle of "destroy and repair", which is to search for a better solution by destructing a part of the solution and reconstructing the damaged part in a different way. Our ALNS pseudocode is presented in Algorithm 1. A detailed ALNS framework can be found in Appendix A.2.

▢ **Algorithm 1** Basic steps of ALNS.

1: $s \leftarrow \mathtt{InitialSolution}, \mathtt{InitialScore}(w^*)$ and $s^{\mathrm{best}} = s$
2: **for** stopping criteria not met **do**
3:     $N^- \leftarrow \mathtt{Choose}(\mathtt{AllDestroyOperators}, w_d^*)$
4:     $N^+ \leftarrow \mathtt{Choose}(\mathtt{AllRepairOperators}, w_r^*)$
5:     $s' \leftarrow \mathtt{DestroyRepairApply}(s, N^-, N^+)$
6:     **if** $s' <$ QualityThreshold **then**
7:         $s' \leftarrow \mathtt{LocalSearch}(s')$
8:     $\mathrm{obj}(s') =$ sum cost (team, travel, overtime) and workload balance penalties
9:     **if** s' satisfies acceptance criterion **then**
10:         $s \leftarrow s'$
11:         **if** $s' < s^{\mathrm{best}}$ **then**
12:             $s^{\mathrm{best}} \leftarrow s'$
13:     **update** RouletteWheel operators performance scores
---

The upper set of graphs in Figure 4 shows an example for the first-stage (initial planning) heuristic outputs. In Stage 1.3 of the first-stage heuristic, we further balance the workload among all operators by including a workload imbalance penalty in the ALNS objective function to penalise the extra units of workload above or below a certain threshold for any service team. The last step of the first-stage heuristic is to notify all customers of their initial appointment slots. Based on the set of routes improved by ALNS, we compute each individual's appointment time from the associated team departure time. To cope with potential last-minute customer cancellations, we expand each individual appointment time into an appointment time window with fixed length and communicate this individual-tailored appointment time window to every registered customer. For example, assuming $T_1 = 4$ hours and a customer's estimated appointment time is at 11:30 am, the first-stage appointment time window for them will be [9:30, 13:30].

5.3 Tour Refinements Stage

At the beginning of the second stage, the list of cancelled customers $\mathbb{I}$ becomes known. So we re-optimise the initial tours to fit the updated-to-date customer information. The lower set of graphs in Figure 4 shows the decision process for our second-stage tour refinement: we remove the cancelled customers from the first-stage tours, compute the estimated arrival times for all non-cancelled customers, improve the service teams workload balance, and notify all the non-cancelled customers of a narrower appointment time window.

The service teams' arrival times to customers and depot are random variables since they depend on travel and service times which are by definition random variables. This lead to our decision of quoting an appointment time window, rather than a specific time point, to every non-cancelled customer during the first and second stages. We re-apply the ALNS improvement heuristic in Stage 2.3, where we not only minimise the total travelling costs, overtime costs, and team workload imbalance but maximise the chance of scheduling the updated appointment times to nest within the first-stage appointment time windows. In this way, we avoid abrupt appointment time modifications, which is essential to service quality

Stage 1:

Stage 1.1:
Form m districs

Stage 1.2:
Form single tour per district

Stage 1.3:
• Apply ALNS to improve workload balance of service teams
• Compute arrival times using estimated travel and service times

Stage 1.4:
• Compute initial appointment time windows using arrival times
• Notify customers of their initial time windows
• Fix service team fleetsize

Stage 2:

Stage 2.1:
Last-minute customer cancelation list revealed

Stage 2.2:
Reform tours without cancelled customers and compute new arrival times using estimated travel and service times

Stage 2.3:
• Apply ALNS to improve routes for better workload balance
• Fit arrival times into time windows from Stage 1.4

Stage 2.4:
• Compute narrower appointment time windows using updated arrival times
• Notify customers of their updated time windows

Figure 4 Initial planning and route refinement stages of the heuristic framework – an example.

and customer satisfaction, even though at the cost of longer service team waiting times. Specifically, we assume a first-stage time window $[T_1^{\text{start}}, T_1^{\text{end}}]$ and a second-stage estimated arrival time a_i at a non-cancelled customer i.

Similar to the first-stage appointment scheduling, we create a narrowed second-stage time window with length $T_2 = 30$min. The time windows are not necessarily centred at their arrival times. This is determined by a linear adjustment $[a_i - T_i^{start}] * c_{idle} = [T_i^{end} - a_i] * c_{wait}$ that forces the center forward in time to cope with more expensive waiting costs, or backward with more expensive idling costs. The ALNS objective term $P' \times \max\{T_1^{\text{start}} - a_i, a_i - T_1^{\text{end}}, 0\}$ penalises any arrival time not nested within the first-stage time window. Besides, we manually adjust the second time window to be $[T_1^{\text{start}}, T_1^{\text{start}} + T_2]$ in the occurrence of any infeasible second time window begins earlier than the first. Likewise, $[T_1^{\text{end}} - T_2, T_1^{\text{end}}]$ applies to any second time window that finishes after the first time window.

5.4 Post-Service Performance Evaluation

The quality of our second-stage refined routes will be evaluated in the post-service evaluation stage. The issue of data over-fitting might occur for our two-stage heuristic, since we only rely on in-sample objective values computed using a discretised set of scenarios n_e clustered from random samples. Therefore, we also evaluate the out-of-sample performance of our solutions using a new and much larger set of benchmark scenarios generated after the model has been solved. This gives a fairer indication of how good our service levels are with an unobserved set of data. The evaluation stage is not counted as a valid solution stage, since no decision-making process is involved.

6   Experiments and Insights

6.1   Experiment Settings

The parameter settings were given in the AIMMS-MOPTA competition guidelines [34]. Specifically, we assume n customers are uniformly located over a 50×50 km geometric grid with the depot located at the origin $(0,0)$. We set the fixed individual team hiring cost $f_m = 100$, hourly team idling time cost $c_{\text{idle}} = 2.5$, hourly overtime cost $c_{\text{over}} = 5$, hourly customer waiting cost $c_{\text{wait}} = 4$. We also define the standard daily workload $L = 8$ hours for each team, the first-stage time window length $T_1 = 2$ hours, and the second-stage time window length $T_2 = 30$ minutes. We assume the travel times between any two nodes are identically distributed with a log-normal distribution. For the customer service time, we select the gamma distribution that is not strictly symmetric in order to avoid generating a negative service time. We assume the expected service time $\hat{s} = \mu_s = 45$ min with its standard deviation set to $\mu_s/2$, the expected travel speed $\hat{v} = 1$ km/min (equivalently, expected travel time $\hat{\tau}_{i,j} = 1$ min/km). Moreover, we assume individual customers all share the cancellation probability defined at a fixed rate 5%.

For a unified measurement, we use CPLEX 20.1.0 as the optimisation solver for both the heuristic framework and exact methods. The whole two-stage heuristic solution computation is performed on a machine with Intel i5-10400F CPU and 16GB RAM installed.

A Sampling-Based Objective Function.   Since the customer cancellation list is random, and so are the travel and service durations, we come up with a sampling-based objective function computed from a number of n_e randomly generated scenarios to guide the second-stage solution process, inspired by the work of [36]:

$$f^*(x) = \frac{1}{n_e} \sum_{i=1}^{n_e} f\big(x, S_i(\tau, s)\big) \tag{6}$$

where $f^*(x)$ is the expected total costs computed from a number of n_e randomly generated scenarios, x is the set of second-stage routes with the cancelled customers removed, S is the sampling function, and $S_i(\tau, s)$ represents the ith scenario with stochastic travel and service times realisation. $f\big(x, S_i(\tau, s)\big)$ represents the total costs of the ith scenario applied to x, and finally n_e is the total size of scenarios.

Scenario Generation.   We introduce a scenario generating procedure to ensure a more diverse set of scenarios is included. First we apply the Monte-Carlo simulation that randomly generates n_s samples, each with an identical pair of travel and service times realisations. We then cluster a fixed number of n_e scenarios from these samples using a k-mean clustering algorithm given that $n_s \gg n_e$. The probability q^ξ of each scenario ξ is estimated using the number of samples clustered together divided by the total number of generated samples. In this way, we are able to capture extreme values using a moderate number of scenarios.

6.2   Observations

The experiment results are given in Table 3, from which we have observed the following points: To begin with, our two-stage heuristic can tackle a larger customer size within a reasonable time. The two-stage heuristic takes no more than 2 minutes on our computer to compute a solution for a 40-customer instance, whereas the deterministic model requires 19 minutes on average, and the stochastic model cannot even terminate within 30 minutes. If

Table 3 Computational Results from different solution methods.

#Scenario	1								10				20				50				100			
Method	SVRP		2-stage Heur		2-stage ALNS		"1-stage" Heur		SVRP		2-stage Heur		SVRP		2-stage Heur		SVRP		2-stage Heur		SVRP		2-stage Heur	
#Customer	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
10	209.23	1.638	169.42	1.28	168.79	1.27	169.07	0.81	211.04	1.93	172.54	2.49	209.10	2.29	164.03	3.84	208.19	3.47	161.28	7.84	205.94	13.20	162.74	14.08
20	236.65	34.55	235.51	15.75	238.43	15.83	234.56	8.16	237.35	192.46	237.71	19.96	-	t*	240.46	24.64	-	t*	237.15	36.56	-	t*	235.76	57.73
30	315.68	364.54	318.77	19.87	318.96	19.87	315.64	15.32	-	t*	318.83	27.68	-	t*	319.55	35.77	-	t*	318.08	50.12	-	t*	318.95	91.42
40	410.30	1115.24	418.28	35.21	418.72	35.48	409.82	23.52	-	t*	417.54	47.04	-	t*	424.14	58.98	-	t*	417.62	95.54	-	t*	417.62	157.54
50	-	t*	521.91	72.87	522.50	73.98	512.59	50.90	-	t*	521.37	88.80	-	t*	521.64	105.65	-	t*	521.82	154.27	-	t*	521.22	239.22
100	-	t*	1012.28	198.84	1016.24	199.51	1020.43	173.96	-	t*	1018.09	256.23	-	t*	1021.63	310.54	-	t*	1004.83	459.90	-	t*	998.48	790.11
150	-	t*	1459.51	609.81	1465.49	612.87	1504.49	586.06	-	t*	1460.89	716.40	-	t*	1460.82	827.44	-	t*	1461.24	1148.10	-	t*	1460.78	1645.79

∗ Score is the expected objective function value averaged from 10 experiments on 100 test instances generated out of 10,000 samples by k-means.

∗ Time is measured in seconds, and t^* refers to the upper limit of computing time which is 1800 s.

∗ All the results (Score and Time) are averaged from 10 experiments. Scenarios used in methods are generated randomly and independently from test instances.

(1) **SVRP** is solved with the proposed time-saving root node solution strategy (pre-selection of fleetsize m) given in Section 4.

(2) **2-Stage Heur** is solved using our two-stage heuristic method with the tour-overlap–breaker local search method without the tour-overlap–breaker local search operator in the ALNS improvement process. This model applies the classical ALNS, which is included here for comparison with our improved ALNS in both speed and outcome.

(3) **2-Stage ALNS** is solved using our proposed two-stage heuristic method with the tour-overlap–breaker local search operator in the ALNS improvement process. This

(4) **"1-stage" Heur** is solved as a comparison to our two-stage heuristic, assuming the full customer cancellation list being available before the initial planning stage. Thus the second-stage re-routing and re-scheduling are excluded from the solution process. We solve this by not removing any customers at the second stage. This comparison tells how much last-minute customer cancellation costs to the business apart from other uncertainties.

we further increase the model's size to 100 customers, none of the exact MIP approaches can terminate within hours, but our two-stage heuristic can still obtain results within 5 minutes, and within 10 minutes for the 150-customer instances.

For the solution quality, our two-stage heuristic provides competitive solutions comparing to CPLEX solutions on the same set of simulated benchmark instances. By comparing same-scenario columns between the exact methods and two-stage heuristic, we observe that within the given time limit, our two-stage heuristic is able to find solutions within 4% of the solutions computed by CPLEX. Even though all exact and heuristic methods columns are non-optimal (since global optimum is extremely difficult to compute, as shown in Fig 1), we want to showcase the fact that our two-stage heuristic is able to provide same-quality solutions and within less amount of time compared to CPLEX. Besides, the two-stage heuristic is more robust in real-life applications and can provide up-to-date decisions at different service preparation stages based on different levels of available information.

Hypothetically, if we obtain the complete customer cancellation information in the first place, we can simply merge the two heuristic stages and deal with only stochastic travel and service times. To determine the additional cost of making multi-stage decisions, we run a parallel experiment "1-stage Heur", assuming complete information for cancelled customers. It achieves lower objective costs than the two-stage heuristic "2-stage Heur", which receives no customer cancellation list but only cancellation probability during the initial planning stage. Yet, our two-stage heuristic is not worse-off in terms of average objective values and computing time from the results. For experiment sets with 100 and 150 customers, "2-stage Heur" outperforms "1-stage Heur" in the expected objective function value although with slightly longer computing time on average. We recognise two potential reasons behind this phenomenon: local search-based heuristics cannot guarantee the global optimum in general, and the solutions computed by "1-stage Heur" being over-fitted to the single scenario than the benchmark instances/scenarios from the evaluation stage.

To conclude, we are able to include last-minute customer cancellations into our solution process and make initial decisions based on probabilistic customer cancellations, all at a reasonable additional cost. The additional cost is mainly due to our requirement to nest the second-stage narrower appointment time window within the first stage's, thus limiting the freedom to optimise the best routes and leading to slightly worse-off solutions. However, no perfect information exists in reality. The differences between one-stage and two-stage solutions can be treated as the costs of "imperfect information", or equivalently, the costs for making a priori decisions and previous-day customer notifications without getting the complete picture.

7   Summary

This paper studied the H-SARA problem, which integrates the fleet-sizing, assignment, routing, and scheduling problems. We have proposed a stochastic MIP model for the H-SARA problem, whose deterministic and stochastic versions are solved with two accelerated methods for small and medium scaled instances. We also developed a tailored two-stage heuristic solution method with an embedded ALNS improvement heuristic, to support a real-life decision-making process taking the evolution of information into account. Our proposed two-stage heuristic shows good performance in terms of computational time and solution quality. It also demonstrates good flexibility and robustness in adapting to multiple scenarios with different travel times, service times, and customer cancellation rates. Using our decision support framework, we can provide time and cost-effective decisions with low idling, waiting, and overtime costs, as well as two sets of customer appointment time windows, and balanced service team workload within geographically clear service zones.

References

1 Jonathan F Bard and Ahmad I Jarrah. Large-scale constrained clustering for rationalizing pickup and delivery operations. *Transportation Research Part B*, 43(5):542–561, 2009. `doi: 10.1016/j.trb.2008.10.003`.

2 Jacques F Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

3 Dimitris J Bertsimas, Patrick Jaillet, and Amedeo R Odoni. A priori optimization. *Operations Research*, 38(6):1019–1033, 1990. `doi:10.1287/opre.38.6.1019`.

4 Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & industrial engineering*, 99:300–313, 2016. `doi:10.1016/j.cie.2015.12.007`.

5 Paola Cappanera, Maria Grazia Scutellà, Federico Nervi, and Laura Galli. Demand uncertainty in robust home care optimization. *Omega*, 80:95–110, 2018. `doi:10.1016/j.omega.2017.08.012`.

6 Mohamed Cissé, Semih Yalçındağ, Yannick Kergosien, Evren Şahin, Christophe Lenté, and Andrea Matta. Or problems related to home health care: A review of relevant routing and scheduling problems. *Operations research for health care*, 13-14:1–22, 2017. `doi:10.1016/j.orhc.2017.06.001`.

7 George B Dantzig, Delbert R Fulkerson, and Selmer M Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954. `doi:10.1287/opre.2.4.393`.

8 Chris Dawson. Royal mail day before delivery time notifications launched. URL: `https://tamebay.com/2019/04/royal-mail-day-before-delivery-time-notifications-launched.html`, April 2019.

9 DPD. Guide to dpd. `https://www.dpd.co.uk/pdf/dpd_sales_guide_2020_v3.pdf`, 2020.

10 Christian Fikar and Patrick Hirsch. A matheuristic for routing real-world home service transport systems facilitating walking. *Journal of Cleaner Production*, 105:300–310, 2015. `doi:10.1016/j.jclepro.2014.07.013`.

11 Christian Fikar and Patrick Hirsch. Home health care routing and scheduling: A review. *Computers & Operations Research*, 77:86–95, 2017. `doi:10.1016/j.cor.2016.07.019`.

12 The National Association for Home Care & Hospice. Basic statistics about home care, 2010. URL: `http://www.nahc.org/wp-content/uploads/2017/10/10hc_stats.pdf`.

13 Bruce Golden, Arjang Assad, Larry Levy, and Filip Gheysens. The fleet size and mix vehicle routing problem. *Computers & Operations Research*, 11(1):49–66, 1984. `doi:10.1016/0305-0548(84)90007-8`.

14 Diwakar Gupta and Brian Denton. Appointment scheduling in health care: Challenges and opportunities. *IIE transactions*, 40(9):800–819, 2008.

15 Elena Valentina Gutiérrez and Carlos Julio Vidal. Home health care logistics management problems: A critical review of models and methods. *Revista Facultad de Ingeniería Universidad de Antioquia*, 68:160–175, 2013.

16 Sandra Gutiérrez, Andrés Miniguano-Trujillo, Diego Recalde, Luis M Torres, and Ramiro Torres. The integrated vehicle and pollster routing problem. *arXiv*, 2019. `arXiv:1912.07356`.

17 Shuihua Han, Ling Zhao, Kui Chen, Zong-wei Luo, and Deepa Mishra. Appointment scheduling and routing optimization of attended home delivery system with random customer behavior. *European Journal of Operational Research*, 262(3):966–980, 2017. `doi:10.1016/j.ejor.2017.03.060`.

18 Vera C Hemmelmayr, Jean-François Cordeau, and Teodor Gabriel Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & operations research*, 39(12):3215–3228, 2012. `doi:10.1016/j.cor.2012.04.007`.

19 S. W Hess, J. B Weaver, H. J Siegfeldt, J. N Whelan, and P. A Zitlau. Nonpartisan political redistricting by computer. *Operations Research*, 13(6):998–1006, 1965. `doi:10.1287/opre.13.6.998`.

20   Solrun G Holm and Ragnhild O Angelsen. A descriptive retrospective study of time consumption in home care services: How do employees use their working time? *BMC Health Services Research*, 14(1):439–439, 2014. `doi:10.1186/1472-6963-14-439`.

21   Simge Küçükyavuz and Suvrajeet Sen. An introduction to two-stage stochastic mixed-integer programming. In *Leading Developments from INFORMS Communities*, pages 1–27. INFORMS, 2017. `doi:10.1287/educ.2017.0171`.

22   Ettore Lanzarone and Andrea Matta. A cost assignment policy for home care patients. *Flexible Services and Manufacturing Journal*, 24(4):465–495, November 2011. `doi:10.1007/s10696-011-9121-4`.

23   Gilbert Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009. `doi:10.1287/trsc.1090.0301`.

24   Canhong Lin, K.L Choy, G.T.S Ho, S.H Chung, and H.Y Lam. Survey of green vehicle routing problem: Past and future trends. *Expert Systems with Applications*, 41(4):1118–1138, 2014. `doi:10.1016/j.eswa.2013.07.107`.

25   Ran Liu, Biao Yuan, and Zhibin Jiang. A branch-and-price algorithm for the home-caregiver scheduling and routing problem with stochastic travel and service times. *Flexible Services and Manufacturing Journal*, 31(4):989–1011, 2019. `doi:10.1007/s10696-018-9328-8`.

26   P.A Maya Duque, M Castro, Kenneth Sörensen, and P Goos. Home care service planning. the case of landelijke thuiszorg. *European journal of operational research*, 243(1):292–301, 2015. `doi:10.1016/j.ejor.2014.11.008`.

27   Jorge Oyola, Halvard Arntzen, and David L Woodruff. The stochastic vehicle routing problem, a literature review, part i: models. *EURO Journal on Transportation and Logistics*, 7(3):193–221, 2018. `doi:10.1007/s13676-016-0100-5`.

28   César Rego, Dorabela Gamboa, Fred Glover, and Colin Osterman. Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European journal of operational research*, 211(3):427–441, 2011. `doi:10.1016/j.ejor.2010.09.010`.

29   María I Restrepo, Louis-Martin Rousseau, and Jonathan Vallée. Home healthcare integrated staffing and scheduling. *Omega (Oxford)*, 95:102057–, 2020. `doi:10.1016/j.omega.2019.03.015`.

30   Carlos Rodriguez, Thierry Garaix, Xiaolan Xie, and Vincent Augusto. Staff dimensioning in homecare services with uncertain demands. *International Journal of Production Research*, 53(24):7396–7410, 2015. `doi:10.1080/00207543.2015.1081427`.

31   Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006. `doi:10.1287/trsc.1050.0135`.

32   Nikolaos V Sahinidis. Optimization under uncertainty: state-of-the-art and opportunities. *Computers & Chemical Engineering*, 28(6-7):971–983, 2004. `doi:10.1016/j.compchemeng.2003.09.017`.

33   Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming – CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. `doi:10.1007/3-540-49481-2_30`.

34   Karmel S. Shehadeh and Mohan Chiriki. 13th aimms-mopta optimization modeling competition. In *Modeling and Optimization: Theory and Applications (MOPTA)*, 2021. URL: `https://coral.ise.lehigh.edu/~mopta/competition`.

35   Yong Shi, Toufik Boudouh, Olivier Grunder, and Deyun Wang. Modeling and solving simultaneous delivery and pick-up problem with stochastic travel and service times in home health care. *Expert systems with applications*, 102:218–233, 2018. `doi:10.1016/j.eswa.2018.02.025`.

36   Kenneth Sörensen and Marc Sevaux. A practical approach for robust and flexible vehicle routing using metaheuristics and monte carlo sampling. *Journal of mathematical modelling and algorithms*, 8(4):387, 2009. `doi:10.1007/s10852-009-9113-5`.

37 Biao Yuan, Ran Liu, and Zhibin Jiang. A branch-and-price algorithm for the home health care scheduling and routing problem with stochastic service times and skill requirements. *International Journal of Production Research*, 53:7450–7464, 2015. `doi: 10.1080/00207543.2015.1082041`.

38 Yang Zhan and Guohua Wan. Vehicle routing and appointment scheduling with team assignment for home services. *Computers & Operations Research*, 100:1–11, 2018. `doi: 10.1016/j.cor.2018.07.006`.

39 Yang Zhan, Zizhuo Wang, and Guohua Wan. Home service routing and appointment scheduling with stochastic service times. *European Journal of Operational Research*, 288(1):98–110, 2021. `doi:10.1016/j.ejor.2020.05.037`.

A   Appendix

A.1   Districting MIP Formulation

Let $[\![1, n]\!]$, be the set of customers and $\{0\}$ be the depot as before. Let $\omega_i \in R^+$ be the activity measure associated with customer i. The number of districts to be formed is the same as the pre-defined number of vehicles m. The average activity measure per district is defined as $\mu = \frac{1}{m} \sum_{i \in [\![1,n]\!]} \omega_i$. We denote $\omega_{\min} \leq 100$ and $\omega_{\max} \geq 100$ as the minimum and maximum percentage of activity measures in a district, respectively. L is the maximum allowed working time. Denote by $d_{i,j}$ the travel (Euclidean) distance between customers i and j. Finally, the decision variable $y_{i,j}$ is equal to one if customer i is assigned to the district centred at customer j, and it is zero otherwise. Here $y_{j,j}$ takes the value of one if customer j is selected to be the district centre. The districting MIP model can be defined as below:

$$\min \sum_{j \in [\![1,n]\!]} \sum_{i \in [\![1,n]\!]} \omega_i d_{i,j}^2 y_{i,j} \tag{7a}$$

$$\sum_{j \in [\![1,n]\!]} y_{i,j} = 1 \qquad\qquad \forall i \in [\![1, n]\!] \tag{7b}$$

$$\sum_{j \in [\![1,n]\!]} y_{j,j} = m \tag{7c}$$

$$y_{i,j} \leq y_{j,j} \qquad\qquad \forall j \in [\![1, n]\!] \tag{7d}$$

$$\sum_{i \in [\![1,n]\!]} \omega_i y_{i,j} \geq \frac{\omega_{\min}}{100} \mu \cdot y_{j,j} \qquad\qquad \forall j \in [\![1, n]\!] \tag{7e}$$

$$\sum_{i \in [\![1,n]\!]} \omega_i y_{i,j} + 2d_{0j} \leq L \qquad\qquad \forall j \in [\![1, n]\!] \tag{7f}$$

$$y_{i,j} \in \{0, 1\} \qquad\qquad \forall i, j \in [\![1, n]\!] \tag{7g}$$

Constraints (7b) require every customer to be assigned to a district. Constraint (7c) requires exactly m districts to be formed. Constraints (7d) state that each formed district must have a center. Constraints (7e) define the minimal workload of any district. Constraint (7f) stresses that the workload within each district, i.e. the activity measure within each district together with the pendulum tour to and from the depot, has to be no more than the total time allowance (or other self-defined upper bound using $\omega_{\max}$).

A.2 ALNS Improvement Heuristic

A.2.1 Destroy and Repair Operators

The algorithm removes a pre-defined number of nodes from the solution together with their linking arcs before adding them back iteratively, with the hope that the newly formed solution yields a smaller objective value. We introduce the whole list of destroy operators below:

1. *Random Removal:* a group of q randomly selected customers are removed from their existing routes and placed inside the customer pool.
2. *Worse Removal:* originally proposed in [31] to remove the q customers with the highest removal gain, which is the difference in cost when this customer is inside an allocated tour, and when the customer is not.
3. *Related Removal:* a single customer is randomly selected and moved together with the $(q-1)$ nearest customers from their tours to the customer pool.
4. *Tour Removal:* randomly remove a single tour. Move all the allocated customers from this single tour to the customer pool.
5. *Longest Tour Break into Half:* break the longest tour found into two smaller tours. Link the start and end of the smaller tours to the depot.
6. *Overcapacitated Tour Break into Half:* break all the infeasible tours (time capacity violated) in the middle and form two smaller tours. Link the start and end of the smaller tours to the depot.

The first three destroy operators are at the customer node level, and the latter three are at the routing level. We set default $q = 5$ from experimental results. Customers inside the customer pool will be re-inserted by an repair operator selected from below [18]:

1. *Greedy Insertion:* Randomly select a customer from the customer pool, insert it into the position that increases the total expected costs by the least. The insertion can be between two consecutive customers or between the depot and a linking customer.
2. *Greedy Insertion Perturbation:* The same mechanism as *Greedy Insertion*. However, the insertion cost of the selected customer at each specific position is influenced by a perturbation factor d between [0.8, 1.2].
3. *Greedy Insertion Forbidden:* The same mechanism as *Greedy Insertion*, only that a customer node cannot be re-inserted to the same position removed from.

Since destroy and repair operators (with local search) allow us to modify the number of existing tours, it therefore is possible to re-optimise the fleet size during the ALNS search. As a result, ALNS allows our first-stage heuristic to be less affected by a poor selection of service team number m at the beginning.

A.2.2 Local Search

Local search methods (*move*, *swap*, and *2-opt*) are applied after each destroy-repair iteration to further improve the repaired solutions. However, since local search is usually computationally expensive, we only wish to apply it to promising candidates whose objective values after the repair stage are within a limit of the best-found incumbent (default 30%). A graphical description of the *move*, *swap*, and *two-opt* methods is given in Figure 5.

A.2.3 Roulette Wheel Selector with Adaptive Weight

We apply the *roulette wheel* (a probabilistic mechanism) to independently select the destroy and repair operators at each iteration. An operator i is selected with probability $\mathsf{ow}_i / \sum_{k=1}^{K} \mathsf{ow}_k$, where K is the group of same-category operators and ow_i is operator i's
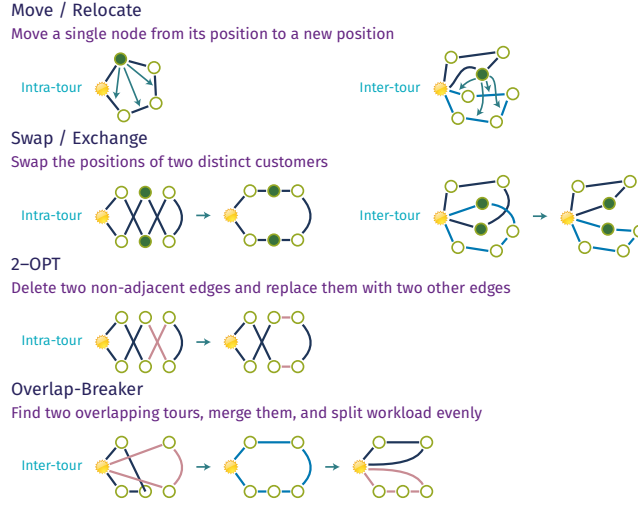
■ **Figure 5** Local Search Operators.

weight. The weight can be interpreted as the operator's "capability" to bring improvement to the incumbent (best current solution), and can be mathematically updated to reflect its in-time performance in the previous N iterations as well as its overall performance throughout the ALNS solution process. We can compute operator i's weight in segment $j + 1$:

$$\mathsf{ow}_{i,j+1} = \begin{cases} \mathsf{ow}_{i,j}(1 - r) + \frac{\pi_i}{Q_i}r & \text{if } \pi_i > 0, \\ \mathsf{ow}_{i,j} & \text{if } \pi_i = 0, \end{cases} \tag{8}$$

where $\mathsf{ow}_{i,j}$ is the weight of operator i in the previous segment j. A segment is a consecutive number of iterations during the solution process. π_i is the score that operator i earned in segment j for contributing to improving the incumbent's quality, and Q_i is the number of times operator i has been employed. Thus $\frac{\pi_i}{Q_i}$ is the average score operator i earns each time it was selected in segment j. This is weighted by a reaction factor r that controls how much the previous segment j determines each operator's overall performance. Here we choose $r = 1/2$ based on experimental results.

A.2.4   Acceptance and Stopping Criteria

ALNS has an embedded *simulated annealing* (SA) meta-heuristic served as the acceptance criterion, which allows the algorithm to accept a newly-found solution s' that not necessarily brings a lower total cost. SA contributes to ALNS's strong capability and robustness in exploring the solution neighbourhood with both diversification and intensification, allowing the search to escape from a local minimum and visit unexplored areas of the search space. Mathematically, we accept the new solution s' with probability $e^{f(s') - f(s)/\mathsf{Tem}}$ where s is the current solution and Tem the initial temperature.

For the stopping criteria, we force the search to terminate after either a certain amount of time or a prescribed number of non-improving iterations is reached.

A.2.5   Further Improvements

To further improve on the real-life practicality of our routes and schedules derived after the districting-first-routing-second construction heuristic (Section 5.2) and ALNS improvement heuristic (A.2), we have considered the following improvements for our first-stage solution: workload balance between teams, multiple tours overlapping minimisation, and single tour self-intersection elimination.

Each service team's assigned workload is bounded by (7e) and (7f), which means a team could still be assigned a much higher or lower workload compared to the rest of the teams. To further balance the workload amongst the teams, we include a soft workload balance penalty $P \cdot \max\{|\frac{\sum_{i \in k} \omega_i - \mu}{\mu}| - \alpha, 0\}$ in the ALNS objective function to penalise the extra units of workload above or below a certain threshold α for any service team (district k) and an average workload μ amongst all districts. We have chosen $\alpha = 0.3$ based on experimental results.

Occasional multiple tours overlapping is unavoidable, especially with a tight number of available service teams. Service durations have a larger scale than the inter-customer travel times, leading to customer assignments prioritising a good fit of customer service times into the remaining workload over the geographical adjacency. The randomness of customer geographical location can result in an unevenly high concentration of customers, challenging for the algorithm to form disjoint, compact, and contiguous driver zones within a reasonable computing time (since local search is computationally expensive). However, the application of *overlap-breaker* or *2-opt* (Figure 5) can remove the majority of overlaps and eliminate twisted tours that self-intersect.

On the Bike Spreading Problem

Elia Costa
Dept. of Information Engineering, University of Padova, Italy

Francesco Silvestri[1] ✉ 🏠 iD
Dept. of Information Engineering, University of Padova, Italy

───── **Abstract** ─────

A free-floating bike-sharing system (FFBSS) is a dockless rental system where an individual can borrow a bike and returns it anywhere, within the service area. To improve the rental service, available bikes should be distributed over the entire service area: a customer leaving from any position is then more likely to find a near bike and then to use the service. Moreover, spreading bikes among the entire service area increases urban spatial equity since the benefits of FFBSS are not a prerogative of just a few zones. For guaranteeing such distribution, the FFBSS operator can use vans to manually relocate bikes, but it incurs high economic and environmental costs. We propose a novel approach that exploits the existing bike flows generated by customers to distribute bikes. More specifically, by envisioning the problem as an Influence Maximization problem, we show that it is possible to position batches of bikes on a small number of zones, and then the daily use of FFBSS will efficiently spread these bikes on a large area. We show that detecting these zones is NP-complete, but there exists a simple and efficient $1 - 1/e$ approximation algorithm; our approach is then evaluated on a dataset of rides from the free-floating bike-sharing system of the city of Padova.

1 Introduction

A *bike-sharing system* (BSS) is a service where an individual can rent a bike and return it after a short term. Nowadays, almost all large cities have adopted a BSS as it is a sustainable transportation system that helps improving air pollution, public health, and traffic congestion [20]. The first BSS dates back to the 1960s (with Witte Fietsen in Amsterdam), and we have seen an explosion of BSSs in the last decade, with now about 2000

---

[1] Corresponding author.

operators currently managing more than 9.7 million bikes around the world [13]. The majority of BSSs are now connected to IT systems that allow to borrow bikes from a smartphone and to collect data on users and rides. Similar systems exist for e-bikes, scooters, mopeds, and cars.

There are two major approaches to BSSs: station-based and free-floating systems. A *station-based bike-sharing system* (SBBSS) represents the most common approach, where a user borrows a bike from a dock and returns it at another dock belonging to the same system; in an SBBSS, the set of origins and destinations of all rides is small and coincides with dock positions. On the other hand, a *free-floating bike-sharing system* (FFBSS) is a sharing model with no docks: each bike has an integrated lock that can be opened on demand with a smartphone app and returned just by closing the lock. As there are no more fixed docks, bikes can be positioned anywhere (hopefully, respecting traffic codes) and the origin/destination of a ride can be any position within the service area. FFBSS is also an interesting solution for the first and last kilometer problem in multimodal transportation cities [16] since the average walking distance of a user to the closest free-floating bike is shorter than SBBSS.

The distribution of bikes in the service area is crucial for to increase user satisfaction. In SBBSS, the main challenge is dealing with hotspots, specifically zones where several rides start (sources) or end (sinks), such as train stations or university campuses. Hotspots are critical: sources and sinks might suffer, respectively, from the lack of available bikes and parking slots in a deck. The operator needs to detect and manage these hotspots: bikes should be collected from sinks and repositioned on sources. The number of hotspots is usually small and several efficient computational strategies have been investigated (see e.g. references in [16]). In FBBSS, hotspots are still critical although customers do not experience the lack of empty slots in the returning docks as in SBBSS.

In addition to dealing with hotspots, FBBSS needs also to guarantee that the entire service area is covered by bikes so that a user leaving from any point can find a near available bike. A study [7] has indeed shown that every additional meter of walking to a shared bike decreases a user's likelihood of using a bike by 0.194% for short distances ($\leq 300m$) and 1.307% for long distances ($> 300m$), implying that a user walking a distance $> 500m$ for reaching the closest bike is highly unlikely to use the system. Moreover, if bikes are well distributed all over the service area, the spatial equity improves since the benefits of the service are not a prerogative of just some zones (e.g., city center) [14]. Assume the service area to be split into zones (e.g., quadrants) where the diameter of each zone is considered a reasonable walking distance (e.g., ≤ 500m): then, the desired goal is that each zone has a sufficient number of bikes.

To distribute bikes over the service area, an FFBSS operator could employ a fleet of vans to manually position bikes in each zone: however, this solution might be economically and environmentally unfeasible due to a large number of zones. In this paper, we provide a novel and alternative approach that distributes bikes over the service area by exploiting the existing customers' bike flows and hence reducing intervention by the FFBSS operator. The idea is to detect a small number of zones, named *seeds*, where bikes are more likely to be spread over the service area by the regular activity of customers during a given time interval. The seeds represent the positions where the FFBSS operator can position batches of bikes, which will then be spread over the entire area by customers, without further interventions from the operator. Formally, we modeled this approach as a variant of the Influence Maximization (IM) problem, which we name the *Bike Spreading (BS) problem*.

To detect these seeds, we first define a weighted graph representing mobility flows: nodes are zones, and weighted edges represent the probability that a bike moves from one node to another. Then, we introduce a diffusion model to analyze how bikes move and a spread score to evaluate the quality of the final bike distribution. Finally, we detect a small subset of nodes that maximizes the spread score and position bikes on these nodes. This formulation results in an NP-complete problem, but we show that there exists a $1 - 1/e$-approximation algorithm to the problem thanks to some properties (e.g., submodularity and monotonicity).

More specifically, the results provided in the paper are the following:

- In Section 3, we introduce the Bike Spreading problem and formalize two versions called `T-BS` and `U-BS`: the `T-BS` version aims at maximizing the number of zones with a minimum amount $\gamma > 0$ of bikes, while the goal of `U-BS` is to uniformly distribute bikes in the service area.
- In Section 4, we analyze the theoretical properties of `T-BS` and `U-BS`: we show their NP-completeness and that `U-BS` satisfies the monotonicity and submodularity properties. By these properties and the result by Nemhauser et al. [15], we get a simple greedy algorithm providing a $1 - 1/e$-approximation for `U-BS`.
- In Section 5, we experimentally investigate the `BS` problem by using data from the free-floating bike service of the city of Padova (Italy). We analyze the performance and quality of the solution of the greedy algorithm, compare the `U-BS` and `T-BS` problems, and make some empirical considerations on the `BS` problem.

2 Preliminaries

2.1 Free-floating bike-sharing service

The studies related to bike-sharing systems mainly involve two topics: demand prediction and rebalancing. Demand analysis involves the understanding of user behavior and providing the most appropriate service (e.g., [12]). Rebalancing has mainly focused on station-based systems (see e.g. [11, 5, 19, 16]), while only a few works have addressed free-floating bikes. Reiss and Bogenberger [17] investigated the relocation strategy and a validation method on Munich's FFBSS; their approach focused on finding the best zones where to relocate bikes to satisfy user demand and minimize bikes' idle time. Pal and Zang [16] and Usamaa et al. [18] focused on finding the best route of relocation vans under costs and time constraints; in [18], picking up faulty bikes was also included. Finally, Caggiani et al. [3] proposed a forecast model aiming at reducing the number of times when a zone has fewer bikes than necessary. These works focused on computing an optimal route to collect and relocate bikes, and on detecting hotspots where to relocate more bikes. To the best of our knowledge, no previous works have studied how to spread bikes over the entire service area and exploited the mobility graph as in our work.

2.2 The Influence Maximization problem

The *Influence Maximization* (`IM`) problem [8] is widely used in social network analysis to detect the most influential users that can efficiently spread information, like news or advertisements; `IM` is also used in epidemiology for analyzing how infections evolve via human interactions. However, differently from news and infections, bikes do not replicate: we then need to redefine the `IM` framework to deal with a fixed amount of objects spreading over the graph. In general, an Influence Maximization problem consists of three components:

- A directed and weighted graph where vertexes represent users, and edges represent the paths where information can propagate from a given node.
- A diffusion model that describes how information moves in the graph. The model specifies the initial status (e.g., the nodes that initially contain the information) and how information distributes. The model advances in steps: in each step, a node detects which other nodes in the neighborhood will receive the information and propagates it.
- An evaluation function $\sigma(\Upsilon)$, which receives a description Υ of how the information is distributed in the graph and provides a non-negative real value. Large values denote better and more desired distributions; e.g., $\sigma(\Upsilon)$ can represent the total number of nodes that have seen some information.

The goal of IM is to find an initial distribution that maximizes the evaluation function $\sigma(\Upsilon)$ after a given number of steps.

The diffusion model is the critical part of IM problems, indeed they delineate the way nodes influence each other. The most commonly used diffusion models [4, 10] are Independent Cascade, Linear Thresholds, Triggering, and Time Aware: these model are progressive models, in the sense that once a node has been influenced it cannot change its status. There are also non-progressive diffusion models: some examples are the Susceptible-Infected-Susceptible (SIS) models [9], widely used in epidemiology.

The bike spreading problem can be viewed as a IM problem where the diffusion model propagates objects (i.e., bikes). The main difference is that objects cannot be replicated and their amount is constant. In contrast, previous works on information or infection diffusion allow replicas: for instance, an infected person can infect many other persons and an image can be shared with friends in a social network.

2.3   Approximating submodular functions

Nemhauser et al. [15] proved that a non-negative, monotone submodular function can be efficiently approximated by a simple greedy algorithm, within a factor $1 - 1/e \sim 0.63$. This result is used by Kempe et al. [8] to obtain an approximate solution for the IM problem under both Independent Cascade and Linear Threshold models.

Consider a function f mapping a set of elements in U to a non-negative real value, i.e. $f : U^* \to \mathbb{R}^+$. Intuitively, a function f is monotone if, by expanding a given input set, the value of the function does not decrease; a function f is submodular if the marginal gain does not increase when adding more elements to the input set.

▶ **Definition 1** (Monotonicity). *Given a function $f : U^* \to \mathbb{R}^+$ where U is a set of elements, function f is said to be* monotone *if, for every $S \subseteq U$ and $v \in U$, we have $f(S \cup \{v\}) \geq f(S)$.*

▶ **Definition 2** (Submodularity). *Given a function $f : U^* \to \mathbb{R}^+$ where U is a set of elements, function f is said to be* submodular *if, for every $S, T \subseteq U$ with $S \subseteq T$ and $v \in U \setminus T$, we have $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$.*

The aforementioned result by Nemhauser et al. [15] is the following:

▶ **Theorem 3** ([15]). *Let $f : U^* \to \mathbb{R}^+$ be a non-negative, monotone submodular function where U is a set of elements. Let $k \geq 1$ be a given value and S^* be a set of k elements in U maximizing the value of f among all sets of size k. Then, there exists a greedy algorithm that returns a set S of k elements such that $f(S) \geq 1 - (1 - 1/k)^k f(S^*) \geq (1 - 1/e)f(S^*)$, where e is the base of the natural logarithm.*

The greedy algorithm is quite simple and it consists of extending a given set S with the node u^* that maximizes the marginal gain, i.e., $u^* = \arg\max_{u \in U}\{f(S \cup \{u\}) - f(S)\}$ until we get a set of k elements.

```
greedy(f, k)
  S = ∅
  for  i = 0 to  k − 1
          u* = arg max_{u∈U}{f(S ∪ {u}) − f(S)}
          S = S ∪ {u*}
  return  S
```

The algorithm takes time $O(k\Delta)$ where Δ is the maximum cost of finding the element of U that maximizes the marginal gain.

3   The Bike Spreading problem

In this section we formalize the Bike Spreading problem. We first propose a general definition based on the Influence Maximization problem, and then we consider two special cases, named `U-BS` and `T-BS` problems, that target specific bike distributions.

The Bike Spreading (`BS`) problem can be viewed as a special case of the Influence Maximization problem, where the entity that is distributed across the graph are items that cannot be replicated, differently from news and infections.[2] We represent a city as a *directed* graph $G = (V, E)$: V is the set of nodes, where every node represents a different zone of the city; E is the set of directed weighted edges capturing the probability of moving from one zone to another. We define $n = |V|$ and $m = |E|$, and let $\Gamma^{IN}(v) = \{w \in V | (w, v) \in E\}$ and $\Gamma^{OUT}(v) = \{w \in V | (v, w) \in E\}$ denote the set of nodes for which v is the destination or the source, respectively. The weight of an edge $e = (u, v) \in E$ represents the *probability* p_e that a bike in node u moves from u to v: intuitively, p_e is the probability that an user renting a bike in zone u ends her/his ride into zone v. We use self-loops to represent bikes that stay in the same node (i.e., for bikes that are not used or are borrowed for a ride starting and ending in the same node). Then, for each node $u \in V$, we have:

$$\sum_{v \in \Gamma^{OUT}(u)} p_{(u,v)} = 1. \tag{1}$$

We assume bikes to be initially positioned on $k \geq 1$ nodes in groups of $L \geq 1$ bikes, and we refer to the initial set of these k nodes as (k, L)-*seed*. Bikes can be damaged or stolen in a free-floating bike system, however, this does not significantly affect the total number of bikes in the system: therefore, we assume the total number of bikes in the graph to be fixed, and we let B denote the total number of bikes, i.e. $B = k \cdot L$.

The diffusion model unfolds in $\tau \geq 1$ discrete steps. At any step $1 \leq t \leq \tau$ and for each vertex $v \in V$, we define the *load* $\ell_{k,L}(v, t, \mathcal{S}) \geq 0$ which represents the (expected) number of bikes in node v after t steps starting from a (k, L)-seed set $\mathcal{S}$. We let $\ell_{k,L}(v, 0, \mathcal{S})$ denote the initial loads: $\ell_{k,L}(v, 0, \mathcal{S}) = L$ for each $v \in \mathcal{S}$, and $\ell_{k,L}(v, 0, \mathcal{S}) = 0$ otherwise. At any time $0 < t \leq \tau$, bikes move according to edge directions and probabilities; each vertex load $\ell_{k,L}(v, t, \mathcal{S})$ is updated as follows:

$$\ell_{k,L}(v, t, \mathcal{S}) = \ell_{k,L}(v, t - 1, \mathcal{S}) + \Delta^{IN}(\mathcal{S}) - \Delta^{OUT}(\mathcal{S})$$

---

[2] The model presented in our paper is not atomic: we allow a bike to be "split" since the value of a node should be understood as the expected number of bikes in that node.

where the two rightmost terms are defined as

$$\Delta^{IN}(\mathcal{S}) = \sum_{u \in \Gamma^{IN}(v)} p_{(u,v)} \ell_{k,L}(u, t-1, \mathcal{S}), \quad \Delta^{OUT}(\mathcal{S}) = \sum_{u \in \Gamma^{OUT}(v)} p_{(v,u)} \ell_{k,L}(v, t-1, \mathcal{S}).$$

Intuitively, the bikes in a node v are partitioned among the outgoing edges according to their probabilities, and then the load of v is decreased by the number of bikes leaving the node (i.e., $\Delta^{OUT}(\mathcal{S})$) and increased by the number of bikes entering in the node (i.e., $\Delta^{IN}(\mathcal{S})$). As $\sum_{u \in \Gamma^{OUT}(v)} p_{(v,u)} = 1$ due to self-loops, the above update can be rewritten as follows:

$$\ell_{k,L}(v, t, \mathcal{S}) = \sum_{u \in \Gamma^{IN}(v)} p_{(u,v)} \ell_{k,L}(u, t-1, \mathcal{S}).$$

We let $\mathcal{L}_{k,L}(t, \mathcal{S}) = (\ell_{k,L}(v_0, t, \mathcal{S}), \ldots, \ell_{k,L}(v_{n-1}, t, \mathcal{S}))$ denote the loads of all nodes in G after $1 \leq t \leq \tau$ steps and with the (k, L)-seed $\mathcal{S}$. We remark that $\ell_{k,L}(v, t, \mathcal{S})$ represents the expected number of bikes in node v after t steps, where all bikes in a node are spread among the neighbors nodes in every time step uniformly at random according to edge probabilities.

To measure the quality of bike distribution among nodes, we introduce the notion of spread. The *spread* $\sigma(\mathcal{L}_{k,L}(t, \mathcal{S}))$ after t steps and with (k, L)-seed $\mathcal{S}$ is a function $\mathbb{R}^n \to \mathbb{R}$ that evaluates the quality of loads $\mathcal{L}_{k,L}(t, \mathcal{S})$. The actual formulation of the spread function depends on the desired distribution in a free-floating bike system. We will see two examples of spread aiming at maximizing the number of nodes with a given minimum load, and at uniformly distributing bikes among all nodes of the graph. We are now ready to define the Bike Spreading problem as:

▶ **Definition 4.** *Given a directed graph $G = (V, E)$, positive integers $L \geq 1$, $k \geq 1$ and $\tau \geq 1$, and spread $\sigma(\cdot)$, the* Bike Spreading (BS) *problem asks for the (k, L)-seed $\mathcal{S}^*$ that maximizes $\sigma(\mathcal{L}_{k,L}(\tau, \mathcal{S}))$. More specifically, the mathematical formulation of the problem is*

$$\mathcal{S}^* = \underset{\mathcal{S} \subseteq V, |\mathcal{S}| = k}{\arg\max} \ \sigma(\mathcal{L}_{k,L}(\tau, \mathcal{S}))$$

such that:
- $\ell_{k,L}(v, 0, \mathcal{S}) = L, \quad \forall v \in \mathcal{S}$
- $\ell_{k,L}(v, 0, \mathcal{S}) = 0, \quad \forall v \in V \setminus \mathcal{S}$
- $\ell_{k,L}(v, t, \mathcal{S}) = \sum_{u \in \Gamma^{IN}(v)} p_{(u,v)} \ell_{k,L}(v, t-1, \mathcal{S}), \quad \forall v \in V \ and \ t = 1, \ldots, \tau$

We remark that in this paper we focus on small values of τ (i.e., the maximum number of steps) and thus we do not analyze the convergence of the spread for $\tau \to +\infty$. As the distribution should happen within a couple of hours from the positioning of bike batches, we expect each bike to be used just a few times and thus $\tau \leq 5$ from a practical point of view.

We now describe the two spread functions used in the paper, namely T-BS and U-BS.

T-BS version

The first example of spread leverages on the idea that a zone is considered well served by the free-floating bike system if there are at least γ bikes (in expectation), where $\gamma > 0$ is a given threshold value. Therefore, the spread counts the number of nodes (i.e., zones) that have at least γ bikes:

$$\sigma_\gamma^{(T)}(\mathcal{L}_{k,L}(\tau, \mathcal{S})) = |\{v \in V | \ell_{k,L}(v, \tau, \mathcal{S}) \geq \gamma\}| \tag{2}$$

We refer to the BS formulation with the spread in Equation 2 as *Threshold BS (T-BS)* problem.

The **T-BS** problem is an all-or-nothing approach, where only nodes receiving a sufficiently large number of bikes are relevant. However such a solution would not penalize skewed distributions: for instance, **T-BS** gives the same score to a distribution of n nodes with load 2γ and a skewed distribution of $n - 1$ nodes with load γ and one node with load $(n + 1)\gamma$. Therefore, if the goal is to maximize the area covered by the service, the ideal distribution should be the uniform distribution with B/n bikes per node. We thus introduce the following spread:

$$\sigma^{(U)}\left(\mathcal{L}_{k,L}\left(\tau, \mathcal{S}\right)\right) = \sum_{v \in V} \sqrt{\ell_{k,L}\left(v, \tau\right)} \tag{3}$$

The maximum value of $\sigma^{(U)}\left(\mathcal{L}_{k,L}\left(\tau, \mathcal{S}\right)\right)$ is reached when bikes are equally distributed over the graph, that is $\ell_{k,L}\left(v, \tau\right) = B/n$. We refer to the **BS** formulation with the spread in Equation 3 as *Uniform BS (U-BS)* problem.

4 Theoretical analysis

In this section, we first prove the NP-completeness of **T-BS** and **U-BS** problems. Then we show that **U-BS** satisfies the monotonicity and submodularity properties and hence, by the result of Nemhauser et al. [15], there exists a greedy algorithm providing a $(1 - 1/e)$-approximate solution to **U-BS**.

4.1 NP-completeness

T-BS is NP-complete

We prove the NP-completeness of **T-BS** with a reduction from the *Minimum Dominating Set* (*MDS*) problem which is known to be NP-complete even in graphs with constant vertex degree $d \geq 3$ [2, 1]. Let $G = (V, E)$ be a simple, connected, undirected graph with degree $d \geq 3$ for all nodes in V. A subset $S \subseteq V$ is a *dominating set* if for every vertex $v \in V \setminus S$, there exists a vertex $u \in S$ such that $(v, u) \in E$; that is, every vertex outside S has at least one neighbor in S. A minimum domination set of G is a domination set of the smallest possible size, and we refer to its size with *domination number* $\gamma(G)$. Since the degree of every node in G equals d, we have $\gamma(G) \leq n - d + 1$, because in a set $S \subseteq V$, $|S| \geq n - d + 1$, every node has at least one neighbor in S.

The decision problem associated with **MDS** is defined as:

- INSTANCE: a undirected graph $G = (V, E)$ with fixed degree $d \geq 3$ and an integer k with $0 < k < n - d + 1$.
- QUESTION: Does a dominating set with $\gamma(G) \leq k$ exist?

The **T-BS** problem can be represented by the following decision problem:

- INSTANCE: a directed and weighted graph $G = (V, E)$ that satisfies Equation 1, integers k, L, γ, τ and λ with $0 < k < n$, $L \geq 1$, $\gamma > 0$, $\tau \geq 1$, and $0 < \lambda \leq n$.
- QUESTION: Does a (k, L)-seed $\mathcal{S} \subseteq V$ exist in G such that $\sigma_\gamma^{(T)}\left(\mathcal{L}_{k,L}\left(\tau, \mathcal{S}\right)\right) \geq \lambda$?

The reduction of **MDS** to **T-BS** is the following. Given an instance of the **MDS** problem on a graph G with degree $d \geq 3$ and integer k, we construct a new weighted and directed graph G' for the **T-BS** problem by directing all edges of G in both directions and by adding a self-loop to each vertex, obtaining a new graph $G' = (V, E')$ with outdegree $d' = d + 1$.

The probability of each edge is then set to $p_e = 1/d'$. We then solve the `T-BS` problem on G' with a seed with size k and $L = d'$ bikes in each vertex in the seed, threshold $\gamma = 1$, step number $\tau = 1$, and $\lambda = n$. The reduction returns yes to the d-`MDS` problem if and only if `T-BS` returns yes, that is if there exists a (k, d')-seed set $\mathcal{S}$ in G' with $\sigma_1^{(T)} (\mathcal{L}_{k,d'} (1, \mathcal{S})) \geq n$.

▶ **Theorem 5.** *The `T-BS` problem is NP-complete.*

Proof. The `T-BS` problem is in NP since a solution can be verified in polynomial time as follows. Let A be the adjacency matrix of a graph G, then matrix A^τ represents the percentage of bikes in a node u that reach a node v, for each $u, v \in V$, using paths of length τ (including possibly self-loops). Matrix A^τ can be computed in $O\left(n^3 \log \tau\right)$ time with the doubling trick. Given a seed $\mathcal{S}$ and an n-dimensional vector $\ell_{\mathcal{S}}$ encoding the initial loads, then the loads at step τ can be computed with $A^\tau \cdot \ell_{\mathcal{S}}$ in $O\left(n^2\right)$ time. Therefore $\sigma_\gamma^{(T)} (\mathcal{L}_{k,L} (\tau, \mathcal{S}))$ for a given seed can be computed in $O\left(n^3 \log \tau\right)$ time.

We now show the correctness of the reduction. We first prove that, if there exists a dominating set $\mathcal{S}$ of size $k' \leq k$ in G, then there exits a (k, d')-seed in G' giving $\sigma_1^{(T)} (\mathcal{L}_{k,d'} (1, \mathcal{S})) \geq n$. Let us assume for simplicity that $\mathcal{S}$ has size $k' = k$: it suffices to add nodes to $\mathcal{S}$ till reaching size k. Since there are $L = d'$ bikes in each node of the seed set, $p_e = 1/d'$ and there are exactly d' outgoing edges from each node, we have that each outgoing edge of a node in $\mathcal{S}$ is crossed by one bike. By definition, the dominating set $\mathcal{S}$ covers each vertex in $V \setminus \mathcal{S}$ with at least one edge; moreover, each node in $\mathcal{S}$ has a self-loop. Therefore, each node in V receives at least on bike and hence $\sigma_1^{(T)} (\mathcal{L}_{k,d'} (1, \mathcal{S})) = n$.

Conversely, if a dominating set $\mathcal{S}$ of size at most k does not exist in G, then there cannot be a set $\mathcal{S}'$ of size k in G' with $\sigma_1^{(T)} (\mathcal{L}_{k,d'} (1, \mathcal{S})) \geq n$. Indeed, since a dominator set of size $\leq k$ does not exists, it means that for any set $\mathcal{S}'$ of $\leq k$ nodes from V there is at least one node $v \in V$ which is not adjacent to nodes in $\mathcal{S}'$. Therefore, for any seed set $\mathcal{S}'$ of k nodes there exists a node v not receiving any bike; it follows that $\ell_{k,d'} (v, 1, \mathcal{S}') = 0$ and thus $\sigma_1^{(T)} (\mathcal{L}_{k,d'} (1, \mathcal{S})) \leq n - 1$. ◀

`U-BS` is NP-complete

We use a reduction from the *Exact Cover by 3-Sets* (*X3C*) problem, as `MDS` does not work for `U-BS` due to its spread function. `X3C` is NP-complete [6], and consists of a covering problem with sets of three elements. Given a set X with $|X| = 3q$, for some integer $q \geq 1$, and a collection C of 3-element subsets of X, `X3C` requires to decide if there exists a subset $C' \subseteq C$ such that C' covers X and every element of X occurs in exactly one set in C' (i.e., C' is an *exact cover* of X). For clarity, consider the following example: let $X = \{1, 2, 3, 4, 5, 6\}$ and $C = \{\{1, 2, 3\}, \{2, 3, 4\}, \{1, 2, 5\}, \{2, 5, 6\}, \{1, 5, 6\}\}$; then, the collection $C' = \{\{2, 3, 4\}, \{1, 5, 6\}\} \subset C$ is an exact cover because each element in X appears exactly once. Note that, if $C = \{\{1, 2, 3\}, \{2, 4, 5\}, \{2, 5, 6\}\}$, then any collection C' cannot be an exact cover: indeed, every pair of sets in C shares at least one entry, and hence every collection C' covers at least one element twice or more. Note that if we do have an exact cover, C' will contain exactly q elements.

The decision problem associated with `X3C` is:

- INSTANCE: a set X, with $|X| = 3q$ for some integer $q \geq 1$, a collection C of 3-element subsets of X.
- QUESTION: does a set $C' \subseteq C$ exist such that every element of X occurs in exactly one member of C'?

The decision problem for U-BS is:
- INSTANCE: a directed and weighted graph $G = (V, E)$ that satisfies Equation 1, integers k, L, τ, λ with $0 < k < n$, $L \geq 1$, $\tau \geq 1$, $\lambda \geq 0$.
- QUESTION: Does a (k, L)-seed $\mathcal{S} \subseteq V$ exist in G such that $\sigma^{(U)}(\mathcal{L}_{k,L}(\tau, \mathcal{S})) \geq \lambda$?

The reduction from X3C to U-BS is the following. Given an instance of the X3C, defined by a set of $3q$ elements $X = \{x_1, \ldots, x_{3q}\}$ and a collection $C = \{c_1, .., c_r\}$ of 3-element subsets of X, we build a directed and weighted graph $G = (V, E)$ for the U-BS problem as follows. With a slight abuse of notation, we let $V = C \cup X$, that is, each element in X and each set in C are represented by a node in V. Then we set $E = E_1 \cup E_2$: E_1 contains an edge (c_i, x_j) if $x_j \in c_i$, for each $c_i \in C$ and $x_j \in X$; E_2 contains a self-loop for each node $x_i \in X$. We observe that nodes in X have only one outgoing edge, while nodes in C have three outgoing edges: then, we set the probability of each self-loop in E_2 to 1, while we set $p_e = 1/3$ for all the remaining edges in E_1. We then run the U-BS problem on G with $k = q$ nodes in the seed, $L = 3$ bikes per node in the seed, $\tau = 1$ steps, and $\lambda = 3q$, and we answer yes to X3C if and only if U-BS returns yes, that is if there exists a $(q, 3)$-seed set $\mathcal{S}$ in G with $\sigma^{(U)}(\mathcal{L}_{q,3}(1, \mathcal{S})) \geq 3q$.

▶ **Theorem 6.** *The U-BS problem is NP-complete.*

Proof. U-BS is in NP as a solution can be checked in polynomial time $O(n^3 \log \tau)$ as shown in the proof of Theorem 5.

We now prove the correctness of the reduction. We first observe that any exact cover for X3C must contain q entries from C, otherwise X is not covered or an element in X is covered by more than one set in C'. We now prove that, if the X3C problem contains an exact cover C', then there exists a $(q, 3)$-seed $\mathcal{S}$ in G with $\sigma^{(U)}(\mathcal{L}_{q,3}(1, \mathcal{S})) \geq 3q$. Let $\mathcal{S}$ be a seed set given by the nodes c_i representing sets in C'. As each node $x \in X$ is covered by exactly one node in C' and the outgoing degree of a node in C' is 3, we have that x receives one bike after the first step. Then: $\sigma^{(U)}(\mathcal{L}_{q,3}(1, \mathcal{S})) = \sum_{v \in V} \sqrt{\ell_{q,3}(v, 1, \mathcal{S})} = \sum_{v \in C} 0 + \sum_{v \in X} 1 = 3q$.

Assume now that X3C does not have an exact cover: then any C' of q sets from C does not cover at least one point of X and covers at least one point of X more than once. Assume by contradiction that U-BS returns a seed $\mathcal{S}$ of $k = q$ nodes with $\sigma^{(U)}(\mathcal{L}_{q,3}(1, \mathcal{S})) \geq 3q$. We claim that $\mathcal{S}$ has no nodes in X. If $\mathcal{S}$ has a node $x \in X$, then the load of x after one step is $h = \ell_{q,3}(x, 1, \mathcal{S}) = \sqrt{3}$ since every node in X has only the self-loop as outgoing edge. Since bikes positioned in nodes of C' move in nodes of X after one step, we get:

$$\sigma^{(U)}(\mathcal{L}_{q,3}(1, \mathcal{S})) = \sqrt{h} + \sum_{v \in X \setminus \{x\}} \ell_{q,3}(v, 1, \mathcal{S}).$$

By the concavity of the square root, the right summation is maximized when all loads are equal and, since $|X| = 3q$, we get $\sigma^{(U)}(\mathcal{L}_{q,3}(1, \mathcal{S})) \leq \sqrt{h} + \sqrt{(3q - 1)(3q - h)}$. The right term of the inequality decreases for $h > 1$, and then $\sigma^{(U)}(\mathcal{L}_{q,3}(1, \mathcal{S})) < 3q$, which is a contradiction. Therefore, we must have that $\mathcal{S}$ contains only nodes in C. However, since there is no exact cover, at least one node in X must receive two or more bikes: by mimic the previous argument, we get that $\sigma^{(U)}(\mathcal{L}_{q,3}(1, \mathcal{S})) < 3q$. Therefore there is no $(q, 3)$-seed giving $\sigma^{(U)}(\mathcal{L}_{q,3}(1, \mathcal{S})) \geq 3q$. ◀

4.2 Approximation algorithms

By the previous hardness results, we do not expect polynomial-time exact algorithms for the T-BS and U-BS problems. In this section, by showing that U-BS satisfies the monotonicity and submodularity properties, we get that the greedy solution in Section 2.3 gives a $(1 - 1/e)$-approximation algorithm for U-BS. The T-BS version does not satisfy the submodularity

property and thus similar theoretical guarantees cannot be provided: however, in the following section, we show that the greedy algorithm experimentally provides a good approximation even for `T-BS`.

We start with a technical lemma and then show that `U-BS` satisfies the monotonicity and submodularity properties.

▶ **Lemma 7.** *Let S and T be two seed sets with $S \subseteq T \subseteq V$, then $\ell_{k,L}(v, t, T) \geq \ell_{k,L}(v, t, S)$ for each $v \in V$ and $t \geq 0$.*

Proof. The proof follows by induction over the number of steps t. It is true in the base case when $\tau = 0$ since:
1. $\ell_{k,L}(v, 0, T) = \ell_{k,L}(v, 0, S) = 0$ for each $v \notin T$;
2. $\ell_{k,L}(v, 0, T) = \ell_{k,L}(v, 0, S) = L$ for each $v \in S$;
3. $\ell_{k,L}(v, 0, T) = L$ and $\ell_{k,L}(v, 0, S) = 0$ for each $v \in T \setminus S$.

Now suppose that the claim is true for $t \geq 0$. Then at step $t + 1$, we have:

$$\ell_{k,L}(v, t+1, T) = \sum_{w \in \Gamma^{IN}(v)} p_{(w,v)} \ell_{k,L}(w, t, T)$$

$$\geq \sum_{w \in \Gamma^{IN}(v)} p_{(w,v)} \ell_{k,L}(w, t, S) = \ell_{k,L}(v, t+1, S).$$

The claim follows.   ◀

▶ **Lemma 8.** *Given the `U-BS` problem with a seed set size k and given parameters τ, L independent of k, then $\sigma^{(U)}(\mathcal{L}_{k,L}(\tau, \mathcal{S}))$ is monotone and submodular.*

Proof. The monotonicity follows from Lemma 7 and the monotonicity of square root. We now consider submodularity. Since the parameters τ, L are given and are independent of k, we define $\sigma(\mathcal{S}) = \sigma^{(U)}(\mathcal{L}_{k,L}(\tau, \mathcal{S}))$ for notational simplicity. By Definition 2, we have to prove that $\sigma(\mathcal{S} \cup \{v\}) - \sigma(\mathcal{S}) \geq \sigma(T \cup \{v\}) - \sigma(T)$ for all $v \in V$ and $\mathcal{S} \subseteq T \subseteq V$. Let Υ_v be the set of nodes in V that can be reached from v with a path of length τ (possibly with self-loops): nodes in Υ_v are all and only the nodes whose load can be affected by the seed in v. We have:

$$\sigma(\mathcal{S} \cup \{v\}) - \sigma(\mathcal{S}) = \sum_{u \in \Upsilon_v} \sqrt{\ell_{k,L}(u, \tau, \mathcal{S} \cup \{v\})} - \sqrt{\ell_{k,L}(u, \tau, \mathcal{S})}$$

$$= \sum_{u \in \Upsilon_v} \sqrt{\ell_{k,L}(u, \tau, \mathcal{S}) + \ell_{k,L}(u, \tau, \{v\})} - \sqrt{\ell_{k,L}(u, \tau, \mathcal{S})}.$$

For any $\beta \geq 0$, we have that $\sqrt{\alpha + \beta} - \sqrt{\alpha}$ is a non increasing function of α, and hence the last term of the previous inequality is lower bounded by

$$\sum_{u \in \Upsilon_v} \sqrt{\ell_{k,L}(u, \tau, T) + \ell_{k,L}(u, \tau, \{v\})} - \sqrt{\ell_{k,L}(u, \tau, T)} = \sigma(T \cup \{v\}) - \sigma(T).$$

We then get $\sigma(\mathcal{S} \cup \{v\}) - \sigma(\mathcal{S}) \geq \sigma(T \cup \{v\}) - \sigma(T)$ that proves the submodularity of `U-BS`.   ◀

Then, from Theorem 3 and the above lemma, we get the following result.

▶ **Corollary 9.** *There exists a $(1 - 1/e)$-approximation algorithm for the `U-BS` problem requiring $O\left(n^3(\log \tau + k)\right)$ time.*

Proof. The result automatically follows from Theorem 3 by the monotonicity and submodularity properties of the spread function of U-BS. The greedy algorithm in Section 2.3 gives the $(1 - 1/e)$-approximation. Computing the spread for a given seed set $\mathcal{S}$ requires $O\left(n^2\right)$ time: loads can indeed be computed by the multiplication $A^\tau \cdot \ell_{\mathcal{S}}$, where A is the adjacency matrix of graph G and $\ell_{\mathcal{S}}$ is an n-dimensional vector encoding nodes in $\mathcal{S}$. Matrix A^τ is computed in $O\left(n^3 \log \tau\right)$ time with the doubling trick. Since the greedy algorithm has k iterations, and each iteration checks $O\left(n\right)$ seeds, the claim follows. ◀

5 Experiments

In Section 5.1, we explain how the input mobility graphs have been obtained, and then in Section 5.2 we show the findings of our analysis. The code and the input graphs are available at `https://github.com/AlgoUniPD/BikeSpreadingProblem`.

5.1 Building the mobility graphs

The input graphs used in the experiments have been built from a dataset containing all rides of the free-floating bike system in Padova (Italy) of the operator Movi by Mobike. The dataset contains 327K rides from May 1st, 2019 to January 30th, 2020. Each ride is described by the anonymized user and bike ids, and by the positions and time stamps of the pick-up and drop-off points. The graphs were constructed by following these three steps.

1. Vertexes are created by snapping each pick-up and drop-off point on a 2-dimensional grid. The grid consists of cells of size $s \times s$ with $s \in \{100\ m, 500\ m\}$. The two grids create two vertex sets of size 9975 and 399 respectively, covering a total area of 99.75 km^2 (see Figure 1b). Grid size should be understood as the maximum distance a user is willing to walk to find a bike.
2. The edge probabilities are constructed by two sets of rides: from 6:30 to 9:00 (morning rides) and from 16:00 to 20:00 (evening rides). In both sets, rides refer to all weekdays of October 2019. For each $u, v \in V$, the probability of an edge (u, v) is set to $n_{u,v}/n_u$, where n_u is the total number of rides originated in the cell represented by node u and $n_{u,v}$ is the total number of rides from node u to node v.
3. The graph is pruned by removing all edges with probabilities lower than a given threshold $\eta \in \{0, 0.01, 0.1\}$. For each node u, the probability mass of the removed edges originating from u is added to the self-loop (u, u) to guarantee that the weights of outgoing edges sum to one. Nodes with no edges or with only a self-loop are removed from the graph.

We thus ended up with 12 graphs $G_{s,\eta,M}$ and $G_{s,\eta,E}$ (M and E mean morning and evening, respectively), whose property is provided in Table 1. The intuition behind the three parameters (grid size s, morning or evening rides, pruning factor η) used for creating the graphs is the following. The different grid size s and pruning factor η are used for generating graphs with a different number of nodes and edges, for testing scalability. The graphs built using morning or evening rides give insight into the mobility flow during the day: the two slots are the morning and evening rush hours where workers commute to or from workplaces (see the daily rides distribution in Figure 1a). From a practical point of view, the weights computed from morning rides should be used if bikes are positioned in the seed nodes during the night to exploit the morning bike flow (equivalently, the evening rides should be used for bikes positioned in the afternoon).
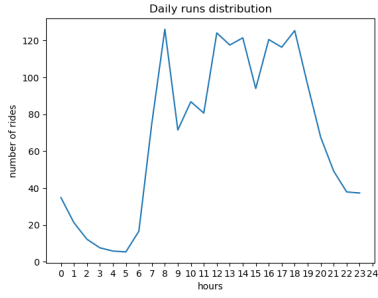
(a)
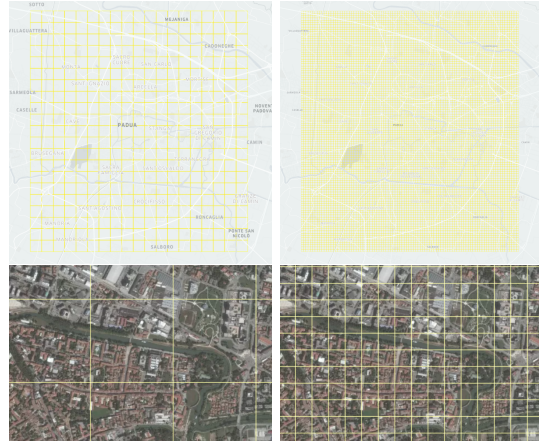
(b)

■ **Figure 1** (a) Daily runs distribution during October 2019. There are three picks: around 8.00 (commuting home to work), 13.00 (lunch time, end of school), around 18 (commuting work to home). (b) Padova subdivided with a grid of 500 m and 100 m, and a detail of the subdivisions. (The underlying street and satellite maps were provided by Kepler.gl.)

■ **Table 1** Properties of the graphs used for the analysis. A graph $G_{s,r,\eta}$ with $s \in \{100\ m, 500\ m\}, r \in \{M, E\}, \eta = \{0, 0.01, 0.1\}$ was obtained with a grid of size s, with morning (M) or evening (E) rides, and pruning factor η.

Graph	η	n	m	avg. degree
$G_{100,0,M}$	0.0	359	1302	3.627
$G_{100,0.01,M}$	0.01	287	954	3.324
$G_{100,0.1,M}$	0.1	139	323	2.324
$G_{500,0,M}$	0.0	111	1196	10.775
$G_{500,0.01,M}$	0.01	107	1068	9.981
$G_{500,0.1,M}$	0.1	75	272	3.627

Graph	η	n	m	avg. degree
$G_{100,0,E}$	0.0	1187	5854	4.932
$G_{100,0.01,E}$	0.01	1099	4986	4.537
$G_{100,0.1,E}$	0.1	222	463	2.086
$G_{500,0,E}$	0.0	142	2625	18.486
$G_{500,0.01,E}$	0.01	125	1810	14.480
$G_{500,0.1,E}$	0.1	92	229	2.489

5.2 Performance and quality

The experimental analysis focuses on the following questions:
(Q1) How close is the solution of the approximate algorithm to the optimal solution?
(Q2) How does the greedy algorithm scale with input size, seed size, and step number?
(Q3) How do the U-BS and T-BS models compare?
(Q4) When bikes should be rebalanced?

All experiments have been executed on an Intel Xeon Processor W-2245 3.9GHz with 128GB RAM and Ubuntu 9.3.0; code was in Python 3. Running times were averaged on 3 executions.

Question Q1: exact vs approximate solutions

Table 2 shows the spread and running time for the brute force exact algorithm and the greedy approximation algorithm with the U-BS version: since the brute force has $O\left(n^k\right)$ time, which is exponential in seed size, we notice a quick increase of the running time even for small seed size. The greedy algorithm is more performing than the brute force, although

◾ **Table 2** Comparison between the brute force and greedy algorithms for the `U-BS` version, with seed size $k \in \{2, 4\}$, $\tau = 1$, $L = 100$. The symbol $*$ means that the instance has not be run due to excessive running time.

Graph	n/m	Brute force $k = 2$		Greedy $k = 2$		Brute force $k = 4$		Greedy $k = 4$	
		σ	time [s]	σ	time [s]	σ	time [s]	σ	time [s]
$G_{500,0.1,M}$	75/272	32.6	0.02	32.6	0.001	43.6	7.20	42.8	0.001
$G_{500,0.01,M}$	107/1068	55.3	0.07	55.3	0.002	63.9	71.36	63.9	0.005
$G_{500,0,M}$	111/1196	57.3	0.07	57.3	0.002	64.1	86.16	63.8	0.005
$G_{100,0,M}$	359/1302	121.0	1.79	121.0	0.021	173.9	18514	123.0	0.040
$G_{100,0,E}$	1187/5854	185.7	99.00	185.7	0.347	$*$	$*$	193.3	0.555

◾ **Table 3** Comparison between the brute force and greedy algorithms for the `T-BS` version, with seed size $k \in \{2, 4\}$, $\tau = 1$, $L = 100$, $\gamma = 1$. The symbol $*$ means that the instance has not be run due to excessive running time.

Graph	n/m	Brute force $k = 2$		Greedy $k = 2$		Brute force $k = 4$		Greedy $k = 4$	
		σ	time [s]	σ	time [s]	σ	time [s]	σ	time [s]
$G_{500,0.1,M}$	75/272	11	0.04	11	0.002	20	15.49	20	0.003
$G_{500,0.01,M}$	107/1068	31	0.22	30	0.008	35	196.93	35	0.015
$G_{500,0,M}$	111/1196	31	0.24	30	0.008	36	234.79	35	0.016
$G_{100,0,M}$	359/1302	51	8.14	51	0.089	$*$	$*$	57	0.176
$G_{100,0,E}$	1187/5854	81	302.17	81	1.03	$*$	$*$	84	2.06

◾ **Table 4** Seed comparison between the brute force algorithm and the approximate algorithm for the `U-BS` and `T-BS` models for $k = 4$ (other parameters: $\tau = 1$, $L = 100$, $\gamma = 1$). The star $*$ after a seed set means that the solution provided by the approximate algorithm has the same spread of the optimal one.

Graph	Seed overlap under `U-BS`	Seed overlap under `T-BS`
$G_{500,0.1,M}$	25%	50%*
$G_{500,0.01,M}$	100%*	75%*
$G_{500,0,M}$	25%	25%

it is visible the n^3 dependence in the running time. The quality of the approximation is quite high, outperforming the theoretical worst-case upper bound of $1 - 1/e$. Similar results hold for `T-BS` as provided in Table 3: the running times show a small increase with respect to the previous table for `U-BS`, mainly due to the conditional statements for checking if a load is smaller than the threshold γ. In all cases, the spreads are very close, although the seed provided by the brute force and the greedy algorithms do not completely overlap. (see Table 4).

Question Q2: scalability

We now analyze the running time of the greedy approach for increasing values of input size, number of steps, and seed size. Since the greedy algorithms for `U-BS` and `T-BS` are almost equivalent, we only provide results for the first one. Table 5 shows the running time on the largest graph ($G_{100,0,E}$), three different seed sizes $k \in \{2, 4, 8\}$ and three different step numbers $\tau \in \{1, 10, 100\}$. For a given k, the times are almost equivalent: τ only affects the initial computation of A^τ, where A is the adjacency matrix of the graph; since the powering requires time $O\left(n^3 \log \tau\right)$, the logarithmic dependence on τ is negligible and hidden by the

Table 5 Running time using $G_{100,0,E}$ with different values of the seed set size k and of the step number τ.

Graph	**k**	**$\tau = 1$** **[ms]**	**$\tau = 10$** **[ms]**	**$\tau = 100$** **[ms]**
$G_{100,0,E}$	2	334.2	352.4	384.4
$G_{100,0,E}$	4	629.8	651.0	657.4
$G_{100,0,E}$	8	1271.8	1277.2	1278.8
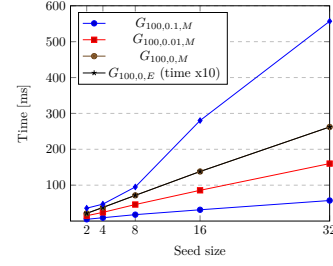
Figure 2 Running time with respect to seed size k. The curve of $G_{100,0,M}$ has been scaled by a factor x10 for better fitting.

cost of the k iterations of the greedy algorithm. The previous table already shows that the running time has a linear dependency on the seed size. Figure 2 expands this analysis by considering different graph sizes. Note that the curve of $G_{100,0,E}$ has been scaled by a factor of 10 to fit the plot space. All curves show a linear dependency in k.

Question Q3: `T-BS` vs `U-BS`

In this experiment, we compare the two models. Intuitively, `T-BS` maximizes the number of nodes with a minimum number γ of bikes: for instance, by setting $\gamma = 1$, the algorithm maximizes the number of cells with at least one bike (in expectation). On the other hand, `U-BS` aims at uniformly distributing bikes among nodes, even if this implies that some nodes have a low expected number of bikes (even < 1). This allows for more fair use of bikes since it increases the load in suburb areas, differently than `T-BS` that facilitates central (and more crowded) areas. In the heatmaps in Figure 3, we compare the two models on the $G_{500,0,M}$ (heatmaps (a) and (b)) and $G_{100,0,M}$ (heatmaps (d) and (e)) graphs. Each heatmap shows how bike distributes, after $\tau = 2$ steps, by positioning 400 bikes in the $k = 4$ seed set selected by the greedy algorithm. We notice that `U-BS` covers a larger fraction of nodes in the map than `T-BS`. The phenomenon is more evident in the 100m grid, where `U-BS` colors a larger number of cells in the east and south suburb areas than `T-BS`. although, the majority of the cells are reached by a small number of bikes, mostly less than one bike in expectation.

Question Q4: When rebalancing?

Consider the graphs $G_{100,0,M}$ and $G_{100,0,E}$: both graphs use the 100m grid and no edge pruning. Graph $G_{100,0,E}$ has a larger number of edges and nodes than $G_{100,0,M}$ (see Table 1), highlighting a change in the use of the FFBSS service from morning and evening: in the morning, rides are mostly directed towards workplaces, the train station, and university departments which are mainly located in the city center and on the east side; in the afternoon, there are much more activities (e.g., having a *spritz* with friends, going to the gym, shopping) and hence the graph covers a wider area of Padova. Bikes can be more spread in the city if we use the more vibrant afternoon for rebalancing bikes. This is confirmed by the simulation of `U-BS` using $G_{100,0,M}$ and $G_{100,0,E}$ in the heatmaps (e) and (f) of Figure 3: by using the evening graph, the bikes significantly spread covering also the north and west parts of the city. Similar results hold for $G_{500,0,M}$ and $G_{500,0,E}$ in heatmaps (b) and (c).

(a) `T-BS` on 500 m, morning.

(b) `U-BS` on 500 m, morning.

(c) `U-BS` on 500 m, evening.

(d) `T-BS` on 100 m, morning.

(e) `U-BS` on 100 m, morning.

(f) `U-BS` on 100 m, evening.
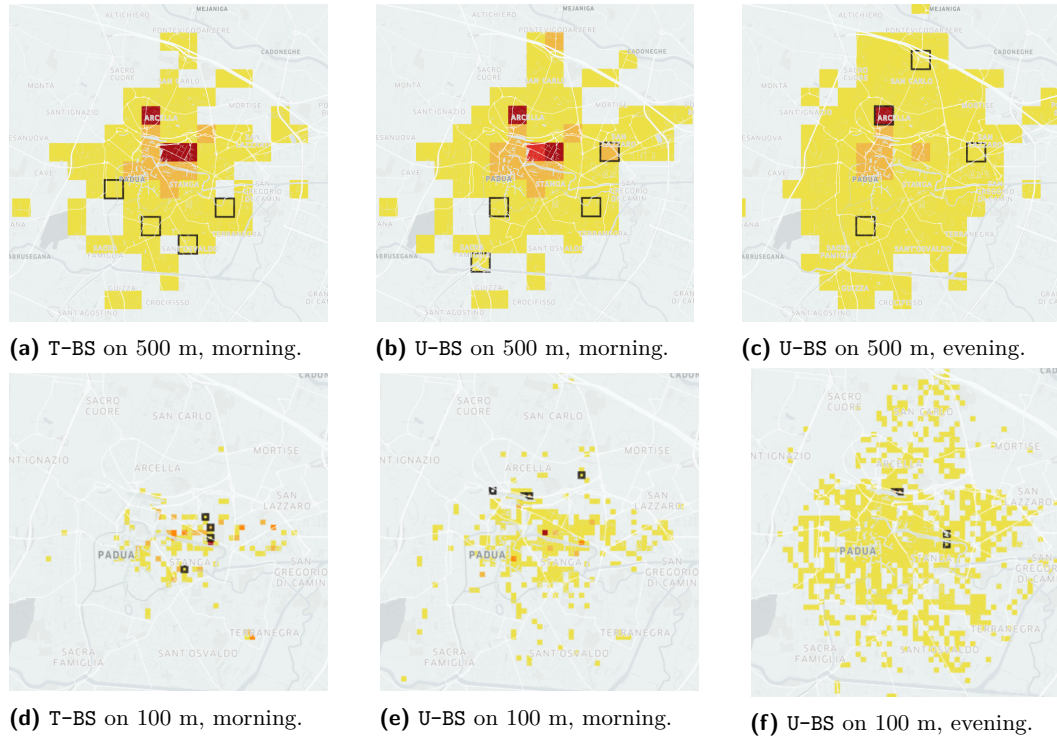
Figure 3 Bike diffusion after $\tau = 2$ steps by positioning $L = 100$ bikes in each node of the seed set of size $k = 4$ selected by the greedy algorithm. For `T-BS`, we set $\gamma = 1$. The cells in the seed set are marked with black contours. Color scale for 500m plots: $(0.0, 2.4]$ (yellow),$(2.4, 4.8], (4.8, 7.2], (7.2, 9.6], (9.6, 12]$ (dark red); Color scale for 100m plots: $(0.0, 1.4]$ (yellow), $(1.4, 2.8], (2.8, 4.2], (4.2, 5.6], (5.6, 7]$ (dark red). (The underlying street maps were provided by Kepler.gl.)

6   Conclusion

In this work, we have proposed a graph approach to spread bikes in a free-floating bike system to cover a large number of zones of the service area; the idea is to select a set of zones where to position bikes and let the mobility flow spread them around the city. The current model assumes that, initially, only seed nodes have bikes while the other nodes are empty. This assumption can be removed by allowing any node to initially have some bikes: it can be shown that submodularity and monotonicity still hold for `U-BS`, and thus the greedy algorithm provides a $1 - 1/e$ approximate solution also in this case. Another extension to investigate is to allow a different distribution of bikes among seed nodes, to balance skewness in bike distribution due to hotspots. Finally, an important research direction is to analyze the behavior of the model on a real free-floating bike system and the differences between the theoretical findings and the actual distribution.

References

1 Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theor. Comput. Sci.*, 237(1-2):123–134, 2000.

2 Valentin Bouquet, François Delbot, Christophe Picouleau, and Stéphane Rovedakis. On minimum dominating sets in cubic and (claw, H)-free graphs. *CoRR*, abs/2002.12232, 2020. `arXiv:2002.12232`.

3   Leonardo Caggiani, Rosalia Camporeale, Michele Ottomanelli, and Wai Yuen Szeto. A modeling framework for the dynamic management of free-floating bike-sharing systems. *Transportation Research Part C: Emerging Technologies*, 87:159–182, 2018.

4   Wei Chen, Alex Collins, Rachel Cummings, Te Ke, Zhenming Liu, David Rincón, Xiaorui Sun, Yajun Wang, Wei Wei, and Yifei Yuan. Influence maximization in social networks when negative opinions may emerge and propagate. In *Proc. of the 11th SIAM Int. Conf. on Data Mining (SDM)*, pages 379–390, 2011.

5   Iris A. Forma, Tal Raviv, and Michal Tzur. A 3-step math heuristic for the static repositioning problem in bike-sharing systems. *Transportation Research Part B: Methodological*, 71:230–247, 2015.

6   Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

7   Ashish Kabra, Elena Belavina, and Karan Girotra. Bike-share systems: Accessibility and availability. *Management Science*, 66(9), 2019.

8   David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 137–146, 2003.

9   William Ogilvy Kermack, A. G. McKendrick, and Gilbert Thomas Walker. A contribution to the mathematical theory of epidemics. *Proc. Royal Society of London, series A*, 115(772):700–721, 1927.

10   Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. Influence maximization on social graphs: A survey. *IEEE Trans. Knowl. Data Eng.*, 30(10):1852–1872, 2018.

11   Zhi Li, Jianhui Zhang, Jiayu Gan, Pengqian Lu, and Fei Lin. Large-scale trip planning for bike-sharing systems. In *Proc. 14th IEEE Int. Conf. on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 328–332, 2017.

12   Lei Lin, Zhengbing He, and Srinivas Peeta. Predicting station-level hourly demand in a large-scale bike-sharing network: A graph convolutional neural network approach. *Transportation Research Part C: Emerging Technologies*, 97:258–276, 2018.

13   Russell Meddin, Paul DeMaio, Oliver O'Brien, Renata Rabello, Chumin Yu, Rahil Gupta, and Jess Seamon. The Meddin bike-sharing world map. Accessed August 17th, 2021. URL: http://bikesharingworldmap.com/.

14   Stephen J. Mooney, Kate Hosford, Bill Howe, An Yan, Meghan Winters, Alon Bassok, and Jana A. Hirsch. Freedom from the station: Spatial equity in access to dockless bike share. *Journal of Transport Geography*, 74:91–96, 2019.

15   George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions – I. *Math. Program.*, 14(1):265–294, 1978.

16   Aritra Pal and Yu Zhang. Free-floating bike sharing: Solving real-life large-scale static rebalancing problems. *Transportation Research Part C: Emerging Technologies*, 80:92–116, 2017.

17   Svenja Reiss and Klaus Bogenberger. Validation of a relocation strategy for Munich's bike sharing system. *Transportation Research Procedia*, 19:341–349, 2016.

18   Muhammad Usama, Yongjun Shen, and Onaira Zahoor. A free-floating bike repositioning problem with faulty bikes. *Procedia Computer Science*, 151:155–162, 2019.

19   Yue Wang and W.Y. Szeto. Static green repositioning in bike sharing systems with broken bikes. *Transportation Research Part D: Transport and Environment*, 65:438–457, 2018.

20   Yongping Zhang and Zhifu Mi. Environmental benefits of bike sharing: A big data-based analysis. *Applied Energy*, 220:296–301, 2018.

A Phase I Simplex Method for Finding Feasible Periodic Timetables

Marc Goerigk[1] ✉ 📙
Network and Data Science Management, Universität Siegen, Germany

Anita Schöbel ✉ 📙
Department of Mathematics, TU Kaiserslautern, Germany
Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM, Kaiserslautern, Germany

Felix Spühler ✉ 📙
Business Information Systems, TU Braunschweig, Germany

─── **Abstract** ───

The periodic event scheduling problem (PESP) with various applications in timetabling or traffic light scheduling is known to be challenging to solve. In general, it is already NP-hard to find a feasible solution. However, depending on the structure of the underlying network and the values of lower and upper bounds on activities, this might also be an easy task.

In this paper we make use of this property and suggest phase I approaches (similar to the well-known phase I of the simplex algorithm) to find a feasible solution to PESP. Given an instance of PESP, we define an auxiliary instance for which a feasible solution can easily be constructed, and whose solution determines a feasible solution of the original instance or proves that the original instance is not feasible. We investigate different possibilities on how such an auxiliary instance can be defined theoretically and experimentally. Furthermore, in our experiments we compare different solution approaches for PESP and their behavior in the phase I approach. The results show that this approach can be especially helpful if the instance admits a feasible solution, while it is generally outperformed by classic mixed-integer programming formulations when the instance is infeasible.

1 Introduction

Railways play an important role in public transportation planning and form an essential component for green logistics and traveling in the future. Timetabling is a key element for planning public passenger transportation. In particular, periodic timetables, e.g., hourly repeated, are of interest from the passengers' perspective because they are easy to remember.

Periodic timetables have been extensively studied in the literature since their introduction by Serafini and Ukovich in 1989 as the periodic event scheduling problem (PESP) [20]. Works on PESP are not only of theoretical interest, as they are already used to optimize timetables in practice. In 2008, Liebchen successfully implemented an optimized timetable for the Berlin Underground [13]. While providing shorter passenger waiting times, it was also possible to reduce the number of trains. In 2006, Kroon et al. optimized the Dutch Railway System [10]. Their timetable was adapted to current and future needs, improving the service significantly and at the same resulting in approximately 40 million Euro additional annual profit.

---

[1] Corresponding author

In real-world applications, safety conditions must be considered which further complicate the problem. For example, trains have to maintain a safety distance when sharing the same rail. This is usually modeled by enforcing a time gap between the departures of two consecutive trains. The added complexity of safety constraints makes it more difficult to find feasible timetables for big instances.

Determining optimal timetables is a complex task. Even finding a feasible timetable is known to be NP-hard [20]. Recently, also the parameterized problem complexity has been studied [14]. The authors show that deciding the feasibility of PESP is W[1]-hard when parameterized by the vertex color number. However, it is easy to see that finding an optimal solution is possible in polynomial time on trees.

Solving large-scale problems to optimality remains out of reach for current families of algorithms, such as the modulo simplex method [7, 15], a matching-based heuristic [17], or methods based on SAT solving [8]. Recent papers [1, 2, 4, 9] provide further progress towards this long-term goal.

In this paper, we provide a new approach for finding feasible timetables. This approach is inspired by the phase I of the classic simplex method for linear programming. A timetabling instance is extended by adding virtual edges to the underlying network, which makes it simple to find a feasible solution in the thus extended network. By minimizing its objective function, a feasible solution to the original problem instance can be found, or a certificate of infeasibility is given.

The remainder of this paper is structured as follows. In Section 2, we briefly recall the formal problem definition and basic properties. We then introduce the phase I approach for finding feasible timetables in Section 3. Using instances from the *LinTim*-library, we evaluate advantages and disadvantages of this method in Section 4, before concluding the paper in Section 5.

2   The Periodic Event Scheduling Problem

In this section, the main definitions of the periodic event scheduling problem (PESP) and two mixed-integer formulations are briefly revisited. For more details, we refer to [11, 12, 16].

An event-activity network (EAN) $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ is a directed graph where nodes represent events (such as the departure or arrival of trains of a directed line) and arcs represent activities (such as drive, wait, transfer and security headway activities). Without loss of generality, we assume the event-activity network $\mathcal{N}$ to be connected. If not, all considerations could be applied to each connected component separately. We also assume that $|\mathcal{E}| \leq |\mathcal{A}|$, i.e., there are at least as many activities as events. This assumption holds for all connected event-activity networks unless the event-activity network is a tree. In this case, all considerations of the PESP are trivial.

For each activity $a \in \mathcal{A}$ the minimal and maximal allowed duration are denoted by L_a and U_a with $0 \leq L_a \leq U_a$ and $L_a, U_a \in \mathbb{N}$. Together, $\Delta_a = [L_a, U_a]$ is called the time span of a. A periodic timetable $\pi \in \mathbb{Z}^{|\mathcal{E}|}$, where $\pi_i \in [0, T-1]$ for all $i \in \mathcal{E}$ with a time period T, is called feasible if for all $a = (i, j) \in \mathcal{A}$ there exist so-called modulo parameters $z \in \mathbb{Z}$ such that $\pi_j - \pi_i + zT \in \Delta_a$. The PESP in the context of timetabling then consists of finding a feasible periodic timetable that minimizes the weighted travel time $\sum_{a=(i,j)\in\mathcal{A}} w_a(\pi_j - \pi_i + z_a T)$ for given (passenger) weights w_a for each $a \in \mathcal{A}$. An instance of the problem is thus defined by the tuple $I = ((\mathcal{E}, \mathcal{A}), w, L, U)$. A well-known mixed-integer programming (MIP) formulation for PESP is the following.

$$\text{(NodeIP)} \quad \min \quad \sum_{a=(i,j)\in\mathcal{A}} w_a(\pi_j - \pi_i + z_a T) \tag{1}$$

$$\text{s.t.} \quad L_a \leq \pi_j - \pi_i + z_a T \leq U_a \qquad \forall a = (i,j) \in \mathcal{A} \tag{2}$$

$$\pi_i \in \mathbb{Z} \qquad \forall i \in \mathcal{E} \tag{3}$$

$$z_a \in \mathbb{Z} \qquad \forall a \in \mathcal{A} \tag{4}$$

We briefly revisit another model that depends on cycles. Any cycle $C \subseteq \mathcal{A}$ is described by the incidence vector $\Gamma(C) \subseteq \{-1,0,1\}^{|\mathcal{A}|}$ with components $\Gamma(C)_a = 1$, if $a \in C$ in forward direction, $\Gamma(C)_a = -1$, if $a \in C$ in backward direction, and $\Gamma(C)_a = 0$ else. Using these incidence vectors as row vectors, we describe a set of cycles $\mathcal{C}$ by a matrix $\Gamma \in \{-1,0,1\}^{|\mathcal{C}|,|\mathcal{A}|}$, that is, it contains $\Gamma(C_a)^t$ in row a for each cycle $C_a \in \mathcal{C}$. Given a spanning tree $\mathcal{T}$, let $\Gamma \in \{-1,0,1\}^{|\mathcal{A}|-|\mathcal{E}|+1,|\mathcal{A}|}$ be the matrix corresponding to its fundamental cycles. Then, Γ is called cycle matrix of $\mathcal{N}$ with respect to $\mathcal{T}$. A vector $\xi \in \mathbb{Z}^{|\mathcal{A}|}$ is a periodic tension w.r.t. π if there exists $z \in \mathbb{Z}^{|\mathcal{A}|}$ such that $\pi_j - \pi_i + z_a T = \xi_a$ for all $a = (i,j) \in \mathcal{A}$. Note that $\xi \in \mathbb{Z}^{|\mathcal{A}|}$ is a tension in $\mathcal{N}$ if and only if $\Gamma\xi = 0$. This results in the following cycle-based mixed-integer program for the PESP:

$$\text{(CBIP)} \quad \min \quad w^t \xi \tag{5}$$

$$\text{s.t.} \quad \Gamma\xi = T\tilde{z} \tag{6}$$

$$L \leq \xi \leq U \tag{7}$$

$$\xi_a \in \mathbb{Z} \quad \forall a \in \mathcal{A} \tag{8}$$

$$\tilde{z}_a \in \mathbb{Z} \quad \forall a \in \mathcal{A} \setminus \mathcal{T} \tag{9}$$

3 A Phase I Approach to PESP

The general idea of phase I approaches can best be recalled looking at the classical simplex algorithm in which a linear program $\min\{c^T x : Ax = b, x \geq 0\}$ with costs $c \in \mathbb{R}^n$, a right-hand side $b \in \mathbb{R}^m$ (w.l.o.g. $b \geq 0$) and a matrix $A \in \mathbb{R}^{m,n}$ is given and we look for an optimal solution $x \in \mathbb{R}^n$. The simplex algorithm needs a feasible solution to get started. If this is not available, the well-known phase I of the simplex algorithm starts: by adding additional columns to the coefficient matrix A, it is extended to $(A|I)$, where I denotes the identity matrix. For the new linear program, the columns of I are chosen as basis, such that $x := (\underbrace{0,\ldots,0}_{\in\mathbb{R}^n}, \underbrace{b^t}_{\in\mathbb{R}^m})^t$ is a feasible starting solution. The auxiliary problem asks to minimize the unit costs of the new variables as auxiliary objective function and is solved by the simplex algorithm (which is now possible since a starting solution is known). If and only if the auxiliary problem has objective value of zero, the original instance is feasible. In this case, a feasible solution to the original instance can be constructed by pivoting the auxiliary variables out of the basis. The generalized scheme of this process is depicted in Figure 1.

For the PESP we now proceed analogously: Given an instance of PESP $I = ((\mathcal{E}, \mathcal{A}), w, L, U)$ we construct an auxiliary instance by extending the given instance to

$$I^{\text{ext}} = ((\mathcal{E}^{\text{ext}}, \mathcal{A}^{\text{ext}}), w^{\text{ext}}, L^{\text{ext}}, U^{\text{ext}}).$$

We show that for I^{ext} a feasible solution can be easily constructed and that the original instance I has a feasible solution if and only if the optimal objective value of I^{ext} is zero.
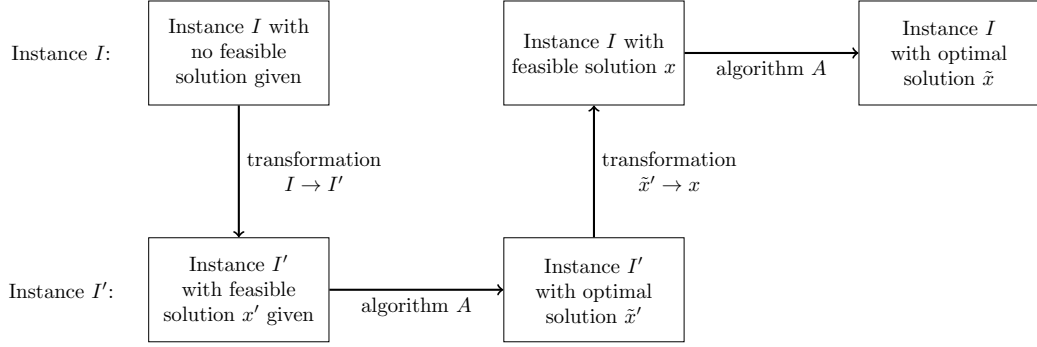
Figure 1 Schematic process of the phase I Approach with the original instance I, the extended instance I' and an algorithm A.

Recall that cycles make the PESP a hard problem. We hence want to make sure that each cycle contains a *flexible* activity, i.e., an activity with bounds $\Delta_a = [0, T-1]$. Such an activity can collect all slack needed to ensure that the timetable satisfies the constraint of the respective cycle in constraint (6).

To this end, let the original instance $I = ((\mathcal{E}, \mathcal{A}), w, L, U)$ be given. We fix a set $A \subseteq \mathcal{A}$ (we will discuss later how this set can be chosen), and define the extended instance as follows. Each activity $a = (i, j) \in A$ of the original instance is replaced by two activities, namely by $a^{\text{old}} = (i, i_a)$ and $a^{\text{virt}} = (i_a, j)$ which are linked by one new event i_a, see Figure 2. The first activity carries the old lower and upper bounds, i.e., L_a and U_a from the original instance are now the bounds of a^{old}. The second activity is a flexible activity which receives $\Delta_{a^{\text{virt}}} = [0, T-1]$ as lower and upper bounds.

Figure 2 Extending an activity $a = (i, j) \in A$ with span $\Delta_a = [L_a, U_a]$ to two activities $a^{old} = (i^{org}, i_a^{virt}) \in \mathcal{A}^{old}$ with span $\Delta_a^{old} = [L_a^{old}, U_a^{old}] := [L_a, U_a]$ and $a^{virt} = (i_a^{virt}, j^{org})$ with span $\Delta_a^{virt} := [0, T-1]$.

Formally, we define $\mathcal{E}^{\text{ext}} = \mathcal{E} \cup \{i_a : a \in A\}$ and $\mathcal{A}^{\text{ext}} = (\mathcal{A} \setminus A) \cup \mathcal{A}^{\text{old}} \cup \mathcal{A}^{\text{virt}}$, where $\mathcal{A}^{\text{old}} = \{(i, i_a) : a = (i, j) \in A\}$ and $\mathcal{A}^{\text{virt}} = \{(i_a, j) : a = (i, j) \in A\}$. As parameters we set:

$$L_a^{\text{ext}} := \begin{cases} L_a & \text{if } a \in \mathcal{A} \setminus A \\ L_{a'} & \text{if } a = (i, i_{a'}) \in \mathcal{A}^{\text{old}}, a' \in A \\ 0 & \text{if } a \in \mathcal{A}^{\text{virt}} \end{cases} \tag{10}$$

$$U_a^{\text{ext}} := \begin{cases} U_a & \text{if } a \in \mathcal{A} \setminus A \\ U_{a'} & \text{if } a = (i, i_{a'}) \in \mathcal{A}^{\text{old}}, a' \in A \\ T - 1 & \text{if } a \in \mathcal{A}^{\text{virt}} \end{cases} \tag{11}$$

$$w_a^{\text{ext}} := \begin{cases} 0 & \text{if } a \in \mathcal{A} \setminus A \\ 0 & \text{if } a \in \mathcal{A}^{\text{old}} \\ 1 & \text{if } a \in \mathcal{A}^{\text{virt}} \end{cases} \tag{12}$$

Given an original instance I and a set A we denote the new instance as $I^{\text{ext}}(A)$ where we leave out A if the context is clear. Clearly, this is again a PESP instance which can hence be formulated by the integer programs provided in Section 2. Note that the objective function

of I^{ext} only includes the virtual activities $\mathcal{A}^{\text{virt}}$. This is in line with the phase I approach for the classic simplex algorithm, in which also only the newly added columns are part of the auxiliary objective function.

Before we discuss how to choose $A \subseteq \mathcal{A}$ and how to find feasible solutions for instances $I^{\text{ext}}(A)$, we provide a basic theorem which states the relation between feasibility of the original instance I and the optimal objective of the extended instance $I^{\text{ext}}(A)$. Note that this statement is independent of the choice of A.

▶ **Theorem 1.** *Let I be an instance of PESP and $A \subseteq \mathcal{A}$ be a set of activities. Then, I is feasible if and only if the objective value $v(I^{\text{ext}}(A))$ of the extended instance is zero.*

Proof. First, let π be a feasible solution for I. Set

$$\pi_i^{\text{ext}} := \begin{cases} \pi_i & \text{if } i \in \mathcal{E} \\ \pi_{j'} & \text{if } i = i_{a'} \text{ with } a' = (i', j') \in A \end{cases},$$

i.e., on the original events $i \in \mathcal{E}$ we leave the timetable as it is while a virtual event $i_{a'}$ on the edge $a' = (i', j')$ obtains the timetable of the end-node j' of a'. To see that π^{ext} is feasible for I^{ext} we have to look at the three types of activities:

- For $a = (i, j) \in \mathcal{A} \setminus A$, both i and j are in $\mathcal{E}$ and feasibility of π^{ext} for I^{ext} follows from feasibility of π for I.
- For $a = (i, i_{a'}) \in \mathcal{A}^{old}$ with $a' = (i, j) \in A$, $\pi_{i_{a'}}^{\text{ext}} - \pi_i^{\text{ext}} + zT_a \in \Delta_a$ holds since due to $\pi_{i_{a'}}^{\text{ext}} = \pi_j$ this is the constraint for the original activity $a' \in A$ which is satisfied, because the timetable π is feasible for I.
- Finally, for $a \in \mathcal{A}^{\text{virt}}$ feasibility is always satisfied since a is a flexible activity with $\Delta_a = [0, T - 1]$.

Note that the objective value of this solution is zero.

For the reverse direction, we start with a feasible timetable π for I^{ext} with objective

$$v(I^{\text{ext}}) = \sum_{a^{\text{virt}} = (i_a, j) \in \mathcal{A}^{\text{virt}}} (\pi_j^{\text{ext}} - \pi_{i_a}^{\text{ext}} + z_{a^{\text{virt}}} T) = 0.$$

Due to the constraints we know that $\pi_j^{\text{ext}} - \pi_{i_a}^{\text{ext}} + z_{a^{\text{virt}}} T \geq L_{a^{\text{virt}}} = 0$ for all $a^{\text{virt}} = (i_a, j) \in \mathcal{A}^{\text{virt}}$, hence we conclude that

$$\pi_j^{\text{ext}} - \pi_{i_a}^{\text{ext}} + z_{a^{\text{virt}}} T = 0 \tag{13}$$

for some $z_{a^{\text{virt}}} \in \mathbb{Z}$.

Given the timetable π^{ext} for I^{ext} we define the timetable π for I by projection, i.e., we just leave the values π_i, $i \in \mathcal{E}$ as they have been in I^{ext}. We now show that this timetable is feasible for the original instance I, i.e., that there exist modulo parameters z_a such that

$$\pi_j - \pi_a + z_a T \in \Delta_a$$

for all $a = (i, j) \in \mathcal{A}$. To this end, we consider the activities in $\mathcal{A} \setminus A$ and in A separately.

- For $a \in \mathcal{A} \setminus A$ feasibility of π for I follows from feasibility of π^{ext} for I^{ext}.
- Now let $a' = (i, j) \in A$. We know that for $a^{\text{old}} = (i, i_{a'}) \in \mathcal{A}^{\text{old}}$ we have

$$L_a \leq \pi_{i_{a'}}^{\text{ext}} - \pi_i^{\text{ext}} + z_{a^{\text{old}}} T \leq U_a \tag{14}$$

where we used that $L_a = L_{a^{\text{old}}}$ and $U_a = U_{a^{\text{old}}}$ according to (10) and (11). Adding (13) and (14) we receive

$$L_a \leq \pi_j - \pi_i + (z_{a^{\text{old}}} + z_{a^{\text{virt}}}) T \leq U_a.$$

Hence, $z_a := z_{a^{\text{old}}} + z_{a^{\text{virt}}} \in \mathbb{Z}$ is the required modulo parameter for $a' \in A$ and the claim is shown. ◀

We can directly conclude that looking at a lower bound while solving I^{ext} may suffice to decide non-feasibility of I.

▶ **Corollary 2.** *Let γ be a lower bound on the objective value of I^{ext}, i.e., $\gamma \leq v(I^{\text{ext}})$. If $\gamma > 0$, I is not feasible.*

We now analyze possibilities how the set $A \subseteq \mathcal{A}$ can be chosen. Keep in mind that we want to find a feasible solution of I^{ext} efficiently. We use a result for a special case, namely, if the event-activity network $\mathcal{N}$ is a tree, the solution of PESP is easy: In the cycle-based formulation, the cycle matrix vanishes and $\xi = L$ is an optimal solution. Based on this result, the general idea to construct an instance with easy-to-find feasible solution is to add the flexible activities to the original activities $A \in \mathcal{A}$ in such a way that $\mathcal{A}^{\text{ext}}$ without the flexible activities is a forest for which a timetable can be found easily for each of its connected components. The following result is easy to verify and therefore given without proof.

▶ **Lemma 3.** *Let $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ be an event-activity network with a spanning tree $\mathcal{T}$. Let $A_1 = \mathcal{A} \setminus \mathcal{T}$, $A_2 \subseteq \mathcal{T}$ and $A = A_1 \cup A_2$ the set for which virtual activities should be added. Finally, let $\mathcal{N}^{\text{ext}} = (\mathcal{E}^{\text{ext}}, \mathcal{A}^{\text{ext}})$ the extended event-activity network where $\mathcal{A}_1^{\text{ext}} \subseteq \mathcal{A}^{\text{ext}}$ corresponds to the activities in A_1 and $\mathcal{A}_2^{\text{ext}} \subseteq \mathcal{A}^{\text{ext}}$ corresponds to the activities in A_2. Then, $(\mathcal{A} \setminus A) \cup \mathcal{A}_1^{\text{old}} \cup \mathcal{A}_2^{\text{old}} \cup \mathcal{A}_2^{\text{virt}}$ defines a spanning tree of the extended event-activity network.*

This means that we can solve PESP on the tree and obtain a feasible solution since all activities which are not in the tree are flexible activities. Specific choices for set A are as follows.

- *full*: Add a virtual activity for each activity in the original-event-activity network.
- *cycle_base*: Add a virtual activity for each fundamental circuit for a given spanning tree.
- *minimal*: Add a virtual activity for each fundamental circuit for a given spanning tree if this circuit does not already contain a flexible activity.

We also mention that the complexity of PESP does not increase when we turn from I to I^{ext}, since the number of cycles in a cycle basis stays the same.

▶ **Lemma 4.** *Let the original event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ with a spanning tree $\mathcal{T}$, the set $A \subseteq \mathcal{A}$ for which virtual activities should be added, and the corresponding extended event-activity network $\mathcal{N}^{\text{ext}} = (\mathcal{E}^{\text{ext}}, \mathcal{A}^{\text{ext}})$ with a spanning tree $\mathcal{T}^{\text{ext}}$ be given. Then, the number of fundamental circuits is the same.*

Proof. The number of fundamental circuits of the original event-activity network is $|\mathcal{A}| - |\mathcal{T}|$ and for the extended network $|\mathcal{A}^{\text{ext}}| - |\mathcal{T}^{\text{ext}}|$. Then, by definition of the sizes of the extended sets of events and activities, it follows that

$$|\mathcal{A}^{\text{ext}}| - |\mathcal{T}^{\text{ext}}| = |\mathcal{A}^{\text{ext}}| - (|\mathcal{E}^{\text{ext}}| - 1) = (|\mathcal{A}| + |A|) - ((|\mathcal{E}| + |A|) - 1)$$
$$= |\mathcal{A}| - |\mathcal{E}| + 1 = |\mathcal{A}| - (|\mathcal{E}| - 1) = |\mathcal{A}| - |\mathcal{T}|. \qquad \blacktriangleleft$$

4   Experiments

4.1   Computational setup

We use the scientific software toolbox *LinTim*[2] [19, 5]. All algorithms except for the phase I approach are already implemented in *LinTim*. The code of the phase I approach is written in Python 3. If required, the mathematical optimization solver *Gurobi*[3] in Version 9 is used and the Python package *networkx*[4] for calculating spanning trees.

---

[2] See https://www.lintim.net/
[3] See https://www.gurobi.com/
[4] See https://networkx.github.io/

All experiments were conducted on a server with 12 Intel(R) Xeon(R) CPU X5675 processors (each at 3.07GHz) and 128 GB RAM. All algorithms have a time limit of 30 minutes and are limited to using one kernel.

The instances that are used for the computational experiments are also part of the *LinTim* toolbox, namely the data sets *toy*, *grid* (bus), *lowersaxony* (rail), *athens* (metro), *bahn-01*, *bahn-02*, *bahn-03*, and *bahn-04* (German high-speed network). On the basis of a given public transportation network with its stops, edges, and line concept, the event-activity network is created. Since the standard event-activity networks are all feasible, headway constraints are added to the event-activity network to complicate the problem and also potentially create infeasible instances. For each event-activity network there are ten different versions with headway times from 1 to 10 minutes.

Table 1 shows the number of events, the number of activities, especially the number of headway activities, and the number of fundamental circuits of the event-activity network for each data set.

Table 1 Size of the event-activity networks of the used *LinTim* data sets. The number of activities includes the number of headway activities.

data set	events	activities	headway activities	fundamental circuits
toy	156	304	116	149
grid	448	901	264	454
lowersaxony	536	1077	388	542
athens	1388	3892	1576	2505
bahn-01	5036	16543	6766	11508
bahn-02	5468	19726	7774	14259
bahn-03	3592	10041	2734	6450
bahn-04	5356	19136	6192	13781

We test nine variations of the described phase I approach, where we use the three different extension methods *minimal*, *cycle_base*, and *full* from Section 3 and the following three algorithms to solve the extended PESP instance I^{ext}:

- *NodeIP*: Solving the node-based MIP (1)-(4)
- *CBIP*: Solving the cycle-based MIP (5)-(9)
- *MNS*: Using the modulo network simplex [15, 7]

NodeIP and *CBIP* are chosen because they are in principle able to solve the PESP optimally, and because they also update the lower bound of the objective function. We use a stopping criterion when reaching a lower bound of the objective value greater than 0. *MNS* is chosen as a heuristic approach that performs well for the PESP. For a detailed description of these methods please refer to [19] and [7]. In the following, the phase I and its combinations are denoted as phase I (<extending method>, <algorithm>). All methods are provided with the same starting solution.

To benchmark the phase I approach, we choose three other algorithms that are already implemented in *LinTim*, i.e., they do not use an extended network. The node-based integer formulation and the cycle-based formulation are chosen because they are straightforward approaches and may also be good for showing infeasibility. We slightly adapt them to make them comparable to the phase I by stopping them when they find the first feasible solution. In the following these variants are denoted as NodeIP-Feas and CBIP-Feas. The third algorithm is the Constraint Propagation, denoted as ConProp, because this algorithm is often used to find feasible solutions, see [6] and [19].

■ **Table 2** Size of the extended event-activity networks for the different extending methods.

data set		original	minimal	cycle_base	full
toy	*events*	156	296	305	460
	activities	304	434	453	608
grid	*events*	448	741	902	1349
	activities	901	1033	1355	1802
lowersaxony	*events*	536	1023	1078	1613
	activities	1077	1509	1619	2154
athens	*events*	1388	3320	3893	5280
	activities	3892	5244	6397	7784
bahn-01	*events*	5036	12405	16544	21579
	activities	16543	19773	28051	33086
bahn-02	*events*	5468	13751	19727	25194
	activities	19726	22031	33985	39452
bahn-03	*events*	3592	6211	10042	13633
	activities	10041	8827	16491	20082
bahn-04	*events*	5356	11555	19137	24492
	activities	19136	17751	32917	38272

In the following, we briefly explain how the run times are determined. For both solvers of the MIP formulations that use *Gurobi*, only the real optimization time is taken as computation time, i.e., without the time for reading the input data or calculating the spanning tree in case of the cycle-based MIP. For the other two algorithms the complete run time is considered, e.g., with reading input, because they could not be integrated in the existing algorithms. However, read-in routines take only a few seconds for the largest instances. Regarding the run times of the phase I, only the run time of the algorithms are taken into account, i.e., without the time that is needed to build the extended event-activity network.

4.2   Results

In the following, we analyze and compare all instances of all data sets together. We have 80 different instances (10 instances for each of the 8 data sets) in total and 120 experiments for each data set (employing each of the 12 algorithms for each of the 10 instances). For all but one instance, namely *bahn-04, headway=3*, we could decide whether they are feasible or not by at least one method. Due to the heuristic nature of MNS, it may happen that it stops without reaching an objective value of zero on the extended instance in phase I and before reaching the time limit. In this case, the run is counted as reaching the time limit.

Table 2 shows the number of events and activities for each way of extending the event-activity network. For the full method it is clear that the number of activities is doubled and hence also the number of events is more than doubled. We observe a similar behavior for the cycle_base method, although not as prominent as in the full case. The minimal method results in very similar numbers of events and activities as the cycle_base method for the smaller instances. Both methods noticeably differ only for the data sets *bahn-03* and *bahn-04* since many full span activities are removed before adding the virtual activities.

Due to the time limit and the heuristic nature of the modulo simplex, not all problems were solved correctly. If MNS did not reach the time limit, but found a solution with objective value greater than zero for a feasible instance, it is counted as time limit. On the other hand, if it reached an objective value greater than zero for an infeasible instance and stopped before
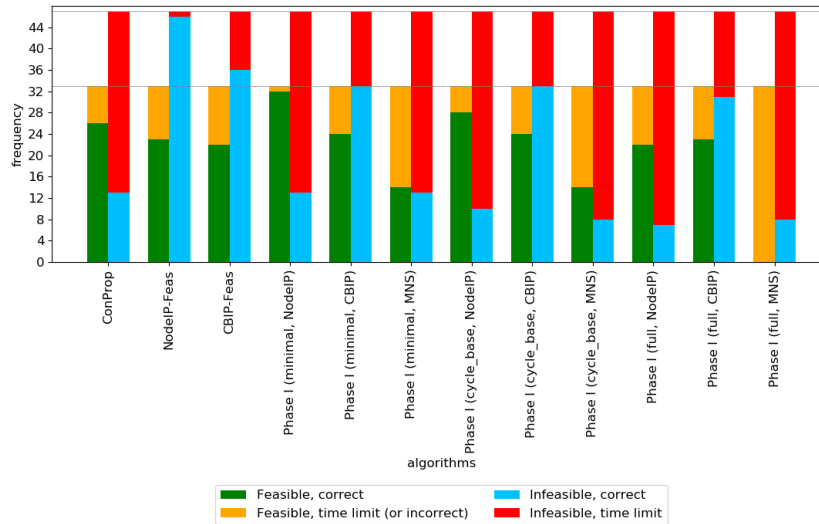
Figure 3 Correctness of all algorithms over all data sets split into feasible and infeasible instances.

the time limit, it is counted as a correct identification (however, the method is not able to prove this correctness). Figure 3 shows the correctness of the algorithms split in feasible and infeasible instances. We note that NodeIP-Feas is best in proving infeasibility. Also CBIP-Feas and phase I with CBIP perform well in proving infeasibility. ConProp and other phase I algorithms perform poorly in comparison. On the other hand, phase I with minimal or cycle_base and NodeIP finds a feasible solution more often than all other algorithms. ConProp and phase I with minimal or cycle_base and CBIP belong to the algorithms that find a feasible solution for most of the instances. Phase I with the full method and MNS does not find a correct feasible solution at all.

In Figure 4, the run times of the algorithms for each instance are ranked, i.e., how often an algorithm was the fastest, second fastest, . . . and how often the algorithm was not correct. For clarity reasons, there is no bar for the time limit. For the feasible instances, NodeIP-Feas and phase I with minimal and cycle_base as extending methods and NodeIP and CBIP as algorithms are the fastest algorithms over all. For the infeasible instances, CBIP-Feas is the fastest algorithm for most instances. Also NodeIP-Feas belongs to the group of the fastest algorithm, followed by phase I with CBIP.

Figure 5 shows a performance profile [3] over all 80 data sets. It shows the ratio of how many instances were solved within the time factor τ of the fastest algorithms for each instance. This means that at $\tau = 1$ the distribution of the fastest algorithms is shown, while at $\tau \approx 10^6$ the percent of solved instances is shown. We see that NodeIP-Feas and CBIP-Feas perform best with regards to the number of correctly solved instances. They are followed by phase I with the minimal and cycle_base method which has similar values. As already observed in previous figures, phase I with MNS and the full method performs worst.

4.3 Discussion

We first discuss the behavior of the algorithms on the different data sets. On the smallest data set *toy*, all algorithms consistently solve the instances correctly. Only phase I with MNS fails to determine the optimal solution within the time limit for some instances. The CBIP-Feas algorithm outperforms all other algorithms for all instances of toy with respect to the run time.
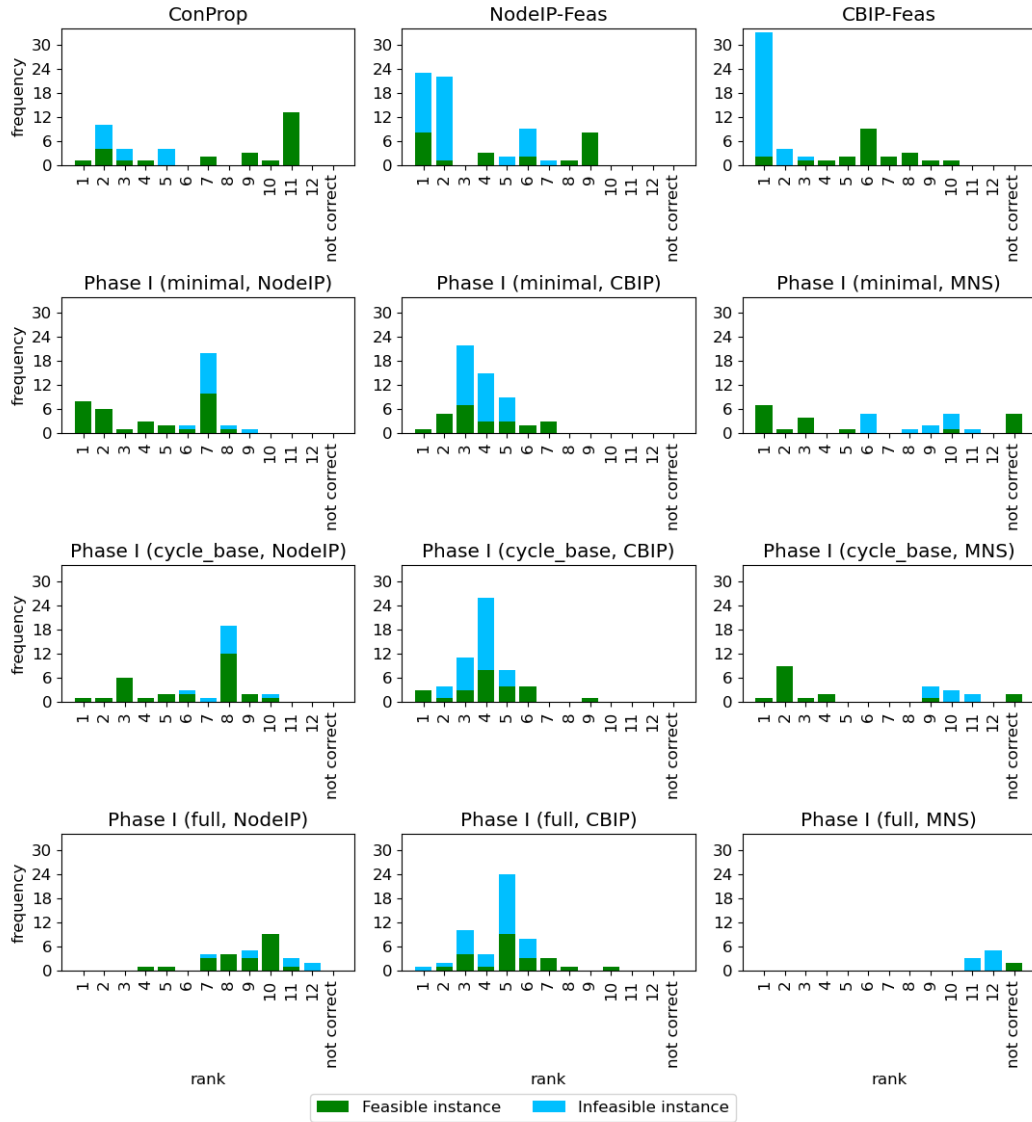
Figure 4 Ranking of run times of all algorithms over all data sets split into feasible and infeasible instances.

For the middle-sized data sets *grid* and *lowersaxony*, classic MIP solvers NodeIP-Feas and CBIP-Feas excel. They solve all instances correctly and within the time limit. The remaining algorithms are not able to solve all instances within the time limit.

The *athens* data set is slightly larger than *grid* and *lowersaxony*, however, the main difference is that the underlying public transportation network has only a few cycles – a characteristic that benefits MNS. All but one algorithm are always correct.

Finally, due to the size of the *bahn* data sets, the time limit is often exceeded. On feasible instances, phase I with NodeIP outperforms all other algorithms. On infeasible instances, NodeIP-Feas is best, followed by CBIP-Feas and phase I with CBIP. NodeIP-Feas is correct for all but one instances.
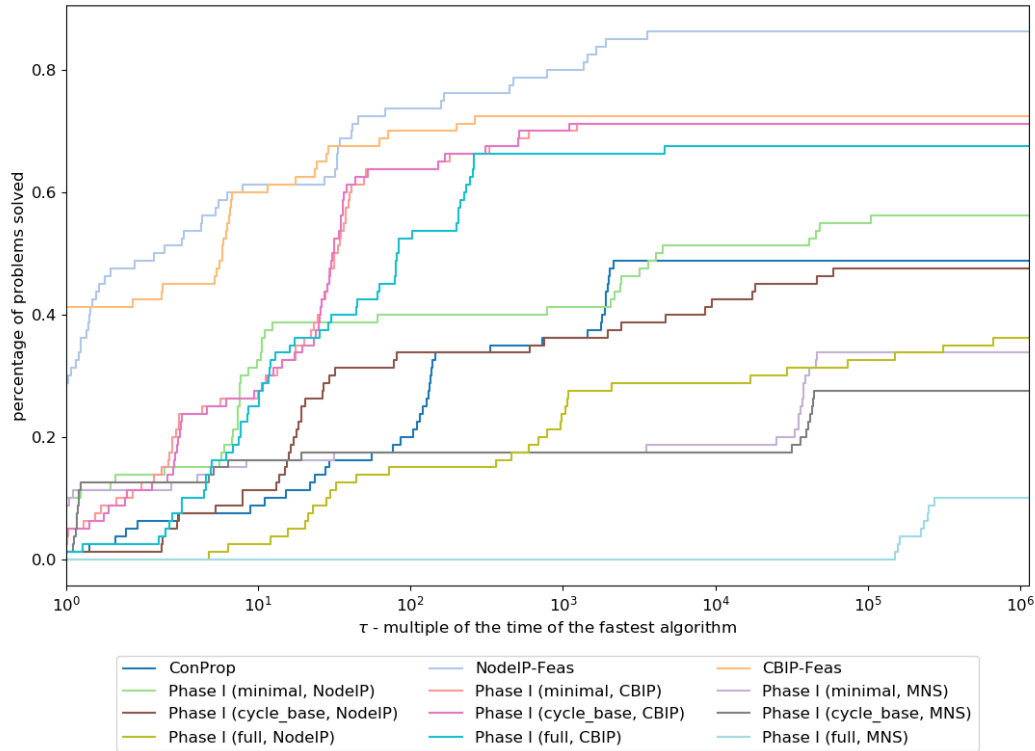
Figure 5 Performance profile of the run time of all algorithms over all data sets.

We now consider the different variants of the phase I approach. We observe that the algorithms can solve more instances correctly when we extend the event-activity network using the minimal method compared to the cycle_base method. When employing the CBIP algorithm the difference between the minimal method and cycle_base method are small. The difference is more prominent for the other two algorithms. Using the extending method full leads to fewer correctly solved instances within the time limit. We observe a similar pattern when analyzing the run times. The phase I algorithms solve phase I faster on average when the event-activity network is extended with the minimal method compared to the cycle_base method. We observe that phase I with the MIP solvers NodeIP and CBIP are correct more often and faster than phase I with MNS. The only data set where phase I with MNS is faster is *athens*.

We distinguish between feasible and infeasible instances to compare phase I with NodeIP to phase I with CBIP. While phase I with NodeIP solves more feasible instances correctly, phase I with CBIP solves more infeasible instances correctly. Phase I with NodeIP outperforms all other phase I approaches for the large and feasible instances; solving all but one instance correctly. To do so, it requires less run time than all other phase I algorithms.

In the next step, we compare the phase I to the established algorithms ConProp, NodeIP-Feas and CBIP-Feas. A direct comparison of the methods is difficult due to their heterogeneous performance on the instances. For that reason, we analyze them in-depth. We focus on the extending methods minimal and cycle_base combined with the algorithms NodeIP and CBIP. phase I with MNS is excluded from the analysis as it only performs well on *athens*. Likewise, the extending method full is excluded as it is outperformed by the other extending methods.

Comparing ConProp to the phase I approach, we observe that the phase I approach outperforms ConProp on multiple instances, with regards to number of correctly solved instances and run time. On feasible instances, phase I with NodeIP performs better. On infeasible instances, phase I with CBIP performs better. Comparing phase I to the MIP solvers, we should distinguish between feasible and infeasible instances. On feasible instances of the large data sets, classic MIP solvers fail to determine a feasible solution. In such scenarios, phase I is a better choice. For the middle-sized data sets no exact statement can be made. On infeasible instances and almost all cases, the NodeIP-Feas and CBIP-Feas perform best.

To conclude, on the studied infeasible instances the phase I approach cannot compete with the classic MIP solvers NodeIP-Feas and CBIP-Feas. On feasible instances, the phase I approach outperforms ConProp, in particular on large data sets. On these instances, classic MIP solvers often fail in determining a feasible solution within the time limit. Finally, we emphasize that the phase I approach outperforms all other algorithms on the data set *athens* with its special structure.

5   Conclusion

Finding periodic timetables is a well-known challenge when designing public transport systems. While finding a timetable with minimum travel time is notoriously difficult, already finding a feasible timetable is NP-hard. Often, such starting solutions are required as part of a local improvement method, such as the modulo network simplex.

In this paper, we developed a new method to find feasible timetables that is inspired by the phase I of the classic simplex method for linear programs. By adding virtual activities to a given event-activity network, we construct an alternative PESP instance for which a feasible solution is trivial to provide. We then solve this extended instance to find a solution that is feasible for the original problem.

We discussed different possibilities of adding virtual activities and conducted an extensive analysis of all combinations of extending methods and PESP algorithms on a set of problems taken from the *LinTim* library. Our results suggest that it is important to differentiate between feasible and infeasible instances when comparing algorithmic performances. While the new phase I approach has a higher success rate on feasible instance, the classic MIP solvers are noticeably better on infeasible instances. For best results, two algorithms may be started in parallel, as proposed in [2]: One to find a feasible solution and one to prove infeasibility.

Future research could focus on developing new extending methods, on the algorithms used for the phase I, and their combination. Furthermore, the behavior of a phase II should be further studied: how does the structure of starting solutions derived from different algorithms impact the subsequent optimization step? Finally, it would be interesting to use the starting solutions also for improving approaches (as in [18]) for integrating timetabling and routing.

References

1   Ralf Borndörfer, Heide Hoppmann, Marika Karbstein, and Niels Lindner. Separation of cycle inequalities in periodic timetabling. *Discrete Optimization*, 35:100552, 2020.

2   Ralf Borndörfer, Niels Lindner, and Sarah Roth. A concurrent approach to the periodic event scheduling problem. *Journal of Rail Transport Planning & Management*, 15:100175, 2020.

3   Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

4   Marc Goerigk and Christian Liebchen. An improved algorithm for the periodic timetabling problem. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017.

5   Marc Goerigk, Michael Schachtebeck, and Anita Schöbel. Evaluating line concepts using travel times and robustness: Simulations with the lintim toolbox. *Public Transport*, 5(3):267–284, 2013.

6   Marc Goerigk and Anita Schöbel. Engineering the modulo network simplex heuristic for the periodic timetabling problem. In *International Symposium on Experimental Algorithms*, pages 181–192. Springer, 2011.

7   Marc Goerigk and Anita Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers & Operations Research*, 40(5):1363–1370, 2013.

8   Peter Großmann, Steffen Hölldobler, Norbert Manthey, Karl Nachtigall, Jens Opitz, and Peter Steinke. Solving periodic event scheduling problems with sat. In *International conference on industrial, engineering and other applications of applied intelligent systems*, pages 166–175. Springer, 2012.

9   Sabrina Herrigel, Marco Laumanns, Jacint Szabo, and Ulrich Weidmann. Periodic railway timetabling with sequential decomposition in the pesp model. *Journal of rail transport planning & management*, 8(3-4):167–183, 2018.

10   Leo Kroon, Dennis Huisman, Erwin Abbink, Pieter-Jan Fioole, Matteo Fischetti, Gábor Maróti, Alexander Schrijver, Adri Steenbeek, and Roelof Ybema. The new dutch timetable: The or revolution. *Interfaces*, 39(1):6–17, 2009.

11   Christian Liebchen. *Periodic Timetable Optimization in Public Transport*. dissertation.de – Verlag im Internet, Berlin, 2006.

12   Christian Liebchen. Periodic timetable optimization in public transport. In *Operations research proceedings 2006*, pages 29–36. Springer, 2007.

13   Christian Liebchen. The first optimized railway timetable in practice. *Transportation Science*, 42(4):420–435, 2008.

14   Niels Lindner and Julian Reisch. Parameterized complexity of periodic timetabling. Technical Report ZIB Report 20-15, Zuse Institute Berlin, 2020.

15   Karl Nachtigall and Jens Opitz. Solving periodic timetable optimisation problems by modulo simplex calculations. In *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

16   Jens Opitz. *Automatische Erzeugung und Optimierung von Taktfahrplänen in Schienenverkehrsnetzen*, volume 1. Springer, 2009.

17   Julius Pätzold and Anita Schöbel. A matching approach for periodic timetabling. In *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016.

18   Philine Schiewe and Anita Schöbel. Periodic timetabling with integrated routing: Towards applicable approaches. *Transportation Science*, 54(6):1714–1731, 2020.

19   Anita Schöbel, Alexander Schiewe, Sebastian Albert, Julius Pätzold, Philine Schiewe, and Jochen Schulz. Lintim: An integrated environment for mathematical public transport optimization. Technical report, Documentation. Technical Report 2020.02, 2020.

20   Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.

Optimal Forks: Preprocessing Single-Source Shortest Path Instances with Interval Data

Niels Lindner ✉ 🆔
Zuse Institut Berlin, Germany

Pedro Maristany de las Casas[1] ✉ 🆔
Zuse Institut Berlin, Germany

Philine Schiewe ✉ 🆔
Department of Mathematics, Technische Universität Kaiserslautern, Germany

── Abstract ──

We investigate preprocessing for single-source shortest path queries in digraphs, where arc costs are only known to lie in an interval. More precisely, we want to decide for each arc whether it is part of some shortest path tree for some realization of costs. We show that this problem is solvable in polynomial time by giving a combinatorial algorithm, using optimal structures that we call forks. Our algorithm turns out to be very efficient in practice, and is sometimes even superior in quality to a heuristic developed for the one-to-one shortest path problem in the context of passenger routing in public transport.

1 Introduction

The shortest path problem is fundamental to combinatorial optimization, and appears in various shapes in numerous applications, not only limited to the field of transportation. Although the classical shortest path problem is very efficiently solvable, it may still be computationally challenging due to the large size of the considered instances, or since frequent recomputations with different parameters are required. One example for the latter situation occurs in route planning in transportation networks, where arc costs depend on time [1]. E.g., travel times are affected by congestion in road networks, and aircraft flight routes depend on weather conditions. In public transportation networks, uncertain travel times plays a role not only during operations, e.g., in the case of delays, but also in the planning phase in the context of line planning, which is often performed before a timetable has been fixed [12].

---

[1]   corresponding author

In these applications, although the travel times at a given time may be hard to predict, adequate lower and upper bounds are known. We therefore consider preprocessing of shortest path instances, where arc costs can be chosen arbitrarily within an interval. This applies not only to time-dependent shortest path problems, but to any situation where bounds on the arc costs are available, e.g., robust shortest path problems [8].

Our basic approach is to remove arcs when they can impossibly be on a shortest path. Ideally, for a given source s and target t, one would like to identify all the arcs that are not part of a shortest s-t-path. Several pruning heuristics have been developed for this purpose [9, 13], however, fast exact algorithms seem out of reach, as this problem is NP-complete [4, 6]. This is why we modify the problem as follows: We want to determine all arcs that cannot be on a shortest path tree rooted at a given source. A heuristic for detecting these arcs has been suggested in [2], but the complexity of the problem remained open.

We show that for a given source vertex s in a digraph on n vertices and m arcs, deciding whether an arc (w, v) can be part of a shortest path tree rooted at s is solvable in polynomial time, and construct an $\mathcal{O}(n(m + n \log n))$ algorithm. This result has been claimed in [7], but some proofs in this master thesis are incomplete or incorrect, the algorithm is more complicated than ours, and there are almost no computational results.

This paper is organized as follows: In Section 2, we reduce this single-source arc pruning problem to what we call the *s-v-w-scenario problem*, the latter serving as a basis of our considerations. Two mixed-integer programming formulations are presented in Section 3. We identify in Section 4 optimal substructures, called *forks*, which allow us to derive our combinatorial polynomial-time algorithm. Section 5 tests our single-source method on several real-world instances, and we compare our results to the one-to-one preprocessing heuristic developed for the purpose of integrated timetabling and passenger routing in [13]. We conclude the paper in Section 6.

2 Cost Scenarios and Weak Arcs

Let $G = (V, A)$ be a digraph. Let $\ell, u \in \mathbb{R}_{\geq 0}^A$ be lower resp. upper bounds for the arc costs, we assume that $\ell_a \leq u_a$ holds for every arc $a \in A$. A *cost scenario* is a vector $c \in \mathbb{R}_{\geq 0}^A$ that satisfies $\ell \leq c \leq u$. For a cost scenario c and vertices $s, t \in V$, we denote by $\Delta_{s,t}(c)$ the cost of a shortest s-t-path in G w.r.t. c. If p is a path in G containing the vertices v and w in this order, we denote by $p_{v,w}$ the v-w-subpath of p. We introduce at first the notion of weak arcs:

▶ **Definition 1** (cf. [6, 14])**.** *Let $s, t \in V$ be vertices in G.*
1. *An arc $a \in A$ is s-t-weak if there is a cost scenario c and a shortest s-t-path w.r.t. c containing a.*
2. *An arc $a \in A$ is s-weak if it is s-t-weak for some $t \in V$.*

The set of s-t-weak arcs defines the smallest subgraph of G that still contains all possible shortest s-t-paths w.r.t. all cost scenarios between ℓ and u. It is therefore desirable to characterize weak arcs algorithmically. However, there is the following negative result:

▶ **Theorem 2** ([3, 4, 6])**.** *Given a digraph G, lower and upper bounds ℓ and u, vertices $s, t \in V$, and an arc $a \in A$, it is strongly NP-complete to decide whether a is s-t-weak.*

In the single-source situation, there is an accessible characterization of weak arcs:

▶ **Lemma 3** ([2, Proposition 3])**.** *Let $s \in V$. An arc $a = (w, v) \in A$ is s-weak if and only if w is reachable from s and $\max\{\Delta_{s,v}(c) - \Delta_{s,w}(c) \mid \ell \leq c \leq u\} \geq \ell_a$.*

For fixed $s, v, w \in V$, we will therefore call $\Delta_{s,v}(c) - \Delta_{s,w}(c)$ the *value* of a cost scenario c.

▶ **Definition 4.** *Given a digraph G, lower and upper bounds ℓ and u, and vertices $s, v, w \in V$ such that v and w are reachable from s, the s-v-w-scenario problem is to find a cost scenario c of maximum value.*

The question whether an arc (w, v) can be removed from G without affecting shortest paths w.r.t. a cost scenario between ℓ and u can therefore be reduced to the solving s-v-w-scenario problem. An example for an optimal cost scenario can be seen in Figure 1.
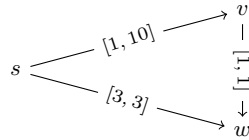
Figure 1 Digraph with interval data on arcs. The only optimal cost scenario is given by $c_{sv} = 10$, $c_{sw} = 3$, $c_{vw} = 1$, the value is 7.
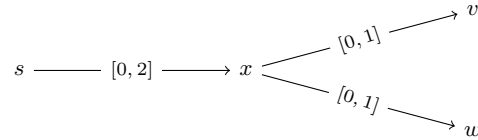
Figure 2 The shown s-v-w scenario instance is used in Example 7. It visualizes the statement in Lemma 6.

▶ **Definition 5.** *Let q be a path in G. The cost scenario defined by q is given by $c_a := \ell_a$ if $a \in q$ and $c_a := u_a$ otherwise.*

The following is a basic, but a rather cryptic self-referencing optimality condition. We refer to Appendix A for the proof.

▶ **Lemma 6** (cf. [6, Theorem 2.5]). *There is an optimal cost scenario c for the s-v-w-scenario problem defined by a shortest s-w-path w.r.t. c.*

▶ **Example 7.** Consider the graph in Figure 2. The cost scenario $c_{sx} = 1$, $c_{xv} = 1$, $c_{xw} = 0$ is optimal and has value 1. The shortest s-w-path w.r.t. c is $q = (s, x, w)$ with $c(q) = 1$. Using the notation from Lemma 6 and its proof, we can construct the cost scenario c^* with $c^*_{sx} = 0$, $c^*_{xv} = 1$, $c^*_{xw} = 0$ by setting the costs of all arcs along the shortest s-w-path $q = (s, x, w)$ w.r.t. c to their lower bound. Then c^* is the cost scenario induced by q, q is still a shortest path w.r.t. c^*, and c^* is an optimal solution to the s-v-w scenario problem with value 1.

3   Mixed-Integer Programming Formulations

In this section we provide two mixed-integer programs that solve the s-v-w-scenario problem. Resolving the maximum of a difference of minima and the linearization of shortest path costs require a few technical steps. The outcome is the program MIP_{I}:

$$\text{Maximize} \qquad \pi_v - \pi_s - \sum_{a \in A} y_a \qquad\qquad\qquad (1a)$$

$$\text{s.t.} \qquad \pi_j - \pi_i \leq c_{ij} \qquad\qquad (i,j) \in A \qquad (1b)$$

$$\sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^-(i)} x_a = \begin{cases} 1 & i = s \\ -1 & i = w \\ 0 & \text{else} \end{cases} \qquad i \in V \qquad (1c)$$

$$u_a(x_a - 1) + c_a \leq y_a \leq u_a x_a \qquad\qquad a \in A \qquad (1d)$$

$$0 \leq y_a \leq c_a \qquad\qquad a \in A$$

$$\ell_a \leq c_a \leq u_a \qquad\qquad a \in A$$

$$x_a \in \{0, 1\} \qquad\qquad a \in A$$

$$\pi_i \in \mathbb{R} \qquad\qquad i \in V$$

▶ **Lemma 8.** $\mathrm{MIP_I}$ *solves the s-v-w-scenario problem.*

Proof. See Appendix B.   ◀

Lemma 6 allows for a reduced mixed integer program $\mathrm{MIP_{II}}$ ([7]):

$$\text{Maximize} \qquad \pi_v - \pi_s - \sum_{a \in A} \ell_a x_a \qquad\qquad\qquad (2a)$$

$$\text{s.t.} \qquad \pi_j - \pi_i \leq u_{ij} - (u_{ij} - \ell_{ij}) x_{ij} \qquad (i,j) \in A \qquad (2b)$$

$$\sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^-(i)} x_a = \begin{cases} 1 & i = s \\ -1 & i = w \\ 0 & \text{else} \end{cases} \qquad i \in V \qquad (2c)$$

$$x_a \in \{0,1\} \qquad\qquad a \in A$$

$$\pi_i \in \mathbb{R} \qquad\qquad i \in V$$

▶ **Lemma 9.** $\mathrm{MIP_{II}}$ *solves the s-v-w-scenario problem.*

Proof. See Appendix B.   ◀

We want to remark that the proof of $\mathrm{MIP_{II}}$ in [7, Proposition 2.2] is incorrect: The author claims that in an optimal solution (π, x), x is always the incidence vector of a shortest s-w-path w.r.t. ℓ. However, as we will see in Remark 13, this is false.

4   Forks

We introduce forks as optimal combinatorial structures solving the s-v-w-scenario problem along with some properties in Section 4.1. Our algorithm is presented in Section 4.2.

4.1   The Theory of Forks

▶ **Definition 10.** *A fork at $x \in V$ is a pair (p, q) of paths in G such that*
1. *q is a shortest s-w-path w.r.t. the cost scenario c defined by q,*
2. *p is a shortest s-v-path w.r.t. the cost scenario c,*
3. *p and q both contain x,*
4. *the s-x-subpaths of p and q are identical,*
5. *the x-v-subpath of p and the x-w-subpath of q are arc-disjoint.*

We call c the cost scenario defined by the fork, *and we call $\Delta_{s,v}(c) - \Delta_{s,w}(c)$ the* value of the fork.

Figure 3 shows an example of a fork at a vertex $x \in V$ of the shown digraph. The following guarantees the existence of an optimal fork:

▶ **Lemma 11.** *For each shortest s-w-path q w.r.t. the cost scenario c defined by q, there is a fork (p, q). In particular, there is an optimal solution c^* to the s-v-w-scenario problem such that c^* is defined by a fork.*
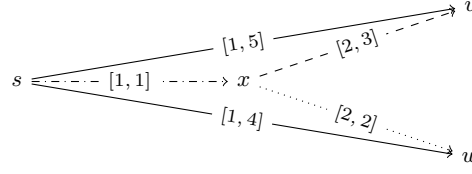
Figure 3 A digraph with a highlighted fork (p,q) at x. The dotted lines represent the s-w-path q. q is also a shortest s-w-path w.r.t. the cost scenario c defined by q. The s-v-path p is represented by the dashed lines and is minimal w.r.t. c. It shares its first arc (s,x) with q. After x, both paths diverge towards their target vertices.

Proof. Let q be a shortest s-w-path w.r.t. the cost scenario c defined by q. Let p be a shortest s-v-path w.r.t. c, and let x be the last common vertex of p and q. As the two s-x-subpaths of p and q are shortest w.r.t. c by subpath optimality, we can replace the s-x-subpath of p with the one of q and still guarantee that p is a shortest s-v-path w.r.t. c.

By Lemma 6, there is an optimal solution c defined by a shortest s-w-path q w.r.t. c. ◄

▶ **Lemma 12.** *Consider a fork (p,q) at x. Then $p_{x,v}$ is a shortest x-v-path w.r.t. u.*

Proof. Let p' be a shortest x-v-path w.r.t. u and assume that $u(p') < u(p_{x,v})$. Then

$$c(p') \le u(p') < u(p_{x,v}) = c(p_{x,v}),$$

as $p_{x,v}$ uses only arcs $a \notin q$ with $c_a = u_a$, and this contradicts p containing a shortest x-v-path w.r.t. c. ◄

▶ Remark 13. It is in general not true that $q_{s,x}$ or $q_{x,w}$ are shortest paths w.r.t. ℓ. For the s-v-w-scenario instance in Figure 1, the only optimal cost scenario c^* with $c_{sv}^* = 10$, $c_{sw}^* = 3$, and $c_{vw}^* = 1$ is induced by a fork at $x = s$ built by the s-v-path $p = (s,v)$ and the s-w-path $q = (s,w)$. In this fork, the $q_{x,w}$-subpath is not a shortest x-w-path w.r.t. ℓ.

The value of a fork at x can be computed only by knowing x and $q_{x,w}$:

▶ **Lemma 14.** *The value of a (p,q) fork at x equals $\Delta_{x,v}(u) - \Delta_{x,w}(c) = \Delta_{x,v}(u) - \ell(q_{x,w})$.*

Proof. Let c be the cost scenario defined by q. Then

$$\begin{aligned}
\Delta_{s,v}(c) - \Delta_{s,w}(c) &= c(p_{s,v}) - c(q_{s,w}) \\
&= c(p_{s,x}) + c(p_{x,v}) - c(q_{s,x}) - c(q_{x,w}) \\
&= c(p_{x,v}) - c(q_{x,w}) \\
&= \Delta_{x,v}(c) - \Delta_{x,w}(c). \\
&= \Delta_{x,v}(u) - \ell(q_{x,w}).
\end{aligned}$$

◄

The following definition is essential for our algorithm:

▶ **Definition 15.** *Let $x \in V$. We call an s-w-path q upper-bound-respecting at x if*
(P1) *q contains x,*
(P2) *for all vertices j of $q_{s,x}$ holds $\ell(q_{j,x}) \le \Delta_{j,v}(u) - \Delta_{x,v}(u)$,*
(P3) *for all vertices j of $q_{x,w}$ holds $\ell(q_{x,j}) \ge \Delta_{x,v}(u) - \Delta_{j,v}(u)$,*
(P4) *q is a shortest s-w-path for the cost scenario defined by q.*

▶ **Lemma 16.** *Let (p,q) be a fork at x. Then q is upper-bound-respecting.*

Proof. Let (p, q) be a fork at x defining the cost scenario c. Properties (P1) and (P4) are clear. For (P2), suppose that j comes before x on q. Then, as p contains a shortest j-v-path w.r.t. c via x, and by Lemma 12,

$$\Delta_{j,v}(u) \geq \Delta_{j,v}(c) = \Delta_{j,x}(c) + \Delta_{x,v}(c) = \Delta_{j,x}(c) + \Delta_{x,v}(u).$$

As $q_{j,x}$ is a shortest j-x-path w.r.t. c, we have $\Delta_{j,x}(c) = c(q_{j,x}) = \ell(q_{j,x})$. It remains to show (P3). Let j be a vertex of q after x. Since p contains a shortest x-v-path and by Lemma 12,

$$\Delta_{x,j}(c) + \Delta_{j,v}(u) \geq \Delta_{x,j}(c) + \Delta_{j,v}(c) \geq \Delta_{x,v}(c) = \Delta_{x,v}(u),$$

and we have $\Delta_{x,j}(c) = c(q_{x,j}) = \ell(q_{x,j})$. ◀

Recall that by Lemma 11, we know that there is an optimal cost scenario for the s-v-w-scenario problem that is defined by a fork. Our combinatorial algorithm will search for upper-bound-respecting paths to solve the s-v-w-scenario problem. The following lemma states that cost scenarios induced by upper-bound-respecting x-w-paths at x with minimal x-w-subpaths can only be better in value for the v-w-scenario problem than forks at x.

▶ **Lemma 17.** *Let c be a cost scenario defined by a fork (p, q) at x. Let q' be an upper-bound-respecting s-w-path at x with minimum $\ell(q')$, defining a cost scenario c'. Then*

$$\Delta_{s,v}(c') - \Delta_{s,w}(c') \geq \Delta_{s,v}(c) - \Delta_{s,w}(c).$$

Proof. First note that

$$\Delta_{s,w}(c') \leq \ell(q') = \ell(q'_{s,x}) + \ell(q'_{x,w}) \leq \ell(q'_{s,x}) + \ell(q_{x,w}),$$

as q is upper-bound-respecting by Lemma 16. Let p' be a shortest s-v-path w.r.t. c', let j denote the last common vertex of p' and q'.

Case 1: j is on $q'_{s,x}$. Then

$$\Delta_{s,v}(c') = c'(q'_{s,j}) + c'(p'_{j,v}) = \ell(q'_{s,j}) + u(p'_{j,v}) \geq \ell(q'_{s,j}) + \Delta_{j,v}(u).$$

Using that $\ell(q'_{s,x}) = \ell(q'_{s,j}) + \ell(q'_{j,x})$, we obtain from the condition (P2)

$$\ell(q'_{s,j}) = \ell(q'_{s,x}) - \ell(q'_{j,x}) \geq \ell(q'_{s,x}) + \Delta_{x,v}(u) - \Delta_{j,v}(u).$$

Inserting this,

$$\Delta_{s,v}(c') \geq \ell(q'_{s,x}) + \Delta_{x,v}(u),$$

so that, with the help of Lemma 14,

$$\Delta_{s,v}(c') - \Delta_{s,w}(c') \geq \Delta_{x,v}(u) - \ell(q'_{x,w}) \geq \Delta_{x,v}(u) - \ell(q_{x,w}) = \Delta_{s,v}(c) - \Delta_{s,w}(c).$$

Case 2: j is on $q'_{x,w}$. Then

$$\Delta_{s,v}(c') = \Delta_{s,x}(c') + \Delta_{x,j}(c') + \Delta_{j,v}(c') = \ell(q'_{s,x}) + \ell(q'_{x,j}) + u(p'_{j,v}) \geq \ell(q'_{s,x}) + \ell(q'_{x,j}) + \Delta_{j,v}(u).$$

By property (P3),

$$\ell(q'_{x,j}) \geq \Delta_{x,v}(u) - \Delta_{j,v}(u),$$

so that

$$\Delta_{s,v}(c') \geq \ell(q'_{s,x}) + \Delta_{x,v}(u),$$

and we find by Lemma 14

$$\Delta_{s,v}(c') - \Delta_{s,w}(c') \geq \Delta_{x,v}(u) - \ell(q_{x,w}) = \Delta_{s,v}(c) - \Delta_{s,w}(c). \qquad ◀$$

4.2   Finding Optimal Cost Scenarios

Lemma 17 motivates Algorithm 1: Iterate over all vertices $x \in V$ and search for an upper-bound-respecting s-w-path q through x with minimum cost w.r.t. ℓ. If this cost equals $\Delta_{s,w}(c)$ for the cost scenario c defined by q, Lemma 17 ensures that the value of c is at least the value of any fork at x. Collecting the values of all those cost scenarios c for all x, we find an upper bound on the value of an optimal fork. But as there is an optimal fork by Lemma 11, we have solved the s-v-w-scenario problem:

▶ **Theorem 18.** *Algorithm 1 solves the s-v-w-scenario problem.*

An optimal fork can in principle be recovered by the procedure indicated in the proof of Lemma 11.

◼ **Algorithm 1** s-v-w-scenario solver.

Input   : digraph $G = (V, A)$, arc cost bounds ℓ, $u \in \mathbb{R}^A_{\geq 0}$, vertices $s, v, w \in V$
Output : $\max\{\Delta_{s,v}(c) - \Delta_{s,w}(c) \mid \ell \leq c \leq u\}$ or $-\infty$

1 $M \leftarrow -\infty$ /* maximum cost scenario value */
2 **if** *v and w are reachable from s* **then**
3   **for** $x \in V$ **do**
4 $q_{s,x} \leftarrow$ shortest s-x-path w.r.t. ℓ subject to (P2)
5 $q_{x,w} \leftarrow$ shortest x-w-path w.r.t. ℓ subject to (P3)
6   **if** $q_{s,x} \neq \mathit{NULL}$ *and* $q_{x,w} \neq \mathit{NULL}$ **then**
7 $q \leftarrow q_{s,x} + q_{x,w}$ /* concatenation of paths */
8 $c \leftarrow$ cost scenario defined by q
9   **if** $\Delta_{s,w}(c) = \ell(q)$ **then**
10 $M \leftarrow \max\{M, \Delta_{s,v}(c) - \Delta_{s,w}(c)\}$
11 **return** M

Implementation

Algorithm 1 returns the value M of an optimal cost-scenario for the s-v-w-scenario problem. M is initialized to $-\infty$ and can be immediately returned in case v or w are not reachable from s. For any other non-trivial input, an efficient implementation of Algorithm 1 requires a shortest-distance matrix Δ w.r.t. the upper bound costs u, to be able to check conditions (P2) and (P3) fast in the shortest path queries triggered in Lines 4 and 5. These lines are executed in the main loop of the algorithm for every $x \in V$ to find an upper-bound-respecting s-w-path at x in two stages. In Line 4, an s-x-subpath $q_{s,x}$ with minimum cost w.r.t. ℓ among the s-x-paths whose vertices fulfill (P2) is computed. Then, in Line 5 an x-w-subpath $q_{x,w}$ that is again minimal w.r.t. ℓ given that all its vertices fulfill (P3) is computed.

To compute both subpaths, we use a modified version of Dijkstra's algorithm w.r.t. the lower bounds ℓ. The query to compute $q_{s,x}$ is run from x to s on the reversed digraph $\overleftarrow{G}$ of G rather than from s to x. Then, while $q_{s,x}$ is being computed, let $q_{x,i}$ be a path extracted from the priority queue for some vertex $i \in V$. For any outgoing arc $(i, j) \in \overleftarrow{A}$ we build new x-j-subpaths $q_{x,j}$ but only those fulfilling $\ell(q_{x,j}) \leq \Delta_{j,v}(u) - \Delta_{x,v}(u)$, which is exactly condition (P2), are further considered in the elsewhere unaltered execution of Dijkstra's algorithm. This query runs from x to s because the left hand side of (P2) evaluates an j-x-subpath. If we would run Dijkstra's algorithm on the original digraph G, we would only

be able to evaluate s-j-subpaths for some $j \in V$. The computation of the $q_{x,w}$-subpath works very similarly. It uses the original digraph G and checks condition (P3) for every new path candidate. Note that if the shortest distance matrix Δ is known, the additional checks during the Dijkstra queries can be done in $\mathcal{O}(1)$ and thus do not have an impact on the overall complexity of the algorithm.

In case our modified Dijkstra find paths $q_{s,x}$ and $q_{x,w}$, their concatenation $q := q_{s,x} + q_{x,w}$ clearly fulfills conditions (P1)-(P3). If we then build the cost scenario c induced by q (Line 8) and run a one-to-all Dijkstra query w.r.t. c starting at s, we can check whether q also fulfills (P4) (Line 9). If it does, q is an upper-bound-respecting s-w-path through x. Thus, it qualifies to possibly update the return value M in case c yields a better value for the s-v-w-scenario problem than the best value known so far (Line 10).

▶ **Theorem 19.** *Algorithm 1 runs in $\mathcal{O}(n(n \log(n) + m))$.*

Proof. The computation of the shortest-distance matrix Δ can be done in $\mathcal{O}(n(n \log(n)+m))$ using Johnson's algorithm [5]. In every iteration of its main loop, the algorithm runs at most three Dijkstra queries (Line 4, Line 5, and Line 9). The distances $\Delta_{s,v}(c)$ (Line 10) and $\Delta_{s,w}(c)$ (Line 9) can be computed in the same Dijkstra query. Since n iterations are performed in total, this results in a running time of $\mathcal{O}(n(n \log(n) + m))$ for the main loop, if we assume that a Fibonacci heap is used. ◀

5   Computational Results

The aim of this section is twofold. First, in Section 5.2, we compute the sets of weak arcs for one-to-all shortest path instances with interval data using $\mathrm{MIP_{II}}$ and Algorithm 1, and show that the latter method is much faster. Secondly, in Section 5.3, we compare our method to an *arc-based pruning heuristic* introduced in [13] to show that our exact method is more effective.

The arc-based pruning from [13] works for the one-to-one shortest path problem with interval data. We assume that every graph has a set V_S of origin vertices and a set V_T of target vertices. Then, for $s \in V_S$ and $t \in V_T$, an arc $(v, w) \in A$ is guaranteed to not lie on any shortest s-t-path if

$$\Delta_{s,t}(u) < \Delta_{s,v}(\ell) + \ell_{v,w} + \Delta_{w,t}(\ell). \tag{3}$$

For a proof, see [13, Theorem 4]. Note that this criterion does not imply that an arc that does not fulfill (3) is s-t-weak. To check (3) algorithmically, two shortest path trees w.r.t. ℓ have to be computed: one rooted at s and one rooted at t on the reversed digraph of G. Then, for every vertex $v \in V$, the distances $\Delta_{s,v}(\ell)$ and $\Delta_{v,t}(\ell)$ are known and (3) can be used to discard irrelevant arcs from G (cf. [13, Algorithm 3]). Since this method discards irrelevant arcs for a fixed s-t-pair only, the set of remaining arcs is not directly comparable to the set of s-weak arcs. Therefore, we repeatedly apply the arc-based pruning from a fixed origin vertex $s \in V_S$ to all vertices $v \in V$. Then, for any s-v-pair, we get a set $A_{s,v}$ of arcs that do not lie on any shortest s-v-path. Consequently, only the arcs in $A_s := \bigcap_{v \in V} A_{s,v}$ are guaranteed to not lie on any shortest path starting at s. The complement of A_s is a superset of the set of s-weak arcs. We call the procedure of computing the sets A_s for all $s \in V_S$ the V_S-V *arc-based pruning*.

For realistic routing instances, it is often interesting to focus only on s-t-pairs for vertices $t \in V_T$. In contrast to the finding of s-weak arcs, the V_S-V arc-based pruning can easily be adapted to consider this setting. We get arcs $A_s^T := \bigcap_{t \in V^T} A_{s,t}$ that are guaranteed to

not lie on a shortest path from s to any target vertex $t \in V^T$. Thus, the complement of A_s^T contains all arcs that can lie on a shortest s-t path with $t \in V_T$. We call the computation of A_s^T the V_S-V_T *arc-based pruning*. In Section 5.3 we compare the sets of arcs returned by both arc-based pruning techniques with the sets of s-weak arcs.

5.1   Instance Description and Implementation Details

Table 3 in Appendix C shows an overview of the used instances and their size. The instances `toy`, `grid`, `lowersaxony`, and `athens` are taken from the open source software framework LinTim [11, 10] which contains algorithms and data sets for public transport planning. While `toy` and `grid` are artificial data sets, `lowersaxony` and `athens` represent the regional train system of Lower Saxony and the metro in Athens, respectively. For most instances, lower and upper bounds on the arcs are distinct, but for instance `grid-fix` the lower and upper bound coincide for a substantial amount of the arcs, as the duration of drive and wait activities is fixed. Instances `*-res` (stands for *res*tricted) and `*-all` vary in the set of transfer stations and thus in the number of arcs. The restriction of transfer stations is done according to [12] such that for instances with fixed drive and wait activities the optimal travel time is not impacted.

The instances `W1` to `W9` model subnetworks of the public transport network of the city of Wuppertal. As with `grid-fix`, drive and wait activities are fixed, moreover, the minimum transfer time is uniform across all stations.

We also consider the airway network above Germany. It is a layered directed graph, whose layers are connected via climb and descend arcs. A path using arcs in a higher layer represents a flight cruising at a higher altitude. A single arc has a copy in multiple layers and different costs in each of them. This discrete set of costs per arc allows us to derive lower and upper bounds on the arcs' costs. We then run all algorithms on a projection of the layered graph in which each arc appears only once, and climbing and descending arcs are ignored.

We use Gurobi 9.1.0 to solve the $\mathrm{MIP_{II}}$ models. Algorithm 1 (see supplementary material) and the arc-based prunings are implemented in C++ and compiled using gcc 7.5.0 and gcc 7.4.0, respectively. Gurobi and Algorithm 1 were run on a computer with an Intel Xeon CPU E5-2670 v2 @ 2.50GHz processor and 128GB of memory. The arc-based prunings are were run on a computer with an AMD Ryzen 5 PRO 2500U @ 2.00GHz processor and 16GB of memory.

5.2   Running Time: $\mathrm{MIP_{II}}$ vs. Algorithm 1

In every instance, we iterate over the origin vertices $s \in V_S$ and solve the resulting s-v-w-scenario problems for all arcs $(w, v) \in A$ using the $\mathrm{MIP_{II}}$ model and using Algorithm 1. For every origin vertex $s \in V_S$, we consider the average running time t_s needed to solve the s-v-w-scenario problems. Finally, in Table 1 we report the average of all t_s values, $s \in V_S$, for every considered graph and both solution approaches.

On the `W*` instances, we observe that the solutions calculated using Algorithm 1 are obtained orders of magnitude faster than using $\mathrm{MIP_{II}}$ models. The `W7` instance is the first for which not all origin vertices can be considered within the time limit of three days. In contrast, the running time of Algorithm 1 remains low even for the biggest instance `W9` since all s-v-w-scenario instances for a fixed origin vertex s can be solved within $3.2s$ in average. On the `toy-res` and `athens-res` instances, all MIPs could be solved but Algorithm 1 is around 4 orders of magnitude faster. On all other instances no optimal solution to the $\mathrm{MIP_{II}}$ models could be found due to memory restrictions or timeouts. The `air-germany` instance is the biggest instance and the running times of Algorithm 1 behave accordingly: on average, solving all s-v-w-scenarios for a fixed origin vertex takes $1551.28s$.

■ **Table 1** Average time in seconds needed to compute the sets of s-weak arcs using Algorithm 1 and the MIP_{II} models. The *Solved* column reports the percentage of the origin vertices $s \in V_S$ for which all s-v-w-scenarios were solved by Gurobi. Algorithm 1 solved all instances. The computations stopped after 72 hours.

Instance	Algorithm 1	MIP_{II}	
Graph Name	Avg. time	Solved [%]	Avg. time
`toy-all`	0.1141	0.0	–
`toy-res`	0.0446	100.0	307.6409
`grid-all`	1.4550	0.0	–
`grid-res`	0.5730	0.0	–
`grid-fix-all`	1.4119	0.0	–
`grid-fix-res`	0.5417	0.0	–
`W1`	0.0019	100.0	4.8697
`W2`	0.0067	100.0	15.6175
`W3`	0.0211	100.0	68.7970
`W4`	0.2150	100.0	620.3328
`W5`	0.1804	100.0	698.2918
`W6`	0.6636	100.0	2466.2169
`W7`	1.1665	46.0	4015.982
`W8`	1.7622	37.2	4936.5306
`W9`	3.2349	15.7	9464.8684
`lowersaxony-all`	0.9811	0.0	–
`lowersaxony-res`	0.3155	0.0	–
`athens-all`	3.3776	0.0	–
`athens-res`	1.1595	100.0	3841.8138
`air-germany`	1551.2769	0.0	–

5.3 Effectiveness: Arc-based Pruning vs. Weak Arcs Solvers

We now compare the sets of s-weak arcs and the sets of remaining arcs after applying the V_S-V and the V_S-V_T arc-based pruning. In Table 2, we report the average cardinality of these sets after computing them for every $s \in V_S$ and $v \in V_T$ or $v \in V$ depending on the arc-based pruning variant. Figure 4 visualizes the same data. Recall that lower numbers are better since they imply that more arcs could be discarded. The remaining arcs after the V_S-V arc-based pruning are always a superset of the s-weak arcs and of the remaining arcs after the V_S-V_T arc-based pruning. There is no theoretical implication relating the size of the latter two sets.

On the `W*` instances, the difference between the set of s-weak arcs and the remaining arcs using the V_S-V arc-based pruning increases as the instances get bigger. For the `W9` instance, 30% of the arcs are s-weak and after the V_S-V arc-based pruning 51% are kept. On all `W*` instances, the V_S-V_T arc-based pruning discards the most arcs but without a clear correlation with the instances' size: for example, compared to the sets of s-weak arcs, it discards 10% more arcs for `W2` and for `W9` the advantage shrinks to only 1%.

On the synthetic and the remaining public transportation instances, we observe interesting results: There are not many target vertices and still the sets of arcs discarded by the V_S-V and V_S-V_T arc-based prunings are almost equal. Additionally, the sets of s-weak arcs turn out to be always smaller. This effect is particularly notable on the `grid-all` and `grid-fix-all` instances, where both arc-based prunings are equally effective and the s-weak arcs are 8% and 10% less, respectively. On the non-synthetic instances `lowersaxony-all` and `athens-all` the arc-based prunings again coincide and the sets of s-weak arcs contain 5% and 3% fewer arcs. Taking the average over the origin vertices, 80% of arcs are s-weak in `lowersaxony-all` and 68% in `athens-all`. Regarding the `air-germany` instance the sets of s-weak arcs contain

Table 2 Average number of remaining arcs for the s-v-w-scenario solver and the arc-based prunings. The averages are built among the number of weak arcs for every fixed vertex $s \in V_S$.

Graph Name	avg. weak arcs s-v-w-solver		avg. remaining arcs V_S-V_T-arc-pruning		avg. remaining arcs V_S-V-arc-pruning	
	tot.	rel.	tot.	rel.	tot.	rel.
`toy-all`	681	0.59	768	0.66	768	0.66
`toy-res`	498	0.48	526	0.50	542	0.52
`grid-all`	1953	0.75	2148	0.83	2148	0.83
`grid-res`	1464	0.62	1563	0.66	1572	0.67
`grid-fix-all`	1817	0.70	2088	0.80	2091	0.80
`grid-fix-res`	1355	0.58	1453	0.62	1504	0.64
`W1`	56	0.40	43	0.30	56	0.40
`W2`	95	0.38	69	0.28	104	0.42
`W3`	155	0.32	123	0.26	170	0.35
`W4`	359	0.34	289	0.27	451	0.43
`W5`	351	0.35	330	0.33	467	0.47
`W6`	587	0.35	520	0.31	846	0.51
`W7`	719	0.32	682	0.30	1127	0.50
`W8`	908	0.31	768	0.26	1350	0.46
`W9`	1177	0.30	1120	0.29	1984	0.51
`lowersaxony-all`	1410	0.80	1492	0.85	1492	0.85
`lowersaxony-res`	964	0.64	1004	0.67	1007	0.67
`athens-all`	2368	0.68	2482	0.71	2482	0.71
`athens-res`	1414	0.51	1454	0.52	1455	0.52
`air-germany`	13610	0.41	16504	0.49	27656	0.83

42% less arcs than the sets of remaining arcs after the V_S-V arc-based pruning. The V_S-V_T arc-based pruning works better than the latter but on average, the sets of remaining arc contain 8% more arcs than the sets of s-weak arcs. Figure 5 contains a plot showing the distribution of the size of the sets of remaining arcs per origin vertex for a representative instance of each type.

6 Conclusion

Deciding whether an arc is s-weak for some source vertex s can be done efficiently at the same complexity as a standard all-to-all shortest path query, using a series of minor modifications of Dijkstra's algorithm. It is hence no surprise that Algorithm 1 runs much faster than commercial solvers on the mixed-integer programming formulation $\mathrm{MIP_{II}}$. In quality, our algorithm performs at least comparable to the arc-based pruning heuristic from [13], for some instances, it is even superior, although that heuristic has been developed for the one-to-one shortest path queries. We hence conclude that our s-v-w-scenario algorithm can serve as a powerful preprocessing tool for shortest path problems in a variety of application contexts.

Beyond testing our method on a larger variety of instances, e.g., road networks, and combining the algorithm with the V_S-V_T arc pruning in an iterative process, a natural question is to tackle the complexity of computing a subgraph of minimum size that contains at least one shortest path for each cost scenario, rather than containing all shortest paths. Another related problem is the detection of arcs that are part of a shortest path trees for all cost scenarios. These are called *strong arcs* in the robust optimization literature (e.g., [14]), and following [7], it seems that similar methods are available here.
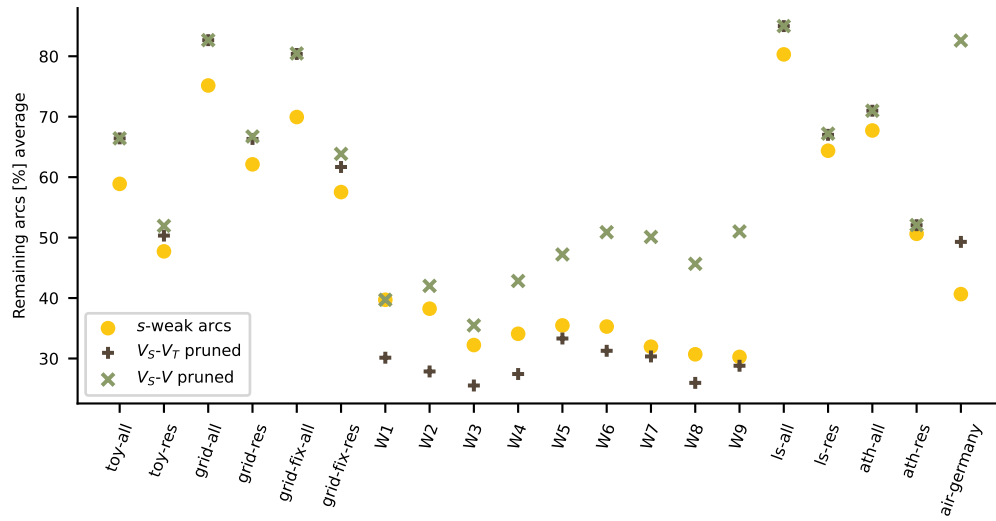
Figure 4 Average cardinality (as a percentage of the total number of arcs in each instance) of the sets of relevant arcs determined using Algorithm 1 and the V_S-V_T and V_S-V arc-based prunings.
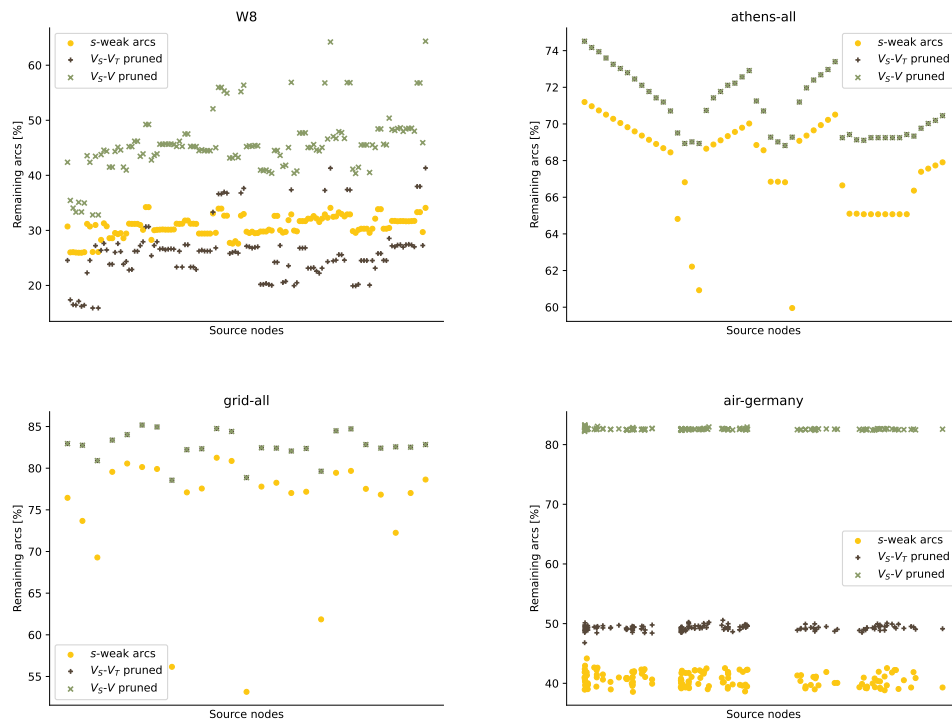
Figure 5 Remaining arcs per origin vertex.

References

1   H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route Planning in Transportation Networks. In *Algorithm Engineering*, 2016. `doi:10.1007/978-3-319-49487-6_2`.

2   D. Catanzaro, M. Labbé, and M. Salazar-Neumann. Reduction approaches for robust shortest path problems. *Computers & Operations Research*, 38(11):1610–1619, 2011. `doi:10.1016/j.cor.2011.01.022`.

3   S. Chanas and P. Zieliński. The computational complexity of the criticality problems in a network with interval activity times. *European Journal of Operational Research*, 136(3):541–550, February 2002. `doi:10.1016/s0377-2217(01)00048-0`.

4   S. Chanas and P. Zieliński. On the hardness of evaluating criticality of activities in a planar network with duration intervals. *Operations Research Letters*, 31(1):53–59, 2003. `doi:10.1016/S0167-6377(02)00174-8`.

5   D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, January 1977. `doi:10.1145/321992.321993`.

6   O. E. Karasan, M. Pinar, and H. Yaman. The Robust Shortest Path Problem with Interval Data. Technical report, Bilkent University, 2001.

7   F. Krellner. Shortest Paths with Interval Data and their Application in Timetabling. Master's thesis, Freie Universität Berlin, 2018.

8   R. Montemanni and L. M. Gambardella. An exact algorithm for the robust shortest path problem with interval data. *Computers & Operations Research*, 31(10):1667–1680, 2004. `doi:10.1016/S0305-0548(03)00114-X`.

9   A. Schienle, P. Maristany, and M. Blanco. A Priori Search Space Pruning in the Flight Planning Problem. In Valentina Cacchiani and Alberto Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASIcs)*, pages 8:1–8:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 2190-6807. `doi:10.4230/OASIcs.ATMOS.2019.8`.

10   A. Schiewe, S. Albert, P. Schiewe, A. Schöbel, and F. Spühler. LinTim - Integrated Optimization in Public Transportation. Homepage. `https://lintim.net`, 2020.

11   A. Schiewe, S. Albert, P. Schiewe, A. Schöbel, and F. Spühler. LinTim: An integrated environment for mathematical public transport optimization. Documentation for version 2020.12. Technical report, TU Kaiserslautern, 2020. URL: `https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-62025`.

12   P. Schiewe. *Integrated Optimization in Public Transport Planning*, volume 160 of *Optimization and Its Applications*. Springer, 2020. `doi:10.1007/978-3-030-46270-3`.

13   P. Schiewe and A. Schöbel. Periodic Timetabling with Integrated Routing: Toward Applicable Approaches. *Transportation Science*, 54(6):1714–1731, 2020. `doi:10.1287/trsc.2019.0965`.

14   H. Yaman, O. E. Karaşan, and M. Ç. Pınar. The robust spanning tree problem with interval data. *Operations Research Letters*, 29(1):31–40, 2001. `doi:10.1016/S0167-6377(01)00078-5`.

A   Proof of Lemma 6

▶ **Lemma 6** (cf. [6, Theorem 2.5]). *There is an optimal cost scenario c for the s-v-w-scenario problem defined by a shortest s-w-path w.r.t. c.*

Proof. Consider an optimal solution c and a shortest s-w-path q w.r.t. c. Further consider the cost scenario c^* defined by q in the sense of Definition 5. We claim at first that q is also a shortest s-w-path w.r.t. c^*. Indeed, let q' be any s-w-path. Then, as q is shortest w.r.t. c,

$$\sum_{a \in q \cap q'} c_a + \sum_{a \in q \setminus q'} c_a = c(q) \leq c(q') = \sum_{a \in q \cap q'} c_a + \sum_{a \in q' \setminus q} c_a,$$

we find

$$\sum_{a \in q \setminus q'} c_a^* = \sum_{a \in q \setminus q'} \ell_a \leq \sum_{a \in q \setminus q'} c_a \leq \sum_{a \in q' \setminus q} c_a \leq \sum_{a \in q' \setminus q} u_a = \sum_{a \in q' \setminus q} c_a^*,$$

and therefore

$$c^*(q) = \sum_{a \in q \cap q'} c_a^* + \sum_{a \in q \setminus q'} c_a^* \leq \sum_{a \in q \cap q'} c_a^* + \sum_{a \in q' \setminus q} c_a^* = c^*(q').$$

In particular, the cost scenario c^* is defined by the shortest s-w-path q w.r.t c^*.

Now let p be a shortest s-v-path w.r.t. c^*. We then have

$$\begin{aligned}
\Delta_{s,v}(c^*) - \Delta_{s,w}(c^*) &= c^*(p) - c^*(q) \\
&= \sum_{a \in p \setminus q} c_a^* - \sum_{a \in q \setminus p} c_a^* \\
&= \sum_{a \in p \setminus q} u_a - \sum_{a \in q \setminus p} \ell_a \\
&\geq \sum_{a \in p \setminus q} c_a - \sum_{a \in q \setminus p} c_a \\
&= c(p) - c(q) \geq \Delta_{s,v}(c) - \Delta_{s,w}(c).
\end{aligned}$$

But as c was optimal, we must have that c^* is optimal as well. ◀

B   Correctness Proofs for $\mathrm{MIP}_{\mathrm{I}}$ and $\mathrm{MIP}_{\mathrm{II}}$

▶ **Lemma 8.** $\mathrm{MIP}_{\mathrm{I}}$ *solves the s-v-w-scenario problem.*

Proof. Let c^* be an optimal cost scenario and let q be a shortest s-w-path w.r.t. c^*. Set $x_a^* := 1$ for all arcs $a \in q$ and $x_a^* := 0$ otherwise. Then x^* satisfies the flow constraints (1c). For all vertices $i \in V$, set $\pi_i^* := \Delta_{s,i}(c^*)$. As $\Delta_{s,i}(c^*) + c_{ij}^* \geq \Delta_{s,j}(c^*)$ for all $(i,j) \in A$, π^* and c^* satisfy (1b). The coupling constraints (1d) yield a vector y^* with the property that $y_a^* = c_a^*$ for $a \in q$ and $y_a^* = 0$ otherwise, so that

$$\sum_{a \in A} y_a^* = \sum_{a \in A} c_a^* x_a^* = c^*(q) = \Delta_{s,w}(c^*).$$

We conclude that the objective value (1a) of this feasible solution is $\Delta_{s,v}(c^*) - \Delta_{s,w}(c^*)$, i.e., the value of c^*.

It remains to show that the optimal objective value of $\mathrm{MIP}_{\mathrm{I}}$ is at most the value of c^*. To this end, let (c, π, x, y) be an optimal solution to $\mathrm{MIP}_{\mathrm{I}}$. For given x, y, c, this optimal solution must satisfy $\pi_v - \pi_s = \Delta_{s,v}(c)$, as

$$\max\{\pi_v - \pi_s \mid \pi_j - \pi_i \leq c_{ij} \text{ for all } (i,j) \in A, \pi_i \in \mathbb{R} \text{ for all } i \in V\}$$

is the dual linear programming formulation of the shortest s-v-path problem w.r.t. c. Moreover, as x indicates some s-w-path by (1c), and analyzing the coupling constraints (1d), we have $\sum_{a \in A} y_a = \sum_{a \in A} c_a x_a \geq \Delta_{s,w}(c)$. We conclude that the optimal value of $\mathrm{MIP}_{\mathrm{I}}$ is at most $\Delta_{s,v}(c) - \Delta_{s,w}(c)$, and this is in turn at most the value of c^*. ◀

▶ **Lemma 9.** $\mathrm{MIP}_{\mathrm{II}}$ *solves the s-v-w-scenario problem.*

Proof. Let c^* be an optimal cost scenario, we can complete c^* to an optimal solution (c^*, π^*, x^*, y^*) to $\mathrm{MIP_I}$ by the proof of Lemma 8. By Lemma 6, we can assume that x^* corresponds to a shortest s-w-path q w.r.t. c^*, and that c^* is defined by q. Clearly, x^* satisfies the flow constraints (2c). Moreover, for arcs $(i, j) \in A$ with $x_{ij}^* = 0$, we have $\pi_j^* - \pi_i^* \le c_{ij}^* \le u_{ij}^*$, so that (2b) holds. Otherwise, if $x_{ij}^* = 1$, we have that $c_{ij}^* = \ell_{ij}$, so that $\pi_j^* - \pi_i^* \le \ell_{ij}$ and hence (2b) are satisfied. In particular, (π^*, x^*) is feasible for $\mathrm{MIP_{II}}$, and we note that $\sum_{a \in A} \ell_a x_a^* = c^*(q^*) = \Delta_{s,w}(c^*)$, so that the objective value (2a) of (π^*, x^*) equals the $\Delta_{s,v}(c^*) - \Delta_{s,w}(c^*)$. This shows that the optimal objective value of $\mathrm{MIP_{II}}$ is at least the value of c^*.

Conversely, let (π, x) be an optimal solution to $\mathrm{MIP_{II}}$. We obtain a feasible solution to $\mathrm{MIP_I}$ by defining $c_a := u_a - (u_a - \ell_a)x_a$ and $y_a := c_a x_a$ for all $a \in A$, and the objective value in $\mathrm{MIP_I}$ remains the same. Applying Lemma 8, the optimal value of $\mathrm{MIP_{II}}$ is at most the value of the optimal scenario c^*. ◀

C   Instance Details

Table 3 Overview of the used instances.

Type	Name	Vertices	Arcs	Origins $|V_S|$	Targets $|V_T|$	$\ell_a = u_a$
Synthetic	`toy-all`	184	1156	8	8	188
	`toy-res`	184	1044	8	8	188
	`grid-all`	442	2598	25	25	392
	`grid-res`	442	2356	25	25	392
	`grid-fix-all`	442	2598	25	25	756
	`grid-fix-res`	442	2356	25	25	756
Public Transport	`W1`	56	142	28	28	142
	`W2`	88	248	36	36	236
	`W3`	122	480	52	52	436
	`W4`	254	1052	80	80	912
	`W5`	242	990	84	84	814
	`W6`	365	1663	102	102	1291
	`W7`	434	2249	111	111	1597
	`W8`	516	2957	129	129	2001
	`W9`	631	3889	140	140	2477
	`lowersaxony-all`	480	1756	34	34	412
	`lowersaxony-res`	480	1498	34	34	412
	`athens-all`	1066	3496	51	51	964
	`athens-res`	1066	2794	51	51	964
Air	`air-germany`	13896	26576	154	125	0

Solving the Dynamic Dial-a-Ride Problem Using a Rolling-Horizon Event-Based Graph

Daniela Gaul[1] ✉ 🄳
Department of Mathematics, Universität Wuppertal, Germany

Kathrin Klamroth ✉ 🄳
Department of Mathematics, Universität Wuppertal, Germany

Michael Stiglmayr ✉ 🄳
Department of Mathematics, Universität Wuppertal, Germany

—— **Abstract** ——

In many ridepooling applications transportation requests arrive throughout the day and have to be answered and integrated into the existing (and operated) vehicle routing. To solve this dynamic dial-a-ride problem we present a rolling-horizon algorithm that dynamically updates the current solution by solving an MILP formulation. The MILP model is based on an event-based graph with nodes representing pick-up and drop-off events associated with feasible user allocations in the vehicles. The proposed solution approach is validated on a set of real-word instances with more than 500 requests. In 99.5% of all iterations the rolling-horizon algorithm returned optimal insertion positions w.r.t. the current schedule in a time-limit of 30 seconds. On average, incoming requests are answered within 2.8 seconds.

1 Introduction

In the dynamic dial-a-ride-problem (DARP) a fleet of vehicles must serve transportation requests defined by origin, destination, load and time windows, that arrive throughout the day. An important application are on-demand ridepooling services which are taxi-like services that process transportation requests submitted via a smartphone app. In contrast to taxi-services, where pooling is usually not allowed, customers with similar origin or destination are assigned to the same ride whenever economically and/or ecologically useful. Thus, ridepooling services are a cheap alternative to taxi-services and private cars with the potential to reduce congestion and particulate pollution in big cities. Some prominent

---

examples are DiDi[3] or UberPool[4]. This paper is motivated by Hol-Mich-App[5], a ridepooling service that was recently established in the city of Wuppertal (Germany). In order to achieve a high level of user acceptance in the competition with individual car transport, an efficient and user-friendly route planning is crucial.

Despite being a highly relevant topic of reserach, the dynamic DARP has been less studied than its static counterpart (see the survey Ho et al. [12]), where it is assumed that all requests are known prior to the start of service. The topic of this paper is the dynamic while still deterministic DARP, i.e., we assume that all information, when received, is known with certainty. An exhaustive and in-depth survey on DARP is given in [12], and a survey on dynamic pick-up and delivery problems can be found in [3]. Solution strategies to the dynamic DARP are often motivated by the requirement to determine immediately a feasible routing including the new requests. A frequently applied solution strategy to dynamic DARP problems combines two approaches (see e.g. [1, 2, 4, 6, 8, 13, 16, 19, 21, 22]): On the one hand, a new request is inserted using fast and simple insertion heuristics. In the idle time between a pair of new requests, on the other hand, a more complex heuristic or meta-heuristic may be used to continually re-optimize the current solution. We give a brief overview on the variants of insertion and re-optimization heuristics used in the literature.

The first and most simple insertion heuristic tries to insert the new request in the current vehicle routes without relocating already assigned users. If a feasible insertion position is found, the new request is inserted in the best insertion position in terms of incremental cost. Variants of this strategy are employed, for example, by Beaudry et al. [2], Carotenuto and Martis [6], Hanne et al. [11], Häll and Peterson [13], Lois and Ziliaskopoulos [16], Madsen et al. [18], Marković et al. [19], Psaraftis [20] and Santos and Xavier [21]. The second variant of an insertion heuristic allows the relocation of already assigned users, thus leading two a higher number of possible insertion positions for the new request. For instance, Attanasio et al. [1] use parallel heuristics to solve the dynamic DARP combining random insertion and tabu search. Berbeglia, Cordeau and Laporte [4] run a tabu search heuristic in parallel with a constraint programming algorithm to determine whether a new request can be inserted feasibly in a given solution or not. Luo and Schonfeld [17] relocate requests which are similar w.r.t. time windows and geographic locations whenever a simple insertion heuristic declares a new request to be infeasible. Vallée et al. [22] propose and analyze three different heuristics aiming at reshuffling already accepted requests if a new request's insertion has been declared infeasible by a service provider's online system. The heuristics are based on ruin and recreate operators and the ejection chain concept [10]. In [8] new unexpected requests may show up at a vehicle stop. In the idle time between two vehicle stops, a neighborhood of the current vehicle route is created. The insertion of the unexpected request is evaluated for all routes in the neighborhood of the current route. A maximum cluster algorithm that finds for each set of users a maximal subset of users that can be served by one vehicle, developed by Häme and Hakula [14], can be used to quickly decide if new requests should be accepted or rejected.

The second phase of a solution approach to the dynamic DARP consists of a reoptimization phase. To improve the current solution in the idle time between a pair of new requests, different variants of local search have been applied. For example, a reinsertion heuristic is used to remove a request from its current route and evaluate the reinsertion of the request into all other routes and/or a swap heuristic exchanges two requests with different routes, see

---

e.g. [17, 19, 16, 21]. In [6] the quality of the solution is sought to be improved by reinserting the entire set of accepted requests. In [13] several ruin and recreate heuristics are combined and compared; in particular ruin methods based on the removal of sequences of requests have proven to improve the quality of solutions. Attanasio et al. [1], Beaudry et al. [2] and Berbeglia et al. [4] make use of (different variants) of tabu search in the improvement phase.

Contribution

As highlighted in the literature review, the standard approach to solve the dynamic DARP is to apply a two-phase algorithm consisting of an insertion heuristic and a reoptimization phase. In this paper, we suggest a more global perspective and aim at the iterative computation of exact optimal solutions that satisfy all feasibility constraints and that respect previous routing decisions. Only when this global optimization exceeds a prespecified time limit of 30 seconds without proving global optimality, the computed schedule is reoptimized in the following iteration. We present computational experiments for real-world data from a ridepooling service in the city of Wuppertal in Germany with up to 500 requests. In all tested instances the average response time was never more than 2.9 seconds. Moreover, a reoptimization was necessary in no more than 0.5% of the iterations. In all other iterations the algorithm returned a globally optimal solution w.r.t. the current situation, which can generally not be guaranteed by common two-phase heuristics.

The remainder of this paper is structured as follows. A formal problem description and an outline of the solution strategy applied in this paper is given in Section 2. In Section 3 the concept of an event-based graph is explained and transferred to the dynamic DARP by associating a dynamic event-based graph with each subproblem of DARP. The corresponding MILP model is introduced in Section 4. Finally, the procedure of updating the event-based graph and solving the MILP model, resulting in a decision on the acceptance of new requests, is outlined in the framework of a rolling-horizon algorithm in Section 5. To validate our approach, computational results on two real-world instances are presented in Section 6. A short summary of our results is given in Section 7. A list of parameters and variables used throughout this paper can be found in the appendix.

2 Problem Description

In this paper, we consider a dynamic DARP in which a finite set of n *transport requests* submitted by *users* have to be either accepted and scheduled or rejected. The transport service is provided by a fleet of K vehicles with capacity Q. All vehicles are situated at a depot, denoted by 0, when the service is started. Let $R := \{1, \ldots, n\}$ denote the transport requests/users. We consider discrete points in time $\tau_1 \leq \cdots \leq \tau_n$ such that request i becomes known at time $\tau_i - \Delta$, where $\Delta \geq 0$ is the predefined time-limit for the update of the current solution (we set $\Delta = 0.5$ minutes in our numerical experiments). Each request $i \in R$ has an associated pick-up location, denoted by i^+, and an associated drop-off location, referred to as i^-. Let $P := \{1^+, \ldots, n^+\}$ denote the set of all pick-up locations and let $D := \{1^-, \ldots, n^-\}$ denote the corresponding set of drop-off locations. Moreover, a number of requested seats $q_i \geq 1$ and a service time of $s_i \geq 0$ minutes (needed to enter or leave the vehicle) is associated with each request $i \in R$. To simplify the notation, we set $q_{i^+} = q_{i^-} := q_i$, $s_{i^+} = s_{i^-} := s_i$ and $q_0 := 0$ as well as $s_0 := 0$. The direct travel time from pick-up location i^+ to drop-off location i^- of request i is denoted by t_i. The maximum acceptable ride time of each request $i \in R$ is bounded from above by L_i. For each request, a pick-up time window $[e_{i^+}, \ell_{i^+}]$ is constructed, where the lower bound equals the desired pick-up time specified by the user.

The drop-off time window $[e_{i^-}, \ell_{i^-}]$ can be computed from the pick-up time window using $e_{i^-} = e_{i^+} + s_{i^+} + t_i$ and $\ell_{i^-} = \ell_{i^+} + s_{i^+} + L_i$. We assume that there is a fixed duration of service T, resulting in a time window $[e_0, \ell_0]$ associated with the depot, where e_0 denotes the start and $\ell_0 := e_0 + T$ denotes the end of service. Every user that is accepted is communicated a pick-up time Γ_i. This time may not be postponed by more than γ minutes.

Due to the dynamic nature of the problem, at any time τ only the requests that have arrived up to time τ are known. In addition, some requests might have been rejected and some of the accepted requests might already have been delivered to their drop-off location at time τ. Therefore, at any time τ, only a subproblem $\mathrm{DARP}(\tau)$ related to the *active requests* $\mathcal{A}(\tau)$ at time τ needs to be considered which comprises all requests that are known but neither rejected nor dropped-off w.r.t. the current solution $\boldsymbol{x}(\tau)$. To distinguish between these different types of requests at a given time τ, let

- $\mathcal{N}(\tau)$ denote the subset of *new requests* that were revealed at time $\tau - \Delta$,
- $\mathcal{S}(\tau)$ denote the subset of *scheduled requests*, i.e. requests that have been accepted but have not been picked-up up to time τ,
- $\mathcal{P}(\tau)$ denote the subset of *picked-up requests* that have not been dropped-off up to time τ,
- $\mathcal{D}(\tau)$ denote the subset of *dropped-off requests* up to time τ and
- $\mathcal{R}(\tau)$ denote the subset of *rejected requests* up to time τ.

Then $\mathcal{A}(\tau) = \mathcal{N}(\tau) \cup \mathcal{S}(\tau) \cup \mathcal{P}(\tau)$ while $\mathcal{D}(\tau), \mathcal{R}(\tau) \not\subseteq \mathcal{A}(\tau)$. Note that the sets $\mathcal{S}(\tau), \mathcal{P}(\tau),$ $\mathcal{D}(\tau), \mathcal{R}(\tau)$ do in fact not only depend on the time τ but also on the solutions determined in previous time steps. Each feasible solution $\boldsymbol{x}(\tau)$ to a subproblem $\mathrm{DARP}(\tau)$ consists of at most K vehicle routes which start and end at the depot. If a user is served by a vehicle, the user has to be picked-up and dropped-off by the same vehicle. On the other hand, a rejected user may not be picked-up or dropped-off by any of the vehicles.

A solution to the dynamic DARP is a strategy that, every time one or more new requests are revealed, modifies the solution of the last subproblem so that each of the new requests is either assigned to a vehicle route or rejected. In the course of assigning new requests to already existing vehicle routes, old requests, if not yet picked-up or dropped-off, might have to be reassigned. However, every request, once accepted, has to be served and every request, once rejected, cannot be served by any vehicle in the following subproblems.

The solution approach we propose in this paper is based on an *event-based MILP* formulation for the static DARP, see [9], which efficiently generates exact solutions to small to medium sized static benchmark problems in a few seconds. The idea of a solution strategy for the dynamic DARP is as follows: 1. An initial solution is obtained by solving the event-based MILP for the requests that are revealed at time $\tau_1 - \Delta$, which is interpreted as the time when the routes are initialized. 2. When new requests arrive at time $\tau_i - \Delta$, $i \geq 2$, the respective users are notified within 30 seconds whether they have been accepted or rejected. Therefore, the vehicle routes up to time τ_i are frozen and the set of active requests $\mathcal{A}(\tau_i)$ is updated. The underlying *event-based graph* is modified by removing all nodes and arcs corresponding to rejected requests and partially removing nodes and arcs corresponding to dropped-off or picked-up users. Nodes and arcs for the new requests are added to the event-based graph. Then the MILP is updated and solved again.

3   Event-Based Graph Model for a Rolling-Horizon

The MILP model for the static DARP suggested in [9] is based on the identification of *events* that represent pick-up or drop-off situations, and of their chronology. It was motivated by the work of Bertsimas et al. [5].

Each event is associated with a Q-tuple that represents a feasible allocation of users to a vehicle with capacity Q. For example, the tuple $(2^+, 5, 3)$ represents an event where user 2 has just been picked-up by a vehicle with capactiy $Q = 3$ and where users 3 and 5 are seated in the vehicle. The first entry of such a Q-tuple always contains the information on the last pick-up location (i^+) or drop-off location (i^-) while all remaining entries of the Q-tuple, representing the remaining users in the vehicle, are sorted in descending order of their respective indices (request numbers). Empty seats are identified by zero entries, and the depot is represented by the node $\mathbf{0} := (0, \ldots, 0)$.

While this formulation of DARP usually requires a large number of events (and hence variables in the associated MILP model), its strength is that feasibility constraints can be easily represented by an associated event-based graph $G = (V, A)$. The node set V of G represents all feasible events, and directed edges in A indicate all possible event sequences. Infeasible user allocations can already be identified (and omitted from V) when generating events, and directed edges between events are introduced if the corresponding event sequence is feasible. Then, every dicycle flow, i.e. every directed circuit, in G represents one vehicle's tour.

In order to extend this concept to the dynamic DARP, we assume that solutions are extended iteratively whenever new requests arrive and introduce a *dynamic event-based graph* $G(\tau) = (V(\tau), A(\tau))$ for the subprobem DARP(τ) at time τ. When new requests are revealed at time $\tau_i - \Delta$, $i \in \{1, \ldots, n\}$, then the event-based graph $G(\tau_i)$ is updated based on the event-based graph $G(\tau_{i-1})$ and the associated solution $\boldsymbol{x}(\tau_{i-1})$ of the last subproblem: Nodes and arcs corresponding to rejected, dropped-off and picked-up users are (partially) removed from the graph while nodes and arcs corresponding to new requests are added.

The node set $V(\tau)$ represents events which are feasible w.r.t. the vehicle capacity Q and also reflect time window and ride time constraints. More precisely, given requests $i, j \in \mathcal{A}(\tau)$, let $f_{i,j}^1, f_{i,j}^2 \in \{0, 1\}$ indicate the feasibility of the paths $j^+ \to i^+ \to j^- \to i^-$ and $j^+ \to i^+ \to i^- \to j^-$, respectively, w.r.t. time window and ride time constraints. By going through all pairs of requests, feasible combinations of users in vehicles (and hence in events in $V(\tau)$) can be easily identified, see [7]. To simplify the notation we set $f_{i,0}^1 = f_{i,0}^2 = f_{0,i}^1 = f_{0,i}^2 = 1$. We now formally define the node set of $G(\tau)$: The set of nodes representing an event in which a user $i \in \mathcal{A}(\tau) \setminus \mathcal{P}(\tau)$ is picked up is called the set of *pick-up nodes* up to time τ and is given by

$$
V_{i^+}(\tau) := \Bigg\{ (v_1, v_2, \ldots, v_Q) \colon v_1 = i^+, \ v_j \in \mathcal{A}(\tau) \cup \{0\} \setminus \{i\}, \ f_{i,v_j}^1 + f_{i,v_j}^2 \geq 1
$$

$$
\forall j \in \{2, \ldots, Q\}, \ (v_j > v_{j+1} \lor v_{j+1} = 0) \ \forall j \in \{2, \ldots, Q-1\}, \ \sum_{j=1}^{Q} q_{v_j} \leq Q \Bigg\}. \quad (1)
$$

Similarly, the set of *drop-off nodes* up to time τ corresponds to events where a user $i \in \mathcal{A}(\tau)$ is dropped off and is given by

$$
V_{i^-}(\tau) := \Bigg\{ (v_1, v_2, \ldots, v_Q) \colon v_1 = i^-, \ v_j \in \mathcal{A}(\tau) \cup \{0\} \setminus \{i\}, \ f_{v_j,i}^1 + f_{i,v_j}^2 \geq 1
$$

$$
\forall j \in \{2, \ldots, Q\}, \ (v_j > v_{j+1} \lor v_{j+1} = 0) \ \forall j \in \{2, \ldots, Q-1\}, \ \sum_{j=1}^{Q} q_{v_j} \leq Q \Bigg\}. \quad (2)
$$

We emphasize that one unique (pick-up or drop-off) location is associated with each event through the identification of the user that is picked up or dropped-off in this particular event. Note also that from the set of all pick-up and drop-off nodes associated with an accepted

user, exactly one pick-up and one drop-off node are contained in the dicycle flow representing the vehicle tour to which the user is assigned in the current solution. The set of nodes $V_{\mathcal{A}(\tau)}$ corresponding to the set of active requests $\mathcal{A}(\tau)$ is given by

$$V_{\mathcal{A}(\tau)} = V_0 \cup \bigcup_{i \in \mathcal{A}(\tau) \setminus \mathcal{P}(\tau)} V_{i^+}(\tau) \cup \bigcup_{i \in \mathcal{A}(\tau)} V_{i^-}(\tau),$$

where the set $V_0 := \{\mathbf{0}\}$ contains only the depot node. Simply put, $V_{\mathcal{A}(\tau)}$ represents the set of nodes that are available at time τ but have not been reached by any vehicle (yet). This set does not include nodes (and hence events) corresponding to users that have been rejected or dropped-off up to time τ since $\mathcal{D}(\tau), \mathcal{R}(\tau) \not\subseteq \mathcal{A}(\tau)$. Moreover, pick-up nodes corresponding to users $\mathcal{P}(\tau)$ are not considered since they have already been reached by a vehicle, where the user has been picked-up. Nodes where a pick-up or drop-off has already been realized up to time τ are referred to as *realized nodes*. As a consequence, each request that is known at time $\tau - \Delta$ falls in one of the following three categories:

- If $i \in \mathcal{N}(\tau) \cup \mathcal{S}(\tau) \cup \mathcal{R}(\tau)$, then no associated node (event) is realized since request i was either rejected or the scheduled pick-up and drop-off times are larger than τ.
- If $i \in \mathcal{P}(\tau)$, then exactly one associated node (event) is a realized node, which is a pick-up node.
- If $i \in \mathcal{D}(\tau)$, then exactly one associated pick-up node (event) and one associated drop-off node (event) is realized.

Let $V_{\mathcal{D}(\tau)}^{\text{realized}}$ denote the set of all realized pick-up and drop-off nodes for each user $i \in \mathcal{D}(\tau)$ and let $V_{\mathcal{P}(\tau)}^{\text{realized}}$ denote the set of all realized pick-up nodes associated with each user $i \in \mathcal{P}(\tau)$. Then the node set $V(\tau)$ is defined as

$$V(\tau) := V_{\mathcal{A}(\tau)} \cup V_{\mathcal{D}(\tau)}^{\text{realized}} \cup V_{\mathcal{P}(\tau)}^{\text{realized}}.$$

Hence, for a user $i \in \mathcal{D}(\tau)$ that has been dropped-off up to time τ only the unique realized pick-up and drop-off nodes are contained in $V(\tau)$, i.e., $V_{i^+}(\tau) := \{v \in V_{\mathcal{D}(\tau)}^{\text{realized}} : v_1 = i^+\}$ and $V_{i^-}(\tau) := \{v \in V_{\mathcal{D}(\tau)}^{\text{realized}} : v_1 = i^-\}$. Analoguously, for a picked-up user $i \in \mathcal{P}(\tau)$ only the unique realized pick-up node is contained in $V(\tau)$, i.e., $V_{i^+}(\tau) := \{v \in V_{\mathcal{P}(\tau)}^{\text{realized}} : v_1 = i^+\}$.

Similar to the node set $V(\tau)$, the arc set $A(\tau)$ of $G(\tau)$ has to reflect the fact that some routing decisions have already been fixed up to time τ in the rolling-horizon framework. This motivates the introduction of the concept of *realized arcs*: Each realized pick-up and drop-off node $v \in V_{\mathcal{D}(\tau)}^{\text{realized}} \cup V_{\mathcal{P}(\tau)}^{\text{realized}}$ is contained in a dicycle flow representing a vehicle's tour. The incoming arc of a realized node, which is part of this dicycle flow, is referred to as *realized arc*. We denote the set of realized arcs by $A^{\text{realized}}(\tau)$. Let $v \in V_{\mathcal{D}(\tau)}^{\text{realized}} \cup V_{\mathcal{P}(\tau)}^{\text{realized}}$ be chosen such that there is no arc $a = (v, w) \in A^{\text{realized}}(\tau)$. Thus, v is the last realized node in the corresponding dicycle flow at time τ. Such nodes indicate the last realized stop on the current tour, from which on the solution may be modified if this is advantageous given the newly revealed requests. We denote the set of "last realized nodes" as $V^{\text{l-realized}}(\tau)$. Then, the arc set $A(\tau)$ is composed of seven subsets that will be further specified below:

$$A(\tau) = \bigcup_{k=1}^{6} A_k(\tau) \cup A^{\text{realized}}(\tau).$$

As in the static case, c.f. [9], $A(\tau)$ represents the set of transits from one event node to another. Let i and j be requests that have been revealed up to time $\tau - \Delta$. Then the six subsets $A_k(\tau)$, $k = 1, \ldots, 6$ are defined as follows:

- The first set $A_1(\tau)$ describes the transit from a pick-up node from a set $V_{i^+}(\tau)$ to a drop-off node from a set V_{j^-}:

$$A_1(\tau) := \Big\{ \big((i^+, v_2, \ldots, v_Q), (j^-, w_2, \ldots, w_Q) \big) \in \big(V_{\mathcal{A}(\tau)} \cup V^{\text{l-realized}}(\tau) \big) \times V_{\mathcal{A}(\tau)} :$$
$$\{j, w_2, \ldots, w_Q\} = \{i, v_2, \ldots, v_Q\} \Big\}.$$

- The transit from a pick-up node from a set $V_{i^+}(\tau)$ to another pick-up node from a set $V_{j^+}(\tau)$ with $j \neq i$ is represented by the following set:

$$A_2(\tau) := \Big\{ \big((i^+, v_2, \ldots, v_{Q-1}, 0), (j^+, w_2, \ldots, w_Q) \big) \in \big(V_{\mathcal{A}(\tau)} \cup V^{\text{l-realized}}(\tau) \big) \times V_{\mathcal{A}(\tau)} :$$
$$\{i, v_2, \ldots, v_{Q-1}\} = \{w_2, \ldots, w_Q\} \Big\}.$$

- $A_3(\tau)$ is comprised of arcs which describe the transit from a drop-off node in a set $V_{i^-}(\tau)$ to a pick-up node in a set $V_{j^+}(\tau)$, $j \neq i$:

$$A_3(\tau) := \Big\{ \big((i^-, v_2, \ldots, v_Q), (j^+, v_2, \ldots, v_Q) \big) \in \big(V_{\mathcal{A}(\tau)} \cup V^{\text{l-realized}}(\tau) \big) \times V_{\mathcal{A}(\tau)} : i \neq j \Big\}.$$

- The transit from a drop-off node from a set $V_{i^-}(\tau)$ to another drop-off node from a set $V_{j^-}(\tau)$, $j \neq i$, is represented by:

$$A_4(\tau) := \Big\{ \big((i^-, v_2, \ldots, v_Q), (j^-, w_2, \ldots, w_{Q-1}, 0) \big) \in \big(V_{\mathcal{A}(\tau)} \cup V^{\text{l-realized}}(\tau) \big) \times V_{\mathcal{A}(\tau)} :$$
$$\{v_2, \ldots, v_Q\} = \{j, w_2, \ldots, w_{Q-1}\} \Big\}.$$

- A dicycle in $G(\tau)$ representing a vehicle tour always contains an arc describing the transit from the depot to a pick-up node in a set $V_{i^+}(\tau)$, as well as an arc describing the transit from a drop-off node from a set $V_{j^-}(\tau)$ to the depot. The following two sets describe these transitions:

$$A_5(\tau) := \Big\{ \big((0, \ldots, 0), (i^+, 0, \ldots, 0) \big) \in V_0 \times V_{\mathcal{A}(\tau)} \Big\},$$

$$A_6(\tau) := \Big\{ \big((j^-, 0, \ldots, 0), (0, \ldots, 0) \big) \in \big(V_{\mathcal{A}(\tau)} \cup V^{\text{l-realized}}(\tau) \big) \times V_0 \Big\}.$$

▶ **Example 1.** We give an example of the changes in the event-based graph for three requests and one vehicle with capacity $Q = 3$. Let $R = \{1, 2, 3\}$. The request data is as follows:

i	q_i	τ_i	$[e_{i^+}, \ell_{i^+}]$	$[e_{i^-}, \ell_{i^-}]$
1	1	5	$[10, 25]$	$[15, 40]$
2	2	15	$[20, 35]$	$[30, 50]$
3	2	45	$[50, 65]$	$[55, 80]$

For the sake of clarity, we assume that all requests are accepted. Furthermore, we assume that the remaining parameters (e.g. travel times) allow all variants of routing described in the following, but are omitted in this example.

When the first request is revealed, we have $\mathcal{A}(\tau_1) = \mathcal{N}(\tau_1) = \{1\}$ and $\mathcal{S}(\tau_1) = \mathcal{P}(\tau_1) = \mathcal{D}(\tau_1) = \emptyset$. The initial graph $G(\tau_1)$ is depicted in Figure 1a. We assume that by the time request 2 is revealed, user 1 has not been picked-up yet, i.e. $\mathcal{N}(\tau_2) = \{2\}$, $\mathcal{S}(\tau_2) = \{1\}$, $\mathcal{A}(\tau_2) = \{1, 2\}$ and $\mathcal{P}(\tau_2) = \mathcal{D}(\tau_2) = \emptyset$. Therefore, we only have to add additional nodes and arcs induced by request 2 as illustrated in $G(\tau_2)$ in Figure 1b. According to the time

(a) Graph $G(\tau_1)$.

(b) Graph $G(\tau_2)$.

(c) Graph $G(\tau_3)$.
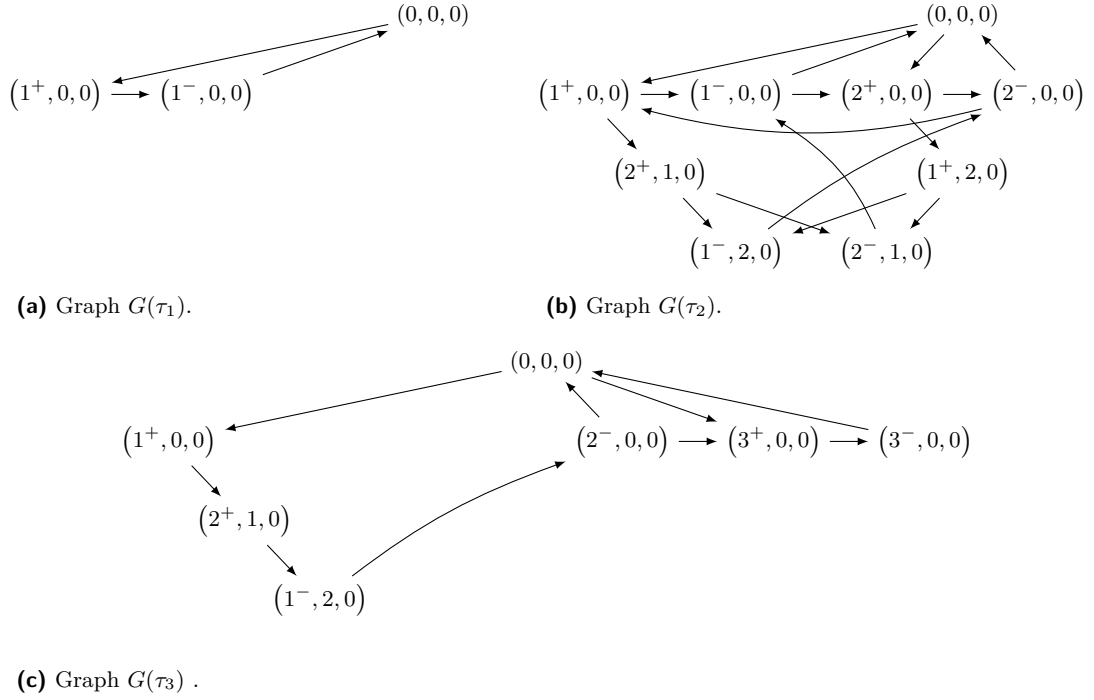
■ **Figure 1** Evolution of the dynamic event-based graph for an instance with three requests.

windows, by the time request 3 is revealed user 1 must have been dropped-off and user 2 must have been picked-up. We assume that user 2 has not been dropped-off yet and that the vehicle tour induced by the current solution is given by the dicycle

$$C_1 = \Big\{ (0,0,0),\ (1^+,0,0),\ (2^+,1,0),\ (1^-,2,0),\ (2^-,0,0),\ (0,0,0) \Big\}.$$

Hence, $\mathcal{N}(\tau_3) = 3$, $\mathcal{A}(\tau_3) = \{2,3\}$, $\mathcal{P}(\tau_3) = \{2\}$, $\mathcal{D}(\tau_3) = \{1\}$ and $\mathcal{S}(\tau_3) = \emptyset$. The corresponding realized nodes are $V_{\mathcal{P}(\tau_3)}^{\text{realized}} = \{(2^+,1,0)\}$ and $V_{\mathcal{D}(\tau_3)}^{\text{realized}} = \{(1^+,0,0),(1^-,2,0)\}$. The set of realized arcs is $A^{\text{realized}}(\tau_3) = \{((0,0,0),(1^+,0,0)),((1^+,0,0),(2^+,1,0)),((2^+,1,0),(1^-,2,0))\}$ and $V^{\text{l-realized}}(\tau_3) = \{(1^-,2,0)\}$. The update of the event-based graph to obtain $G(\tau_3)$ is illustrated in Figure 1c. Note that there are no nodes $v \in V(\tau_3)$ that simultaneously contain users 1 (i.e., 1^+ or 1^-) and 3 (i.e., 3^+ or 3^-) as $1 \notin \mathcal{A}(\tau_3)$, which means that according to equations (1) and (2) there are no shared nodes. Similarly, the seats requested by users 2 and 3 combined exceed the vehicle capacity of three.

4   Event-Based MILP for a Rolling-Horizon

Based upon the event-based graph model we update and solve an MILP problem in a rolling-horizon strategy whenever new requests arrive, that is, at times $\tau = \tau_j$ for $j = 1, \ldots, n$. Every subproblem $\text{DARP}(\tau)$ can be modeled as a variant of a minimum cost flow problem with additional constraints in the dynamic event-based graph $G(\tau) = (V(\tau), A(\tau))$.

For the MILP formulation of $\text{DARP}(\tau)$ we use the following additional parameters and variables:

Since every node in the dynamic event-based graph $G(\tau) = (V(\tau), A(\tau))$ corresponds to a uniquely determined geographical location, we can associate routing costs $c_a \geq 0$ and a travel times $t_a \geq 0$ with the respective arcs $a \in A(\tau)$ in $G(\tau)$. Let $\delta^{\text{in}}(v,\tau)$ and $\delta^{\text{out}}(v,\tau)$ denote

the set of incoming and outgoing arcs of v, respectively. A solution of $\mathrm{DARP}(\tau)$ is denoted by $\boldsymbol{x}(\tau)$ and is composed of the following variables: The binary variables x_a with $a \in A(\tau)$ are equal to one if and only if arc $a \in A(\tau)$ is used by a vehicle. A feasible tour of a vehicle is then represented by a dicycle C in the dynamic event-based graph $G(\tau)$ such that $x_a = 1$ for all $a \in C$. Note that due to the structure of the event-based graph, the pick-up and drop-off node of any user are contained in the same dicycle C, representing the assignment of the user to the respective vehicle. If a vehicle has reached the last drop-off location in the dicycle representing its route, it will wait at its current location for new requests until it has to start its journey back to the depot to arrive there before the end of service ℓ_0. Since requests might be rejected, we introduce a binary variable p_i for each $i \in \mathcal{A}(\tau) \setminus \mathcal{P}(\tau)$ with $p_i = 1$ indicating that request i is accepted. To model the beginning of service at a node $v \in V(\tau)$, i.e. the time at which a vehicle arrives at the location represented by v to pick-up or drop-off passengers, we use continuous variables B_v. The continuous variables d_i, $i \in \mathcal{A}(\tau)$ measure a user's excess ride time compared to his or her earliest drop-off time.

The parameters x_a^{old} and B_v^{old} are used to store the values of the variables x_a and B_v from the previous iteration in the rolling-horizon framework. Once a vehicle has departed from a location, we cannot divert it from its next destination (as this brings technical difficulties related to the calculation of distances, see [3]). Also, if an arc has been realized up to time τ, it has to be included in a dicycle flow in all later subproblems. Therefore, if $\tau > \tau_1$ then all partial routes up to time τ and hence all variables x_a corresponding to the set

$$A^{\mathrm{fixed}}(\tau) := \{(v,w) \in A(\tau) \colon x_{(v,w)}^{\mathrm{old}} = 1, \tau \geq B_w^{\mathrm{old}} - t_{(v,w)}\}$$

are fixed in the MILP corresponding to the current subproblem $\mathrm{DARP}(\tau)$. The set of realized arcs $A^{\mathrm{realized}}(\tau)$ is a subset of the set of fixed arcs $A^{\mathrm{fixed}}(\tau)$. We set $A^{\mathrm{fixed}}(\tau_1) = \emptyset$. Furthermore, let $A^{\mathrm{new}}(\tau)$ be the set of all arcs that have not been contained in the graph corresponding to the previous subproblem. We have $A^{\mathrm{new}}(\tau_1) = A(\tau_1)$.

For the remainder of this section, let $j \in \{1, \dots, n\}$ be arbitrary but fixed. To prepare the MILP formulation of $\mathrm{DARP}(\tau_j)$, we define a set of travel time constraints $(C_{v,w}(\tau_j))$ for all $(v,w) \in A^{\mathrm{new}}(\tau_j) \setminus \delta^{\mathrm{out}}(\mathbf{0}, \tau_j)$:

$$B_w \geq \max\{B_v, \tau_j\} + s_{v_1} + t_{(v,w)} - M_{v,w}(\tau_j) \cdot (1 - x_{(v,w)}), \qquad (C_{v,w}(\tau_j))$$

$$\text{where} \quad M_{v,w}(\tau_j) \geq \begin{cases} \ell_{v_1} - e_{w_1} + s_{v_1} + t_{(v,w)} & \text{if } B_v \geq \tau_j \\ \tau_j - e_{w_1} + s_{v_1} + t_{(v,w)} & \text{otherwise} \end{cases}$$

is a sufficiently large constant. The constraints $(C_{v,w}(\tau_j))$ guarantee that for all arcs $(v,w) \in A^{\mathrm{new}}(\tau_j) \setminus \delta^{\mathrm{out}}(\mathbf{0}, \tau_j)$ the beginning of service at a node w is greater than or equal to the earliest departure time at a preceding node v plus the time needed to travel from node v to node w. If $(v,w) \in A^{\mathrm{new}}(\tau_j) \setminus \delta^{\mathrm{out}}(\mathbf{0}, \tau_j)$, then the arc (v,w) is related to a new request that has been revealed at time $\tau_j - \Delta$. This implies that travel from v to w can start no earlier than $\max\{B_v, \tau_j\} + s_{v_1}$. Note that in this case constraint $(C_{v,w}(\tau_j))$ can be linearized by rewriting it using two constraints where $\max\{B_v, \tau_j\}$ is once replaced by B_v and once by τ_j. We are now ready to formulate the event-based $\mathrm{MILP}(\tau_j)$ for each subproblem $\mathrm{DARP}(\tau_j)$.

Event-Based MILP(τ_j) for a Rolling-Horizon.

$$\min \omega_1 \sum_{a \in A(\tau_j)} c_a\, x_a + \omega_2 \sum_{i \in \mathcal{A}(\tau_j) \setminus \mathcal{P}(\tau_j)} (1 - p_i) + \omega_3 \sum_{i \in \mathcal{A}(\tau_j)} d_i, \qquad (3\mathrm{a})$$

$$\text{s.t.} \sum_{a \in \delta^{\mathrm{in}}(v, \tau_j)} x_a - \sum_{a \in \delta^{\mathrm{out}}(v, \tau_j)} x_a = 0 \qquad \forall v \in V(\tau_j), \qquad (3\mathrm{b})$$

$$\sum_{\substack{a\in\delta^{\mathrm{in}}(v,\tau_j)\\v\in V_{i+}}} x_a = p_i \qquad \forall i \in \mathcal{A}(\tau_j)\setminus\mathcal{P}(\tau_j), \tag{3c}$$

$$\sum_{a\in\delta^{\mathrm{out}}(\mathbf{0},\tau_j)} x_a \leq K, \tag{3d}$$

$$e_0 \leq B_{\mathbf{0}} \leq \ell_0, \tag{3e}$$

$$e_{i+} + (\ell_{i+}-e_{i+})\left(1 - \sum_{a\in\delta^{\mathrm{in}}(v,\tau_j)} x_a\right) \leq B_v \leq \ell_{i+} \quad \forall i \in \mathcal{A}(\tau_j)\setminus\mathcal{P}(\tau_j),\ v \in V_{i+}(\tau_j), \tag{3f}$$

$$e_{i-} \leq B_v \leq e_{i+} + L_i + s_{i+} + (\ell_{i+}-e_{i+})\sum_{a\in\delta^{\mathrm{in}}(v,\tau_j)} x_a \quad \forall i \in \mathcal{A}(\tau_j),\ v \in V_{i-}(\tau_j), \tag{3g}$$

$$B_v \leq \ell_{i+}\left(1 - \sum_{a\in\delta^{\mathrm{in}}(v,\tau_j)} x_a\right) + (\Gamma_i+\gamma)\sum_{a\in\delta^{\mathrm{in}}(v,\tau_j)} x_a \qquad \forall i \in \mathcal{S}(\tau_j), \forall v \in V_{i+}(\tau_j), \tag{3h}$$

$$B_w - B_v - s_{i+} \leq L_i \qquad \forall i \in \mathcal{A}(\tau_j),\ v \in V_{i+}(\tau_j),\ w \in V_{i-}(\tau_j), \tag{3i}$$

$$B_w \geq \tau_j + t_{(v,w)}x_{(v,w)} \qquad \forall(v,w) \in \delta^{\mathrm{out}}(\mathbf{0},\tau_j)\setminus A^{\mathrm{fixed}}(\tau_j), \tag{3j}$$

$$(C_{v,w}(\tau_k)) \qquad \forall(v,w) \in A^{\mathrm{new}}(\tau_k)\setminus\delta^{\mathrm{out}}(\mathbf{0},\tau_k),\ \forall k=1,\dots,j, \tag{3k}$$

$$d_i \geq B_v - e_{i-} \qquad \forall i \in \mathcal{A}(\tau_j),\ \forall v \in V_{i-}(\tau_j), \tag{3l}$$

$$p_i = 1 \quad \forall i \in \mathcal{S}(\tau_j), \tag{3m}$$

$$x_{(v,w)} = 1,\ B_w = B_w^{\mathrm{old}} \qquad \forall(v,w) \in A^{\mathrm{fixed}}(\tau_j), \tag{3n}$$

$$p_i \in \{0,1\} \qquad \forall i \in \mathcal{A}(\tau_j)\setminus\mathcal{P}(\tau_j), \qquad d_i \geq 0 \quad \forall i \in \mathcal{A}(\tau_j), \tag{3o}$$

$$x_a \in \{0,1\} \quad \forall a \in A(\tau_j), \qquad\qquad B_v \geq 0 \quad \forall v \in V(\tau_j). \tag{3p}$$

The objective function (3a) minimizes the total routing cost, the total excess ride time and the number of unaccepted requests, where $\omega_1, \omega_2, \omega_3 > 0$ are weighting parameters that can be adapted to represent the respective importance of these optimization criteria. The flow conservation constraints (3b) ensure that only dicycle flows in $G(\tau_j)$ are feasible. Every accepted user has to be picked-up at one of its pick-up nodes by exactly one vehicle (3c). Constraint (3d) is a capacity constraint on the number of vehicles. The constraints (3e)–(3g) are time-window constraints for the vehicles to arrive at events (nodes). Constraints (3h) guarantee that the start of service at a pick-up node of a user $i \in \mathcal{S}(\tau_j)$ which has not been picked-up yet, is not later than the pick-up time Γ_i communicated to the user plus an additional constant γ. Furthermore, the maximum ride time of a user is bounded by constraint (3i), while constraints (3j)–(3k) model the travel-time from node to node. Constraints (3l) measure a user's excess ride time. The constraints (3m) ensure that a request is contained in a vehicle's route if and only if it is accepted (indicated by $p_i = 1$). Finally, constraints (3n) ensure that the next solution respects the partial routes up to time τ_j, including the scheduled service times that are inherited from the previous iteration. Vehicle capacity, pairing and precedence constraints are ensured by the structure of the event-based graph. Furthermore, it guranatees that picked-up users will not be relocated to any other vehicle and that they will eventually be dropped-off. Note that requests that have been accepted but have not been picked-up or dropped-off yet may be assigned to other vehicles in the next iteration.

5   A Rolling-Horizon Algorithm

We now present the essential aspects of the rolling-horizon algorithm. The approach is based on iteratively updating the dynamic event-based graph whenever new requests arrive, given the information obtained from the previous solution. Then the corresponding MILP is resolved. For each new request we have to determine whether it can be feasibly integrated into the existing schedule. If this is possible, then a schedule including the new request that minimizes routing costs and excess ride time is computed. We impose a time limit of 30 seconds to decide how to process new requests. If the solution returned by the MILP solver is not yet known to be optimal due to this time limit, then the solution is reoptimized in the next iteration. Note that this reoptimization can only consider variables that have not yet been fixed due to the advanced time. In the following, let δ be a timer that ensures this time limit by measuring the time in minutes needed to execute lines 4–8 in Algorithm 1.

Algorithm 1 Rolling-horizon algorithm for dynamic DARP.

---
1 $(x, B, p, d) = \text{solve}(\text{MILP}(\tau_1))$
2 **for** $i = 2 \ldots n$ **do**   `// new requests `$\mathcal{N}(\tau_i)$` are revealed`
3 $\quad$ Start timer $\delta = 0$
4 $\quad$ Determine $\mathcal{D}(\tau_i)$, $\mathcal{R}(\tau_i)$ and $\mathcal{P}(\tau_i)$
5 $\quad$ $\mathcal{A}(\tau_i) = \mathcal{A}(\tau_{i-1}) \cup \mathcal{N}(\tau_i) \setminus (\mathcal{D}(\tau_i) \cup \mathcal{R}(\tau_i))$
6 $\quad$ Compute dynamic event-based graph $G(\tau_i)$
7 $\quad$ Determine set of fixed arcs $A^{\text{fixed}}(\tau_i)$ $\quad$ `// fix partial routes up to `τ_i
8 $\quad$ $(x, B, p, d) = \text{solve}(\text{MILP}(\tau_i))$ and stop prematurely when $\delta = \Delta$
9 $\quad$ **foreach** request $i \in \mathcal{N}(\tau_i)$ **do**
10 $\quad\quad$ **if** $p_i = 1$ **then**
11 $\quad\quad\quad$ accept request i
12 $\quad\quad$ **else**
13 $\quad\quad\quad$ reject request i

An initial feasible solution containing the initial requests is obtained by solving $\text{MILP}(\tau_1)$. Every time one or more new requests are revealed at times τ_i, $i \in \{2, \ldots, n\}$, the set of active requests is updated as $\mathcal{A}(\tau_i) = \mathcal{A}(\tau_{i-1}) \cup \mathcal{N}(\tau_i) \setminus (\mathcal{D}(\tau_i) \cup \mathcal{R}(\tau_i))$ and the dynamic event-based graph corresponding to the current time τ_i is computed. Note that we do not have to recompute the whole graph in each iteration: All not realized pick-up and drop-off nodes (up to time τ_i) corresponding to dropped-off and denied users and all not realized pick-up nodes (up to time τ_i) corresponding to picked-up users are removed from the graph together with all incident arcs. On the other hand, new nodes and arcs corresponding to new requests are added to the graph and the MILP is updated accordingly. To assure that vehicle routes computed for the current subproblem $\text{DARP}(\tau_i)$ are consistent with the routes that have been executed up to time $\tau_i - \Delta$, the corresponding variables have to be fixed up to time τ_i before solving the next subproblem $\text{MILP}(\tau_i)$.

6   Computational Results

In this section we assess the performance of Algorithm 1 based on real data from Hol-Mich-App, a dial-a-ride service in the city of Wuppertal launched in 2020. We use two instances that differ w.r.t. the length of the planning horizon and the number of requests. *Su_8_22* is an instance with $n = 254$ transportation requests based on accumulated data

from nine consecutive Sundays in January and February 2021 with service hours from 8 a.m. until 10 p.m., i.e. $T = 840$ minutes. Sa_6_3 consists of $n = 519$ requests and is based on accumulated data from nine consecutive Saturdays in January and February 2021 with service hours from 6 a.m. until 3 a.m. the next morning, i.e. $T = 1260$ minutes. Note that due to the Covid-19 pandemic the demand for ridepooling services was rather low and hence we accumulated requests to obtain realistic instances. Moreover, the ridepooling cabs which are equipped with six seats were not allowed to transport more than three passengers at a time, i.e., $Q = 3$. We used linear regression to approximate unknown travel times from distances and from the known travel times between the pick-up and drop-off locations of the requests. More precisely, the costs c_a were computed in an OpenStreetMap network of Wuppertal using OSMnx[6], a Python API to OpenStreetMap, and all unknown travel times t_a were computed from the regression line $t_a = 1.8246\,c_a + 2.369$. The length of the pick-up time window for each user is 25 minutes, and the lower bound of the pick-up time window is equal to the time when the transportation request was submitted plus the response time of the algorithm, i.e. $e_i = \tau_i$. Moreover, the maximum ride time of request i is equal to $t_i + \max(10, 0.75\,t_i)$ minutes. The service time for every request is set to 0.75 minutes and the number of requested seats varies from one to three, i.e. $q_i \in \{1, 2, 3\}$. The drop-off time window is computed based on the pick-up time window, the direct travel time, the maximum ride time and the service time. The maximum delay of communicated pick-up time is set to $\gamma = 5$ minutes. After some preliminary testing, the parameters in the objective function (3a) are set to $\omega_1 = 1$, $\omega_2 = 60$ and $\omega_3 = 0.1$. Due to the accumulation of request data, we were not given a fixed number of vehicles by the service provider. An evolution of the number of requests during service hours is depicted in Figure 2 in the appendix. In the peak hour, there are 51 requests in instance Sa_6_3 and 32 requests in instance Su_8_22. The average length of a direct trip, i.e. driving from pick-up to drop-off location without any additional stops, in both instances is 8.4 minutes. In our tests we evaluate different fleet sizes and solve instance Sa_6_3 with $K \in \{12, 14, 16\}$ and instance Su_8_22 with $K \in \{6, 8, 10\}$ vehicles. Algorithm 1 was implemented in C++ and all computations were carried out on an Intel Core i7-8700 CPU, 3.20 GHz, 32GB memory using CPLEX 12.10. The computational results can be found in Table 1. For all instances we report the following average values per accepted request: the routing costs (C), the excess ride time in minutes (E), the waiting time from the time of submitting the request until the time of pick-up in minutes (W), the trip length in minutes (TL), the average time to answer a new request in seconds (A), the percentage of requests that are rejected (R), and the number of times CPLEX was terminated prematurely due to a timeout (CT). Furthermore, we listed the average detour factor (DF), the mean occupancy (MO), the percentage of empty mileage (EM) and the system efficiency (SE), which are measures to evaluate the operational efficiency of ridepooling systems. The computation is based on [15] and can be found in Section C.

The results confirm that Algorithm 1 can quickly answer and schedule new requests. No CPLEX timeouts occured in any run of a Su_8_22 instance. Thus, all 254 requests are either inserted optimally in the given schedule, given the solution of the preceding iteration, or they are rejected due to infeasibility or inacceptable costs. For the larger Sa_6_3 instances very few timeouts occured, and CPLEX terminated prematurely only one or two times out of the 404 iterations[7]. This affected the insertion of five out of 519 requests. The relative MIP gap in these iterations ranged from 0.4% to 0.5%. Moreover, a reoptimization was necessary only in

---

[6] https://github.com/gboeing/osmnx
[7] There are less than 519 iterations since several requests are revealed at the same time.

Table 1 Computational results for instances from Hol-mich-App.

Instance	K	C	E	W	TL	DF	MO	EM	SE	A	R	CT
Sa_6_3	12	4.4	12.2	9.7	11.6	1.1	1.6	0.3	1.0	2.9	3.5	1
Sa_6_3	14	4.4	12.2	9.6	11.7	1.1	1.6	0.3	1.0	2.8	3.3	2
Sa_6_3	16	4.4	11.8	9.4	11.5	1.1	1.5	0.3	1.0	2.7	3.1	1
Su_8_22	6	4.7	15.9	13.0	12.0	1.2	1.5	0.3	0.9	0.5	3.5	0
Su_8_22	8	4.6	12.3	10.0	11.5	1.1	1.5	0.3	0.9	0.4	1.6	0
Su_8_22	10	4.6	11.8	9.7	11.4	1.1	1.5	0.3	0.9	0.3	1.6	0

0.5% of the iterations, which implies that only a very low percentage of requests was rejected while there would have been a feasible and profitable insertion position. From comparing the results for *Su_8_22* and *Sa_6_3* for the different fleet sizes, it becomes evident that by the use of additional vehicles the average routing costs, the average excess ride time, the average waiting time and the average trip length (except *Sa_6_3* with $K = 14$) per accepted user decrease or remain constant. The average detour factor, the mean occupancy, the percentage of empty mileage and the system efficiency remain (nearly) constant for the different values of K, while the percentage of rejected requests decreases with an increasing number of vehicles. The average time to answer new requests ranges from 2.7 to 2.9 seconds (*Sa_6_3*) and 0.3 to 0.5 seconds (*Su_8_22*) on average, demonstrating that Algorithm 1 is stable under different vehicle configurations.

7   Conclusions

We present a rolling-horizon approach for the solution of the dynamic dial-a-ride-problem that is based on adaptively updating an event-based MILP formulation. Numerical experiments on medium-sized instances from a recently established ridepooling service in the city of Wuppertal confirm the efficiency and reliability of this approach. By adapting the weighting parameters in the objective function, different preferences w.r.t. service cost and customer satisfaction can be implemented. The approach can also be used to assess the quality gain when increasing the fleet size or when changing other parameters in the model.

References

1   Andrea Attanasio, Jean-François Cordeau, Gianpaolo Ghiani, and Gilbert Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, 2004. `doi:10.1016/j.parco.2003.12.001`.

2   Alexandre Beaudry, Gilbert Laporte, Teresa Melo, and Stefan Nickel. Dynamic transportation of patients in hospitals. *OR Spectrum*, 32(1):77–107, 2008. `doi:10.1007/s00291-008-0135-6`.

3   Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010. `doi:10.1016/j.ejor.2009.04.024`.

4   Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem. *INFORMS Journal on Computing*, 24(3):343–355, 2012. `doi:10.1287/ijoc.1110.0454`.

5   Dimitris Bertsimas, Patrick Jaillet, and Sébastien Martin. Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, 67(1):143–162, 2019. `doi:10.1287/opre.2018.1763`.

6   Pasquale Carotenuto and Fabio Martis. A double dynamic fast algorithm to solve multi-vehicle dial a ride problem. *Transportation Research Procedia*, 27:632–639, 2017. `doi:10.1016/j.trpro.2017.12.131`.

7   Jean-François Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006. `doi:10.1287/opre.1060.0283`.

8   Luca Coslovich, Raffaele Pesenti, and Walter Ukovich. A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*, 175(3):1605–1615, 2006. `doi:10.1016/j.ejor.2005.02.038`.

9   Daniela Gaul, Kathrin Klamroth, and Michael Stiglmayr. Event-based MILP models for ride-hailing applications. *arXiv*, 2021. submitted to European Journal of Operations Research. `arXiv:2103.01817`.

10   Fred Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1-3):223–253, 1996. `doi:10.1016/0166-218x(94)00037-e`.

11   Thomas Hanne, Teresa Melo, and Stefan Nickel. Bringing robustness to patient flow management through optimized patient transports in hospitals. *Interfaces*, 39(3):241–255, 2009. `doi:10.1287/inte.1080.0379`.

12   Sin C. Ho, W.Y. Szeto, Yong-Hong Kuo, Janny M.Y. Leung, Matthew Petering, and Terence W.H. Tou. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421, 2018. `doi:10.1016/j.trb.2018.02.001`.

13   Carl H. Häll and Anders Peterson. Improving paratransit scheduling using ruin and recreate methods. *Transportation Planning and Technology*, 36(4):377–393, 2013. `doi:10.1080/03081060.2013.798488`.

14   Lauri Häme and Harri Hakula. A maximum cluster algorithm for checking the feasibility of dial-a-ride instances. *Transportation Science*, 49(2):295–310, 2015. `doi:10.1287/trsc.2013.0495`.

15   Christian Liebchen, Martin Lehnert, Christian Mehlert, and Martin Schiefelbusch. Betriebliche Effizienzgrößen für Ridepooling-Systeme. In *Making Connected Mobility Work*, pages 135–150. Springer Fachmedien Wiesbaden, 2021. `doi:10.1007/978-3-658-32266-3_7`.

16   Athanasios Lois and Athanasios Ziliaskopoulos. Online algorithm for dynamic dial a ride problem and its metrics. *Transportation Research Procedia*, 24:377–384, 2017. `doi:10.1016/j.trpro.2017.05.097`.

17   Ying Luo and Paul Schonfeld. Online rejected-reinsertion heuristics for dynamic multivehicle dial-a-ride problem. *Transportation Research Record: Journal of the Transportation Research Board*, 2218(1):59–67, 2011. `doi:10.3141/2218-07`.

18   Oli B. G. Madsen, Hans F. Ravn, and Jens Moberg Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60(1):193–208, 1995. `doi:10.1007/bf02031946`.

19   Nikola Marković, Rahul Nair, Paul Schonfeld, Elise Miller-Hooks, and Matthew Mohebbi. Optimizing dial-a-ride services in maryland: Benefits of computerized routing and scheduling. *Transportation Research Part C: Emerging Technologies*, 55:156–165, 2015. `doi:10.1016/j.trc.2015.01.011`.

20   Harilaos N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980. `doi:10.1287/trsc.14.2.130`.

21   Douglas O. Santos and Eduardo C. Xavier. Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications*, 42(19):6728–6737, 2015. `doi:10.1016/j.eswa.2015.04.060`.

22   S. Vallee, A. Oulamara, and W. Ramdane Cherif-Khettaf. New online reinsertion approaches for a dynamic dial-a-ride problem. *Journal of Computational Science*, 47:101199, 2020. `doi:10.1016/j.jocs.2020.101199`.

A   Parameters and Variables

■ **Table 2** List of parameters.

Parameter	Description
n	number of transport requests
R	set of transport requests
i^+, i^-	pick-up and drop-off location of request i
P, D	set of pick-up and set of drop-off locations
Δ	time allowed to communicate an answer to new requests
$\tau_i - \Delta$	time at which request i is revealed
τ	current time
$\mathcal{A}(\tau)$	set of active requests for subproblem DARP(τ)
$\mathcal{N}(\tau)$	new requests revealed at $\tau - \Delta$
$\mathcal{S}(\tau)$, $\mathcal{P}(\tau)$, $\mathcal{D}(\tau)$, $\mathcal{R}(\tau)$	subsets of R of scheduled, picked-up, dropped-off and rejected requests up to time τ
K	fleet of vehicles
Q	vehicle capacity
q_i	load associated with request i
s_i	service duration associated with request i
$[e_j, \ell_j]$	time window associated with request location j
T	maximum duration of service
t_i	direct travel time from pick-up loaction i^+ to drop-off location i^-
L_i	maximum ride time associated with request i
Γ_i	pick-up time communicated to user i
γ	maximum delay of communicated pick-up time
$f_{i,j}^1$, $f_{i,j}^2$	feasibility of paths $j^+ \to i^+ \to j^- \to i^-$ and $j^+ \to i^+ \to i^- \to j^-$
$G(\tau) = (V(\tau), A(\tau))$	event-based graph corresponding to subproblem DARP(τ)
$V_{i+}(\tau)$, $V_{i-}(\tau)$	set of pick-up nodes and set of drop-off nodes corresponding to request i and DARP(τ)
$V_{\mathcal{A}(\tau)}$	set of nodes corresponding to active requests $\mathcal{A}(\tau)$ and DARP(τ)
$V_{\mathcal{D}(\tau)}^{\text{realized}}$, $V_{\mathcal{P}(\tau)}^{\text{realized}}$	set of realized drop-off and set of realized pick-up nodes corresponding to DARP(τ)
$V^{\text{l-realized}}(\tau)$	set of last realized nodes corresponding to DARP(τ)
$A^{\text{realized}}(\tau)$	set of realized arcs corresponding to DARP(τ)
$A^{\text{fixed}}(\tau)$	set of fixed arcs corresponding to DARP(τ)
$A^{\text{new}}(\tau)$	set of arcs that have not been contained in the arc set of the last subproblem
c_a, t_a	routing cost and travel time on arc a
$\delta^{\text{in}}(v, \tau)$, $\delta^{\text{out}}(v, \tau)$	incoming arcs and outgoing arcs of node v corresponding to DARP(τ)
x_a^{old}, B_v^{old}	value of variables x_a and B_v obtained from last subproblem solved
ω_1, ω_2, ω_3	weighting parameters
δ	timer in minutes to measure time while executing Algorithm 1

Table 3 List of variables.

Variable	Description
p_i	binary variable indicating if user i is transported or not
B_v	continuous variable indicating the start of service time at node v
x_a	binary variable indicating if arc a is used or not
d_i	continuous variable indicating the excess ride time of user i w.r.t. e_{i-}

B   Additional Data to Computational Results
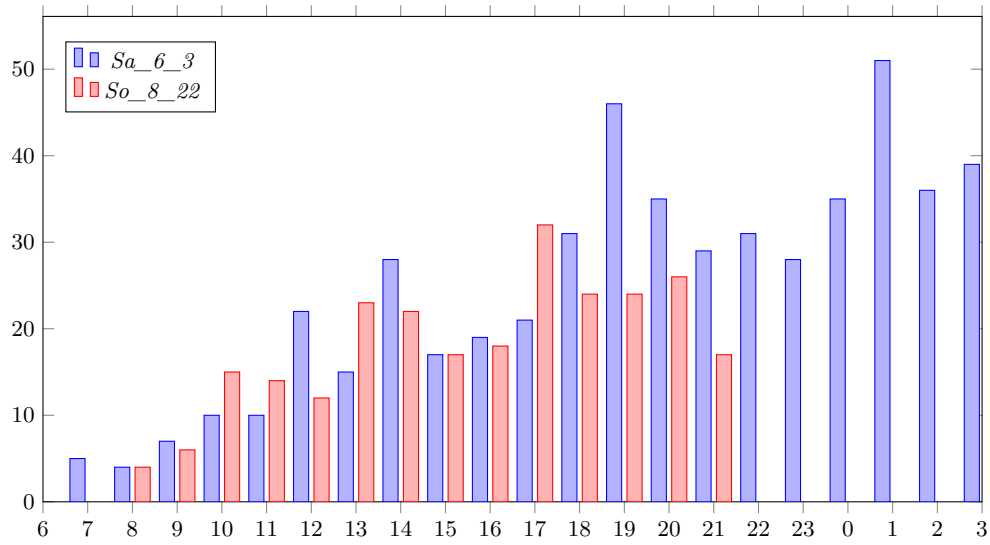
Figure 2 Evolution of number of requests during service hours.

C   Measurung the Operational Efficiency of Ridepooling Systems

The computation of the following efficiency measures are based on [15].

$$\text{average detour factor} = \frac{\text{passenger kilometers driven}}{\text{passenger kilometers booked}}$$

$$\text{mean occupancy} = \frac{\text{passenger kilometers driven}}{\text{vehicle kilometers occupied}}$$

$$\text{percentage of empty mileage} = \frac{\text{empty mileage}}{\text{total vehicle kilometers}}$$

$$\text{system efficiency} = \frac{\text{mean occupancy} \cdot (1 - \text{ percentage of empty mileage})}{\text{average detour factor}}$$

Solving the Periodic Scheduling Problem: An Assignment Approach in Non-Periodic Networks

Vera Grafe ✉
Technische Universität Kaiserslautern, Germany

Anita Schöbel ✉
Technische Universität Kaiserslautern, Germany
Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM, Kaiserslautern, Germany

──── **Abstract** ────

The periodic event scheduling problem (PESP) is a well researched problem used for finding good periodic timetables in public transport. While it is based on a periodic network consisting of events and activities which are repeated every period, we propose a new periodic timetabling model using a *non-periodic* network. This is a first step towards the goal of integrating periodic timetabling with other planning steps taking place in the aperiodic network, e.g. passenger assignment or delay management. In this paper, we develop the new model, show how we can reduce its size and prove its equivalence to PESP. We also conduct computational experiments on close-to real-world data from Lower Saxony, a region in northern Germany, and see that the model can be solved in a reasonable amount of time.

1 Introduction

An important aspect of optimising public transport is finding a good periodic timetable. From the passengers' point of view, short travelling times are desirable, which can be achieved by making the timetable as tight as possible. This problem is known as the *Periodic event Scheduling Problem (PESP)* and is well researched. It uses a periodic event-activity network in which each node represents many arrivals or departures, namely one per period. In this paper we develop a new model for the PESP in a (larger) aperiodic network. We first give our motivation why such a model is needed.

Tight periodic timetables minimise travelling times, but are very prone to delays which are inevitable in reality and highly dissatisfactory for the passengers. Hence, apart from short travelling times, a good timetable should also have some degree of delay resistance. Many concepts and ideas on how to increase robustness of a timetable against delays exist, see [15]. However, none of these approaches uses the promising concept of *recoverable robustness* introduced by [13]. The aim is to find a periodic timetable with small travelling times such that in every delay scenario from a given set it is possible to find a disposition timetable which fulfils some quality criteria. To this end, we have to integrate timetabling and delay management. Timetables are determined in a periodic network, but delay management is done in an aperiodic network, since in general delays do not occur periodically. In order to integrate delay management into timetabling, we hence have to find a way to solve both problems in the same network. The same holds for integrating passengers' assignment since also the demand does not occur periodically.

One way for such an integration is to develop a timetabling model which computes a periodic timetable in an aperiodic network, which is the goal of this paper. We call the new model ***P**eriodic **T**imetabling in **A**periodic Network* (PTTA).

Periodic timetabling is well studied in the literature. The PESP was first introduced in [27]. It aims at finding a feasible periodic timetable. Instead of only considering the feasibility problem, one can also consider different objective functions. In [17] this was done by minimising the waiting times of the transferring passengers. An alternative formulation which can be solved much faster uses cycle bases, see, e.g. [17, 9, 21]. The problem was solved with a branch-and-bound approach in [16] and with a genetic algorithm in [19]. The modulo simplex [18, 7] and a fast matching approach [20] are more recent heuristics for solving PESP. An approach running several solution methods in parallel was presented in [2]. Computing a periodic timetable in an aperiodic network was already considered in [28]. As opposed to our model, in [28] the decision on which transfer activities are needed is not part of the optimisation process but is fixed before by a simple heuristic rule. In [1] the problem is considered only for a single train line between two stations. A model putting an emphasis on passenger satisfaction and including the passenger routing is proposed in [22]. It uses the assumption that all drive and dwell times are fixed and does not consider track safety constraints. For a survey on timetabling we refer to [3, 6].

The remainder of this paper is structured as follows: The PESP is briefly reviewed in Section 2. In Section 3 we introduce the new timetabling model and make several modifications to the model such that it better meets our needs. In Section 4 we compare the new model PTTA2 to the established model PESP and show that they are equivalent. We present some computational results in Section 5 and conclude the paper with some final remarks and suggestions for further research in Section 6.

2   The Periodic Event Scheduling Problem

A model often used for periodic timetabling is the Periodic Event Scheduling Problem (PESP), which was introduced in [27]. In the PESP we are given a period T together with a set of events $\underline{\mathcal{E}}$, which either correspond to the arrival or the departure of a traffic line at some station. Furthermore, we have activities $\underline{\mathcal{A}}$, which represent processes between the events. Together, we obtain an *event-activity-network* (EAN) $\mathcal{N} = (\underline{\mathcal{E}}, \underline{\mathcal{A}})$ in which the events are represented as nodes and the activities as arcs. We distinguish several different types of activities. *Driving activities* model a train line driving from one station to another, while *waiting activities* represent a line waiting at a station. Passengers have the possibility to transfer between different lines, which is included by the *transfer activities*. If a line has a frequency higher than one, i.e. the line is served several times in one period, we want to spread the rides equally over the period. This is done by *synchronisation activities*. *Headway activities* are used to model safety regulations requiring a minimal distance between two consecutive departures or arrivals, or the safety restriction on single-track lines. They usually come in pairs, since it is not clear beforehand in which order the two departures will take place. Given an EAN $\mathcal{N} = (\underline{\mathcal{E}}, \underline{\mathcal{A}})$, we want to find a periodic timetable with period T, which is a mapping $\pi \colon \underline{\mathcal{E}} \to \{0, \dots, T-1\}$ assigning a time to every event. To simplify notation we set $\pi_i := \pi(i)$ for $i \in \underline{\mathcal{E}}$. For every activity $a \in \underline{\mathcal{A}}$ a lower bound $L_a \in \mathbb{N}$ and an upper bound $U_a \in \mathbb{N}$ are given. L_a is the minimal time necessary to perform the activity a, while U_a is the maximal time allowed for a. A timetable is *feasible* if it respects the bounds on the activities, i.e. for every activity $a = (i, j) \in \underline{\mathcal{A}}$ we require $\pi_j - \pi_i + z_a T \in [L_a, U_a]$ for some $z_a \in \mathbb{Z}$. The modulo parameter z_a takes the periodicity into account.
The PESP asks for a feasible timetable. In timetabling we additionally want to minimise the total travelling time summed over all passengers. For $a \in \underline{\mathcal{A}}$ let $w_a \in \mathbb{N}$ be the number of passengers using activity a. The following is the basic IP formulation for PESP:

$$\min \sum_{a=(i,j)\in\underline{\mathcal{A}}} w_a \cdot (\pi_j - \pi_i + z_a T) \qquad\qquad\qquad \text{(PESP)}$$

$$\pi_j - \pi_i + z_a T \leq U_a \qquad\qquad a = (i,j) \in \underline{\mathcal{A}} \qquad (1)$$

$$\pi_j - \pi_i + z_a T \geq L_a \qquad\qquad a = (i,j) \in \underline{\mathcal{A}} \qquad (2)$$

$$\pi_i \in \{0, \dots, T-1\} \qquad\qquad\qquad i \in \underline{\mathcal{E}} \qquad (3)$$

$$z_a \in \mathbb{Z} \qquad\qquad\qquad\qquad a \in \underline{\mathcal{A}}. \qquad (4)$$

Details about periodic timetabling can be found in the literature on PESP, a good introduction is given in [12, 17].

3 A New Timetabling Model

As mentioned before, we want to compute a timetable in an aperiodic EAN. While in a periodic EAN the events represent the arrivals or departures of a *line* at some station (for unit line frequencies), in an aperiodic EAN they model the arrival or departure of a single *trip*. A trip is the journey of a vehicle from the beginning of a line to its end, i.e. one line can yield several trips (based on the number of periods and the frequency of the line). Hence, instead of only considering the lines, we consider all trips of the lines separately. This means we have to "roll out" the periodic EAN to an aperiodic one in a time interval $[t_{\min}, t_{\max}]$, a procedure which is also used in delay management, where a timetable is given and used for rolling out. Since we want to determine the timetable, we cannot use this roll-out procedure. Nevertheless, we first repeat how the roll-out is done for a given timetable (based on [14]) and then explain our procedure which leaves the timetable open.

Rolling out with a given timetable. For every $i \in \underline{\mathcal{E}}$ set

$$\pi_{\mathrm{first}}(i) := \min\{\pi_i + kT : \pi_i + kT \geq t_{\min}, k \in \mathbb{Z}\},$$

$$\pi_{\mathrm{last}}(i) := \max\{\pi_i + kT : \pi_i + kT \leq t_{\max}, k \in \mathbb{Z}\}.$$

These are the first respectively last times the event i occurs in the considered time horizon. The roll-out process then works as follows:

- For every $i \in \underline{\mathcal{E}}$ and $1 \leq s \leq K_i := \left\lfloor \frac{\pi_{\mathrm{last}}(i) - \pi_{\mathrm{first}}(i)}{T} \right\rfloor + 1$ construct an aperiodic event i_s with $\pi_{i_s} = \pi_{\mathrm{first}}(i) + (s-1)T$. Let $\mathcal{E}(i) := \{i_s : 1 \leq s \leq K_i\}$ be the set of aperiodic events corresponding to the periodic event i.
- For every $a = (i,j) \in \underline{\mathcal{A}} \backslash \underline{\mathcal{A}}_{\mathrm{head}}$ (where $\underline{\mathcal{A}}_{\mathrm{head}}$ is the set of headway activities) and $i_s \in \mathcal{E}(i)$ determine $j_t \in \mathcal{E}(j)$ (if it exists) such that $L_a \leq \pi_{j_t} - \pi_{i_s} \leq U_a$. We create an aperiodic activity $a_{st} = (i_s, j_t)$ and set $L_{a_{st}} = L_a$, $U_{a_{st}} = U_a$ and $w_{a_{st}} = w_a$. For each pair $a = (i,j), a' = (j,i) \in \underline{\mathcal{A}}_{\mathrm{head}}$ of headway activities and $s \in \mathcal{E}(i), t \in \mathcal{E}(j)$ create two aperiodic activities $a_{st} = (i_s, j_t), a_{ts} = (j_t, i_s)$ with $L_{a_{st}} = L_a$ and $L_{a_{ts}} = T - U_a$. If j_t does not exist we are at the end of $[t_{\min}, t_{\max}]$ and nothing has to be done.

Note that in [14] the activities in the rolled out network do not have upper bounds, since these are ignored in delay management. Since we do timetabling, we want to respect the upper bounds and add them also in the rolled out EAN. Another particularity are the headway activities which ensure a security distance between two consecutive departures. Since it is not clear which of the two events will take place first, they come in pairs. For

every pair of these headway arcs a_{st}, a_{ts} exactly one of them is chosen for which the lower bound has to be respected, i.e. the pair $a = (i,j), a' = (j,i) \in \underline{\mathcal{A}}_{\text{head}}$ yields the following constraints:

For all $1 \le s \le K_i, 1 \le t \le K_j$ either $\pi_{j_t} - \pi_{i_s} \ge H_{ij}$ or $\pi_{i_s} - \pi_{j_t} \ge H_{ji}$,

where $H_{ij} = L_a$, $H_{ji} = L_{a'}$. For further details, we refer to [14]. (Note that the problem can be interpreted as a resource-constrained machine scheduling problem, see, e.g., [4, 26]). A common assumption is that

$$0 \le L_a \le T - 1 \text{ and } L_a \le U_a \le L_a + T - 1 \text{ for all } a \in \mathcal{A}. \tag{5}$$

In this case, the j_t in the roll-out process is uniquely determined, if it exists. If we do not use this assumption, we may have to choose one of several possible j_t. We will later introduce a rule how to make this choice, but for now it is enough to choose an arbitrary one.

The goal of this paper is to compute the timetable in the rolled out EAN. Hence, we cannot use the timetable when rolling out. However, the timetable information is important for determining the activities between the correct arrival and departure events. This is shown in Figure 1 where in (c) and (d) two different timetables are used for the roll-out leading to two different aperiodic networks. Since we do not know beforehand which activities will be needed for the optimal timetable, we allow all possibilities (see part (b) of Figure 1) and leave it to the optimization to choose the correct activities together with the optimal timetable.

We hence adapt the procedure in the following way.

Rolling out without knowing the timetable.
- For every periodic event $i \in \underline{\mathcal{E}}$ and $1 \le s \le K := \lfloor \frac{t_{\max} - t_{\min}}{T} \rfloor + 1$ create an aperiodic event i_s. Let $\mathcal{E}(i) := \{i_s : 1 \le s \le K\}$ be the set of all aperiodic events corresponding to i. The set of all events is $\mathcal{E} := \cup_{i \in \underline{\mathcal{E}}} \mathcal{E}(i)$.
- For every periodic activity $a = (i,j) \in \underline{\mathcal{A}} \backslash \underline{\mathcal{A}}_{\text{head}}$, for exactly one arc $a = (i,j)$ of every pair of headway activities and for every $1 \le s, t \le K$ create a *possible (aperiodic) activity* a_{st} with $L_{a_{st}} = L_a$, $U_{a_{st}} = U_a$ and $w_{a_{st}} = w_a$. Let $\mathcal{A}(a) := \{a_{st} = (i_s, j_t) : 1 \le s, t \le K\}$ be the set of possible activities corresponding to a. The set of all possible activities is

$$\mathcal{A} := \bigcup_{a \in \underline{\mathcal{A}}} \mathcal{A}(a). \tag{6}$$

The final network $(\mathcal{E}, \mathcal{A})$ is called the *rolled out network*.

We remark that when rolling out with a timetable, the number K_i of aperiodic events corresponding to a periodic event i depends on i. This is not the case when rolling out without knowing the timetable, where we have a constant K. However, this only makes a difference if our planning horizon $[t_{\min}, t_{\max}]$ covers a fractional number of periods. E.g. if we consider 3.5 periods, some events will take place three times and some four times. Since this depends on the timetable, we cannot make this distinction when rolling out without knowing the timetable, where we have to consider each event four times. If we assume that we only consider whole periods, K_i is constant for all $i \in \underline{\mathcal{E}}$ and thus both procedures yield the same number of events.

The rolled out network contains not only the actual activities, but all possibilities for the activities. Thus, when fixing the timetable we have to simultaneously solve an assignment problem: for each periodic activity we have to choose exactly one of the corresponding arcs in every considered period. In order to do so we introduce a binary variable

$$u_a = \begin{cases} 1 & \text{if } a \text{ is chosen,} \\ 0 & \text{otherwise.} \end{cases}$$
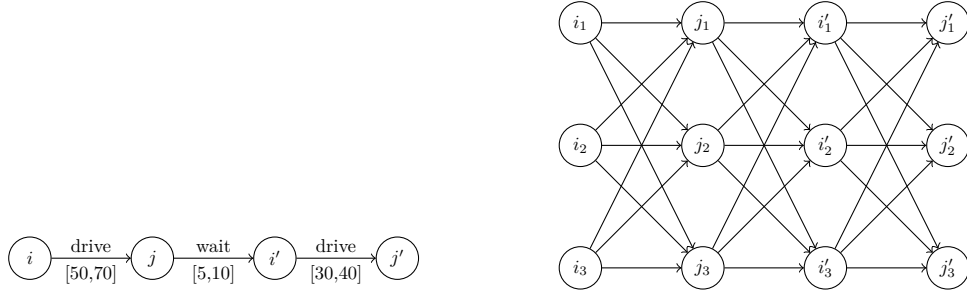
(a) Periodic EAN with $[L_a, U_a]$ given below the arcs.

(b) EAN rolled out with all possible activities.
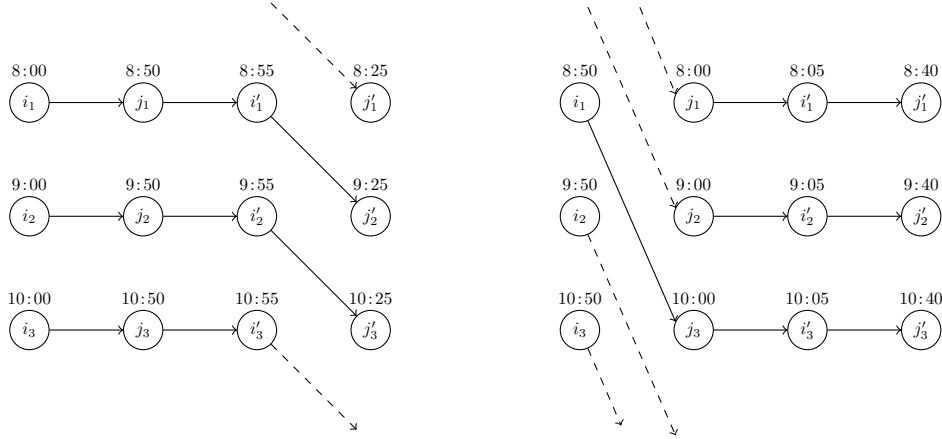
(c) Rolled out EAN after choosing a feasible timetable and the corresponding activities. The dashed arcs indicate the connections entering or leaving the planning horizon.

(d) Rolled out EAN with another feasible timetable, which results in different activities.

Figure 1 Rolling out a periodic EAN without knowing the timetable for the time interval $[8:00,10:59]$ with $T = 60$, i.e. $K = 3$.

Furthermore, for $a \in \mathcal{A}$ we set $b_a := \left\lceil \frac{U_a}{T} \right\rceil$ and $K_a := K - b_a$. It will become clear later why we need this notation. Below we give the first idea of the constraints we need. The first correct formulation will be given in PTTA1.

$$\min \sum_{a=(i_s,j_t)\in\mathcal{A}} w_a \cdot u_a(\pi_{j_t} - \pi_{i_s}) \tag{7}$$

$$\text{s.t. } \pi_{j_t} - \pi_{i_s} + M(u_a - 1) \leq U_a \qquad a = (i_s, j_t) \in \mathcal{A} \tag{8}$$

$$\pi_{j_t} - \pi_{i_s} + M(1 - u_a) \geq L_a \qquad a = (i_s, j_t) \in \mathcal{A} \tag{9}$$

$$\pi_{i_s} - \pi_{i_{s-1}} = T \qquad i_s \in \mathcal{E}, 2 \leq s \leq K \tag{10}$$

$$\sum_{t:a'=(i_s,j_t)\in\mathcal{A}} u_{a'} = 1 \qquad a = (i,j) \in \underline{\mathcal{A}}, 1 \leq s \leq K_a \tag{11}$$

$$\pi_i \geq t_{\min} \qquad i \in \mathcal{E} \tag{12}$$

$$\pi_{i_1} \leq t_{\min} + T - 1 \qquad i \in \underline{\mathcal{E}} \tag{13}$$

$$\pi_i \in \mathbb{N} \qquad i \in \mathcal{E} \tag{14}$$

$$u_a \in \{0, 1\} \qquad a \in \mathcal{A}. \tag{15}$$

The objective function minimises the total travelling time over all passengers. In the case that an activity a is chosen, i.e. $u_a = 1$, constraints (8) and (9) ensure that the upper and lower bounds for this activity are respected. If a is not selected, the constraints become redundant for appropriately chosen M. Constraints (10) are called *synchronisation constraints* and ensure that the timetable has period T. For every periodic activity the assignment constraint (11) chooses exactly one of the corresponding aperiodic activities in every period in such a way that it fits to the timetable constraints (8) and (9). For the last b_a periods it is possible that no feasible choice exists and hence, we omit the constraints for these periods. We will later explain why this problem cannot occur for the other periods. Constraints (12) and (13) enforce that no event is scheduled earlier than $t_{\min}$ and that the first event takes place in the first period we consider. Finally, to ensure that at least one period is considered in constraints (11) we assume $b_a < K$ for all $a \in \mathcal{A}$, i.e. the planning horizon is sufficiently large.

Can we disregard $s > K_a$ in the assignment constraints?

As mentioned above, if we have a timetable π and an $s > K_a$ there may be no t such that $\pi_{j_t} - \pi_{i_s} \in [L_a, U_a]$, since the time of the event we would theoretically have to choose exceeds the planning horizon, as already seen in Figure 1 for the dashed arcs. Hence, we disregard the last b_a periods for the assignment. We will show in Lemma 3 that this indeed does not exclude optimal solutions.

However, disregarding the last b_a periods causes a problem with the objective function. Since setting $u_a = 1$ increases the objective value and we minimise, for every $a = (i, j) \in \mathcal{A}, s > K_a$ we will always have $u_{(i_s, j_t)} = 0$ for every t in an optimal solution. Hence, the passengers in the last b_a periods are (falsely) not considered. Fortunately, we can use the following trick to overcome this problem: Due to periodicity the contribution to the objective function of these passengers is the same as in all other periods. This means that we can correct this mistake in the objective function by replacing it by

$$\min \sum_{a=(i_1, j_t) \in \mathcal{A}} w_a \cdot u_a (\pi_{j_t} - \pi_{i_1}) \cdot K.$$

We obtain the following formulation:

$$\min \sum_{a=(i_1, j_t) \in \mathcal{A}} w_a \cdot u_a (\pi_{j_t} - \pi_{i_1}) \cdot K \qquad \text{(PTTA1)}$$
$$\text{s.t. } (8) - (15)$$

Analysis of the headway constraints

Note that when rolling out with a timetable we handled the headway activities differently than when rolling out without knowing a timetable. For the PESP it is known that even without knowing the order of the events, one headway constraint suffices to cover a pair of headway activities. This is also true in our case, i.e. both ways of handling the headways are equivalent. The proof can be found in the appendix.

▶ **Lemma 1.** *Let $a = (i, j), a' = (j, i) \in \mathcal{A}_{\text{head}}$. The following statements are equivalent:*
(a) *For all $1 \leq s, t \leq K$ we have either $\pi_{j_t} - \pi_{i_s} \geq L_a = H_{ij}$ or $\pi_{i_s} - \pi_{j_t} \geq L_{a'} = H_{ji}$.*
(b) *For all $1 \leq s \leq K_a$ there is some $1 \leq t \leq K$ such that $\pi_{j_t} - \pi_{i_s} \in [L_a, U_a] = [H_{ij}, T - H_{ji}]$.*

In the following, for simplicity, we will always handle the headways as given by the constraints in (b), regardless whether we roll out with or without using a given timetable. In the following we analyse and strengthen PTTA1.

Can we linearise the quadratic objective function?

This can be done using standard techniques. We introduce a new variable F_a for $a = (i_1, j_t) \in \mathcal{A}$ to obtain the following equivalent formulation:

$$\min \sum_{a=(i_1,j_t)\in\mathcal{A}} w_a F_a \cdot K \qquad \text{(PTTA2)}$$

$$\text{s.t. } (8) - (15)$$

$$F_a \geq M(u_a - 1) + \pi_{j_t} - \pi_{i_1} \qquad a = (i_1, j_t) \in \mathcal{A} \qquad (16)$$

$$F_a \in \mathbb{N} \qquad a = (i_1, j_t) \in \mathcal{A}. \qquad (17)$$

It is straightforward to prove that the linearisation is correct, i.e. PTTA1 and PTTA2 are equivalent.

How to choose M?

▶ **Lemma 2.** $M := t_{\max} + T - 1 + \max_{a\in\mathcal{A}} L_a$ *is sufficiently large.*

Proof. We have to show that for every $a = (i_s, j_t) \in \mathcal{A}$ the following inequalities hold:

- $M \geq \pi_{i_s} - \pi_{j_t} + L_a$
- $M \geq \pi_{j_t} - \pi_{i_s} - U_a$
- $M \geq \pi_{j_t} - \pi_{i_s} - F_a$

In order to see this we use the following observations. First, using constraints (10) inductively yields $\pi_{i_s} = \pi_{i_1} + (s-1)T$. Second, by constraints (13) we know that $\pi_{i_1} \leq t_{\min} + T - 1$. And finally, by choice of K we have $KT \leq t_{\max} - t_{\min} + T$. Putting all this together we obtain

$$\pi_{i_s} = \pi_{i_1} + (s-1)T \leq \pi_{i_1} + (K-1)T \leq t_{\min} + KT - 1 \leq t_{\max} + T - 1.$$

Thus, we have $\pi_{i_s} - \pi_{j_t} + L_a \leq \pi_{i_s} + L_a \leq M$, which shows the first inequality. Similarly, we obtain the other two. ◀

Reducing the number of variables and constraints

So far, we have considered every combination (i_s, j_t) for $(i, j) \in \underline{\mathcal{A}}$ and $1 \leq s, t \leq K$. However, for some of these we can show that they cannot be selected in a feasible solution.

▶ **Lemma 3.** *Let* $(i, j) \in \underline{\mathcal{A}}$ *and* $1 \leq s \leq K$. *Then for* $a = (i_s, j_t)$ *with* $t \geq s + 1 + b_a$ *or* $t \leq s - 1$ *we have* $u_a = 0$ *in any feasible solution.*

Proof. We have $t_{\min} \leq \pi_{i_1}, \pi_{j_1} \leq t_{\min} + T - 1$, which implies $1 - T \leq \pi_{j_1} - \pi_{i_1} \leq T - 1$. By periodicity we obtain for $t \geq s + 1 + b_a$:

$$\pi_{j_t} - \pi_{i_s} = (\pi_{j_1} + (t-1)T) - (\pi_{i_1} + (s-1)T) \geq 1 - T + (t-s)T$$
$$\geq 1 - T + (1 + b_a)T \geq 1 + U_a > U_a.$$

Similarly, for $t \leq s - 1$ we have:

$$
\begin{aligned}
\pi_{j_t} - \pi_{i_s} &= (\pi_{j_1} + (t-1)T) - (\pi_{i_1} + (s-1)T) \\
&\leq T - 1 + (t-s)T \leq T - 1 - T = -1 < L_a
\end{aligned}
$$

By constraints (8) and (9) it follows $u_a = 0$.   ◄

Hence, we only have to consider (i_s, j_t) for $s \leq t \leq s + b_a$. In particular, for $s \leq K_a$ we only have to consider $t \leq K$, i.e. all relevant j_t are in the planning horizon. We adapt $\mathcal{A}(a)$ in (6) and now use the smaller sets

$$
\mathcal{A}(a) := \{a_{st} = (i_s, j_t) : 1 \leq s \leq K, s \leq t \leq \min\{s + b_a, K\}\}. \tag{18}
$$

Note that this may be a significant reduction, e.g. under the assumption (5) we have $U_a \leq L_a + T - 1 \leq 2(T - 1)$ and hence $b_a \leq 2$.

We can reduce the activities we have to consider even further with the following reasoning: Because of the periodicity of the timetable, the choice of $u_{(i_1, j_t)}$ already determines the value of u for later periods. Hence, we only need to consider variables $u_{(i_1, j_t)} \in \mathcal{A}$ with i_1 being the event in the first period instead of $u_{(i_s, j_t)} \in \mathcal{A}$ for all i_s with $(i_s, i_t) \in \mathcal{A}$. This affects constraints (8), (9), (11), and (15) in PTTA2 and reduces the number of variables and constraints in our formulation considerably leading to the following IP. Note that we also use the reduced set $\mathcal{A}$ resulting from (18).

$$
\min \sum_{a = (i_1, j_t) \in \mathcal{A}} w_a F_a \cdot K \tag{PTTA3}
$$

$$
\pi_{j_t} - \pi_{i_1} + M(u_a - 1) \leq U_a \qquad\qquad a = (i_1, j_t) \in \mathcal{A} \tag{19}
$$

$$
\pi_{j_t} - \pi_{i_1} + M(1 - u_a) \geq L_a \qquad\qquad a = (i_1, j_t) \in \mathcal{A} \tag{20}
$$

$$
\pi_{i_s} - \pi_{i_{s-1}} = T \qquad\qquad i_s \in \mathcal{E}, 2 \leq s \leq K \tag{21}
$$

$$
\sum_{t : a = (i_1, j_t) \in \mathcal{A}} u_a = 1 \qquad\qquad (i, j) \in \underline{\mathcal{A}} \tag{22}
$$

$$
F_a \geq M(u_a - 1) + \pi_{j_t} - \pi_{i_1} \qquad\qquad a = (i_1, j_t) \in \mathcal{A} \tag{23}
$$

$$
\pi_i \geq t_{\min} \qquad\qquad i \in \mathcal{E} \tag{24}
$$

$$
\pi_{i_1} \leq t_{\min} + T - 1 \qquad\qquad i \in \underline{\mathcal{E}} \tag{25}
$$

$$
\pi_i \in \mathbb{N} \qquad\qquad i \in \mathcal{E} \tag{26}
$$

$$
u_a \in \{0, 1\} \qquad\qquad a = (i_1, j_t) \in \mathcal{A} \tag{27}
$$

$$
F_a \in \mathbb{N} \qquad\qquad a = (i_1, j_t) \in \mathcal{A}. \tag{28}
$$

▶ **Lemma 4.** *PTTA2 and PTTA3 are equivalent.*

The proof is in the appendix.

4   Comparison of PTTA2 and PESP

We now want to compare the new assignment-based model with the established model PESP. We consider the version PTTA2. Let an instance of PESP be given. We roll out the EAN without knowing a timetable. Suppose we can solve either PESP or PTTA2 quickly. Does this help to find a solution of the other problem? More precisely, we are interested in the following questions:

(a) Let $(\tilde{\pi}, z)$ be a feasible (optimal) solution for PESP. Can we use it to construct a feasible (optimal) solution for PTTA2?

(b) Let (π, u, F) be a feasible solution for PTTA2. Can we use it to construct a feasible (optimal) solution for PESP?

We start with (a). Let a periodic timetable be given. As an intermediate step we consider the roll-out w.r.t this timetable. The following lemma ensures that for any realization i_s of event i (except for those at the end of the planning horizon) we can choose a corresponding realization j_t feasible for the rolled out constraint (i_s, j_t).

▶ **Lemma 5.** *Let $(\tilde{\pi}, z)$ be a feasible solution for PESP and π the solution constructed in the roll-out process. Let $a = (i, j) \in \underline{A}$ and $k, l \in \mathbb{Z}$ such that $\pi_{\mathrm{first}}(i) = \tilde{\pi}_i + kT$ and $\pi_{\mathrm{first}}(j) = \tilde{\pi}_j + lT$. For any choice of $1 \leq s \leq K$ and $t := z_a + k - l + s$ with $t \leq K$, the bounds on activity (i_s, j_t) are fulfilled, i.e. $\pi_{j_t} - \pi_{i_s} \in [L_a, U_a]$.*

Proof. By definition of π we have $\pi_{i_s} = \pi_{\mathrm{first}}(i) + (s-1)T = \tilde{\pi}_i + (k+s-1)T$ and $\pi_{j_t} = \pi_{\mathrm{first}}(j) + (t-1)T = \tilde{\pi}_j + (l+t-1)T$. Hence, it follows

$$\pi_{j_t} - \pi_{i_s} = \tilde{\pi}_j - \tilde{\pi}_i + (l - k - s + t)T = \tilde{\pi}_j - \tilde{\pi}_i + z_a T \in [L_a, U_a]. \qquad \blacktriangleleft$$

▶ **Corollary 6.** *In the situation of Lemma 5 for $1 \leq s \leq K_a$ there exists an $s \leq t \leq s + b_a$ with $\pi_{j_t} - \pi_{i_s} \in [L_a, U_a]$.*

Proof. We remark that by Lemma 3 it follows that for t as chosen in Lemma 5 we have $s \leq t \leq s + b_a$. Since $s \leq K_a$, this implies $t \leq s + b_a \leq K_a + b_a = K$, so by Lemma 5 we obtain $\pi_{j_t} - \pi_{i_s} \in [L_a, U_a]$. $\qquad \blacktriangleleft$

As mentioned already for the roll-out process for a given timetable, the choice of t has not to be unique in the general case and we could choose one of the possibilities arbitrarily. From now on, we will choose t as in Lemma 5.

We can use these results to construct a solution for the rolled out network. Let an instance of PESP $(\underline{\mathcal{E}}, \underline{\mathcal{A}})$ be given and $(\mathcal{E}, \mathcal{A})$ be the EAN received by rolling out without knowing a solution. Let $(\tilde{\pi}, z)$ be a solution for PESP. We define π as in the roll-out process with the timetable given, i.e. $\pi_{i_s} = \pi_{\mathrm{first}}(i) + (s-1)T$. Furthermore, for $a = (i_s, j_t) \in \mathcal{A}$ we choose k, l as in Lemma 5 and set

$$u_a = \begin{cases} 1 & \text{if } t = z_a + k - l + s, \\ 0 & \text{otherwise,} \end{cases}$$

and for $a = (i_1, j_t)$ we set

$$F_a = \begin{cases} \pi_{j_t} - \pi_{i_1} & \text{if } u_a = 1, \\ 0 & \text{otherwise.} \end{cases}$$

This construction gives us a feasible solution for PTTA2 in the rolled out network as the following lemma shows. The proof can be found in the appendix.

▶ **Lemma 7.** *Let $(\tilde{\pi}, z)$ be a solution for PESP with objective value $\tilde{f}$. Then (π, u, F) as defined above is a feasible solution for PTTA2 and the corresponding objective value is $f = K\tilde{f}$.*

We now turn to (b). Again, let an instance of PESP $(\underline{\mathcal{E}}, \underline{\mathcal{A}})$ be given and $(\mathcal{E}, \mathcal{A})$ be the EAN received by rolling out without knowing a solution. Let (π, u, F) be a feasible solution to PTTA2. For $i \in \underline{\mathcal{E}}$ we set

$$\tilde{\pi}_i := \pi_{i_1} \mod T,$$

i.e. there is some $r_i \in \mathbb{Z}$ such that $\pi_{i_1} = \tilde{\pi}_i + r_i T$. For $a = (i, j) \in \underline{\mathcal{A}}$ there is some t such that $u_{(i_1, j_t)} = 1$. Set

$$z_a := r_j - r_i + t - 1.$$

Also this construction works, i.e. we get a feasible solution for PESP with bounded objective function value. Again, the proof can be found in the appendix.

▶ **Lemma 8.** *Let (π, u, F) be a feasible solution to PTTA2 with objective value f. Then $(\tilde{\pi}, z)$ as defined above is a feasible solution for PESP and for its objective value $\tilde{f}$ we have $\tilde{f} \leq f \cdot \frac{1}{K}$.*

Putting the two constructions together, we finally conclude that we can in fact construct an optimal solution for PESP if we know an optimal solution for PTTA2 and vice versa. In particular, it makes no difference whether one computes a solution with PTTA2 or rolls out a solution obtained with PESP, i.e. in this sense, PTTA2 and PESP are equivalent. The proof directly follows from Lemma 7 and Lemma 8 (see appendix).

▶ **Corollary 9.** *If $(\tilde{\pi}, z)$ is an optimal solution for PESP, the solution (π, u, F) constructed in Lemma 7 is optimal for PTTA2. On the other hand, if (π, u, F) is an optimal solution for PTTA2, the solution $(\tilde{\pi}, z)$ constructed in Lemma 8 is optimal for PESP.*

5 Computational Experiments

In this section, we test the performance of the new models when solving the IP formulations with Gurobi and compare them to PESP. We use data of the regional railway network in a region of Lower Saxony in northern Germany, since they have a size for which our integer programs can still be solved in reasonable time. The dataset is part of the open-source software framework LinTim, see [24, 23]. We use LinTim to generate different line concepts and the resulting EANs. An overview of the number of lines $|\mathcal{L}|$ and the size of the EANs is given in Table 1. We solve PTTA2 and PTTA3 for different time horizons (we vary the number of periods from $K = 3, 4, \ldots, 15$), observe the run time and compare it to the run time when solving PESP. We implemented the IP models in Python and ran them on an Lenovo laptop with Intel(R) Core(TM) i5-10310U CPU @ 1.70GHz, 2.21 GHz and 16 GB RAM using the solver Gurobi 9.1.1 ([10]). The results are shown in Figure 2.

Table 1 Size of the periodic EAN for the used line concepts.

Line concept	$\|\mathcal{L}\|$	$\|\mathcal{E}\|$	$\|\mathcal{A}\|$
line concept 1	5	180	262
line concept 2	6	196	314
line concept 3	6	212	372
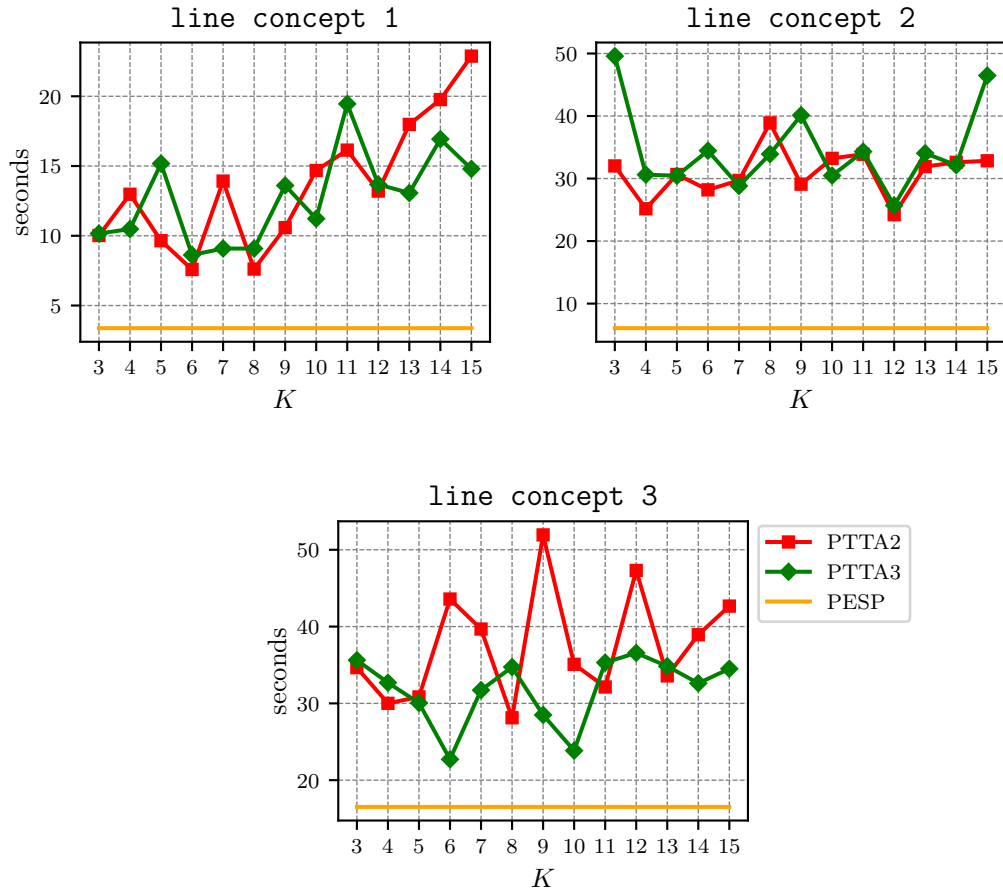
Figure 2 Average run time for different line concepts with varying K.

We first note that, as expected due to the higher number of variables and the additional assignment constraints, for all versions of PTTA the solver takes much longer than for PESP. However, recall that our motivation was to integrate delay management – a task the PESP is not suited for – so we do not have the aspiration to beat the PESP when doing pure timetabling. Since PTTA3 only solves the assignment for the first period, while PTTA2 does this for all periods, one would expect it to be faster solvable than PTTA2. Indeed, we can see this behaviour in the instance `line concept 3`. For `line concept 2` both models perform quite similar. In the instance `line concept 1` we can observe that for larger K the run time of PTTA2 increases more than for PTTA3, which can again be explained with PTTA3 only solving the assignment in the first period. An exception is the peak of PTTA3 at $K = 11$. However, inspecting the progress of the solver shows that the optimal solution was actually found much earlier and the most part of the run time was dedicated to proving optimality, so we treat this as an random outlier. The instance `line concept 3`, which is the largest one, shows the largest variance. Investigating the solving process shows that also here the solver often has difficulties to determine that the incumbent solution is indeed optimal, a well known phenomenon for many integer problems. Thus, providing dual bounds has the potential to speed up the solving process significantly.

6   Conclusion

We have developed a new model for periodic timetabling which uses a non-periodic network as basis. We have shown that the new model is equivalent to PESP and that – although this was not our main focus – the achieved run times are acceptable. We also derived a streamlined version which uses significantly less variables and constraints.

The new model opens many possibilities for future research. An obvious line of research is to strengthen its IP formulation, e.g. by using dual bounds, to speed up the solving process. A possible extension of our model could be to allow more flexibility in the synchronisation constraints, e.g. to allow that the differences between repetitions of events are not exactly T but in some interval $[T - \epsilon, T + \epsilon]$. Our main interest, however, is to use the model for integration purposes. Here, the following topics are of particular interest.

First, we plan to use the new aperiodic model for integrating timetabling and delay management in a two-stage model. This is necessary if the practically relevant concept of recovery robustness [13, 8] is to be used in which we look for a timetable that can be recovered by a suitable delay management strategy (see [11, 5] for an overview on delay management). Note that the reduced model PTTA3 cannot be used in this context since for delay management all periods need to be considered separately. Second, the new model can also be used for dealing with timetabling problems with different line frequencies. This topic is only scarcely treated in the literature on PESP, its main difficulty being to distribute passengers on the different possible transfer activities before knowing the timetable. We currently use PTTA to get an optimal distribution of passengers even if the frequencies between incoming and outgoing trains differ from each other.

Finally, we suppose that the model can also be used to integrate timetabling and passenger routing as done in [25].

References

1   Ralf Borndörfer and Christian Liebchen. When Periodic Timetables Are Suboptimal. In *Operations Research Proceedings 2007*, pages 449–454. Springer Berlin Heidelberg, 2008. `doi:10.1007/978-3-540-77903-2_69`.

2   Ralf Borndörfer, Niels Lindner, and Sarah Roth. A concurrent approach to the periodic event scheduling problem. *Journal of Rail Transport Planning & Management*, 15, 2020. `doi:10.1016/j.jrtpm.2019.100175`.

3   Valentina Cacchiani and Paolo Toth. Robust train timetabling. In *Handbook of Optimization in the Railway Industry*, pages 93–115. Springer, 2018. `doi:10.1007/978-3-319-72153-8_5`.

4   Carla Conte and Anita Schöbel. Identifying dependencies among delays. In *proceedings of IAROR 2007*, 2007. ISBN 978-90-78271-02-4.

5   Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schöbel. Delay propagation and delay management in transportation networks. In *Handbook of Optimization in the Railway Industry*, pages 285–317. Springer, 2018. `doi:10.1007/978-3-319-72153-8_13`.

6   Laura Galli and Sebastian Stiller. Modern challenges in timetabling. In *Handbook of Optimization in the Railway Industry*, pages 117–140. Springer, 2018. `doi:10.1007/978-3-319-72153-8_6`.

7   Marc Goerigk and Anita Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers and Operations Research*, 40(5):1363–1370, 2013. `doi:10.1016/j.cor.2012.08.018`.

8   Marc Goerigk and Anita Schöbel. Recovery-to-optimality: A new two-stage approach to robustness with an application to aperiodic timetabling. *Computers and Operations Research*, 52:1–15, 2014.

9 Rob M. P. Goverde. Improving Punctuality and Transfer Reliability by Railway Timetable Optimization. *Proceedings of the 5th TRAIL Annual Congress*, 1999.

10 Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2021.

11 Eva König. A review on railway delay management. *Public Transport*, 12(2):335–361, 2020.

12 Christian Liebchen. *Periodic timetable optimization in public transport*. PhD thesis, TU Berlin, 2006. `doi:10.1007/978-3-540-69995-8_5`.

13 Christian Liebchen, Marco E. Lübbecke, Rolf H. Möhring, and Sebastian Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. *Lecture Notes in Computer Science*, 5868:1–27, 2009. `doi:10.1007/978-3-642-05465-5_1`.

14 Christian Liebchen, Michael Schachtebeck, Anita Schöbel, Sebastian Stiller, and André Prigge. Computing delay resistant railway timetables. *Computers & Operations Research*, 37:857–868, 2010. `doi:10.1016/j.cor.2009.03.022`.

15 Richard M. Lusby, Jesper Larsen, and Simon Bull. A survey on robustness in railway planning. *European Journal of Operational Research*, 266:1–15, 2018.

16 Karl Nachtigall. Periodic network optimization with different arc frequencies. *Discrete applied mathematics*, 69:1–17, 1996.

17 Karl Nachtigall. *Periodic network optimization and fixed interval timetables*. PhD thesis, University of Hildesheim, 1998.

18 Karl Nachtigall and Jens Opitz. Solving periodic timetable optimisation problems by modulo simplex calculations. In *OpenAccess Series in Informatics*, volume 9, 2008. `doi:10.4230/OASIcs.ATMOS.2008.1588`.

19 Karl Nachtigall and Stefan Voget. A genetic algorithm approach to periodic railway synchronization. *Computers and Operations Research*, 23(5):453–463, 1996. `doi:10.1016/0305-0548(95)00032-1`.

20 Julius Pätzold and Anita Schöbel. A Matching Approach for Periodic Timetabling. In Marc Goerigk and Renato Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ATMOS.2016.1`.

21 Leon Peeters. *Cyclic Railway Timetable Optimization*. PhD thesis, Erasmus Universiteit Rotterdam, 2003.

22 Tomáš Robenek, Yousef Maknoon, Shadi Sharif Azadeh, Jianghang Chen, and Michel Bierlaire. Passenger centric train timetabling problem. *Transportation Research Part B: Methodological*, 89:107–126, 2016. `doi:10.1016/j.trb.2016.04.003`.

23 Alexander Schiewe, Sebastian Albert, Philine Schiewe, Anita Schöbel, and Felix Spühler. LinTim - Integrated Optimization in Public Transportation. URL: `https://www.lintim.net`.

24 Alexander Schiewe, Sebastian Albert, Philine Schiewe, Anita Schöbel, and Felix Spühler. LinTim: An integrated environment for mathematical public transport optimization. Documentation for version 2020.12, 2020. URL: `https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-62025`.

25 Philine Schiewe and Anita Schöbel. Periodic timetabling with integrated routing: Towards applicable approaches. *Transportation Science*, 54(6):1714–1731, 2020. `doi:10.1287/trsc.2019.0965`.

26 Anita Schöbel. Capacity constraints in delay management. *Public Transport*, 1(2):135–154, 2009.

27 Paolo Serafini and Walter Ukovich. A Mathematical Model for Periodic Scheduling Problems. *SIAM Journal on Discrete Mathematics*, 2:550–581, 1989. `doi:10.1137/0402049`.

28 Pieter Vansteenwegen and Dirk Van Oudheusden. Developing railway timetables which guarantee a better service. *European Journal of Operational Research*, 173(1):337–350, 2006. `doi:10.1016/j.ejor.2004.12.013`.

A   Proofs

Proof of Lemma 1

Proof. First, note that $U_a = T - H_{ji} \leq T$ for $a = (i, j) \in \mathcal{A}_{\text{head}}$, i.e. $K_a = K - \left\lceil \frac{U_a}{T} \right\rceil = K - 1$.
"$(a) \Rightarrow (b)$" Let $1 \leq s \leq K - 1$. We consider the event j_K in the last period. Since the event i_s takes place in the s-th period, we have $\pi_{i_s} < \pi_{j_K}$. In particular, $\pi_{i_s} - \pi_{j_K} < 0 \leq H_{ji}$ and hence, by (a), we have $\pi_{j_K} - \pi_{i_s} \geq H_{ij}$. Let now t be minimal such that $\pi_{j_t} - \pi_{i_s} \geq H_{ij} = L_a$.
It remains to show that $\pi_{j_t} - \pi_{i_s} \leq T - H_{ji} = U_a$.
First case: $t > 1$. By minimality of t we have $\pi_{j_{t-1}} - \pi_{i_s} < H_{ij}$ and hence, $\pi_{i_s} - \pi_{j_{t-1}} \geq H_{ji}$. This yields $\pi_{j_t} - \pi_{i_s} = \pi_{j_{t-1}} + T - \pi_{i_s} \leq T - H_{ji} = U_a$.
Second case: $t = 1$. Assume $\pi_{j_1} - \pi_{i_s} > T - H_{ji}$. Then $\pi_{j_1} - \pi_{i_{s+1}} = \pi_{j_1} - \pi_{i_s} - T > -H_{ji}$, i.e. $\pi_{i_{s+1}} - \pi_{j_1} < H_{ji}$. Hence, we must have $\pi_{j_1} - \pi_{i_{s+1}} \geq H_{ij}$, which in particular means that $\pi_{j_1} \geq \pi_{i_{s+1}}$. Since j_1 takes place in the first period and i_{s+1} in the $s + 1$-th period, this is a contradiction. Thus, our assumption was false and we have $\pi_{j_1} - \pi_{i_s} \leq T - H_{ji} = U_a$.
"$(b) \Rightarrow (a)$" We first consider $1 \leq s \leq K - 1$. By assumption there is some t' such that $\pi_{j_{t'}} - \pi_{i_s} \in [H_{ij}, T - H_{ji}]$. For $t \geq t'$ we have $\pi_{j_t} - \pi_{i_s} \geq \pi_{j_{t'}} - \pi_{i_s} \geq H_{ij}$. On the other hand, for $t < t'$ we have $\pi_{j_t} \leq \pi_{j_{t'}} - T$ and hence $\pi_{i_s} - \pi_{j_t} \geq \pi_{i_s} - \pi_{j_{t'}} + T \geq H_{ji}$. Thus, for every t one of the conditions is fulfilled.
It remains to show the claim for $s = K$. Using the assumption for $s' = K - 1$ yields the existence of some t' such that $\pi_{j_{t'}} - \pi_{i_{K-1}} \in [H_{ij}, T - H_{ji}]$. In particular, $\pi_{j_{t'}} \geq \pi_{i_{K-1}}$, which implies $t' \geq K - 1$.
First case: $t' = K - 1$. We have $\pi_{j_K} - \pi_{i_K} = (\pi_{j_{K-1}} + T) - (\pi_{i_{K-1}} + T) = \pi_{j_{K-1}} - \pi_{i_{K-1}} \geq H_{ij}$. Furthermore, for $t \leq K - 1$ it follows $\pi_{i_K} - \pi_{j_t} = \pi_{i_{K-1}} + T - \pi_{j_t} \geq \pi_{i_{K-1}} + T - \pi_{j_{K-1}} \geq H_{ji}$, where the last inequality follows from $\pi_{j_{K-1}} - \pi_{i_{K-1}} \leq T - H_{ji}$.
Second case: $t' = K$. For every $t \leq K$ we have $\pi_{j_t} - \pi_{i_K} \leq \pi_{j_K} - \pi_{i_K} = \pi_{j_K} - \pi_{i_{K-1}} - T \leq -H_{ji}$, which implies $\pi_{i_K} - \pi_{j_t} \geq H_{ji}$.   ◀

Proof of Lemma 4

Proof. "$\Rightarrow$" Let (π, u, F) be a solution for PTTA2. For $a = (i_1, j_t)$ set $u'_a := u_a$. Clearly, (π, u', F) is a feasible solution for PTTA3 and the objective values coincide. "$\Leftarrow$" Let (π, u', F) be a solution for PTTA3. For $a = (i_s, j_t) \in \mathcal{A}$ set $u_a := u'_{(i_1, j_{t-s+1})}$. Note that since $a \in \mathcal{A}$ we have $s \leq t \leq s + b_a$ and therefore $1 \leq t - s + 1 \leq 1 + b_a$, which implies that also $(i_1, j_{t-s+1}) \in \mathcal{A}$. We show that (π, u, F) is a feasible solution for PTTA2:

- Let $a = (i_s, j_t) \in \mathcal{A}$. We have

$$\pi_{j_t} - \pi_{i_s} + M(u_a - 1) = (\pi_{j_{t-s+1}} + (s-1)T) - (\pi_{i_1} + (s-1)T) + M(u_a - 1)$$
$$= \pi_{j_{t-s+1}} - \pi_{i_1} + M(u'_{(i_1, j_{t-s+1})} - 1) \leq U_{(i_1, j_{t-s+1})} = U_a,$$

   which shows constraints (8). Analogously be obtain (9).
- Let $(i, j) \in \underline{\mathcal{A}}, 1 \leq s \leq K$. We have

$$\sum_{t : a = (i_s, j_t) \in \mathcal{A}} u_a = \sum_{t : a = (i_1, j_{t-s+1}) \in \mathcal{A}} u'_a = 1$$

   and hence, (11) holds.
- Constraints (10) and (12) to (17) follow immediately.

Consequently, (π, u, F) is a feasible solution for PTTA2 with the same objective value as (π, u', F).   ◀

Proof of Lemma 7

Proof. We check that (π, u, F) fulfils all constraints:

- (8) and (9) are fulfilled by choice of u and Lemma 5.
- Let $i_s \in \mathcal{E}, 2 \leq s \leq K$. By definition of π it follows

$$\pi_{i_s} - \pi_{i_{s-1}} = (\pi_{\text{first}}(i) + (s-1)T) - (\pi_{\text{first}}(i) + (s-2)T) = T,$$

  which proves (10).

- Let $a = (i,j) \in \underline{\mathcal{A}}, 1 \leq s \leq K_a$. By Lemma 5 we have $\pi_{j_t} - \pi_{i_s} \in [L_a, U_a]$ for $t = z_a + k - l + s$, which by Lemma 3 implies $t \leq s + b_a$. In particular, $(i_s, j_t) \in \mathcal{A}$. By choice of u it follows $\sum_{t:a=(i_s,j_t)\in\mathcal{A}} u_a = 1$, i.e. constraints (11) are fulfilled.
- Constraints (12) to (15) are obviously fulfilled.
- Let $a = (i_1, j_t) \in \mathcal{A}$.
  First case: $u_a = 1$. $F_a = \pi_{j_t} - \pi_{i_1} = M(u_a - 1) + \pi_{j_t} - \pi_{i_1}$.
  Second case: $u_a = 0$. $F_a = 0 > -M + \pi_{j_t} - \pi_{i_1} = M(u_a - 1) + \pi_{j_t} - \pi_{i_1}$.
  Hence, constraints (16) are fulfilled.
- For (17), $F_a \in \mathbb{Z}$ is clear. Note that by (9) $u_a = 1$ is only possible if $\pi_{j_t} \geq \pi_{i_1}$, which in particular means that $F_a \geq 0$ and therefore $F_a \in \mathbb{N}$.

Hence, (π, u, F) is indeed a feasible solution. For the objective value we obtain:

$$f = K \cdot \left(\sum_{a=(i_1,j_t)\in\mathcal{A}} w_a F_a \right) = K \cdot \left(\sum_{a=(i_1,j_t)\in\mathcal{A}:u_a=1} w_a(\pi_{j_t} - \pi_{i_1}) \right)$$

$$\overset{(*)}{=} K \cdot \left(\sum_{a=(i,j)\in\underline{\mathcal{A}}} w_a(\tilde{\pi}_j - \tilde{\pi}_i + z_a T) \right) = K \cdot \tilde{f},$$

where $(*)$ follows from the proof of Lemma 5. ◄

Proof of Lemma 8

Proof. Let $a = (i,j) \in \underline{\mathcal{A}}$. The following holds:

$$\begin{aligned}
\tilde{\pi}_j - \tilde{\pi}_i + z_a T &= (\pi_{j_1} - r_j T) - (\pi_{i_1} - r_i T) + z_a T \\
&= (\pi_{j_t} - (t-1)T - r_j T) - (\pi_{i_1} - r_i T) + z_a T \\
&= \pi_{j_t} - \pi_{i_1} - (r_j - r_i + t - 1)T + z_a T \\
&= \pi_{j_t} - \pi_{i_1} \in [L_a, U_a].
\end{aligned}$$

Hence, $(\tilde{\pi}, z)$ is a feasible solution to PESP. For the objective value we have:

$$\begin{aligned}
\tilde{f} &= \sum_{a=(i,j)\in\underline{\mathcal{A}}} w_a(\tilde{\pi}_j - \tilde{\pi}_i + z_a T) \\
&= \sum_{a=(i_1,j_t)\in\mathcal{A}:u_a=1} w_a(\pi_{j_t} - \pi_{i_1}) \\
&\overset{(*)}{\leq} \sum_{a=(i_1,j_t)\in\mathcal{A}:u_a=1} w_a F_a \\
&\overset{(**)}{\leq} \sum_{a=(i_1,j_t)\in\mathcal{A}} w_a F_a = f \cdot \frac{1}{K}.
\end{aligned}$$

Here, $(*)$ follows from (16) and $(**)$ from $F_a \geq 0$. ◄

Proof of Corollary 9

Proof. Let $(\tilde{\pi}, z)$ be an optimal solution for PESP with objective value $\tilde{f}$. By Lemma 7 we obtain a feasible solution (π, u, F) for PTTA with objective value $f = K\tilde{f}$. Assume this is not optimal, i.e. there is a solution (π', u', F') with objective value $f' < f$. By Lemma 8 we get a solution $(\bar{\pi}, \bar{z})$ for PESP with objective value $\bar{f} \le f' \cdot \frac{1}{K} < f \cdot \frac{1}{K} = \tilde{f}$, which is a contradiction to $(\tilde{\pi}, z)$ being an optimal solution.

On the other hand, let (π, u, F) be an optimal solution to PTTA with objective value f. Lemma 8 yields a feasible solution $(\tilde{\pi}, z)$ for PESP with objective value $\tilde{f} \le f \cdot \frac{1}{K}$. Assume $(\tilde{\pi}, z)$ is not optimal, i.e. there is a solution $(\bar{\pi}, \bar{z})$ with objective value $\bar{f} < \tilde{f}$. By Lemma 7 we receive a solution (π', u', F') for PTTA with objective value $f' = K\bar{f} < K\tilde{f} \le f$, a contradiction. ◄

Fast Map Matching with Vertex-Monotone Fréchet Distance

Daniel Chen ✉
Apple, Cupertino, CA, USA

Christian Sommer ✉
Apple, Cupertino, CA, USA

Daniel Wolleb ✉
Apple, Cupertino, CA, USA

── Abstract ──────────────

We study a generalization for map matching algorithms that includes both geometric approaches such as the Fréchet distance and global weight approaches such as those typically used by Hidden Markov Models. Through this perspective, we discovered an efficient map matching algorithm with respect to the *vertex-monotone Fréchet distance* while using a heuristic tie-breaker inspired by global weight methods. While the classical Fréchet distance requires parameterizations to be monotone, the vertex-monotone Fréchet distance allows backtracking within edges. Our analysis and experimental evaluations show that relaxing the monotonicity constraint enables significantly faster algorithms without significantly altering the resulting map matched paths.

1 Introduction

With the widespread availability of receivers for the Global Positioning System (GPS) in modern cars and smartphones came the rise of large databases of GPS traces (also called trajectories). These trajectory databases are valuable sources for various applications like map construction [8, 2, 25], map refinement and correction [42, 26], traffic estimation [6, 9, 35], travel time estimation [28, 47], dynamic routing [43, 14, 12, 13], ride sharing [18], mobility studies [19, 36], location mining [53], and many more. As a first step, GPS traces often get mapped to a road network via a non-trivial process called *map matching*. Beyond the applications mentioned above, map matching can also help with indexing the trajectory database to support fast retrieval of traces [30, 21].

Given a GPS trace as a sequence of (latitude, longitude) pairs, possibly equipped with time stamps and auxiliary information, a map matching algorithm is expected to return a connected sequence of road segments or edges (i.e., a path) in the road network that the input trace originally traversed. In some situations, the match is rather obvious, e.g., for a straight-line trace along an isolated stretch of road. The main challenge of map matching lies in the interplay of noisy GPS observations and dense road networks, especially in urban settings with complex intersections, highway crossings, and stacked roads, where skyscrapers may further block or otherwise interfere with the satellite signal. In such scenarios there can be many candidate paths for a given GPS trace, with several viable options for the path the device was actually traveling on.

Beyond qualitative aspects, another important consideration is the performance of a map matching algorithm. Contemporary trajectory databases are large, as is their growth, and many of the applications mentioned above typically benefit from low-latency processing of the most recent traces. Furthermore, road networks change over time, at least locally,

which in turn may require a decent amount of reprocessing of historical traces. Faster map matching performance can then be leveraged to achieve better tradeoffs between the number of cores required for map matching, the latency of the final result, and the amount of data that can be processed.

1.1   Related Work

Within the field of trajectory mining, map matching has been studied in many different models. For a general survey, we refer to Zheng [52]. For a survey on online map matching (where we are given only a few recent GPS points), we refer to Quddus et al. [39].

In this paper, we are interested in *offline* map matching, where the input consists of a typically longer GPS trace that should be matched to the map. Offline map matching allows to match many traces to a map simultaneously [31]. However, most papers, including our work, study the scenario where each trace is matched separately and independently.

A survey by Wei et al. [48, 49] distinguishes between incremental and global approaches. While incremental algorithms were successful at the SIGSPATIAL Cup 2012 [3, 45], global approaches are generally more accurate [7]. Weight-based, global approaches, like the Hidden Markov Model by Newson and Krumm [32], are widely used [24, 34] and also work in the online setting [23]. However, as Wei et al. [48, 49] argue, they require careful parameter tuning and make strong independence assumptions on the distribution of the GPS errors. Geometric approaches largely avoid parameter tuning by simply minimizing a distance function between the trace and a matching path.

The Fréchet distance is frequently used in many different variations for map matching [50, 11, 44, 4, 10] and other applications [27] and is also the main ingredient in our algorithm. Indeed, our algorithm borrows from much of the prior research into Fréchet distance based map matching, but our focus is on map matching high sampling rate and relatively low error GPS traces in a small number of microseconds per crumb, while quickly rejecting those with high error. There are many large data sets consistent with these properties, including the one in SIGSPATIAL Cup 2012, and they are often a more reliable source of information than high error or sparsely sampled GPS traces. By focusing on these data conditions and by relaxing the Fréchet distance, we are able to avoid more complex machinery in [50, 11, 44] and optimize absolute runtimes for low error traces.

There is also prior work on map matching with respect to the Fréchet distance while restricting candidates to shortest paths on the graph [10]. While our algorithm does not enforce such a restriction, we do prefer shorter routes when breaking ties. Indeed, map matching algorithms based purely on the Fréchet distance may sometimes choose an arbitrary matched path out of the set of paths with the same Fréchet distance. Wei et al. [48, 49] presented an algorithm that combines a Fréchet distance and a weight-based approach that prefers shorter and closer routes, and we build on their result with the goal of improving the running time without sacrificing accuracy.

1.2   Contributions

We present a novel map matching algorithm based on the vertex-monotone Fréchet distance [46, 27]. There are several ingredients that contribute substantially to performance and quality. We discuss the following three in more detail:

- Wenk et al. [50] compute the weak Fréchet distance by pruning the search space using the road network geometry. Pursuing an analogous approach, we get a significant performance boost.

- We present a novel trace simplification technique to further improve the running time.
- To distinguish between different paths of the same distance, we introduce a global weight function, similar to the one used by Wei et al. [48, 49], to achieve a high matching accuracy.

Finally, we propose a general framework that provides a unified view on both Hidden Markov Models and Fréchet distance approaches. We argue that they both can be seen as path searches in parametrization spaces, which motivates some of the tradeoffs we chose when designing our algorithm. In our experimental evaluation, we observe that our algorithm is significantly faster than previously published results, which makes it a viable and competitive alternative to popular HMM-based methods.

2 Preliminaries

2.1 Fréchet distances between curves

In map matching, we want to find a matching path through the road network that is closest to a given trace of a GPS device under some measure of similarity. To do this, we model both paths on the network and GPS traces as curves in $\mathbb{R}^2$. One natural distance function between curves in $\mathbb{R}^2$ is the *Fréchet distance*.

A illustrative mental picture for the Fréchet distance is the following: A dog and its owner go for a walk. The dog strolls along the first curve while the owner walks along the second curve going sometimes faster, sometimes slower, in an effort to keep the leash of the dog as short as possible for the entire path. The shortest possible leash length that allows both of them to traverse their paths while never walking backwards is precisely what is called the Fréchet distance between the two curves.

In the following, we define the Fréchet distance as well as three known variants and their relations (weak, discrete, and vertex-monotone Fréchet distance).

▶ **Definition 1** (Fréchet distance [20, 5]). *For two curves given as continuous maps $\pi : [1, n] \to \mathbb{R}^2$ and $\sigma : [1, m] \to \mathbb{R}^2$, the* Fréchet distance *is defined as*

$$d_{FD}(\pi, \sigma) = \inf_{\substack{f:[0,1]\to[1,n] \\ g:[0,1]\to[1,m]}} \max_{t\in[0,1]} \|\pi(f(t)) - \sigma(g(t))\|_2 \,, \tag{1}$$

where f and g are continuous and monotonically increasing functions with $f(0) = 1$, $f(1) = n$, $g(0) = 1$, $g(1) = m$.

If we omit the monotonicity constraint, i.e. allow the owner and the dog to backtrack along their curves, we get what is known as the *weak Fréchet distance*. While it can lead to efficient algorithms, as proposed by Wenk et al. [50], the weak Fréchet distance can be arbitrarily smaller than the Fréchet distance, as e.g. noted by Chen et al. [11]. In particular, a matched trace might correspond to driving back and forth on a one-way street to better match a loop in the trace trajectory, e.g. at a highway intersection.

Another way to relax the problem while still enforcing monotonicity is to look only at the vertex positions, assuming that both curves are polygonal. This means that the dog and the owner progress in discrete steps and in each step they each either jump to the next vertex or stay put. The main issue with the *discrete Fréchet distance* is that it can be large even for two curves that visually are very close if one of the two curves is sampled very coarsely. In particular, if a long, straight section of a highway is modeled as a single segment, even a trace that follows it very closely would have a large discrete Fréchet distance. One

way to handle this would be to supersample the geometry to a sufficiently high granularity, but this would increase the complexity of the free space diagram. Indeed, as we will see later on, reducing the complexity of the free space diagram through geometry simplification significantly improves our runtimes.

Finally, we introduce yet another variant of the Fréchet distance, which we use in our map matching algorithm. Consider the setting where the dog and the owner have to walk continuously on two polygonal curves, but where the monotonicity constraint is relaxed to allow for backtracking within each straight-line segment, but not past any vertex. This is known as the *vertex-monotone Fréchet distance*, which was defined by van Leusden [46]. Below is a formal definition (adding the boundary constraints missing in [46]).

▶ **Definition 2** (Vertex-monotone Fréchet distance [46])**.** *For two polygonal curves given as linearly-interpolated, continuous maps $\pi : [1, n] \to \mathbb{R}^2$ and $\sigma : [1, m] \to \mathbb{R}^2$, the* vertex-monotone Fréchet distance *is defined as*

$$d_{VMFD}(\pi, \sigma) = \inf_{\substack{f:[0,1]\to[1,n] \\ g:[0,1]\to[1,m]}} \max_{t\in[0,1]} \|\pi(f(t)) - \sigma(g(t))\|_2, \tag{2}$$

where f and g are continuous with $f(0) = 1$, $f(1) = n$ and if $f(t) > i$ for any $t \in [0, 1]$ and $i \in [\![1, n]\!]$, then also $f(t') > i$ for any $t' > t$, and likewise for g.

We can observe the following order between all these distance functions for any pair of polygonal curves (here d_{WFD} refers to the weak Fréchet distance, and d_{DFD} refers to the discrete Fréchet distance):

$$d_{\mathrm{WFD}} \le d_{\mathrm{VMFD}} \le d_{\mathrm{FD}} \le d_{\mathrm{DFD}} \tag{3}$$

This follows from the fact that the mapping functions f and g get more and more constrained from left to right. Moreover, we note that using the triangle inequality, we can also bound

$$d_{\mathrm{FD}} \le d_{\mathrm{VMFD}} + D, \tag{4}$$

where D is the length of the longest line segment on π and σ.

Furthermore, note that we can bring d_{VMFD} arbitrarily close to d_{FD} by simply subdividing long segments of π and σ. Also note that d_{FD} can differ from d_{VMFD} even on undirected graphs because d_{FD} allows the mapping to switch directions only at the vertices.

2.2   Map matching problem

The map matching problem has two inputs: The first one is a *GPS trace T* that describes the trip of a driver as n points (also called *crumbs*), each specifying a position $p_i \in \mathbb{R}^2$ and monotonically increasing time stamps $t_i \in \mathbb{R}$. For simplicity, we assume that the times are normalized to $t_1 = 1$ and $t_n = n$. We model the route of the driver as a 2D-curve consisting of the polyline given by $p_1, \ldots, p_n$ and the time parametrization implied by $t_1, \ldots, t_n$, where for any time t with $t_i < t < t_{i+1}$, we have that $T(t)$ is the linear interpolation between p_i and p_{i+1}, see Figure 1.

The second input is a *road network*, given as a directed graph $G = (V, A)$ with vertices V and arcs A, where each vertex $v_i \in V$ has a location $\ell_i \in \mathbb{R}^2$, and each arc $a = (v_i, v_j) \in A$ has the *shape* of the straight line $\overline{\ell_i, \ell_j}$. Note that others often allow for arbitrary polylines as the shape of an arc, which we can simply model as subdividing arcs with vertices of degree two.
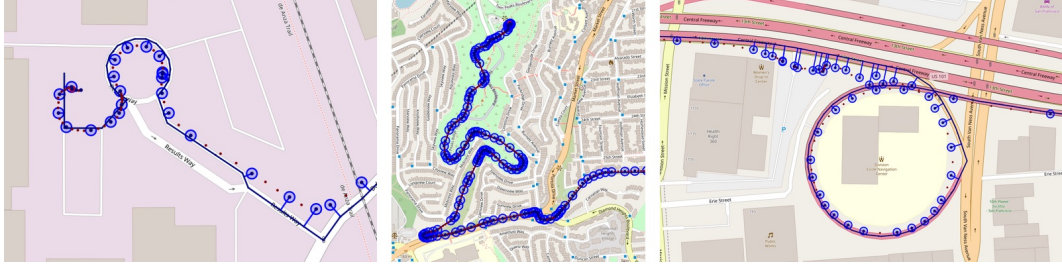
■ **Figure 1** Visualization of three map matching situations. Crumbs shown in red were dropped in the simplification step prior to map matching. Crumbs shown in blue got matched to the road network. The dark blue path shows the matched path P, and the light blue lines show the matching of crumbs onto the road network. Sometimes, GPS is very accurate and the trace lines up perfectly with the geometry of the road network (middle). Oftentimes, the trace and the network do not align perfectly due to the distance to the middle of the road on the map (left) or due to noisy GPS measurements (right).

Let Π be the set of paths through G, where a path P is a sequence of m vertices $v_1, \ldots, v_m$ that are connected by arcs, i.e., $(v_i, v_{i+1}) \in A$ for $1 \leq i < m$. We also view P as the polygonal 2D-curve described by the sequence of arcs and parametrized such that $P(i) = v_i$.

We are looking for a path $P \in \Pi$ that *best* reflects the journey that the driver took through the road network. To define what *best* means in this context, we look at two parametrizations: $f : [0,1] \mapsto [1,n]$ for the trace T and $g : [0,1] \mapsto [1,m]$ for the map matched path P. We now use the distance measures defined in Section 2.1 to restrict f and g, and to specify which path P we desire.

▶ **Definition 3** (Vertex-monotone Fréchet distance map matching). *Given trace T and network G, the* vertex-monotone Fréchet distance map matching problem *asks to find the path $P \in \Pi$ as well as parametrizations f and g such that $d_{VMFD}(T, P)$ is minimized.*

Note that for map matching, it is reasonable to relax the boundary condition of g in d_{VMFD} to $g(0) \in [1,2]$ and $g(1) \in [m-1, m]$ so as to allow for the beginning and end of the trace to be each matched to the interior of an arc.

2.3 Free-space diagram

An important tool to compute the Fréchet distance between curves and also for map matching is the so-called *free-space diagram* (also sometimes called *free-space surface*). It is defined as the sublevel set of the Euclidean distance function with respect to the parameter space of the trace and the graph. For a threshold $\epsilon > 0$, the free space $F_{\leq \epsilon}$ is the set of pairs of points, one on the trace T, one on the graph G, so that they are within distance ϵ of each other:

$$F_{\leq \epsilon}(T, G) = \{(p, q) \mid p \in T, q \in G, \|p - q\|_2 \leq \epsilon\}. \tag{5}$$

This notion is useful because searching for a map matching can now be seen as finding a path with certain properties through this free-space diagram. Namely, we are looking for a vertex-monotone path from (p_1, u) to (p_n, v) for some arcs $u, v \in A$. Let us call the individual elements of this Cartesian product *cells* and its borders *intervals* of the free-space diagram. We refer to Figure 2 for an illustration.
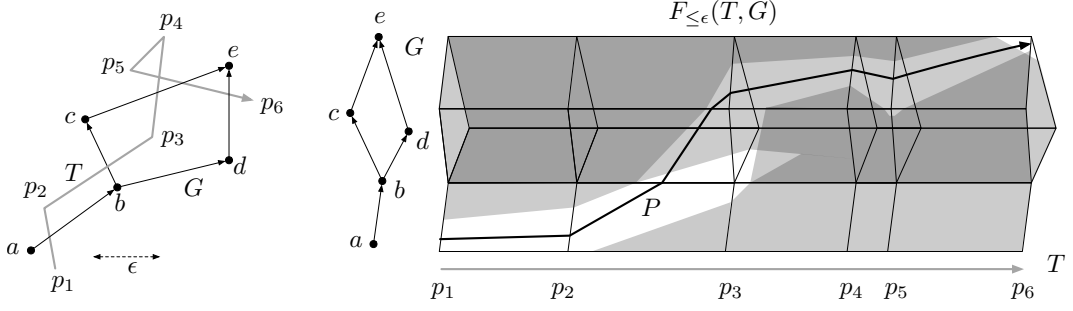
■ **Figure 2** (left) A trace T in grey through the arcs of a graph G. (right) The free-space diagram with T stretched out horizontally and G in the vertical direction shows unreachable areas in dark color. Any vertex-monotone path through the reachable free-space, like the path P in black, we consider a valid map matching with distance at most ϵ. Note how the example is not monotone between p_4 and p_5 along arc (c, e). Also note how for this distance ϵ, there is a path through the reachable free-space that maps T to the path $a \to b \to c \to e$, but there is none along $a \to b \to d \to e$ as vertex d is too far away from the trace.

3   General Framework for Map Matching Algorithms

To motivate our algorithm, we introduce a general framework for map matching that includes more than just Fréchet distance map matching.

For a discrete GPS trace $T : [\![1, n]\!] \mapsto \mathbb{R}^2$, a map matching can be formalized as a function $M : [\![1, n]\!] \mapsto G$, where G is an embedded graph representing the road network. We observe that if we take the Cartesian product of any continuous relaxation $\tau : [1, n] \mapsto \mathbb{R}^2$ and $\mu : [1, n] \mapsto G$ of T and M, we get $\pi = \tau \times \mu : [1, n] \mapsto \mathbb{R}^2 \times G$. Moreover, any such $\rho = \tau \times \mu$ where $\tau(i) = T(i)$ where $i \in \mathbb{Z}$ yields a map matching $M : [\![1, n]\!] \mapsto G$ by restricting the domain of μ to integers. Therefore, it is natural to consider a class of map matching algorithms that optimize a cost function $\gamma(\rho)$ on functions $\rho = \tau \times \mu : [1, n] \mapsto \mathbb{R}^2 \times G$ where $\tau(i) = T(i)$. We claim that both classical Hidden Markov Model (HMM for short) and Fréchet map matching approaches fall into this class of algorithms.

3.1   Hidden Markov Model

The HMM algorithm by Newson and Krumm [32] is one of the most popular map matching algorithms, and is the basis of map matching implementations in libraries such as Graph-Hopper [24] and MapBox [34]. The general approach of this algorithm is to find matched roads $\{r_i\}_{i=1}^{n}$ that maximize

$$\Pr(T(n)|r_n) \prod_{i=1}^{n-1} \Pr(T(i)|r_i) \Pr(d_i). \tag{6}$$

In the original paper, these probabilities are defined with model parameters σ_z, β as

$$\Pr(T(i)|r_i) = \frac{1}{\sqrt{2\pi}\sigma_z} e^{-0.5(\|T(i)-x_i\|_2/\sigma_z)^2} \tag{7}$$

where x_i is the point on r_i closest to $T(i)$ and

$$\Pr(d_i) = \frac{1}{\beta} e^{-d_i/\beta} \tag{8}$$

where

$$d_i = \left| \|T(i) - T(i+1)\|_2 - \|x_i - x_{i+1}\|_2 \right|. \tag{9}$$

Note that maximizing (6) is equivalent to minimizing the following cost function:

$$\gamma(\rho) = -\sum_{i=1}^{n} \log \Pr(T(i)|r_i) - \sum_{i=1}^{n-1} \log \Pr(d_i). \tag{10}$$

This cost function can be computed from $\rho = \tau \times \mu$, as r_i is simply the road that $\mu(i)$ lies on, and

$$d_i = \left| \|\tau(i) - \tau(i+1)\|_2 - \int_i^{i+1} \nu_i(t)dt \right| \tag{11}$$

where $\nu_i(t) = \mu'(t)$ if μ traverses the shortest path on G from $\mu(i)$ to $\mu(i+1)$, and $\nu_i(t) = \infty$ otherwise.

Now, the problem can be viewed through the lens of finding the least costly path with respect to γ through $\mathbb{R}^2 \times G$. To make this efficient to optimize, the HMM algorithm restricts $\mu(i)$ to projected points of road within a small neighborhood of each $T(i)$. This discretization of the problem naturally suggests defining vertices in $(T(i), x_i) \in \mathbb{R} \times G$ with costs $-\log \Pr(T(i)|r_i)$ and shortcut edges of cost $-\log \Pr(d_i)$ between them and then using Dijkstra's algorithm for shortest paths to find the least costly path with respect to γ. The algorithm of Tang et al. [45], for example, behaves in a similar fashion, although it does not explicitly model the problem as a HMM.

3.2 Fréchet distance

Map matching by the Fréchet distance [4] also naturally falls in this framework. To see this, we simply plug in the definition of the Fréchet distance and note that minimizing the Fréchet distance is the same as minimizing the cost function

$$\gamma(\rho) = \max_{i \in [1,n]} \|\tau(i) - \mu(i)\|_2 \tag{12}$$

if τ is a monotonic parameterization of the input trace, and μ monotone on its projected path on the graph and $\gamma(\rho) = \infty$ otherwise. We note that it is this monotonicity condition that precludes a simple implementation of this path optimization problem using Dijkstra's algorithm [16] in map matching. Instead, classical algorithms for Fréchet map matching find optimal values for γ through binary or parametric search.

3.3 Motivation for our algorithm

With this framework in mind, our goal becomes finding a function γ that captures how "good" a map matching is, and that, at the same time, is efficient to optimize. We observe that if we use the vertex-monotone Fréchet distance instead of the regular Fréchet distance, we are optimizing (12) without global monotonicity constraints, and such an optimization can be implemented with a single Dijkstra's search. One issue with raw Fréchet map matching is that there may be many ties with the same γ. Wei et al. [48, 49] found that using a HMM-like objective function on matchings with the same Fréchet distance gave good map matching results, which inspired us to use a similar secondary optimization in our map matching algorithm. This secondary optimization, like the HMM, can be implemented using Dijkstra's algorithm, notably on an even smaller portion of the parameter space. Details for our algorithm are described in Section 4.

4 Algorithm

On a high level, our algorithm for vertex-monotone Fréchet distance map matching combines three existing ideas in a new way:

- We follow the general two-step approach of Wei et al. [48, 49] to first compute the necessary Fréchet distance for a match to exist and then to optimize the path within that distance threshold using a secondary objective function.
- We build a representation of the reachable free-space on the fly. This was not done by Wei et al. but by Wenk et al. [50] in the context of using the weak Fréchet distance.
- Finally, we introduce the vertex-monotone Fréchet distance metric to map matching to overcome the shortcomings of the weak Fréchet distance described in Section 2.1 and to significantly improve the running time compared to the standard Fréchet distance.

4.1 Overview

Our algorithm works in two steps: First, we determine the vertex-monotone Fréchet distance between T and G by running a *minimum bottleneck path* search through the free-space diagram of T and G. The only parameter of this search is a maximum distance D, which we use to decide whether there is a match for this trace or not.

Second, we search for the best path among all the ones with minimum distance. For this tie-breaking step, we use a global weight-function to trade off minimizing the length of the matching path P through the graph with minimizing the distances between each point $p_i = \tau(t)$ of T and its corresponding match $x_i = \mu(t)$ on G.

As pointed out by Wei et al. [48, 49], such a second step is necessary to differentiate among the often many paths P that have the same vertex-monotone Fréchet distance. Note that a single point with high GPS error and possibly large distance from a nearest road segment may significantly increase the number of possible segments for many other points. For instance, such a weight function allows us to prefer the straight main street over a nearby parallel parking road as it avoids a detour, even if the GPS signal of a car driving on the main street is off and some individual crumbs appear to be closer to the parking road than the main street (see Figure 3 for an example).

4.2 Distance computation

The first phase of our algorithm has two goals: Determining the vertex-monotone Fréchet distance $d_{\mathrm{VMFD}}^* = \min_{P \in G} d_{\mathrm{VMFD}}(P, G)$ as well as, if $d_{\mathrm{VMFD}}^* \leq D$, exploring the entire reachable free-space of $F_{\leq d_{\mathrm{VMFD}}^*}$.

4.2.1 Auxiliary interval graph

To this end, we traverse the free-space surface in increasing order of the vertex-monotone Fréchet distance. We look at this as a graph traversal problem, where the vertices are the border intervals of the cells of the free-space diagram. In particular, we look at *vertical intervals* $(p_i, (v_j, v_k))$, determined by a crumb $p_i \in T$ and an arc $(v_j, v_k) \in A$, and at *horizontal intervals* $((p_i, p_{i+1}), v_j)$, determined by two consecutive crumbs $p_i, p_{i+1} \in T$ and a vertex $v_j \in V$. Each interval naturally defines a distance, namely the distance between the point and the arc for vertical intervals and between the crumb line and the vertex for horizontal ones, that we use as a vertex weight in the interval graph.

To define the connectivity of this auxiliary interval graph, we consider all the options of a vertex-monotone parametrization. Vertical intervals $(p_i, (v_j, v_k))$ only have two options to continue: We can move from p_i to p_{i+1} to the next vertical interval $(p_{i+1}, (v_j, v_k))$,
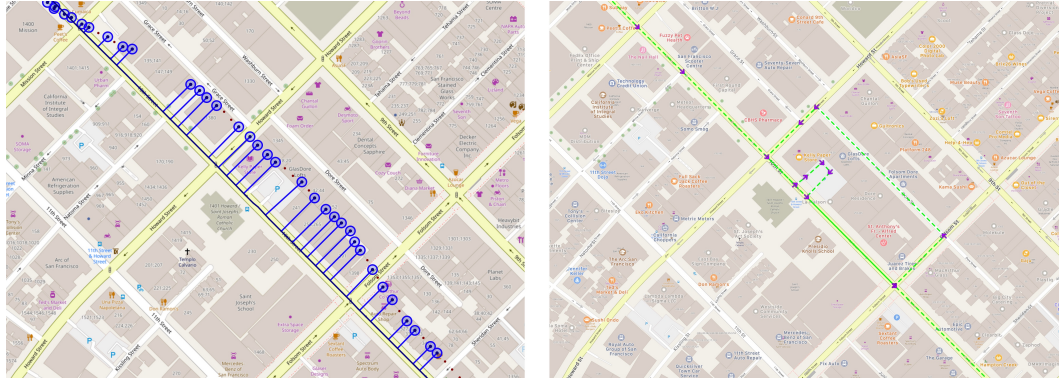
Figure 3 (left) A trace proceeding along 10th Street in San Francisco from northwest to southeast with its correctly map matched path computed with the vertex-monotone Fréchet distance map matching algorithm (with simplificiation enabled). Note how the GPS points are consistently shifted by about 45 meters to the northeast for the central part of the depicted trace. (right) The path computed by the GraphHopper HMM implementation (without simplification). Observe how the shifted points unsettle the algorithm and entice the path to enter into the parking lot and then to even go once around the entire block to travel through Dore Street. Dore Street is significantly closer to the observed GPS points and thus corresponds to a higher likelihood in the HMM. However, the much longer matching path and the fact that one has to use it in the opposing direction (as Howard Street and Folsom Street are one-way streets) make it clear that this is not the correct match. This example illustrates why we believe that the independence assumption for GPS errors often does not hold in practice and that, due to its geometric nature, our proposed algorithm is more robust to such consistent offsets.

which corresponds to matching the crumb line (p_i, p_{i+1}) to within the arc (v_j, v_k), or we can proceed to the end of the arc, so to the horizontal interval $((p_i, p_{i+1}), v_k)$, which corresponds to matching the end of (v_j, v_k) to within the interval (p_i, p_{i+1}). Horizontal intervals $((p_i, p_{i+1}), v_j)$ have two continuations for each outgoing arc of v_j: For an arc (v_j, v_k), we can go to another horizontal interval $((p_i, p_{i+1}), v_k)$, saying that we match all of (v_j, v_k) to within (p_i, p_{i+1}), or we can go to the vertical interval at the end of the crumb line, so to $(p_{i+1}, (v_j, v_k))$, saying that we match the end of (p_i, p_{i+1}) to within the arc (v_j, v_k).

Note how we do not add interval arcs for going to the previous crumb or to the beginning of an arc. Those would be the arcs needed to allow for searching the weak Fréchet distance. We refer to Figure 4 for an illustration of this interval graph on the free-space diagram and the possible out-arcs of each interval vertex.

4.2.2 Correctness

We now argue that computing the vertex-monotone Fréchet distance corresponds to finding a path with the smallest possible maximum vertex weight, the so-called *minimum bottleneck path*, through this interval graph starting at any vertical interval involving p_1 and ending at any vertical interval involving p_n. To prove this, we observe that for any two line segments in $\mathbb{R}^2$ the maximum distance between the two always involves at least one of the four endpoints. This means that for any cell of the free-space surface and any two points on neighboring or opposite border intervals of the cell (which thus define two line segments), there is always a monotone (but not necessarily increasing) parametrization within the cell, so that the maximum distance is at one of the two points. Hence we only need to worry about the
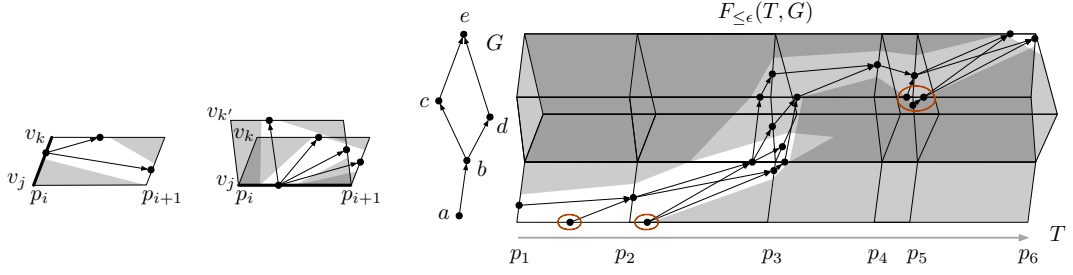
Figure 4 (left) A vertical and a horizontal interval of a free-space graph (in bold) with their vertex and their successors. We draw the vertex at the position within the interval that represents the point on the arc closest to the crumb, or the point on the crumb line closest to the vertex, respectively. Note how two of the arcs go backwards with respect to one of the parameters, so they correspond to non-monotone parametrizations. (right) The partial interval graph of the free-space diagram for a distance threshold ϵ. Note that there are multiple paths from $(p_1, (a, b))$ to $(p_5, (c, e))$ (differing around p_3) that each correspond to a vertex-monotone Fréchet distance map matching. Also note the five vertices circled in red that are in the free-space but are not reachable from the only starting vertex $(p_1, (a, b))$.

intersections of the parametrization with the grid lines of the free-space diagram. As vertex-monotonicity allows us to pick any parametrization point on an interval irrespective of the parametrization point on previous intervals, we can always pick the point that corresponds to the projection of the crumb onto the arc (for vertical intervals) or of the vertex onto the crumb line (for horizontal intervals), which is the interval weight we defined above.

4.2.3   Implementation

Let us now dive into the implementation of this bottleneck path search. We start by searching for all roads that are within distance D of the starting crumb p_1. Note that this is the only spatial query in our algorithm (unlike HMM, which searches for nearby roads for every crumb). For all these candidate roads, we build their vertical intervals and insert them into a priority queue keyed by the interval weight. We now run a modified version of Dijkstra's single-source shortest path algorithm [16] to find the minimum bottleneck path to any vertical interval containing p_n. Once we find such a path, we keep exploring the graph for as long as the bottleneck does not increase. If at any point the bottleneck reaches D, we abort the search.

It is important to note that we build the interval graph on the fly, i.e., we do not enumerate all interval vertices and arcs of the free-space diagram, but only traverse those reachable within distance d^*_{VMFD} from the start. We store all index triples of the reachable intervals in a hash set to be able to quickly constrain our later path optimization to this subgraph.

4.2.4   Running time

The running time of this distance computation depends on two things: the initial spatial index lookup and the bottleneck path search. We use a Geohash-based [33] hash-table lookup as the spatial index, which for a constant radius D looks up a constant number of hash-table entries and thus runs in time linear to the number of roads returned. In particular, the running time is independent of $|A|$, the size of the graph. The bottleneck path search runs in time $\mathcal{O}(n^* \log n^* + m^*)$, where n^* and m^* are the number of vertices and arcs of the free-space diagram $F_{\leq d^*_{\text{VMFD}}}$ being explored.

Note that while there are faster algorithms in the worst case, i.e., ways of shaving the log, for the minimum bottleneck path problem on undirected graphs (using linear-time median pivoting and shrinking of connected components, see [37]) and on directed acyclic graphs (processing the vertices in topological order voids the need for a priority queue), Djikstra's allows us easily to limit our search space on the graph, and give up if the bottleneck distance becomes too large.

It is also worth comparing our running time with that of the regular Fréchet distance. To observe the monotonicity constraint of the Fréchet distance, we can not always pick the point with the smallest distance along the interval of the free-space. In fact, the range of parametrization points that is feasible on an interval depends on the choice on the previous interval as well as on the final Fréchet distance that we target. Hence, when computing the Fréchet distance, one usually solves the decision problem, i.e., computes the earliest reachable point on each interval for a given ϵ, and then performs a binary or parametric search to find d_{FD}^*. For that decision problem no priority queue is needed, so the running time can be bounded by $\mathcal{O}(\log(D) \cdot (n' + m'))$ where n' and m' are the number of vertices and arcs of the free-space diagram of $F_{\leq D}$. To stress why computing the vertex-monotone Fréchet distance is much faster than the Fréchet distance, it is important to note that the running times for the two distance measures do not just replace the log of the binary search with the log of the priority queue, but that computing d_{VMFD}^* only involves looking at $F_{\leq d_{\mathrm{VMFD}}^*}$ while computing d_{FD}^* involves looking at $F_{\leq 2d_{\mathrm{FD}}^*}$ or even $F_{\leq D}$ depending on the implementation of the parameter search. As the complexity of the reachable free-space grows roughly quadratic with the search radius, overshooting the search radius when solving the decision problem can heavily influence the running time.

4.3 Path optimization

After determining d_{VMFD}^*, the second step of our algorithm is to select the best path P^* among all those with $d_{\mathrm{VMFD}}(T, P) = d_{\mathrm{VMFD}}^*$.

For this tie-breaking step, we minimize a global weight function inspired by Wei et al. [48, 49], namely

$$w(P) = \sum_{i=1}^{n} (\Delta_i + \Delta_{i+1}) \, \|x_i - p_i\|_2 + \alpha \sum_{i=1}^{n-1} l_i, \tag{13}$$

where α is a constant, $\Delta_i = \|p_{i-1} - p_i\|_2$, x_i is the point in the graph where p_i gets matched to and l_i is the length of the matched path between x_i and x_{i+1}. For the sentinel cases, we use $\Delta_1 = \Delta_{n+1} = \beta$, for some constant β.

The first sum in $w(P)$ measures the closeness of the matched path to the trace, trying to encourage P to follow T not just at the extreme point (where d_{VMFD}^* is determined) but also everywhere else. By weighing each crumb matching distance with the distance to the previous and next crumb, we weigh those crumbs higher where the user was moving faster or the sampling frequency is lower, which makes this formula independently of the sampling rate (similar to the t_i term by Wei et al.).

The second sum of $w(P)$ measures the length of the matching path P, discouraging P from taking local detours in areas where the free-space allows multiple options. We found that setting $\alpha = D$ and $\beta = 8D$ works well, basically reducing the number of parameters of our whole algorithm to a single one D, which can easily be set based on the maximum GPS error one expects to see in T.

One intuition for why we like this weight formula is that in the simple case where T and P are both straight horizontal lines with a constant vertical offset of a "typical" GPS error of $\frac{D}{2}$ between them, both summands correspond to (roughly) twice the size of the area of

the rectangle spanned by T and P independent of the sampling rate or driving speed. While the qualities of such a weight function are undeniably subjective, we argue that ours is a slightly more natural way to trade off trace proximity and path length than the one by Wei et al., especially since it involves the same units (meters squared) on both sides of the sum.

4.3.1 Shortest path interpretation

This weight $w(P)$ can be optimized using a shortest-path computation on the reachable free-space of $F_{\leq d^*_{\text{VMFD}}}$. The summands $(\|p_{i-1} - p_i\|_2 + \|p_{i+1} - p_i\|_2) \|x_i - p_i\|_2$ correspond one-to-one to vertex weights for all the n vertical intervals on every P. The summands for $\alpha \cdot l_i$ can be split up to arc weights on the interval graph, where each interval arc measures the progress (forward or backward) along the graph arc involved. As an example, an interval arc from one horizontal interval to another has to account for the entire length of the arc in the graph. Using the hash set of reachable intervals from the first step, we can again use Dijkstra's single-source shortest path algorithm to find P^* as well as all the x_i in time $\mathcal{O}(n^* \log n^* + m^*)$. Note that in our implementation, we do not use Fibonacci heaps for the priority queue, so the asymptotic running time of our implementation is in fact $\mathcal{O}(n^* + m^* \log n^*)$.

4.4 Trace simplification

In this section, we describe an additional, optional speed-up technique in our implementation: if the temporal and spatial information in the trace T indicates that the user was travelling at a constant speed and direction for several crumbs, we can reduce the complexity of the free-space diagram by subsampling the trace before invoking the map-matching algorithm. If we then interpolate the resulting map matching, we can expect the upsampled x_i on P to closely match the positions we would have gotten when map matching the original trace.

There are many well-known curve simplification algorithms and the addition of a time component is straightforward for many of them. In particular, for a simplification threshold of 1 meter, we experimented with the algorithm by Ramer [40], Douglas and Peucker [17], the one by Reumann and Witkam [41] and a dynamic programming approach to optimally select the minimum number of points. In the end, we found the following approach to give a very good trade-off between the time spent simplifying the trace and the time saved in map matching (which closely correlates to the number of crumbs dropped). Our algorithm closely resembles the algorithm FrechetSimp by Agarwal et al. [1, Section 3.1], but instead of preserving the Fréchet distance, we also preserve the speed of the trace, which can be important in many applications. We found that simplification with a threshold of just one meter at any provided timestamp resulted in a significant reduction in runtime.

4.4.1 Doubling-search simplification

This algorithm scans through the trace from beginning to end and incrementally decides which crumbs to keep and which ones to skip. It does so with a slow-start binary search, i.e., it first tries skipping 1, then 2, then 4, then 8, . . . , points until it fails (i.e., at least one of the skipped crumbs would be interpolated more than 1 meter away from its original position) and then uses regular binary search to find how many points can be skipped. Note that the predicate "can x points be skipped?" is not monotone in x, meaning that it might be possible to skip 6 points but not 5 points. In that sense, this algorithm is doing a best effort but is not guaranteed to greedily skip as many points as possible. What is guaranteed though is that if for some x we have that for all $y \leq x$, "y points can be skipped" holds, then the algorithm will skip at least x points (it might get lucky and skip even more points).

The running time of this algorithm is $\mathcal{O}(n \log n)$: The cost for checking whether x points can be skipped is $\mathcal{O}(x)$ and whenever we skip y points, the cost for doing so is bounded by the slow-start in $\mathcal{O}(1) + \mathcal{O}(2) + \mathcal{O}(4) + \cdots + \mathcal{O}(y) + \mathcal{O}(2y) = \mathcal{O}(y)$ and the binary search in $\mathcal{O}(y \log y)$. Note that this worst case only occurs if a long stretch of points can be skipped (in which case it is offset by time savings in the map matcher). However, the running time drops to linear if for example at least every tenth point has to be taken. We refer to Figure 5 for illustrations of this trace simplification.
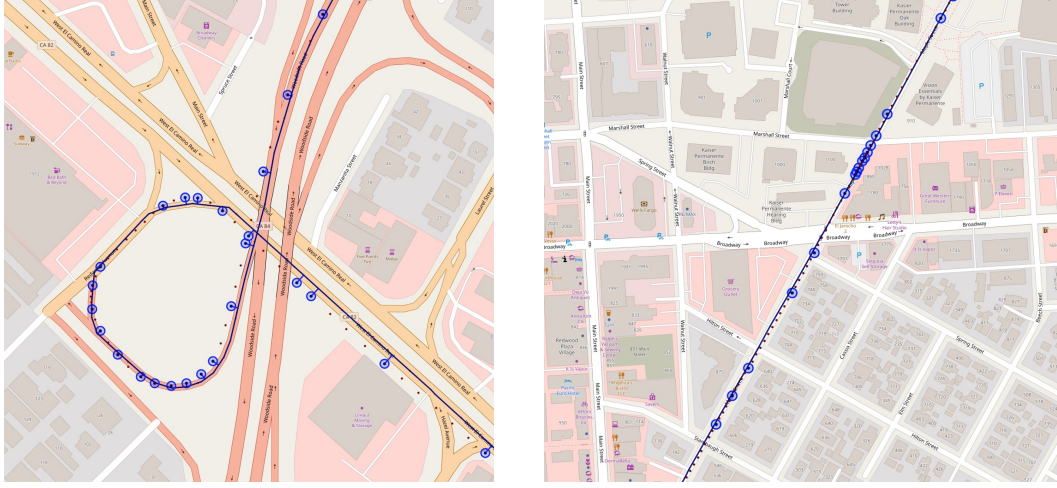
Figure 5 (left) This shows the spatial component of our trace simplification. On the straight sections many crumbs get skipped, while in the curved parts of the loop most crumbs are kept to ensure an accurate interpolation. (right) This shows the temporal component of our trace simplification. While the entire trace shown is a straight line, the driver had to stop at the intersection with Marshall Street. Therefore, many crumbs are retained there, whereas up to ten crumbs are skipped in the area where the car was driving at a uniform speed.

5   Experiments

Table 1 Experimental results: map matching times normalized to microseconds per crumb (shown as mean $\pm$ standard deviation). Lower is better. The algorithm leveraging vertex-monotone Fréchet distance (VMFD) is roughly 4 times faster than the regular Fréchet distance (FD) implementation, and one to two orders of magnitude faster than the GraphHopper HMM implementation. The addition of our geometric trace simplification method (columns marked with +S) results in a roughly two-fold speedup for the Fréchet distance-based methods, while only a small improvement can be observed for the GraphHopper HMM.

	Map matching time in μs per crumb					
	VMFD		FD		HMM	
Data set	+S		+S		+S	
SF Bay Area	1.9 ± 3.5	4.4 ± 7.1	6.8 ± 11	19.2 ± 31	519	662
Hong Kong	1.2 ± 2.2	2.3 ± 5.4	4.0 ± 10	8.4 ± 18	51	60

The main focus of our experimental evaluation is on performance. While processing times for a single trace matter, for most of our applications throughput matters most. In the following, we are reporting processing times normalized by crumb for two data sets, each with thousands of traces and millions of crumbs.

■ **Table 2** Experimental results for the Bay Area data set for different maximum (vertex-monotone) Fréchet distances D: (left) Map matching times normalized to microseconds per crumb (shown as mean ± standard deviation). Lower is better. (right) Number of matching traces (out of 7736). The vertex-monotone Fréchet distance (VMFD) consistently outperforms the regular Fréchet distance (FD) by roughly a factor of 4, both with (+S) and without simplification.

| | Map matching time in μs per crumb | | | | Number of matching traces | | | |
| | VMFD | | FD | | VMFD | | FD | |
D	+S		+S		+S		+S	
4m	0.5 ± 0.5	0.4 ± 0.5	0.9 ± 1.3	1.0 ± 1.2	74	68	72	66
8m	0.6 ± 5.3	0.5 ± 0.6	1.3 ± 2.1	1.5 ± 1.9	407	408	403	402
16m	0.8 ± 5.8	0.9 ± 1.3	2.2 ± 4.0	3.3 ± 4.9	2297	2298	2287	2288
32m	1.2 ± 5.6	1.9 ± 1.3	4.0 ± 6.5	7.9 ± 11	4072	4067	4057	4051
64m	1.9 ± 3.5	4.4 ± 7.1	6.8 ± 11	19.2 ± 31	5467	5463	5456	5454
128m	3.6 ± 5.7	11.5 ± 24	13.1 ± 26	48.8 ± 102	6218	6216	6217	6215
256m	10.3 ± 23	38.3 ± 86	39.2 ± 104	166 ± 405	6875	6873	6875	6873

■ **Table 3** Experimental results for the Hong Kong data set: (left) Map matching times normalized to microseconds per crumb (shown as mean ± standard deviation). Lower is better. (right) Number of matching traces (out of 3889). The vertex-monotone Fréchet distance (VMFD) consistently outperforms the regular Fréchet distance (FD) by roughly a factor of 4, both with (+S) and without simplification.

| | Map matching time in μs per crumb | | | | Number of matching traces | | | |
| | VMFD | | FD | | VMFD | | FD | |
D	+S		+S		+S		+S	
4m	0.4 ± 0.4	0.3 ± 0.4	0.7 ± 0.9	0.8 ± 0.9	10	8	10	8
8m	0.4 ± 0.4	0.3 ± 0.4	0.9 ± 1.3	1.2 ± 1.7	80	81	79	80
16m	0.5 ± 0.6	0.5 ± 0.7	1.3 ± 2.3	1.8 ± 3.0	377	376	377	376
32m	0.7 ± 1.0	1.0 ± 1.8	2.2 ± 4.8	3.6 ± 7.2	760	759	759	758
64m	1.2 ± 2.2	2.3 ± 5.4	4.0 ± 10	8.4 ± 18	1139	1137	1135	1133
128m	2.7 ± 5.6	6.2 ± 11	10.4 ± 29	27.3 ± 59	1611	1606	1610	1605
256m	7.5 ± 15	22.9 ± 49	30.0 ± 78	108 ± 256	2186	2178	2183	2175

5.1   Data sets

We used two sets of traces for our performance experiments. Both sets consist of Open-StreetMap traces that we downloaded using JOSM [29]. We preprocessed both trace sets by splitting traces at any gap of more than 15 seconds to ensure a high sampling rate and by only keeping traces with at least 60 crumbs so that they provide reasonable context for disambiguation. The first set consists of 7736 traces in the San Francisco Bay Area with a total of 3.62M crumbs. The second set consists of 3889 traces in Hong Kong, totaling at 1.63M crumbs. For the maps in our experiment, we extracted the street network out of OpenStreetMap osm.pbf files [22, 38] using RoutingKit [14, 15]. Note that this extraction step eliminates non-drivable segments such as hiking paths. The map of Northern California contains 11.7M vertices and 23.6M arcs, and the map of Hong Kong contains 280k vertices and 470k arcs.

5.2   Algorithms

We compare three algorithms, all implemented in Java:

1. vertex-monotone Fréchet distance map matching algorithm as described in Section 4 with $D = 64$ meters (and varying D later)
2. Fréchet distance map matching algorithm with the same maximum distance $D = 64$ meters
3. HMM algorithm [32] in the open-source library GraphHopper [24] (default settings)

We ran all experiments on a MacBook Pro from 2019 with a 2.3 GHz 8-core Intel Core i9 CPU and 32 GB RAM. Our implementation is single threaded, however, so only one core is being used. We ran all algorithms 3 times in a row and used the measurements of the last run.

5.3   Quality

We tested our algorithm on the ACM GIS Cup 2012 [3] data set, with corrections from Wei et al. [48, 49]. We disabled preprocessors and simplification, as the data set required a matched road for each crumb. Our algorithm resulted in a 97.75% match rate, which is in line with match rates reported in [3, 48, 49]. We visually inspected the differences from the provided ground truth, and they appeared to be ambiguous from a geometric standpoint. If certain modeling assumptions are desired, the optimization function can be modified to include them, and we leave that open for future work. The relaxation to vertex monotonicity can sometimes alter map matching results at the end points of segments, for example when the GPS signals fluctuate back and forth for a few meters for cars waiting at traffic lights, but the sequence of road segments matched does not change.

We also noticed that in some scenarios, such as shifted GPS locations in urban canyons, using a Fréchet distance type distance as our primary objective function has significant advantages over relying on distributional assumptions such as those in a HMM. Error distributions may not be independent, and modeling assumptions that rely on independence can result in unexpected map matching results, as for example in Figure 3.

5.4   Performance results

For an overview of experimental results, see Table 1.

Without trace simplification, we measured $4.4\mu s$ per crumb for our algorithm and $662\mu s$ per crumb for the GraphHopper HMM on the Bay Area traces. On the OSM Hong Kong traces, we measured $2.3\mu s$ per crumb for our algorithm and $60\mu s$ per crumb for the GraphHopper HMM. Therefore, our implementation is one to two orders of magnitude faster than the GraphHopper HMM.

For the regular Fréchet distance, we implemented a search procedure using a slow-start binary search up to at most 64 meters and down to a precision of 1 meter. We found the vertex-monotone Fréchet distance implementation to be roughly 4 times faster than the regular Fréchet distance implementation.

We also measured the impact of the trace simplification described in Section 4.4 and report a roughly two-fold speedup for an accuracy of 1 meter which results in about 2 out of 3 crumbs being dropped. Note that both vertex-monotone Fréchet distance and the regular Fréchet distance benefit from simplification. For completeness, we also ran the GraphHopper HMM on the simplified traces. As the simplification is a geometric simplification, both Fréchet distance map matching variants are fairly stable, whereas there is less reason to

■ **Figure 6** Box plots of the running times per crumb for the traces in the Bay Area dataset grouped by their (vertex-monotone) Fréchet distance. The last column [unm.] contains the traces that did not match under the maximum distance of $D = 64$ meters. We observe that the running time per crumb grows relatively slowly with increasing Fréchet distance. Furthermore, the traces that do not match are faster to process than the ones that do.

believe that the HMM model would be stable under this simplification. As can be seen in Table 1, it also turned out that the GraphHopper HMM does not benefit much from simplification, most likely because the overall lengths of paths being computed does not get reduced by much.

Finally, we also studied the effect of decreasing and increasing the maximum Fréchet distance parameter D from our default value of 64 meters. As Tables 2 and 3 show, the map matching time grows superlinear with D while the number of additional matched traces does not. Especially in the Bay Area data set, where most traces already match with 32 or 64 meters, increasing to larger D is not worth the extra computational effort in many applications. Manual inspection of the additionally matched traces with $D > 64$ meters showed that these are almost exclusively traces of hiking, biking, ferry, cablecar or train trips which therefore justifiably do not match against the street network at $D = 64$ meters. We also provide insight into the distribution of processing times by Fréchet distance in Figures 6 and 7.

5.5   Literature comparison

While run on different graphs, traces and computers, and possibly implemented with different programming languages, some previous papers report throughput numbers in their experimental sections, which we list here.

- For a sampling interval of 1 second, Wei et al. [48, 49] report a map matching time of around $400\,000\mu s$ per crumb for their Fréchet distance implementation and of about $1000\mu s$ per crumb for the HMM by Newson and Krumm [32] (numbers from [48, page 7, logarithmic plot]).
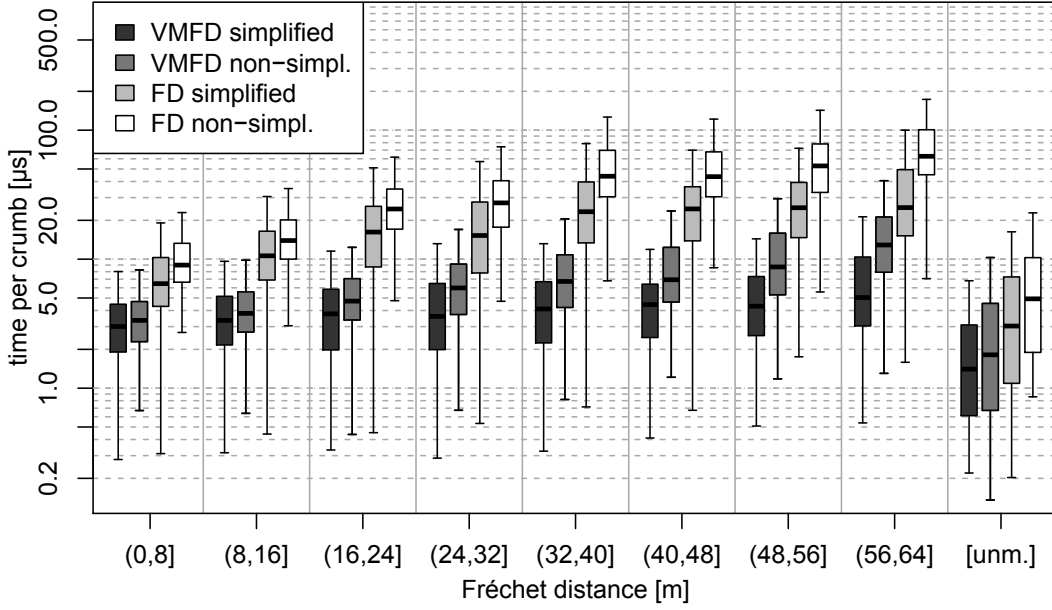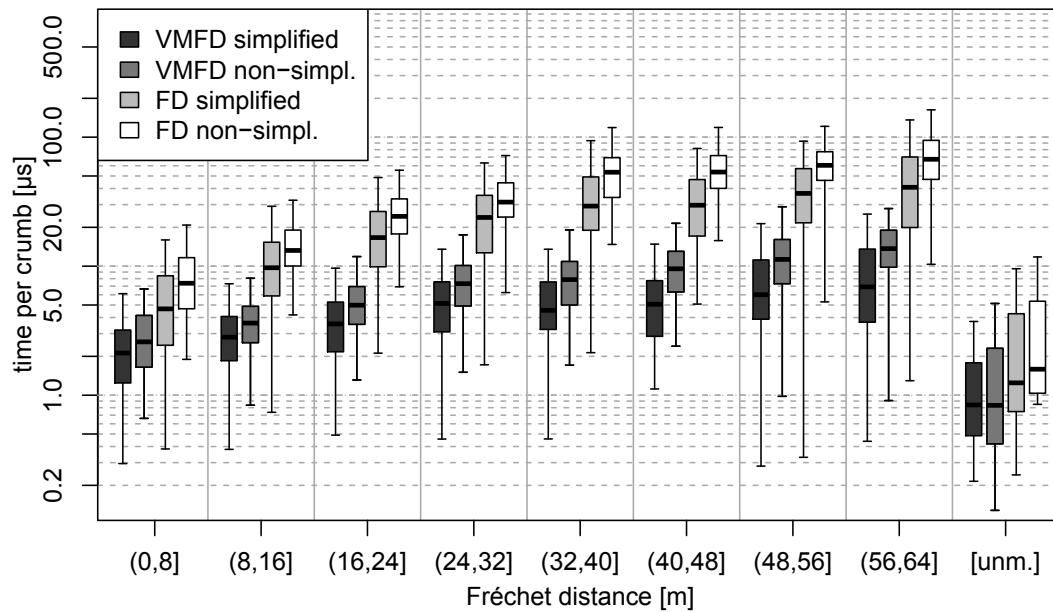
Figure 7 Box plots of the running times per crumb for the traces in the Hong Kong dataset grouped by their (vertex-monotone) Fréchet distance. The last column [unm.] contains the traces that did not match under the maximum distance of $D = 64$ meters. Same observation as for Figure 6.

- Tang et al. [45], the winners of the SIGSPATIAL cup 2012 on map matching, report a time of about $67\mu s$ per crumb [45, page 4, Figure 2].
- Finally, Yang and Gidofalvi [51] measured their optimized HMM at 22 to $40\mu s$ per crumb.

Many of these algorithms are implemented in C++, which allows for significantly more optimization when compared with our Java implementation. Even so, our algorithm exceeds previously reported throughputs by an order of magnitude and with its geometric guarantees is a strong alternative to HMM-based algorithms.

6 Conclusion

We propose the metric of vertex-monotone Fréchet distance as an effective alternative for efficient map matching. We show that Fréchet distance-based map matching can be fast even while combining the power of geometric optimization with global weight methods. Our algorithm requires very little parameter tuning and does not make strong assumptions on the distribution of GPS errors.

References

1 Pankaj K Agarwal, Sariel Har-Peled, Nabil H Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005.

2 Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19(3):601–632, 2015.

3 Mohamed Ali, John Krumm, Travis Rautman, and Ankur Teredesai. ACM SIGSPATIAL GIS cup 2012. In *20th International Conference on Advances in Geographic Information Systems, SIGSPATIAL'12*, pages 597–600, 2012.

4   Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. Matching planar maps. *J. Algorithms*, 49(2):262–283, 2003. Announced at SODA'03.

5   Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995.

6   Javed A. Aslam, Sejoon Lim, Xinghao Pan, and Daniela Rus. City-scale traffic estimation from a roving sensor network. In *10th ACM Conference on Embedded Network Sensor Systems, SenSys'12*, pages 141–154. ACM, 2012.

7   Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *31st International Conference on Very Large Data Bases, VLDB'05*, pages 853–864. ACM, 2005.

8   Lili Cao and John Krumm. From GPS traces to a routable road map. In *17th International Symposium on Advances in Geographic Information Systems, SIGSPATIAL'09*, pages 3–12. ACM, 2009.

9   Pablo Samuel Castro, Daqing Zhang, and Shijian Li. Urban traffic modelling and prediction using large scale taxi GPS traces. In *10th International Conference on Pervasive Computing, Pervasive'12*, volume 7319 of *Lecture Notes in Computer Science*, pages 57–72. Springer, 2012.

10   Erin W. Chambers, Brittany Terese Fasy, Yusu Wang, and Carola Wenk. Map-matching using shortest paths. *ACM Transactions on Spatial Algorithms and Systems*, 6(1):6:1–6:17, 2020.

11   Daniel Chen, Anne Driemel, Leonidas J. Guibas, Andy Nguyen, and Carola Wenk. Approximate map matching with respect to the Fréchet distance. In *14th Workshop on Algorithm Engineering and Experiments, ALENEX'11*, pages 75–83, 2011.

12   Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. *Transportation Science*, 51(2):566–591, 2017.

13   Daniel Delling, Dennis Schieferdecker, and Christian Sommer. Traffic-aware routing in road networks. In *34th IEEE International Conference on Data Engineering, ICDE'18*, pages 1543–1548, 2018.

14   Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *ACM J. Exp. Algorithmics*, 21(1):1.5:1–1.5:49, 2016.

15   Julian Dibbelt, Ben Strasser, and Dorothea Wagner. RoutingKit. `https://github.com/RoutingKit/RoutingKit`, 2020.

16   Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

17   David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

18   Gregory D. Erhardt, Sneha Roy, Drew Cooper, Bhargava Sana, Mei Chen, and Joe Castiglione. Do transportation network companies decrease or increase congestion? *Science Advances*, 5(5), 2019.

19   Nivan Ferreira, Jorge Poco, Huy T. Vo, Juliana Freire, and Cláudio T. Silva. Visual exploration of big spatio-temporal urban data: A study of new york city taxi trips. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2149–2158, 2013.

20   M Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 22(1):1–72, 1906.

21   Stefan Funke, Tobias Rupp, André Nusser, and Sabine Storandt. PATHFINDER: storage and indexing of massive trajectory sets. In *16th International Symposium on Spatial and Temporal Databases, SSTD'19*, pages 90–99. ACM, 2019.

22   Geofabrik GmbH. Geofabrik OSM NorCal map. `https://download.geofabrik.de/north-america/us/california/norcal.html`, 2020.

23   Chong Yang Goh, Justin Dauwels, Nikola Mitrovic, Muhammad Tayyab Asif, Ali Oran, and Patrick Jaillet. Online map-matching based on hidden Markov model for real-time traffic sensing applications. In *15th International IEEE Conference on Intelligent Transportation Systems, ITSC'12*, pages 776–781, 2012.

24   GraphHopper. Map matching based on graphhopper. `https://github.com/graphhopper/map-matching`, 2020.

25   Songtao He, Favyen Bastani, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. Roadrunner: improving the precision of road network inference from GPS trajectories. In *26th International Conference on Advances in Geographic Information Systems, SIGSPATIAL'18*, pages 3–12. ACM, 2018.

26   Abdeltawab M. Hendawi, Sree Sindhu Sabbineni, Jianwei Shen, Yaxiao Song, Peiwei Cao, Zhihong Zhang, John Krumm, and Mohamed H. Ali. Which one is correct, the map or the GPS trace. In *27th International Conference on Advances in Geographic Information Systems, SIGSPATIAL'19*, pages 472–475. ACM, 2019.

27   Roel Jacobs. Constructing maps by clustering trajectories. Master's thesis, TU Eindhoven, 2016.

28   Erik Jenelius and Haris N. Koutsopoulos. Travel time estimation for urban road networks using low frequency probe vehicle data. *Transportation Research Part B: Methodological*, 53:64–81, 2013.

29   JOSM. An extensible editor for openstreetmap. `https://josm.openstreetmap.de`, 2020.

30   Benjamin B. Krogh, Christian S. Jensen, and Kristian Torp. Efficient in-memory indexing of network-constrained trajectories. In *24th ACM International Conference on Advances in Geographic Information Systems SIGSPATIAL'16*, pages 17:1–17:10. ACM, 2016.

31   Yang Li, Qixing Huang, Michael Kerber, Lin Zhang, and Leonidas J. Guibas. Large-scale joint map matching of GPS traces. In *21st International Conference on Advances in Geographic Information Systems, SIGSPATIAL'13*, pages 214–223. ACM, 2013.

32   Paul Newson and John Krumm. Hidden Markov map matching through noise and sparseness. In *17th ACM International Symposium on Advances in Geographic Information Systems, SIGSPATIAL'09*, pages 336–343. ACM, 2009.

33   Gustavo Niemeyer. geohash.org is public! `https://web.archive.org/web/20080305223755/http://blog.labix.org/#post-85`, 2008.

34   Patrick Niklaus. Matching GPS traces to a map. `https://blog.mapbox.com/matching-gps-traces-to-a-map-73730197d0e2`, 2015.

35   Bei Pan, Yu Zheng, David Wilkie, and Cyrus Shahabi. Crowd sensing of traffic anomalies based on human mobility and social media. In *21st International Conference on Advances in Geographic Information Systems, SIGSPATIAL'13*, pages 334–343. ACM, 2013.

36   Gang Pan, Guande Qi, Wangsheng Zhang, Shijian Li, Zhaohui Wu, and Laurence Tianruo Yang. Trace analysis and mining for smart cities: issues, methods, and applications. *IEEE Communications Magazine*, 51(6), 2013.

37   Matthias Peinhardt and Volker Kaibel. On the bottleneck shortest path problem. Technical report, Technical Report ZIB-Report 06-22, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2006.

38   OpenStreetMap project. OSM extract Hong Kong. `http://download.openstreetmap.fr/extracts/asia/china/`, 2020.

39   Mohammed A. Quddus, Washington Y. Ochieng, and Robert B. Noland. Current map-matching algorithms for transport applications: State-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15(5):312–328, 2007.

40   Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256, 1972.

41   K Reumann and APM Witkam. Optimizing curve segmantation in computer graphics. In *International Computing Symposium 1973*, pages 467–472, 1974.

42   Stefan Schrödl, Kiri Wagstaff, Seth Rogers, Pat Langley, and Christopher Wilson. Mining GPS traces for map refinement. *Data Mining and Knowledge Discovery*, 9(1):59–87, 2004.

43   Dominik Schultes and Peter Sanders. Dynamic highway-node routing. In *6th International Workshop on Experimental Algorithms, WEA'07*, volume 4525 of *Lecture Notes in Computer Science*, pages 66–79. Springer, 2007.

44   Junichi Shigezumi, Tatsuya Asai, Hiroaki Morikawa, and Hiroya Inakoshi. A fast algorithm for matching planar maps with minimum Fréchet distances. In *4th International ACM Workshop on Analytics for Big Geospatial Data, BigSpatial@SIGSPATIAL'15*, pages 25–34, 2015.

45   Youze Tang, Andy Diwen Zhu, and Xiaokui Xiao. An efficient algorithm for mapping vehicle trajectories onto road networks. In *International Conference on Advances in Geographic Information Systems, SIGSPATIAL'12*, pages 601–604. ACM, 2012.

46   Rolf van Leusden. A novel algorithm for computing the Fréchet distance. Master's thesis, TU Eindhoven, 2013.

47   Yilun Wang, Yu Zheng, and Yexiang Xue. Travel time estimation of a path using sparse trajectories. In *20th ACM International Conference on Knowledge Discovery and Data Mining, KDD'14*, pages 25–34. ACM, 2014.

48   Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. Map matching by Fréchet distance and global weight optimization. *Technical Paper*, page 19, 2013.

49   Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. Map matching: comparison of approaches using sparse and noisy data. In *21st International Conference on Advances in Geographic Information Systems, SIGSPATIAL'13*, pages 434–437. ACM, 2013.

50   Carola Wenk, Randall Salas, and Dieter Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *18th International Conference on Scientific and Statistical Database Management, SSDBM'06*, pages 379–388, 2006.

51   Can Yang and Gyozo Gidofalvi. Fast map matching, an algorithm integrating hidden Markov model with precomputation. *International Journal of Geographical Information Science*, 32(3):547–570, 2018.

52   Yu Zheng. Trajectory data mining: An overview. *ACM Transactions on Intelligent Systems and Technology*, 6(3):29:1–29:41, 2015.

53   Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from GPS trajectories. In *18th International Conference on World Wide Web, WWW'09*, pages 791–800. ACM, 2009.

Robustness Generalizations of the Shortest Feasible Path Problem for Electric Vehicles

Payas Rajan ✉ 🆔
Department of Computer Science & Engineering,
University of California, Riverside, CA, USA

Moritz Baum ✉
Apple, Cupertino, CA, USA

Michael Wegner ✉
Apple, Cupertino, CA, USA

Tobias Zündorf ✉
Apple, Cupertino, CA, USA

Christian J. West ✉
Apple, Cupertino, CA, USA

Dennis Schieferdecker ✉
Apple, Cupertino, CA, USA

Daniel Delling ✉
Apple, Cupertino, CA, USA

───── **Abstract** ─────

Electric Vehicle routing is often modeled as a Shortest Feasible Path Problem (SFPP), which minimizes total travel time while maintaining a non-zero State of Charge (SoC) along the route. However, the problem assumes perfect information about energy consumption and charging stations, which are difficult to even estimate in practice. Further, drivers might have varying risk tolerances for different trips. To overcome these limitations, we propose two generalizations to the SFPP; they compute the shortest feasible path for *any* initial SoC and, respectively, for *every* possible minimum SoC threshold. We present algorithmic solutions for each problem, and provide two constructs: *Starting Charge Maps* and *Buffer Maps*, which represent the tradeoffs between robustness of feasible routes and their travel times. The two constructs are useful in many ways, including presenting alternate routes or providing charging prompts to users. We evaluate the performance of our algorithms on realistic input instances.

1 Introduction

Several factors can cause an Electric Vehicle (EV) to get stranded along a route: They often have shorter ranges than internal combustion (IC) vehicles, charging stations can be sparse and fragmented among different providers. For drivers, this *stranding risk* manifests as *range anxiety* and *range stress* [18, 25, 40, 41, 44, 46]. To alleviate range anxiety, route planning for EVs must consider battery constraints while selecting routes [19, 33, 34, 48, 49].

Previous work [7, 8] models EV routing with charging stops as the NP-hard Shortest Feasible Path Problem (SFPP): Given a road network modeled as a weighted, directed graph with energy consumptions and travel times on each edge; charging stations on a subset of vertices and their respective concave charging functions; a source vertex, a destination vertex and a starting battery SoC, find a path that minimizes the total travel time including charging time while maintaining a non-zero battery SoC at all points along the route. The Charging Function Propagation (CFP) algorithm solves SFPP in exponential time and space.

In practice, however, the shortest feasible path might not be sufficient. First, the energy consumptions on edges are derived from estimation models that are not perfectly accurate [14, 20, 42, 43]. Second, the energy consumption of an EV depends on several factors that

are difficult to even estimate: driver aggressiveness, age of the battery, wear and tear of the EV. Each of these factors can affect the energy consumption significantly [1, 22]. Third, users may have varying risk tolerances, and thus a one-size-fits-all approach is not sufficient to alleviate range anxiety when serving routes for a large number of EV drivers.

In this work, we introduce two generalizations of SFPP which are used to compute two constructs, the *Starting Charge Map (SCM)* and *Buffer Map (BM)*. Both *SCM* and *BM* are computed between a source vertex s and target vertex t. Evaluating SCM_{st} for a valid starting SoC β_s gives the corresponding shortest feasible path between s and t, while evaluating BM_{st} for buffer energy $\mathfrak{b}$ returns a shortest feasible path where the SoC is guaranteed to never drop below $\mathfrak{b}$ along the route.

The *SCM* and *BM* allow route planning systems to access a larger set of alternative feasible paths than the standard CFP algorithm, which returns only a single feasible path. This variety in paths has several applications–recommending EV drivers alternative routes, generating suggestions like charging extra at s to save travel time, or letting users choose the degree of acceptable risk for a trip. Both problems can be solved by brute force approaches that run CFP for all possible values of β_s or $\mathfrak{b}$. However, since β_s or $\mathfrak{b}$ can take an infinite number of possible values, such an approach would simply not terminate. In this paper, we make the following contributions:

- We introduce the *Starting Charge Map (SCM)* and *Buffer Map (BM)* that encapsulate a set of alternative routes to help alleviate range anxiety for a wide variety of EV drivers.
- Computing *SCM* and *BM* using standard CFP requires several expensive runs of the algorithm. We present fast, exact algorithms that compute the two abstractions with acceptable real-time performance on large graphs.
- We evaluate our algorithms on realistic instances, using real-world road networks of California and Oregon, an energy consumption model taken from a Nissan Leaf 2013 [20], and a dataset of public EV charging stations [2]. Our results show good performance even without the use of preprocessing techniques for shortest path queries.

2   Related Work

Most current EVs suffice for a majority of trips that drivers take, as shown in [39]. However, *range anxiety*, defined as an EV driver's fear of getting stranded along a route is often cited as a major hindrance to widespread EV adoption [24, 26]. Prior work shows that perceived range anxiety is inversely related to the degree of drivers' *trust* in the EVs [31, 44, 25, 46, 32]. Route planning for EVs, therefore, has two objectives: Minimize travel times under battery constraints, and reduce *surprise* for the driver to minimize range anxiety.

Early works on EV route planning like [3, 45] consider the problem of minimizing energy consumption along routes insted of standard route planning formulations that minimize travel time [4]. Since then, many additions have been proposed to the EV routing problem to make it more realistic. Several newer variants consider battery-swapping stations [19] or charging functions [7, 12, 37, 50]. Some works [29, 48, 49] model EV routing as a multicriteria Dijkstra's search [35], which returns a set of pareto-optimal routes that are not dominated in either travel time or energy consumption. Conversely, some other works like [7, 12] present EV routing as an extension of the Constrained Shortest Path problem. These problems seek to minimize total travel time including charging time, while constraining the total energy consumption of paths to levels allowed by realistic battery capacities. Another line of research considers "profile queries", which look for all optimal shortest paths depending on a certain state [47], e.g., the initial state of charge of an EV [10, 13] or the current point in time [11, 17, 23].

Underlying all EV routing algorithms is an assumption that the energy consumptions assigned to graph edges are accurate. In practice, this is difficult to achieve with existing energy consumption models [14, 20, 21, 42, 43, 36]. EV energy consumption is affected by several factors including traffic conditions, driver aggressiveness, battery health and regular wear-and-tear of the vehicle, which are hard to estimate. Recently, [1] showed that each of these factors can impact the energy consumption along short routes by as much as 40%. Similarly, [43] show a high variance in EV energy consumptions for short trips. To mitigate the effects of inaccurate estimates, [46] recommend holding a *safety margin* between 12 and 23% of battery capacity. Only few EV routing algorithms [22, 30] accommodate buffer energy for variance in energy consumption estimates or provide robust routes.

3 Preliminaries

Our setup is similar to the standard shortest feasible path problem [7, 8]. We consider a road network modeled as directed graph $G = \langle V, E \rangle$, with V the set of vertices and $E : V \times V$ the set of edges. We are given two edge weight functions $d : E \to \mathbb{R}_{\geq 0}$ and $c : E \to \mathbb{R}$ that assign the travel time and energy consumption to each $e \in E$. An $s - t$ *path* in G is a sequence of adjacent vertices $P = [s = v_1 v_2 \ldots v_n = t]$, such that $\forall 1 \leq i \leq n, (v_i, v_{i+1}) \in E$ holds.

For a path P, the total *driving time* is $d(P) = \Sigma_{i=1}^{n-1} d(v_i, v_{i+1})$. The *consumption profile*, $f_P : [0, M] \to [-M, M] \cup \{-\infty\}$ is a function that maps the starting SoC β_s to residual SoC β_t at t after traversing P. f_P can be negative due to energy recuperation along P, or $-\infty$ if it is not possible to traverse P with starting SoC β_s. $f_P(\beta)$ can be computed using a 3-tuple $\langle \text{in}_P, \text{cost}_P, \text{max}_P \rangle$, where in_P is the minimum SoC required at s to traverse P, $\text{cost}_P = \Sigma_{i=1}^{n-1} c(v_i, v_{i+1})$, and out_P is the maximum SoC possible at t after traversing P [19]. Conversely, we define an *inverse consumption profile* $f_P^{-1} : [-M, M] \to [0, M] \cup \{\infty\}$, which maps residual SoC β_t to the starting SoC β_s. We evaluate both functions as:

$$f_P(\beta) = \begin{cases} -\infty & \text{if } \beta < \text{in}_P \\ \text{out}_P & \text{if } \beta - \text{cost}_P > \text{out}_P \\ \beta - \text{cost}_P & \text{otherwise} \end{cases} , \quad f_P^{-1}(\beta) = \begin{cases} \infty & \text{if } \beta > \text{out}_P \\ \text{in}_P & \text{if } \beta + \text{cost}_P < \text{in}_P \\ \beta + \text{cost}_P & \text{otherwise} \end{cases}$$

Let $f_\phi(\beta)$ and $f_\phi^{-1}(\beta)$ be identity SoC profiles that always map a given SoC β to itself. Given two paths $P = [v_1 v_2 \ldots v_k]$ and $Q = [v_{k+1} \ldots v_n]$, we can get the concatenation $P \circ Q = [v_1 \ldots v_k v_{k+1} \ldots v_n]$ and a linked consumption profile $f_{P \circ Q}$ as $\text{in}_{P \circ Q} = max\{\text{in}_P, \text{cost}_P + \text{in}_Q\}$, $\text{out}_{P \circ Q} = min\{\text{out}_Q, \text{out}_P - \text{cost}_Q\}$, and $\text{cost}_{P \circ Q} = max\{\text{cost}_P + \text{cost}_Q, \text{in}_P - \text{out}_Q\}$, if $\text{out}_P \geq \text{in}_Q$; otherwise, $P \circ Q$ is infeasible and $f_{P \circ Q} \equiv -\infty$. Lastly, an (inverse) SoC profile f_1 is said to *dominate* f_2 if $\forall \beta \in [0, M], f_1(\beta) \geq f_2(\beta)$.

A set $S \subseteq V$ marks the available charging stations on the road network. Each $v \in S$ is assigned a concave, monotonically increasing *charging function* $\text{cf}_v : \mathbb{R}_{\geq 0} \to [0, M]$ that maps the charging time at v to the resultant SoC after charging. Conversely, we also define the inverse charging function $\text{cf}_v^{-1} : [0, M] \to \mathbb{R}_{\geq 0}$. To obtain the time it takes to charge from β_1 to β_2, we compute $\text{cf}^{-1}(\beta_2) - \text{cf}^{-1}(\beta_1)$.

▶ **Definition 1.** *A shortest feasible path P between a source $s \in V$ and a target $t \in V$ for an EV with a starting SoC $\beta_s \in [0, M]$ is one that minimizes the total trip time (travel time + charging time) while maintaining a non-negative battery SoC at all points on P.*

For this work, we add two constraints to the original definition of charging functions: First, we require that all charging functions have a minimum initial SoC of 0 and are able to fully charge EVs to M SoC. This constraint is realistic as any real-world charging station can charge an EV with an empty battery to its full capacity. Second, similar to [7], we require all charging functions to be *piecewise linear*.

3.1   Charging Function Propagation (CFP)

CFP [7, 8] is a generalization of the bicriteria Dijkstra's algorithm [35] with two major differences: First, the set of labels at a vertex represent all possible tradeoffs between charging time and the resultant SoC after charging at the last station, and second, the decision how much to charge at a station is taken at the immediately following station the EV visits. This is because the amount of charge needed at $u \in S$ is dependent on energy consumed by the EV between u and the next station $v \in S$. If v_i and v_j are two consecutive charging stations on a path $P = [v_1 \ldots v_n]$, we call the subpath $[v_i \ldots v_j]$ a *leg* of P.

Assume that we want to find a shortest feasible path between $s, t \in V$ for starting SoC β_s. For all $v \in V$, we maintain sets $L_{uns}(v)$ for unsettled and $L_{set}(v)$ for settled labels. For vertex v, a label of the CFP search is a 4-tuple $\ell = \langle \tau_v, \beta_u, u, f_{[u\ldots v]} \rangle$ where τ_v is the total travel time from s to v except the charging time at the last charging station u, β_u is the EV's SoC on arriving at u and $f_{[u\ldots v]}$ is the consumption profile of subpath $[u \ldots v]$. The CFP search propagates through G as follows:

1. *At s:* A label $\ell = \langle 0, \beta_s, s, f_\phi \rangle$ is added to the travel time ordered min-priority queue PQ.
2. *Search reaches a non-charging vertex $v \neq t$:* Let path $P = [s = v_1 \ldots v_k = v]$ and total travel time $\tau_P = \Sigma_{i=1}^{k-1} d(v_i, v_{i+1})$. Create label $\langle \tau_P, \beta_s, s, f_P \rangle$ and add to $L_{uns}(v)$.
3. *Search reaches first charging station vertex $v \neq t$:* Let path $P = [s \ldots v]$ and total travel time over P be τ_P. Create label $\langle \tau_P, f_P(\beta_s), v, f_\phi \rangle$ and add to $L_{uns}(v)$.
4. *Search reaches a non-charging vertex $v \neq t$:* Let $\ell = \langle \tau_v, \beta_u, u, f_{[u\ldots v]} \rangle$ be the current label extracted from PQ. Since u is the last charging station, let subpath $P = [u \ldots v]$ and the total travel time over P be τ_P. Add label $\langle \tau_{[s\ldots v]}, f_{[s\ldots v]}(\beta_s), u, f_P \rangle$ to $L_{uns}(v)$.
5. *Search reaches a subsequent charging vertex $v \neq t$:* Let $\ell = \langle \tau_v, \beta_u, u, f_{[u\ldots v]} \rangle$ be the current label extracted from PQ. Since u is the last charging station, let *leg* $\mathcal{L} = [u \ldots v]$ of path $P = [s \ldots v]$, and the total travel time over P be τ_P. Compute the *SoC function* $b_\ell(\tau) := \tau_P + f_\mathcal{L}(\mathrm{cf}_u(\beta_u, \tau - \tau_P))$. Since all charging functions are assumed to be piecewise linear, it suffices to create one label per breakpoint of b_ℓ [7]. For breakpoint $B = (\tau_B, SoC_B)$, create a label $\langle \tau_B, SoC_B, v, f_\phi \rangle$ and add to $L_{uns}(v)$.
6. *Search reaches destination t:* Terminate and backtrack to extract a path from s to t.

The label sets for all $v \in V$ are used to minimise the total number of dominance checks among labels for v. L_{uns} is implemented as a min-heap with total feasible travel time as the key, and the following invariant is maintained: The minimum label ℓ in $L_{uns}(v)$ is not dominated by any label in $L_{set}(v)$. Label ℓ dominates ℓ' iff $b_\ell(\tau) \geq b_{\ell'}(\tau)$ when $\tau \geq 0$.

As the number of labels created during CFP search can be exponential, the algorithm belongs to the EXPTIME class. A combination of A* search using potential functions and Contraction Hierarchies [28] can be used to speed up CFP on large graphs in practice. When both speedup techniques are combined, the result is called the CHArge algorithm [7, 8].

4   Starting Charge Maps

▶ **Definition 2.** *For a given source $s \in V$ and target $t \in V$, a* starting charge map $SCM_{st} : [0, M] \rightarrow P$ *is a function that maps a starting charge β_s to the corresponding shortest feasible path P.*

An SCM is a generalization of the shortest feasible path problem where the starting SoC β_s is unknown. First, it can be used to *recommend* users faster routes that they can take if the starting SoC is higher. For example, given an SCM, it is trivial to generate recommendations for EV drivers like "The best path with your current SoC takes 45 minutes,

but you might save 10 minutes if you spend 15 more minutes charging at your present location before starting your trip". Such recommendations can be particularly useful to EV drivers for routes with flexible starting times. Second, different trips taken by an EV user might have *different levels of risk aversion*, and an *SCM* can be used to show users feasible paths that suit the current scenario. As an example, consider two EV trips, the first through an urban area with a high density of charging stations during daytime, and a second trip through a sparsely populated area after nightfall. In the first scenario, most drivers might trade off a higher stranding risk for shorter travel times, while the preferences might be reversed for the second route. *SCM*s can be used to explore such alternatives and present them to the driver. Asking the driver to charge longer might reduce the risk, while allowing to start with a lower SoC usually increases the risk. Lastly, in most applications, routes are computed on a server and sent to the users on mobile clients. Since battery constraints apply for EV routing, more information about the vehicle needs to be sent to the server than for regular Internal Combustion (IC) vehicles. If instead of individual routes, *SCM*s are computed and sent to the client for display, the current SoC no longer needs to be sent to the routing server, which may result in *better privacy* for the drivers.

A brute force approach to compute SCM_{st} is to run the CFP algorithm for all values in $[0, M]$. However, since $[0, M]$ contains an infinite number of values, this is clearly not feasible. Even if we discretize the domain and restrict it to only percentage values that are multiples of a small fixed integer k, running CFP $\frac{100}{k}$ times, once each for $\{0, k, 2k, 3k, ..., 100\}\%$, would still be too slow for interactive routing applications where queries need to be answered quickly. A better approach is to run a series of binary searches in the starting SoC range $[0, M]$ such that on iteration i, the search returns a breakpoint starting SoC $\beta \in [0, M]$, where the shortest feasible paths for starting SoC β and $(\beta + \epsilon)$ differ by at least one edge. However, if $|SCM_{st}| = N$, such an approach would take $N \log N$ runs of the CFP algorithm. In the next section, we present an algorithm that computes SCM_{st} in N runs.

4.1 Reverse Charging Function Propagation

First, we introduce the following intermediate problem:

▶ **Definition 3.** *The Reverse Shortest Feasible Path (RSFP) Problem:*
Given a graph $G = \langle V, E \rangle$, edge weight functions $d : E \to \mathbb{R}_{\geq 0}$ and $c : E \to \mathbb{R}$ that represent travel time and energy consumption on edges respectively, a source $s \in V$ and a target $t \in V$, a set $S \subseteq V$ marked as charging stations, and an SoC β_t, find a shortest path P such that SoC never drops below 0 along P and has a residual SoC at least β_t at t.

As RSFP is closely related to the regular shortest feasible path problem, it can be solved with a *reverse* variant of the CFP algorithm. Note that several operations needed for CFP are not symmetric, e.g., $f(P \circ Q) \neq f(Q \circ P)$. Following, we detail the Reverse Charging Function Propagation (RCFP) algorithm and extend it to compute Starting Charge Maps.

The Reverse CFP works on a backward graph G', obtained by reversing the directions of all edges in G. The RCFP search starts at t with residual SoC β_t and propagates towards s. At $v \in V$, a label ℓ' is defined as $\langle \tau_t, \beta'_u, u, f_{[v...u]} \rangle$, with τ_t the total travel time on subpath $[t \ldots v]$, u the last charging station encountered in the search, β'_u the SoC *after* charging at u, and $f_{[v...u]}$ the consumption profile of subpath $[v \ldots u]$.

A key difference between forward and reverse CFP search labels is that while a label ℓ for the forward search contains β_u, the SoC *before* charging at the last charging station u, ℓ' stores β'_u, the SoC *after* charging at u. Computing β'_u is *only* possible in reverse CFP search, because of the following: As forward CFP search reaches v, only the exact energy

(a) CFP runs on G with piecewise linear, concave charging functions.

(b) RCFP runs on backward graph G' with inverse charging functions.
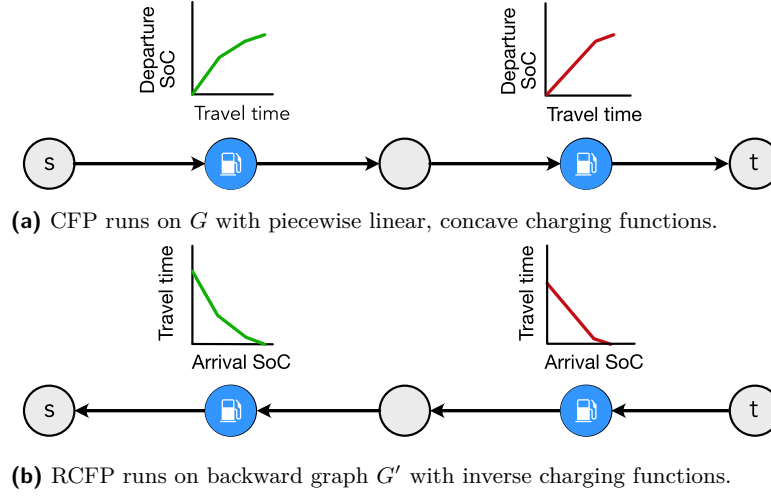
Figure 1 Comparing SFP and RSFP problem setups. While charging functions map the time spent charging to an EV's SoC at *departure*, inverted charging functions map the EV's SoC at *arrival* at the charging station to the least possible charging time required to reach target.

consumption on $[u \ldots v]$ is known, and therefore CFP needs to keep track of all possible charging scenarios at *previous* charging station u until the search reaches t or the next charging station. However, in RCFP search, the exact energy consumption between v and the target or *next* charging station u is known. Thus, as RCFP search reaches a charging station or origin, we know *exactly* how much charge is needed to travel from v to u, and have residual SoC β'_u. Both forward and reverse CFP maintain two label sets for each $v \in V$: $L_{uns}(v)$ for unsettled and $L_{set}(v)$ for settled labels.

Further, for RCFP, we transform all cf_v to inverse charging functions cf_v^{-1}. At $v \in S$, cf_v^{-1} returns the time required to charge an empty battery to resultant SoC $\beta'$. Note that under our assumptions, the inverse charging functions are piecewise linear, convex and monotonically decreasing. RCFP propagates through G' as follows:

1. *At t:* A label $\ell' = \langle 0, \beta_t, t, f_\phi^{-1} \rangle$ is added to the travel time ordered min-priority queue.
2. *Search reaches a non-charging vertex $v \neq s$:* Let path $P = [v = v_1 \ldots v_k = t]$ and total travel time be $\tau_P = \Sigma_1^k d(v_i, v_{i+1})$. Create label $\langle \tau_P, \beta_t, t, f_P^{-1} \rangle$ and add to $L_{uns}(v)$.
3. *Search reaches first charging station vertex $v \neq s$:* Let path $P = [v \ldots t]$, total travel time over P be τ_P. Create label $\langle \tau_P, f_P^{-1}(\beta_t), v, f_\phi^{-1} \rangle$ and add to $L_{uns}(v)$.
4. *Search reaches a non-charging vertex $v \neq t$:* Let $\ell = \langle \tau_v, \beta_u, u, f_{[u \ldots v]} \rangle$ be the current label extracted from PQ. Since u is the last charging station, let subpath $P = [u \ldots v]$ and the total travel time over P be τ_P. Add label $\langle \tau_{[s \ldots v]}, f_{[s \ldots v]}^{-1}(\beta_s), u, f_P^{-1} \rangle$ to $L_{uns}(v)$.
5. *Search reaches a subsequent charging vertex $v \neq s$:* Let $\ell' = \langle \tau_t, \beta'_u, u, f_{[u \ldots v]}^{-1} \rangle$ be the current label extracted from PQ. Since u is the last charging station, let *leg* $\mathcal{L} = [u \ldots v]$ and path $P = [v \ldots t]$. Let the total travel time over P be τ_P. Next, compute the *Starting SoC function* $b'_{\ell'}(\beta) := \tau_P + max(0, \mathrm{cf}_u^{-1}(f_\ell^{-1}(\beta)) - \mathrm{cf}_u^{-1}(\beta'_u))$. Again, since we assume that all inverted charging functions are piecewise linear, it suffices to create one label per breakpoint of $b'_{\ell'}$. For breakpoint $B = (\tau_B, SoC_B)$, create a label $\langle b'_{\ell'}(SoC_B), SoC_B, v, f_P^{-1} \rangle$ and add to $L_{uns}(v)$.
6. *Search reaches destination s:* Terminate and backtrack to extract a path from t to s.

A label ℓ'_1 is said to dominate ℓ'_2 iff $b'_{\ell'_1}(\beta) \leq b'_{\ell'_1}(\beta)$ for $\beta \geq 0$.
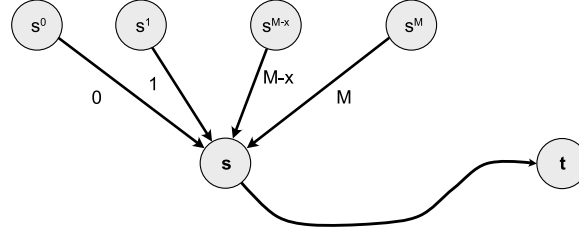
■ **Figure 2** "Virtual" vertices added to the graph.

▶ **Lemma 4.** *If a shortest feasible $s - t$ path exists, running the RCFP algorithm from t to s with $\beta_t = 0$ finds it.*

Proof. Let P be a shortest feasible $s - t$ path in G. Now, we show that RCFP computes the correct solution (travel time and starting SoC) for P. We distinguish three cases:

- *P contains no charging stop:* The linking operation on (inverse) consumption profiles is associative [10]. Further, the order in which labels are added to $L_{uns}(v)$ does not affect the correctness of the algorithms. Therefore, a shortest feasible path is found regardless of search direction and the correctness of RCFP follows from that of CFP [8].
- *P contains a single charging stop:* Let u be the charging stop on P, which divides P into subpaths $[s \ldots u]$ and $[u \ldots t]$. As the RCFP search starts from t and reaches u, the departure SoC at u is set to $\text{in}_{[u \ldots t]}$, the minimum SoC required to ensure feasibility of P. Charging more at u only increases the charging time without any corresponding decrease in travel time, which in turn increases the total travel time along P, violating the assumption that P is the *shortest* feasible path. On subpath $[s \ldots u]$, the RCFP search proceeds as in case (1).
- *P contains multiple charging stops:* Let u and u' be two consecutive charging stations on P, which divide P into subpaths $[s \ldots u], [u \ldots u']$ and $[u' \ldots t]$. Lemma 2 in [8] shows that for CFP, the *optimal* departure time at u always corresponds to charging to either $\text{in}_{[u \ldots u']}$ or to a breakpoint of cf_u. Similarly, after the RCFP search reaches u, the departure time at u' always corresponds to charging to either $\text{in}_{[u \ldots u']}$, or to a breakpoint of $\text{cf}_{u'}^{-1}$, which is optimal.

Next, we show that the minimum time label in RCFP search is not dominated by other labels and reaches s the first. The first claim follows from the dominance criterion for RCFP, which is symmetric to that of CFP: A label ℓ_v is dominated if it results in a higher total travel time for every possible initial SoC at v. This implies that a dominated label can not result in a unique optimal solution, since replacing the sub-path to the target it represents with the sub-path of the label dominating it would result in a better or equal solution. Lastly, since labels are ordered by travel time at all $L_{uns}(v)$, the label with minimum total travel time reaches s first.   ◀

4.1.1   Computing SCM with Reverse CFP

If the Reverse CFP algorithm does not terminate when the search reaches s and is instead allowed to continue to run until PQ is empty, we would have the set of all pareto-optimal feasible paths from s to t at s. This set of pareto-optimal feasible paths forms the Starting Charge Map between vertices s and t.

▶ **Theorem 5.** *If the RCFP algorithm is run from $t \in V$ with $\beta_t = 0$ until the priority queue is empty, the Pareto-set of labels at every $s \in V$ is equivalent to SCM_{st}.*

Proof. We prove Theorem 5 by showing that after running the RCFP from t, a starting SoC β_s, the label set at s contains a label that corresponds to $SCM_{st}(\beta_s)$. For this, we add temporary *virtual* vertices $s^{(M-x)}$ and an edge from $s^{(M-x)}$ to s with energy consumption x to the network, as depicted in Figure 2. From Lemma 4, we know that RCFP can compute a shortest feasible path P from $s^{(M-x)}$ to t. Note that by construction, P must contain s and $in_{[s...t]} \leq x$, since $(M - x)$ energy is consumed on the edge from $s^{(M-x)}$ to s. Thus, a label ℓ must exist at s that represents the shortest feasible path from s to t and requires an initial SoC of at most $x.\ell$ corresponds to $SCM_{st}(x)$. Since the computation of RCFP in the network without $s^{(M-x)}$ is independent of the existence of $s^{(M-x)}$, the RCFP algorithm has to compute the label ℓ before the priority queue runs empty even if $s^{(M-x)}$ is not part of the network. ◀

5 Buffer Maps

Like Starting Charge Maps, a Buffer Map is a generalization of the Shortest Feasible Path Problem; albeit instead of unknown starting charge β_s, the lower bound of minimum allowed SoC along the path is raised from 0 to an arbitrary $\mathfrak{b} \in [0, M]$. Formally,

▶ **Definition 6.** *A* buffer map $BM_{st} : [0, M] \to P$ *between a source $s \in V$ and target $t \in V$ is a function that maps a given* buffer *SoC $\mathfrak{b} \in [0, M]$ to the corresponding shortest feasible path P such that the EV maintains at least $\mathfrak{b}$ SoC at all points in P.*

Further, like SCMs, Buffer Maps can be used to show alternative routes to EV drivers who can decide upon the degree of acceptable stranding risk along the route. However, a key difference between the two abstractions and their usage is that while SCMs are used to get alternative routes depending on the *starting state* of the EV, alternative routes in buffer maps differ on the basis of *projected EV behaviour along the route*. In this way, alternative routes in BMs offer strong guarantees against stranding risk for EV drivers; not surprisingly, they are also more expensive to compute. Note that this problem would also qualify as what is referred to as "profile query" in the literature, since we ask for an optimal solution for arbitrary initial SoC [10, 13]. However, unlike [10, 13], we consider a multi-criteria variant of this problem and also allow intermediate charging stops.

For each distinct $\mathfrak{b}$, a run of the CFP algorithm can yield a shortest feasible path with the minimum SoC equal to $\mathfrak{b}$. A brute force approach to computing a Buffer Map is to run CFP several times, setting $\mathfrak{b}$ to each value in $[0, M]$. However, this approach is not feasible since the interval $[0, M]$ contains infinite values. In the next subsection, we present an exact, practical algorithm to compute a buffer map.

5.1 Iterative Charging Function Propagation

Each EV path consists of a sequence of legs. We define:

▶ **Definition 7.** *Given an SoC $\mathfrak{b} \in [0, M]$, a* critical leg *of a shortest feasible path P is one on which the SoC drops to $\mathfrak{b}$.*

CFP computes the exact amount of charge that an EV charges at every station along a feasible route P in order to minimize total travel time. However, to ensure that the minimum SoC of the EV along P never drops below a given $\mathfrak{b} \in [0, M]$, the EV must charge extra on

the charging stations adjacent to critical legs along P. The amount of extra energy to charge at such stations is exactly equal to that required to maintain at least $\flat$ SoC along the route, and is called the *buffer energy*.

Our approach to computing a Buffer Map BM_{st} for a given source $s \in V$ and target $t \in V$ works in *iterations*. Every iteration starts with choosing a value $\flat' \in [0, M]$. An augmented variant of CFP is run that returns a shortest feasible path P' such that the minimum SoC of the EV along P' is equal to $\flat'$. A collection of all such P' constitutes the set of paths in BM_{st}. Therefore, our approach has two components: first, an augmented variant of the CFP that respects the buffer SoC $\flat'$, and second, an algorithm that computes the increase in $\flat'$ on every iteration.

5.1.1 Augmenting CFP

The first iteration of our algorithm starts with $\flat' = 0$. The augmented CFP search starts from s with an SoC β_s and propagates towards t. Assume that the search requires charging at consecutive stations u' and u, and reaches $v \in V$. Let the breakpoints of $\mathrm{cf}_{u'}$ be $[B^1_{u'} B^2_{u'} \ldots B^m_{u'}]$, where $B^i_{u'} = (\tau^i_{u'}, SoC^i_{u'}), 1 \leq i \leq m$ where SoC^i_i is EV's resultant SoC after charging for time $\tau^i_{u'}$. Similarly, the breakpoints of cf_u are $[B^1_u B^2_u ... B^n_u]$.
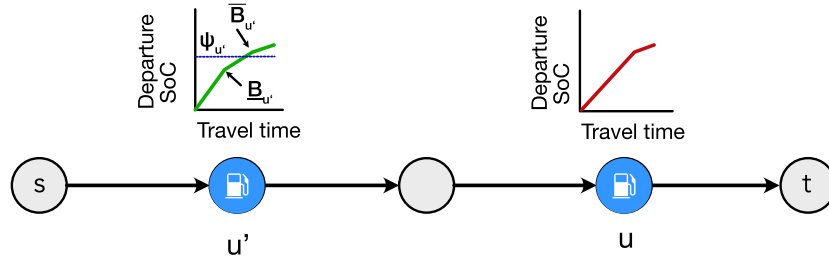
Figure 3 Augmented CFP setup.

Recall that CFP sets the amount of charge added to the EV at a station only after the search reaches the next charging station. Let the EV's SoC be $\psi_{u'}$ at departure after charging at station u'. Also, let $\underline{B}_{u'} = (\underline{\tau_{u'}}, \underline{SoC_{u'}})$ be the breakpoint of $\mathrm{cf}_{u'}$ with SoC immediately lesser or equal to $\psi_{u'}$, and $\overline{B}_{u'} = (\overline{\tau_{u'}}, \overline{SoC_{u'}})$ be the next breakpoint after $\underline{B}_{u'}$. Therefore, $\underline{SoC}_{u'} \leq \psi_{u'} < \overline{SoC}_{u'}$. Figure 3 shows an example of the $\underline{B}_{u'}$ and $\overline{B}_{u'}$ corresponding to a given $\psi_{u'}$. Similarly, given cf_u and ψ_u, $\underline{SoC}_u \leq \psi_u \leq \overline{SoC}_u$.

At $v \in V$, a label of the search is given by $\mathfrak{l} = \langle \tau_v, \beta_u, u, f_{[u...v]}, \rho_v, \delta_v \rangle$, where τ_t, β_u, u, and $f_{[u...v]}$ are analogous to regular CFP, ρ_v is the time required to add unit buffer energy to the EV on the current path, and δ_v is the maximum SoC up to which it can be charged without a loss in charging rate (due to concavity of charging functions).

▶ **Lemma 8.** *Let P be a shortest feasible $s - t$ path with k charging stops on P and $\flat$ be the minimum allowed SoC along P. Assume that the EV arrives at i^{th} charging station with SoC α_i, charges for t_i time, and departs with SoC ψ_i. Further, let C be the charging stations at the beginning of critical legs in P. To increase the buffer energy along P by ϵ, increasing departure SoC ψ_i to $(\psi_i + \epsilon)$ on all stations in C is an optimal solution if:*
(1) *On charging stations in C, $f(\psi_i + \epsilon) - f(\psi_i) = \epsilon$, i.e. charging ϵ more increases the residual SoC at t by ϵ.*
(2) *On all non-critical legs, the minimum allowed SoC is at least $\flat + \epsilon$.*

(3) *For all charging stations in C, the charging function is differentiable and does not have breakpoints with SoCs in range $[\psi_i, (\psi_i + \epsilon)]$.*

(4) *For all charging stations at the end of a critical leg, the charging function is differentiable and does not have a breakpoint in SoC range $[\alpha_i, (\alpha_i + \epsilon)]$.*

Proof. First, note that increasing ψ_i at all charging stations in C by ϵ is sufficient to increase the total buffer by ϵ – this follows immediately from conditions (1) and (2).

Let a *solution* S be the set of charging stops and charging times along path P, resulting from an Augmented CFP run between vertices s to t. We will show that no other solution can result in a lower total travel time along path P without changing at least one edge in P. Assume for contradiction, a solution S' has a lower total travel time than S along same path P. In order to increase the buffer energy for S by ϵ, ψ_i for each charging station in C must be increased by at least $(\psi_i + \epsilon)$. This can only be achieved by charging additional energy at a station on P.

Let j be the charging stop closest to t at which charging time differs between S and S'. We claim that there must be a critical leg after departing from j and that its departure SoC is $(\psi_j + \epsilon)$ – if this were not the case, we could decrease the departure SoC at j to $(\psi_j + \epsilon)$, which would be sufficient to increase buffer energy by ϵ, giving us a faster solution and contradicting the assumption that S is optimal. This implies that we can decrease the departure SoC at j to $(\psi_j + \epsilon)$, which is sufficient to increase buffer energy by ϵ, which gives us a faster solution contradicting the assumption that S is optimal. Since the departure SoC on j is equal to $(\psi_j + \epsilon)$, the arrival SoC at j must be greater. In other words, we charge more at some other stop i so we can charge less at j. But then, we can create a faster solution for buffer energy $\mathfrak{b}$ as follows: there exists a $\delta > 0$ such that we can charge δ more at i and charge δ less at j (since the charging function is concave and differentiable around ψ_j, the charging rate remains the same as for $(\psi_j + \epsilon)$). This contradicts the fact the solution S for buffer SoC $\mathfrak{b}$ was optimal. ◀

The CFP search starts from s with $\rho_s = 0$ and $\delta_s = M$. Assume that the search reaches charging station u after charging at a prior station u'. Let $\mathfrak{l}_u = \langle \tau_v, \beta_u, u, f_{[u...v]}, \rho_v, \delta_v \rangle$ be the current label extracted from priority queue. If $\psi_{u'} = \mathfrak{b}'$, i.e. the leg $[u' \ldots u]$ is a critical leg, we set $\delta_u = \min(\delta_{u'}, SoC_{B_2} - SoC_{B_1}, \overline{SoC}_u - \mathrm{cf}_u(f_{[u'...u]}(\psi_{u'})))$, where SoC_{B_1} and SoC_{B_2} are the SoC of the first and second breakpoints of the SoC function of $\mathfrak{l}_u$. We also set $\rho_u = \rho_{u'} + \frac{(\overline{\tau_{u'}} - \tau_{u'})}{(SoC_{u'} - \underline{SoC_{u'}})} - \frac{(\overline{\tau_u} - \tau_u)}{(SoC_u - \underline{SoC_u})}$. If $[u' \ldots u]$ is not a critical leg, the $\delta_u = \min(\delta_{u'}, \mathrm{in}_{[u'...u]} - \mathfrak{b}')$, and $\rho_u = \rho_{u'}$.

A label $\mathfrak{l}_u$ dominates $\mathfrak{l}'_u$ if the SoC function of $\mathfrak{l}_u$ dominates the SoC function of $\mathfrak{l}_{u'}$, and $\rho_u \leq \rho_{u'}$. In other words, a label $\mathfrak{l}$ dominates $\mathfrak{l}'$ if it represents a faster path to which the buffer energy can be added at a faster rate.

5.1.2   Computing $\mathfrak{b}'$ for the next iteration

On an iteration, we let the augmented CFP run and collect the complete set of non-dominated labels at target t. Let the set of labels collected at t be $\mathfrak{L}$. The Augmented CFP search guarantees that every $\mathfrak{l}_i \in \mathfrak{L}$ represents a feasible path where the SoC along the path does not drop below $\mathfrak{b}$. Let the label $\mathfrak{l}_{min}$ have the minimum total travel time of all $\mathfrak{l} \in \mathfrak{L}$. Next, we need to determine the maximum buffer energy that can be added at stations adjacent to critical legs of the shortest feasible path P found in the current iteration, while ensuring that no other feasible path becomes a better (faster) choice than P. We can solve this problem geometrically on an X-Y plane, where X and Y axis represent the buffer SoC and the total travel time of the EV respectively.
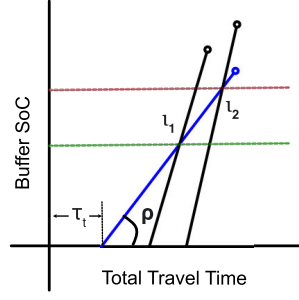
Figure 4 Each line on the X-Y plane represents a label $\mathfrak{l}_i \in \mathfrak{L}$ for iteration N. Highlighted blue *label line segment* represents the minimum time label $\mathfrak{l}_{min}$. The slope of blue label line segment is $\rho \in \mathfrak{l}_{min}$, and the X-intercept is equal to travel time $\tau_t \in \mathfrak{l}_{min}$. It intersects with two other label line segments at ι_1 and ι_2. Similarly, let intersection points be $\{\iota_1, \iota_2, ..., \iota_n\}$ if $\mathfrak{L}$ contains more labels. b' for the next iteration is equal to the minimum buffer SoC in $\{\iota_1, \iota_2, ..., \iota_n\}$ (SoC of shown green line).

For a label $\mathfrak{l}_t$, we draw a *label line segment* with slope $\rho_t \in \mathfrak{l}_t$, and the X-intercept equal to the total travel time τ_t of $\mathfrak{l}$. Further, the maximum ordinate of the line segment is given by $\delta_t \in \mathfrak{l}_t$. Figure 4 shows an example where $\mathfrak{L}$ contains three labels. The next step is to find the *globally minimum buffer SoC*, δ_{min} to which the EV can be charged the fastest among all labels in $\mathfrak{L}$. To find such a value, we start with the label line segment for $\mathfrak{l}_{min}$, and find its intersections with all other label line segments on the plane. Let the set of such intersections be $\{\iota_1, \iota_2, ..., \iota_n\}$. Since Figure 4 has only three label line segments, it shows two intersection points ι_1 and ι_2. Thus, δ_{min} is given by the buffer SoC of the intersection point that lowest on the Y-axis in the plane. For the next iteration, we set $\mathfrak{b}' = \delta_{min}$ and add the feasible path represented by $\mathfrak{l}_t$ to the buffer map BM.

▶ **Lemma 9.** *The global delta selection algorithm is correct, i.e. no feasible path has a lower total travel time and can add buffer energy faster than the chosen route given by the algorithm.*

Proof. We prove geometrically. Since all charging functions are convex with a positive slope, the slopes of all label line segments in the X-Y plane are positive. Further, since $\mathfrak{l}_{min}$ has the smallest X-intercept, in buffer SoC interval $[0, \text{SoC of } \iota_1]$, no other label in $\mathfrak{L}$ can charge the EV to a higher buffer SoC in lesser time.   ◀

As we increase $\mathfrak{b}'$ on each iteration, the augmented CFP search becomes more selective and the number of feasible paths from s to t decreases, since only on fewer paths would an EV be able to maintain a higher minimum SoC. The iterations terminate when $\mathfrak{b}'$ becomes high enough so the CFP search does not return any feasible paths.

▶ **Theorem 10.** *The Iterative CFP algorithm terminates and computes BM_{st} correctly.*

Proof. We have already argued that we compute ρ and δ correctly for labels propagated by the Augmented CFP search, and that for label $\mathfrak{l}$ it gives us the minimum additional required charging time in order to increase the buffer energy by any value in $[0, \delta]$ on the feasible path represented by $\mathfrak{l}$. We now show that the solutions added to the buffer map are indeed optimal and there is no remaining path with a shorter time for some value of buffer energy. Assume for contradiction, that we add a label $\mathfrak{l}$ to the buffer map, for which there exists a label $\mathfrak{l}'$ that offers a faster solution for some buffer energy. Observe that this implies that it is not a part of the Pareto set at the target, since the global delta computation finds the best label in that set by lemma 9. We can now distinguish two cases:

1. $\mathfrak{l}'$ *represents a feasible path with at least one critical leg:* Since $\mathfrak{l}'$ can not have a faster (minimum) traversal time than $\mathfrak{l}$ by construction (the algorithm selected $\mathfrak{l}$ and added it to the buffer map because it is the label with minimum travel time), it can only become the better solution after adding additional charge so it yields shorter total travel time for higher buffer energy. In other words, $\mathfrak{l}'$ offers a better charging rate and therefore is not dominated by $\mathfrak{l}$, which implies that it (or another dominating label) must be a member of the Pareto set. This must result in a lower intersection point on the Y-axis than δ during the global delta computation, which contradicts our assumption.

2. $\mathfrak{l}'$ *represents a feasible path with no critical leg:* This implies it has no charging stop (if there was a label with a charging stop but no critical leg, we could always charge less to obtain a faster solution). This means it cannot be dominated by $\mathfrak{l}$ because it has $\rho = 0$, and therefore it or another single-leg path must be a part of the Pareto set, which implies that it is taken into account when computing the global value of δ, again leading to a contradiction. ◀

Several factors can affect the total number of iterations required to compute BM: the distance between s and t, the total number of charging stations required to reach from s to t, which in turn depends on the parameters of the EV under consideration. The number of iterations further depends on the number of breakpoints in charging functions along the feasible paths from s to t. However, in practice, the number of iterations remains small for the following reasons: First, $\mathrm{cf}_u, u \in S$ are usually simple, linear functions up to 80% charge and only have a small number of breakpoints in the $80 - 100\%$ range. Next, most EV trips tend to not have a large number of charging stops along the way, and as EV ranges increase, this number would further decrease.

6   Experiments

We implemented our algorithms in C++ using Apple clang version 10.0.1 with $-O3$ optimizations. All experiments were run on macOS 10.14.6 using a Mac Pro 6,1 with a quad-core Intel Xeon E5 (3.7 GHz base clock). The processor has 256 KB of per-core L2 and 10 MB of shared L3 cache. The machine has 64 GBs of DDR3-ECC memory clocked at 1866 MHz.

6.1   Preparing a realistic EV Routing instance

Table 1 Our road network is taken from OpenStreetMap, public charging stations data from the Alternative Fuel Data Center [2], elevations from NASADEM [38] and an energy consumption model from a Nissan Leaf 2013 [20].

Dataset	Vertices	Edges	|Ch. stations|
Oregon (contracted)	502327	710107	323
California (contracted)	2547618	3741891	1406

We extract the road networks of Oregon and California from OpenStreetMap (OSM)[1] and label each edge with travel time equal to geographic distance divided by the maximum allowed speed for the road segment type. We contract all vertices with degrees ≤ 2 for our experiments, keeping only the largest connected component of the network. Table 1 shows the size of road networks after contraction.

---

[1] https://openstreetmap.org/

Next, we add the elevation to each vertex of the network, taken by sampling the NAS-ADEM elevation dataset at $30m$ resolution [38]. The elevation is required to compute the energy consumption on every edge of the network, which we derive from a microscopic EV energy consumption model for a Nissan Leaf 2013 [20].

Lastly, we extract the locations of public EV charging stations in Oregon and California from the Alternative Fuels Data Center [2]. For each charging station in the dataset, we mark the vertex geographically closest to it as the charging station. We assign each charging station vertex one of three charging functions: i) a *slow* linear function that charges the EV to full battery in 120 minutes; ii) a *fast* charging function that charges the EV to 80% in 30 minutes and to full capacity in 60 minutes, and iii) a *fastest* charging function that charges to 80% capacity in 20 minutes, and to full in 40 minutes. We arbitrarily assign 60% of all charging stations the *slow* charging function, another 30% stations the *fast*, and the remaining 10% the *fastest* charging functions.

To allow for tests with reasonable running times, we make it easier for a label to dominate another in the (Reverse) CFP search. We do this by adding a constant *slack energy consumption* ϵ to the dominance criterion in all three algorithms. Given labels ℓ_1 and ℓ_2, ℓ_1 dominates ℓ_2 iff all breakpoints of ℓ_1's SoC function have a higher energy than breakpoints of ℓ_2's SoC function after decreasing each breakpoint by ϵ energy. We set ϵ to 1% of the total battery capacity of the EV. Similar modifications to the dominance criteria have been proposed in earlier work, e.g. see [5, 12].

6.2   Reverse Shortest Feasible Path Queries & Starting Charge Maps

■ **Table 2** Average performance of 1000 queries running RCFP vs. variants of standard CFP. The EV is always assumed to start with 100% SoC at source. CFP with stopping criterion terminates after finding only one feasible route, and is therefore much faster than regular CFP which returns all feasible routes. RCFP can be seen to perform at par with CFP without stopping criterion. Time shown in seconds, also shown – no. of labels extracted from priority queue, alternative routes to t, and the no. of times search reached target. Targets found differ between RCFP and CFP because of the difference in dominance criteria.

	Alg.	**16 kWh**				**32 kWh**			
		Time	kLabels	|Routes|	Targets	Time	kLabels	|Routes|	Targets
Oregon	CFP (Stp)	1.767	933	0.709	709	1.510	873	0.895	895
	CFP	3.853	1758	4.962	709	3.629	1973	5.634	895
	RCFP	4.861	2477	7.703	710	3.502	2370	7.176	895
California	CFP (Stp)	34.847	10141	0.722	722	21.805	8645	1.0	1000
	CFP	70.076	19684	8.88	722	61.171	21596	9.837	1000
	RCFP	66.096	22571	11.467	724	46.617	22191	11.645	1000
		64 kWh				**128 kWh**			
		Time	kLabels	|Routes|	Targets	Time	kLabels	|Routes|	Targets
Oregon	CFP (Stp)	0.877	730	1.0	1000	0.678	621	1.0	1000
	CFP	2.648	1859	5.205	1000	2.521	1751	5.04	1000
	RCFP	2.641	2071	6.599	1000	2.241	1877	5.76	1000
California	CFP (Stp)	13.919	6631	1.0	1000	7.221	5006	1.0	1000
	CFP	47.197	18273	8.429	1000	25.182	13752	6.304	1000
	RCFP	36.568	19079	9.897	1000	16.255	12220	6.563	1000

Table 2 shows the results of running 1000 SFP and RSFP queries with several standard EV battery capacities (16, 32, 64, and 128 kWh) between random vertices in the road networks of Oregon and California. The table compares the performance of three algorithms–forward

CFP with *stopping criterion*, which makes the search terminates as soon as it reaches t; *full* forward CFP that runs till all pareto-optimal feasible paths from s to t are found; and the Reverse CFP algorithm as presented in Section 4.

We find that the CFP with stopping criterion performs at least a factor of two faster than full CFP that computes the pareto-optimal set of feasible paths. This is hardly surprising as the full CFP offers a richer set of routes which planners can use, in lieu of more computational overhead. However, if faster queries are desirable at the cost of alternative routes, the same technique can be applied to the reverse CFP algorithm with little effort. Target pruning [9] is another closely related technique that can achieve the same goal.

We observe that for both networks, SFP and RSFP query times generally decrease with increase in range of the EV, with a notable exception of capacity increase from 16 to 32 kWh, in which case the reverse search query times increase for the Oregon network and full CFP query times for the California network. This can be explained as follows: As the battery capacity increases, the (R)CFP search is able to reach vertices farther away. However, with increase in range, the slack energy ϵ also increases, making it easier for a label to dominate another, so fewer labels are settled in the search. The net effect of the two opposing factors, in this case, is that the total query time increases.

6.3 Iterative CFP and Buffer Maps

■ **Table 3** Average performance of Iterative CFP to answer 1000 Buffer Map queries (with 50 and 100% starting SoC) between random vertices on the Oregon road network.

	Range	Time (s)	kLabels	Iterations	Avg. $\lvert BM \rvert$	Targets
50%	16 kWh	67.405	30754	7.03	6.103	640
	32 kWh	99.310	45685	9.987	9.061	878
	64 kWh	32.571	25441	9.37	8.538	1000
	128 kWh	17.440	15316	7.013	6.359	1000
100%	16 kWh	198.395	57545	10.573	9.57	709
	32 kWh	85.484	47106	14.285	13.29	895
	64 kWh	53.706	37930	14.225	14.22	1000
	128 kWh	14.240	16183	10.611	9.635	1000

Table 3 shows the results of 1000 Iterative CFP queries between random vertices in the Oregon network. We do not report the running times for California, since they were found to be impractical with some queries running for more than 3 hours.

The total running time of the Iterative CFP algorithm has two components: The cost of Augmented CFP runs and the cost of computing the minimum global δ energy in each round. The cost of global delta computation is negligible in practice, since the number of Augmented CFP labels reaching the target vertex is often low. In Table 3, note that an Augmented CFP run takes longer than full CFP. This is caused due to inclusion of an additional parameter (charging rate) in the dominance criteria of the Augmented CFP.

The Iterative CFP is slower than the other algorithms discussed. This is expected as the algorithm involves running several iterations of an exponential-time shortest path computation. Our networks do not use standard speedup techniques like Contraction Hierarchies (CHs) [28], their multicriteria variant [27], or CRP [15, 16], though. Applying any of these techniques can significantly reduce query times by reducing the number of vertices explored to find shortest paths. A combination of speedup techniques such as CHs and A* search could be further applied for even greater speedups [6] at the cost of additional complexity.

7 Conclusion and Future Work

In this paper, we introduced Starting Charge Maps and Buffer Maps, which are helpful in preventing EV users' range anxiety and enable other use cases (such as minimizing trip time by charging more at home). Both problems require extending the known Shortest Feasible Path problem, essentially increasing its output by another dimension. Similar to profile queries in time-dependent route planning [11], this requires more sophisticated algorithms for Buffer Maps, which is reflected in the running times we observed in our experimental evaluation. For Starting Charge Maps, however, we proposed a simple and elegant approach which is in large parts symmetric to the known CFP algorithm and, as a result, computes them with similar running times, as out experimental results confirm.

Possible future work includes (heuristic) improvements of the Buffer Map search, or integration with A* and CH for faster queries as done by the CHArge algorithm [7, 8]. We may further consider related problem settings such having the SoC buffer dependent on the distance between charging stops.

References

1 Yazan Al-Wreikat, Clara Serrano, and José Ricardo Sodré. Driving behaviour and trip condition effects on the energy consumption of an electric vehicle under real-world driving. *Appl. Energy*, 297:117096, September 2021.

2 Alternative Fuels Data Center. Electric vehicle charging station locations. `https://afdc.energy.gov/fuels/electricity_locations.html`, 2021. Accessed: 2021-6-9.

3 Andreas Artmeier, Julian Haselmayr, Martin Leucker, and Martin Sachenbacher. The shortest path problem revisited: Optimal routing for electric vehicles. In *KI 2010: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 309–316. Springer, Berlin, Heidelberg, September 2010.

4 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. In *Algorithm Engineering*, Lecture Notes in Computer Science, pages 19–80. Springer, Cham, 2016.

5 Lucas S Batista, Felipe Campelo, Frederico G Guimarães, and Jaime A Ramírez. A comparison of dominance criteria in many-objective optimization problems. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2359–2366, June 2011.

6 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, , and Dorothea Wagner. Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *Algorithms*, 2008.

7 Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf. Shortest feasible paths with charging stops for battery electric vehicles. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 44. ACM, November 2015.

8 Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf. Shortest feasible paths with charging stops for battery electric vehicles. *Transportation Science*, 53(6):1627–1655, November 2019.

9 Moritz Baum, Julian Dibbelt, Lorenz Hübschle-Schneider, Thomas Pajor, and Dorothea Wagner. Speed-Consumption tradeoff for electric vehicle route planning. In *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, OpenAccess Series in Informatics (OASIcs), pages 138–151. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014.

10 Moritz Baum, Julian Dibbelt, Thomas Pajor, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Energy-Optimal routes for battery electric vehicles. *Algorithmica*, December 2019.

11   Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Dynamic Time-Dependent route planning in road networks with user preferences. In *Experimental Algorithms*, volume 9685 of *Lecture Notes in Computer Science*, pages 33–49, Cham, 2016. Springer International Publishing.

12   Moritz Baum, Julian Dibbelt, Dorothea Wagner, and Tobias Zündorf. Modeling and engineering constrained shortest path algorithms for battery electric vehicles. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 87, 2017.

13   Moritz Baum, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Consumption profiles in route planning for electric vehicles: Theory and applications. In *16th International Symposium on Experimental Algorithms (SEA 2017)*. drops.dagstuhl.de, 2017.

14   Cedric De Cauwer, Wouter Verbeke, Thierry Coosemans, Saphir Faid, and Joeri Van Mierlo. A Data-Driven method for energy consumption prediction and Energy-Efficient routing of electric vehicles in Real-World conditions. *Energies*, 10(5):608, May 2017.

15   Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Customizable route planning. In *Experimental Algorithms*, pages 376–387. Springer, Berlin, Heidelberg, May 2011.

16   Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Customizable route planning in road networks. *Transportation Science*, 51(2):566–591, May 2015.

17   Daniel Delling and Dorothea Wagner. Time-Dependent route planning. In *Robust and Online Large-Scale Optimization*, Lecture Notes in Computer Science, pages 207–230. Springer, Berlin, Heidelberg, 2009.

18   Matthias Eisel, Ilja Nastjuk, and Lutz M Kolbe. Understanding the influence of in-vehicle information systems on range stress – insights from an electric vehicle field experiment. *Transp. Res. Part F Traffic Psychol. Behav.*, 43:199–211, November 2016.

19   Jochen Eisner, Stefan Funke, and Sabine Storandt. Optimal route planning for electric vehicles in large networks. *AAAI*, 25(1), August 2011.

20   Chiara Fiori, Kyoungho Ahn, and Hesham A Rakha. Power-based electric vehicle energy consumption model: Model development and validation. *Appl. Energy*, 168:257–268, April 2016.

21   Chiara Fiori, Vittorio Marzano, Vincenzo Punzo, and Marcello Montanino. Energy consumption modeling in presence of uncertainty. *IEEE Trans. Intell. Transp. Syst.*, pages 1–12, 2020.

22   Matthew William Fontana. *Optimal routes for electric vehicles facing uncertainty, congestion, and energy constraints*. PhD thesis, Massachusetts Institute of Technology, 2013.

23   Luca Foschini, John Hershberger, and Subhash Suri. On the complexity of Time-Dependent shortest paths. *Algorithmica*, 68(4):1075–1097, April 2014.

24   Thomas Franke and Josef F Krems. Interacting with limited mobility resources: Psychological range levels in electric vehicle use. *Transp. Res. Part A: Policy Pract.*, 48:109–122, February 2013.

25   Thomas Franke, Isabel Neumann, Franziska Bühler, Peter Cocron, and Josef F Krems. Experiencing range in an electric vehicle: Understanding psychological barriers: Experiencing range. *Appl. Psychol.*, 61(3):368–391, July 2012.

26   Thomas Franke, Nadine Rauh, Madlen Günther, Maria Trantow, and Josef F Krems. Which factors can protect against range stress in everyday usage of battery electric vehicles? toward enhancing sustainability of electric mobility systems. *Hum. Factors*, 58(1):13–26, February 2016.

27   Stefan Funke and Sabine Storandt. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In *Proceedings of the Meeting on Algorithm Engineering & Experiments*, pages 41–54, Philadelphia, PA, USA, 2013. Society for Industrial and Applied Mathematics.

28   Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

29   Michael T Goodrich and Paweł Pszona. Two-phase bicriterion search for finding fast and efficient electric vehicle routes. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 193–202. ACM, November 2014.

30   Gerhard Huber, Klaus Bogenberger, and Hans van Lint. Optimization of charging strategies for battery electric vehicles under uncertainty. *IEEE Trans. Intell. Transp. Syst.*, pages 1–17, 2020.

31   Malte F Jung, David Sirkin, Turgut M Gür, and Martin Steinert. Displayed uncertainty improves driving experience and behavior: The case of range anxiety in an electric car. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2201–2210. ACM, April 2015.

32   Johannes Kester. Security in transition(s): The low-level security politics of electric vehicle range anxiety. *Security Dialogue*, 50(6):547–563, December 2019.

33   Yan Li, Pratik Kotwal, Pengyue Wang, Yiqun Xie, Shashi Shekhar, and William Northrop. Physics-guided energy-efficient path selection using on-board diagnostics data. *ACM/IMS Trans. Data Sci.*, 1(3):1–28, September 2020.

34   Yan Li, Shashi Shekhar, Pengyue Wang, and William Northrop. Physics-guided energy-efficient path selection: a summary of results. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 99–108. ACM, November 2018.

35   Ernesto Queirós Vieira Martins. On a multicriteria shortest path problem. *Eur. J. Oper. Res.*, 16(2):236–245, May 1984.

36   Michail Masikos, Konstantinos Demestichas, Evgenia Adamopoulou, and Michael Theologou. Energy-efficient routing based on vehicular consumption predictions of a mesoscopic learning model. *Appl. Soft Comput.*, 28:114–124, March 2015.

37   Sören Merting, Christian Schwan, and Martin Strehler. Routing of electric vehicles: Constrained shortest path problems with resource recovering nodes. In *OASIcs-OpenAccess Series in Informatics*, volume 48. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 2015.

38   J P L Nasa. NASADEM merged DEM global 1 arc second V001 [dataset]. `http://dx.doi.org/10.5067/MEaSUREs/NASADEM/NASADEM_HGT.001`, 2020. Accessed: 2021-6-9.

39   Zachary A Needell, James McNerney, Michael T Chang, and Jessika E Trancik. Potential for widespread electrification of personal vehicle travel in the united states. *Nature Energy*, 1:16112, August 2016.

40   Dario Pevec, Jurica Babic, Arthur Carvalho, Yashar Ghiassi-Farrokhfal, Wolfgang Ketter, and Vedran Podobnik. Electric vehicle range anxiety: An obstacle for the personal transportation (r)evolution? In *2019 4th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–8, June 2019.

41   Dario Pevec, Jurica Babic, Arthur Carvalho, Yashar Ghiassi-Farrokhfal, Wolfgang Ketter, and Vedran Podobnik. A survey-based assessment of how existing and potential electric vehicle owners perceive range anxiety. *J. Clean. Prod.*, 276:122779, December 2020.

42   Xuewei Qi, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew J Barth. Data-driven decomposition analysis and estimation of link-level electric vehicle energy consumption under real-world traffic conditions. *Transp. Res. Part D: Trans. Environ.*, 64:36–52, October 2018.

43   Payas Rajan and Chinya V Ravishankar. The phase abstraction for estimating energy consumption and travel times for electric vehicle route planning. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '19, pages 556–559, New York, NY, USA, 2019. ACM.

44   Nadine Rauh, Thomas Franke, and Josef F Krems. User experience with electric vehicles while driving in a critical range situation – a qualitative approach. *IET Intel. Transport Syst.*, 9(7):734–739, July 2015.

45   Martin Sachenbacher, Martin Leucker, Andreas Artmeier, and Julian Haselmayr. Efficient energy-optimal routing for electric vehicles. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, pages 1402–1407. AAAI Press, August 2011.

46   Stefan Sautermeister, Max Falk, Bernard Bäker, Frank Gauterin, and Moritz Vaillant. Influence of measurement and prediction uncertainties on range estimation for electric vehicles. *IEEE Trans. Intell. Transp. Syst.*, 19(8):2615–2626, August 2018.

47   René Schönfelder and Martin Leucker. Abstract routing models and abstractions in the context of vehicle routing. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, June 2015.

48   Sabine Storandt. Quick and energy-efficient routes: Computing constrained shortest paths for electric vehicles. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '12, pages 20–25, New York, NY, USA, 2012. ACM.

49   Sabine Storandt and Stefan Funke. Cruising with a Battery-Powered vehicle and not getting stranded. In *AAAI*, volume 3, page 46, 2012.

50   Martin Strehler, Sören Merting, and Christian Schwan. Energy-efficient shortest routes for electric and hybrid vehicles. *Trans. Res. Part B: Methodol.*, 103(Supplement C):111–135, September 2017.

A Branch-Price-And-Cut Algorithm for Stochastic Crowd Shipping Last-Mile Delivery with Correlated Marginals

Marco Silva[1] ✉
INESC TEC, Porto, Portugal

João Pedro Pedroso ✉
INESC TEC, Porto, Portugal
University of Porto, Portugal

Ana Viana ✉
INESC TEC, Porto, Portugal
Politechnic of Porto, Portugal

Xenia Klimentova ✉
INESC TEC, Porto, Portugal

—————— **Abstract** ——————

We study last-mile delivery with the option of crowd shipping, where a company makes use of occasional drivers to complement its vehicle's fleet in the activity of delivering products to its customers. We model it as a data-driven distributionally robust optimization approach to the capacitated vehicle routing problem. We assume the marginals of the defined uncertainty vector are known, but the joint distribution is difficult to estimate. The presence of customers and available occasional drivers can be random. We adopt a strategic planning perspective, where an optimal a priori solution is calculated before the uncertainty is revealed. Therefore, without the need for online resolution performance, we can experiment with exact solutions. Solving the problem defined above is challenging: not only the first-stage problem is already NP-Hard, but also the uncertainty and potentially the second-stage decisions are binary of high dimension, leading to non-convex optimization formulations that are complex to solve. We propose a branch-price-and-cut algorithm taking into consideration measures that exploit the intrinsic characteristics of our problem and reduce the complexity to solve it.

1 Introduction

Last-mile delivery is defined as the movement of goods from a transportation depot to the final delivery destination, which is typically a personal residence. Due to its importance and competitive value, last-mile delivery has prompted many companies to seek creative and innovative solutions.

---

[1] corresponding author

In this paper, we consider a setting in which a company not only has a fleet of capacitated vehicles and drivers available to make deliveries, but may also use the services of occasional drivers (ODs) who are willing to make deliveries using their vehicle in return for a small compensation. Under such business model, a.k.a crowd shipping, the company seeks to make all the deliveries at the minimum total cost, i.e., the cost associated with their vehicles and drivers plus the compensation paid to the ODs.

The advantages of crowd-shipping are numerous and are not only related to economic issues, since the compensation for the ODs is generally less than the cost associated with delivering using its own capacitated vehicles. If relying on the idea of individuals sharing their potentially under-utilized property, sharing vehicles can lead to a reduction in polluting emissions, energy consumption, noise and traffic congestion.

The application of crowd shipping alluded to above gives rise to new and interesting variants of the routing problem. It has been addressed as an extension of the classical vehicle routing problem (VRP) or the traveling salesman problem, being modeled under different deterministic, stochastic and/or dynamic optimization approaches.

In this work we adopt a data-driven stochastic approach where we model uncertainty as the probability of each customer to be delivered by an OD, a.k.a outsourced, or to be absent. We name them skipped customers. This probability, modeled as a Bernoulli distribution, should be easy to compute from historical data. Different from other crowd shipping last-mile delivery works in the literature, we do not assume that the uncertain events are independent ([8, 10, 13, 22]). Furthermore, because estimating correlations from potentially high dimensional uncertainty historical data can be very difficult (as is our case with many customers), we propose a worst-case probability approach where the joint probability of customers uncertainty is not known. We are interested in analyzing the effect of this assumption in the results when compared to the independent uncertainty assumption.

We consider a two-stage model with recourse. In the first stage, only the ordering in which the customers will be visited is defined. The company's vehicle routes are set only in the second stage after the uncertainty is revealed. Furthermore, we assume that each company's vehicle can serve a limited number of customers. A route is defined by starting at the depot, then following the order defined in the first stage, but skipping outsourced or absent customers and returning to the depot if the maximum number of customers have been delivered or if there are no more customers to be delivered. A new route is started from the depot going to the next not outsourced customer and following the same scheme as in the previous route. Potentially, many vehicle routes are set.

We are also interested in analyzing the potential cost savings associated with this recourse when compared to the case of reoptimization, when a different optimal decision is made for each scenario of uncertainty.

The main contributions and results of this work are:

- A novel data-driven worst-case probability paradigm for crowd shipping last-mile delivery, advancing the state-of-the-art in this topic. We model uncertainty in a way that it can capture customers that are absent or outsourced to ODs.

- A mixed-integer linear optimization formulation based on a distributionally robust formulation solved with a branch-price-and-cut algorithm approach, where we can capture characteristics of the problem to reduce the complexity to solve it.

- Computational evidence on the capability of the proposed model, that reflects a more realistic assumption of correlated marginals, to obtain solutions that can improve those provided using more simplified assumptions.

In what follows, in Section 2 we review relevant approaches to solve variants of the problem and contextualize our approach. In Section 3 we elaborate on the literature on distributionally robust optimization that we leverage to formulate our problem. In Section 4 we formally present our problem, the model and the formulation we have defined, exploiting the problem's characteristics to reduce the complexity of the algorithms proposed to solve it. Section 5 details the algorithm developed to be able to solve larger instances. Next, in Section 6 we present and discuss the computational results. Finally, in Section 7 we present the conclusion of the work done.

2 Literature review

Here we focus on the literature most relevant to compare to our approach. We are interested not only in crowd shipped last-mile related publications, but also in works that deal with similar problems under the concept of customer uncertainty.

2.1 Crowd shipping routing

A seminal work on last-mile delivery with crowd shipping is proposed in [4]. The authors study a deterministic approach where the customers' locations and the ODs parameters are input data. The model proposed is a combination of an assignment problem, where ODs are assigned to customers based on pre-defined assignment rules, with a capacitated VRP where routes are defined for vehicles passing through customers not served by ODs. For each OD and customer combination, a compensation fee to be paid for the outsourcing is also defined. Furthermore, each OD always accepts deliveries assigned to her/him. Under these assumptions, a customer is only outsourced to an OD if the overall solution is optimal. The pricing mechanism, meaning how compensation fees are defined, undertakes a critical part of the algorithm and is discussed in more detail by the authors. The authors develop a multi-start heuristic to handle instances with more than 25 customers.

Differently, in [5] the authors develop a dynamic solution alternative, where the solution is adjusted every time new information is available. They consider a service platform that automatically creates matches between parcel delivery tasks and ODs. The matching of tasks, drivers, and dedicated vehicles in real-time gives rise to a new variant of the dynamic pickup and delivery problem. They propose a rolling horizon framework and develop an exact solution approach to solve the matching problem each time new information becomes available.

The authors in [10] introduce a dynamic and stochastic routing problem in which the demand, arrives over time, as also does part of the delivery capacity, in the form of in-store customers willing to make deliveries. They develop two rolling horizon dispatching approaches to the problem: one that considers only the state of the system when making decisions, and one that also incorporates probabilistic information about future online orders and in-store customer arrivals.

In [9], the authors consider stochastic ODs and define routes for the company vehicles and the ODs based on their destination. They consider time windows when the ODs may appear and use a two-stage model in which partial routes of the company vehicles are defined in the first stage and, after the ODs are revealed, they adjust deliveries in the second stage. A penalty is paid for non served customers. They develop a Mixed Integer Linear formulation for the problem and special techniques to expedite the resolution. The stochastic solution is based on a scenario approach and they assume a uniform distribution of scenarios. Results are reported on instances with up to 20 customers and 3 ODs.

In [13] the authors consider that customers can be offered or not to potential ODs and that there is a known probability of them being accepted. They develop a heuristic to identify which customers will be offered to ODs and what will be the exact expected value of the associated solution by scenario enumeration. The probabilities of acceptance are considered independent. Computational experiments are conducted on randomly generated instances of 15 customers.

2.2   Routing with customer uncertainty

One of the first works addressing routing with customer uncertainty was presented in [17] that defines a problem of routing through a set of customers where only a random subset of them needs to be visited: The Probabilistic Traveling Salesman Problem. Assuming that the probability distribution is known and that it is equal to all customers and independent, the authors derive closed-form expressions for computing efficiently the expected length of any given tour.

In [6] the authors extend the previous work by considering a probabilistic variant of the classical VRP, in which demands and/or customer presence are stochastic. They introduce a recourse strategy where, in the second stage, not only absent customers are skipped, but also the route is broken and a detour happens every time the capacity of the vehicle is reached. Another contribution of the work is to elaborate on the need that many times arises of looking for strategic planning solutions, where an a priori sequence among all customers of minimal expected length is calculated, rather than solving the problem only when the demand becomes known. Assuming that the probability distribution is known, different to each customer and independent, they find closed-form expressions and algorithms to compute the expected length of an a priori sequence.

To solve the two previous models, integer L-Shaped branch-and-cut algorithms were proposed in [19] and in [14]. The authors could solve instances with up to 9 uncertain customers.

A specialized branch-and-bound algorithm is presented in [2] for the probabilistic traveling salesman problem under the a priori strategy. They adapt existing algorithms for the deterministic traveling salesman problem using the closed expected value evaluation expression defined in [17] and present numerical results for instances of up to 18 customers. The same authors present in [3] another branch-and-bound approach, this time using parallelization techniques, solving instances of up to 30 customers.

An approximation algorithm is presented in [18], for the VRP with probabilistic customers. They propose a two-stage stochastic optimization set-partitioning formulation where, in the first stage, a dispatcher determines a set of vehicle routes serving all potential customer locations, before actual requests for service realize. In the second stage, vehicles are dispatched after observing the subset of customers requiring service; a customer not requiring service is skipped from its planned route at execution. A column generation framework that allows for solving the problem to a given optimality tolerance is proposed. For a time limit of six hours, instances of up to 40 customers were solved.

The works presented so far assume that uncertain variables are independent. Nevertheless, in many planning problems, the correlations among individual events contain crucial information. The underlying correlations, possibly caused by some common trigger factors (e.g., weather, holidays, geographic location), are often difficult to predict or analyze, which makes the planning problem complicated. Estimating the correlations is hard, particularly when this includes the huge sample size required to characterize joint distribution since they are potentially high-dimensional. This can be our case, even when the estimation of their one-dimensional marginals is rather accurate.

Focusing on this issue from a general perspective, the authors in [1] study the possible loss incurred by ignoring these correlations, and propose a new concept called Price of Correlations (POC) to quantify that loss. They show that the POC has a small upper bound for a wide class of cost functions, including uncapacitated facility location, Steiner tree and submodular functions, suggesting that the intuitive approach of assuming independent distribution may work well for these stochastic optimization problems. On the other hand, they demonstrate that for some cost functions, POC can be particularly large.

Alternatives to the VRP with the assumption of independent uncertainty can be found in the works of [12] and [15], where the authors model using concepts from distributionally robust optimization (DRO), where it is assumed that probability distributions are not completely known and a worst-case probability distribution formulation is optimized.

3 Distributionally robust optimization (DRO)

Distributionally robust optimization is a robust formulation for stochastic programming problems and dates back to the work of [23], exploiting the concept of a worst-case probability distribution (see, e.g., [7, 11, 16]).

In this modeling approach, after defining a set $\mathcal{P}$ of feasible probability distributions that is assumed to include the true distribution $\mathbb{P}$, the objective function is reformulated with respect to the worst-case expected cost over the choice of a distribution in this set. This leads to solving the Distributionally Robust Optimization Problem

$$\min_{z \in Z} \max_{\mathbb{P} \in \mathcal{P}} \mathbb{E}_{\mathbb{P}}[h(z, \xi)], \tag{DROP}$$

where $h(z, \xi)$ is a cost function in z that depends on some vector of random parameters ξ, and $\mathbb{E}_{\mathbb{P}}$ is the expectation taken with respect to the random vector ξ given that it follows the probability distribution $\mathbb{P}$. The set $\mathcal{P}$ is called the ambiguity set.

The ambiguity set $\mathcal{P}$ is a key ingredient of any distributionally robust optimization model. It is a natural alternative when the question of how should one make decisions in the presence of a large amount of uncertain data arises and the correlations are not known. Since an ambiguity set only characterizes certain properties of the unknown true probability distribution, its estimation requires fewer data and can often be done using historical records, being suitable for data-driven approaches.

Since the introduction of distributionally robust optimization, several ambiguity sets have been proposed (e.g., [11, 21, 24]). It is shown that under specific assumptions over these ambiguity sets, many problems can be reformulated as convex optimization problems that can be efficiently solved by commercial solvers.

4 Stochastic crowd shipping last-mile delivery with correlated marginals

The two-stage approach defined in Section 1 is suitable under an a priori strategic planning process. The first stage decision will minimize the average total cost considering all scenarios under a worst-case probability paradigm. The total cost is given not only by the vehicle's routes cost but also by the total compensation fee paid to ODs.

A vital modeling decision of our approach is that uncertainty is customer-related. We can express not only the customer absence, but also uncertainty related to outsourcing the delivery service to an OD. It is different from the current crowd shipping last-mile delivery models, where uncertainty is related to the OD (e.g. [9, 10]). It is suitable for planning purposes and has the advantage that we can reduce the complexity of the problem to be solved by not having to introduce explicit OD's constraints, such as their quantity, capacity and routes, in the problem formulation. In our model, this reflects intrinsically in the customer's Bernoulli probability distribution that can be estimated from available historical data.

We define a compensation fee to be paid to the OD for each customer. In our model, it pays for only a small detour around each customer. It is equivalent to the idea that the customer will only be crowd shipped if there is an OD located very near him. It is compatible with the case where a delivery company would utilize crowd shipping with an emphasis on reducing environmental impacts, like traffic and gas emissions, and not on transforming it into an opportunity for professional services. Potential ODs are offered to outsource customers against the defined compensation fee. If they are available, and therefore accept, in the second stage the compensation fee is paid and the outsourcing is done.

A typical setting would be the use of in-store shoppers, who are willing to drop off packages for online customers on their route back home. In return, these in-store shoppers are offered a small compensation to reimburse their travel costs partially. As the participants are usually free to use any means of transportation to perform the delivery, we refer to them using ODs.

4.1   Problem formulation

Let $G = (V, A)$ be a directed graph, where $V = \{0, \ldots, N\}$ is the set of vertices and $A = \{(i, j) | i, j \in V\}$ is the set of arcs. Set V consists of a depot (vertex 0) and a subset C of customers' represented by locations ($C = \{1, \ldots, N\}$). We assume that the graph is symmetric, meaning that the cost or distance to transverse between two customers is the same regardless of the direction. Such feature is exploited in the algorithms developed to solve the problem. With each arc is associated a non-negative cost or distance c_{ij}. This cost or distance satisfies triangular inequalities. We also assume that the vehicles to be used as the company fleet are identical and can serve up to Q customers.

Vector $\xi = (\xi_1, \ldots, \xi_N)$ defines an uncertain scenario, $\xi_i = 1$ iff $i \in C$ is skipped, 0 otherwise. The support of the joint distribution, Ξ, includes all possible combinations of the scenario's components. We index scenarios using indicator $w \in W$. For each scenario with customer i being skipped there is a marginal probability, m_i, and a compensation fee, f_i, associated. As a remark, note that f_i is the compensation fee paid to the OD, weighted by the probability of the customer being outsourced. We assume that the uncertain components are not independent and the joint distribution is unknown.

We initially formulate our problem as in (DROP), where now $h(z, \xi)$ is the cost of delivery of the second stage routes and z defines the first-stage ordering.

To reduce complexity of the algorithm, we reformulate the problem exploiting some of its characteristics, as follows. We define our ambiguity set as

$$\mathcal{P} = \{\mathbb{P} \, | \, \mathbb{P}\{\xi \in \Xi\} = 1; \text{ marginals } m_i \text{ for } \xi_i = 1, i \in C\},$$

and since our uncertainty is binary, we reformulate DROP as in Proposition 1.

▶ **Proposition 1.** *Formulation DROP applied to our ambiguity set can be reformulated as*

$$\min_{z \in Z} \quad s - \sum_{i \in C} m_i u_i$$

$$s.t. \quad s - \sum_{i \in C} \xi_i^w u_i \geq h(z,w) \quad \forall w \in W \qquad \text{(DROR)}$$

$$s \geq 0, \, u_i \geq 0 \qquad\qquad \forall i \in C$$

where s, u_i, $i \in C$ are dual variables defined in our reformulation. We abuse notation and express the second-stage cost function now in terms of the first stage variables and the uncertainty index, $h(z,w)$.

Proof. We defer a step by step reformulation to Appendix A.   ◀

The next step is to define the first and second stage formulations, including the cost function $h(z,\xi)$. The first stage is defined solely by a ordering for serving the customers. The following variables are used:

- First-stage main variable

    $z_{i,j} = 1$ iff customer i is served before customer j.
- First-stage auxiliary variables

    $z_{i,j,r}^1 = 1$ iff customer r is served in between customers i and j

    $z_{i,j,r}^2 = 1$ iff customer r is served before customers i and j

    $z_{i,j,r}^3 = 1$ iff customer r is served after customers i and j

The second stage is defined in a way that we can calculate the cost of a route given the ordering of the first stage and the scenario to be considered. We define the following sets of main and auxiliary second-stage variables, where now we include the depot in the ordering as it will be always the first and last to be served in each route:

- Main variables

    $y_{w,i,j} = 1$ iff, for scenario ξ^w, depot or customer j is served right after depot or customer i. This means that all customers r in between i and j are outsourced in this scenario.

    $v_{w,i,j} = 1$ iff, for scenario ξ^w, vehicle capacity, Q, is reached at customer i and j is the next not skipped customer. This means that before customer i, in scenario ξ^w, there are $kQ - 1$ customers, where $k \in \{1, \ldots, \left\lfloor \frac{|C|}{Q} \right\rfloor\}$ and that all customers r in between i and j are outsourced in this scenario.

- Auxiliary variables

    $y_{w,i,t}^1 = 1$ iff, for scenario ξ^w and given a ordering of customers, there are t customers before i, $t \in \{0, \ldots, |C| - 1\}$. It indicates the position of a customer for each scenario.

We can now define the cost function $h(z,\xi)$. The cost function sums up the cost of each arc transpassed considering all routes plus the cost of the outsourced customers. We have already stated that each variable $y_{w,i,j} = 1$ defines an arc that is transpassed and each variable $v_{w,i,j} = 1$ defines a detour to the depot. This way we define the cost function as

$$h(z,w) = \sum_{i \in C} f_i \xi_i^w + \sum_{\substack{i,j \in V \\ i \neq j}} c_{i,j} y_{w,i,j} + \sum_{\substack{i,j \in C \\ i \neq j}} (c_{i,0} + c_{0,j} - c_{i,j}) v_{w,i,j}, \qquad (1)$$

where we index uncertainty with indicator w.

With all variables and cost function defined we reformulate DROR as in Proposition 2.

▶ **Proposition 2.** *With variables and cost function defined, Formulation DROR can be reformulated as*

$$\min \quad s + \sum_{i \in C} m_i u_i \tag{DROC}$$

$$s.t. \quad s + \sum_{i \in C} u_i \xi_i^w \geq \sum_{i \in C} f_i \xi_i^w + \sum_{i,j \in V} c_{i,j} y_{w,i,j} + \sum_{i,j \in C} (c_{i,0} + c_{0,j} - c_{i,j}) v_{w,i,j}$$

$$z_{i,j} + z_{j,i} = 1$$

$$z_{i,j} + z_{j,r} + z_{r,i} \leq 2$$

$$z_{i,j,r}^1 \geq z_{i,r} + z_{r,j} - 1$$

$$z_{i,j,r}^2 \geq z_{r,i} + z_{r,j} - 1$$

$$z_{i,j,r}^3 \geq z_{i,r} + z_{j,r} - 1$$

$$y_{w,i,j} \geq 1 - \xi_i^w + 1 - \xi_j^w + z_{i,j} + \sum_{r \in C} (\xi_r^w z_{i,j,r}^1 + z_{i,j,r}^2 + z_{i,j,r}^3) - |C|$$

$$y_{w,0,i} \geq 1 - \xi_i^w + \sum_{j \in C} (\xi_j^w z_{j,i} + z_{i,j}) - |C| + 1$$

$$y_{w,i,0} \geq 1 - \xi_i^w + \sum_{j \in C} (\xi_j^w z_{i,j} + z_{j,i}) - |C| + 1$$

$$v_{w,i,j} \geq y_{w,i,j} + \sum_{k \in \{1,\ldots,\lfloor \frac{|C|}{Q} \rfloor\}} y_{w,i,kQ-1}^1 - 1$$

$$\sum_{t \in \{0,\ldots,|C|-1\}} y_{w,i,t}^1 = 1 - \xi_i^w$$

$$\sum_{t \in \{0,\ldots,|C|-1\}} t y_{w,i,t}^1 \leq \sum_{j \in C} (1 - \xi_j^w) z_{j,i}$$

$$\sum_{i \in C} y_{w,i,t}^1 \leq 1$$

$$s \geq 0, \; u_i \leq 0$$

$$z_{i,j,r}^1, \; z_{i,j,r}^2, \; z_{i,j,r}^3 \in [0,1], \; z_{i,j} \in \{0,1\}$$

$$y_{w,i,t}^1, y_{w,i,j}, \; y_{w,0,i}, \; , y_{w,i,0}, \; v_{w,i,j} \in [0,1]$$

where the constraints and variables are valid $\forall w \in W$, $\forall i,j,r \in C$, $i \neq j \neq r$, and $\forall t \in \{0,\ldots,|C|-1\}$, when not stated otherwise.

Proof. We defer a step by step reformulation to Appendix B. ◀

5   Algorithm

Formulation (DROC) is challenging to solve. Not only it englobes an NP-Hard linear ordering problem based on binary $z_{i,j}$ variables with a weak linear relaxation, as evidence by our experiments, but also, it is defined by an exponential number of constraints and variables indexed by uncertain scenarios. To solve it we propose a branch-price-and-cut algorithm (BPC). Algorithm 1 summarizes the main steps undertaken to perform BPC. The directives of the implementation of the algorithm are:

- A customized branching rule based on the incremental ordering of the sequence of the visit of the customers. This branching rule permits that we fix many binary variables simultaneously to their lower or upper bounds at a node while producing feasible regions of equitable sizes after branching.

- A symmetry breaking strategy to limit the number of branchings. This is a way to eliminate partial orderings of customers that will not contribute to arriving at an optimal solution and therefore gain greater computational efficiency by eliminating nodes of our branching tree.
- At each node solve a relaxed restricted version of the formulation. The restricted version is composed of a finite number of scenarios.
- Initial tests indicate that the node relaxation is weak and may consume significant time. On the other hand, the independent marginal distribution version of the formulation provides a lower bound that is easy to calculate at each node. We then use this alternative as a lower bound to prune the nodes before proceeding with the calculation of the relaxed restricted version of our problem.
- Each node is solved to optimality and is pruned by its lower bound.
- Each node's integer solution is validated against new scenarios. A separation subproblem with a column and row generation approach is used to separate invalid integer solutions.
- New scenarios inserted re-initiate the process of solving the node relaxed problem.
- Valid integer solutions are tested against the incumbent solution and the correspondent node is pruned afterwards.
- Fractional solutions are branched.
- The algorithm runs until no more nodes are available to test or when a time limit is reached

▌ **Algorithm 1** Branch-price-and-cut (BPC) algorithm.

Input $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ ▷ Q, set C, vectors c, f, m
Initialize
//Nodes list ← root node, Incumbent solution ← Heuristic, Lower bound ← $-\infty$
while There are still nodes to be branched in the Nodes list **do**
$\quad$ Node Select $\qquad\qquad\qquad\qquad\qquad\qquad$ ▷ Select node based on search criteria
$\quad$ Initialize scenarios $\qquad\qquad\qquad\qquad\qquad$ ▷ Add scenarios from parents node
$\quad$ Prune $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ ▷ by Independent lower bound
$\quad$ **while** There are still scenarios to be added **do**
$\qquad$ Solve
$\qquad$ Prune $\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ ▷ by Node solution-lower bound
$\qquad$ Scenario Separation subproblem $\qquad\qquad\qquad\qquad\qquad$ ▷ If integer
$\quad$ **end while**
$\quad$ Update if new Incumbent solution $\qquad\qquad\qquad\qquad$ ▷ Prune if better value
$\quad$ Branch node
$\quad$ Prune $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ ▷ by symmetry
$\quad$ Update Nodes List
end while
Return optimal solution - order of customers to visit and expected cost

Appendix C details the implementation of each feature of the algorithm.

6 Experiments and Computational Results

For this Section, the objective of our experiments is two-fold: we want to analyze the effect of considering dependent marginals from a solution perspective and we are interested in analyzing the effect of the recourse strategy defined for our problem. To pursue this objective,

we implement additional algorithms to compare the solution of the different approaches. All algorithms are coded in Julia ([20]) using JuMP package and Cplex 12.7 and run in an Intel Xeon Cluster. A limit of 25200 seconds (7 hours) of computing time is given for each instance.

6.1 Instances

As test instances, we adapt the ones in [22], generated from instances in the TSPLIB by truncating them to the first $n+1$ vertices (one depot and n customers) for different values of n and assigning values to m_i and f_i according to different criteria. The number of customers used is $|C| \in \{6, 10, 14, 18\}$. Five instances for each number of customers are generated.

The compensation fee f_i for each customer i is set to a fixed small value, to avoid zero compensation fees, plus a value proportional to the minimal detour considering all pairs of customers $r, j \in C, i \neq j \neq r$ and given by $\min_{j,r \in C} c_{j,i} + c_{i,r} - c_{j,r}$. We assume that the pairs (m_i, f_i) generated are coherent, meaning that the compensation paid will reflect the associated probability to skip customers.

The professional fleet vehicle capacity is defined by $Q = \left\lfloor \frac{|C|}{3} \right\rfloor$.

With the instances generated from TSPLIB, we create 4 different sets of instances based on specific probability assignment rules as described below, arriving at 80 instances. All results presented by the number of customers is an average of all of their respective instances.

Instance Set A- Probability m_i is linearly proportional to the vertex's distance from the depot, with $m_i = 0.95$ for the farthest delivery point.
Instance Set B- As in set A, but we assigned probabilities with inverse proportionality to their distance from the depot. The rationale is that, in real applications, far delivery points might be inaccessible and harder to crowdsource.
Instance Set C- Here we assume that all probabilities are equal, having $m_i = 0.3$.
Instance Set D- In this case, we select probabilities at random.

6.2 Additional algorithms

We present in Table 1 a general description of different variations of Algorithm BPC. These variants were developed to run exact solutions to similar problems found in the literature, but using the same algorithmic approach that we have established for BPC. We want to compare solutions and time performance of these different problems and algorithms.

■ **Table 1** Algorithms variants.

Algorithm Code	Description
$INDPCAP$	Independent Marginals
$DETM$	Deterministic version
$REOPT$	Reoptimization strategy

6.3 Price of correlation

In this section, we analyze the effect of considering dependent marginals. For doing so, we run a set of instances against our algorithm but also against algorithm $INDPCAP$ that implements the same recourse but considers marginals independent. For a particular problem

Table 2 Price of Correlation.
Indep is the % savings average when compared to deterministic solution for $INDPCAP$.
Dep is the % savings average when compared to deterministic solution for BPC.
CG is the correlation gap average, as defined in Section 6.3.
We use the best solution provided by the algorithm under the time limit.

	Set A			Set B			Set C			Set D		
$\|C\|$	Dep	Indep	CG	Dep	Indep	CG	Dep	Indep	CG	Dep	Indep	CG
6	44.26	51.92	1.31	14.12	37.72	1.38	13.54	34.75	1.32	30.08	40.82	1.33
10	49.98	59.00	1.40	26.08	50.53	1.63	18.90	45.58	1.49	30.75	52.11	1.44
14	48.04	54.34	1.44	23.69	45.96	1.52	18.05	38.74	1.33	27.81	42.99	1.33
18	44.55	52.02	1.44	26.82	47.76	1.44	17.28	40.19	1.43	25.84	44.19	1.34

instance, let z_I be the optimal decision assuming independent marginals distribution. [1] define an indicator called correlation gap (CG) as an upper bound to the price of correlation (POC), that is given by

$$CG = \frac{\mathbb{E}_{\mathbb{P}^{D(z_I)}}[h(z_I, \xi)]}{\mathbb{E}_{\mathbb{P}^I}[h(z_I, \xi)]},$$

where $\mathbb{P}^I$ is the independent Bernoulli distribution with marginals m_i, and $\mathbb{P}^{D(z_I)}$ is the worst-case distribution for decision z_I.

We use the same indicator as a measure of the effectiveness of using a worst-case distribution formulation. A small CG indicates that the decision-maker can take the independent marginal distribution solution as an approximation of the worst-case distribution without involving much risk.

Table 2 presents, for each set of instances, the percentage of savings achieved by algorithms $INDPCAP$ (Indep) and BPC (Dep) solutions. It also shows the correlation gap (CG) calculated for these solutions. We note that the absolute saving values of each algorithm are not as important - as that depends strongly on the compensation fees - as the relationship between them. We can see that the correlation gap (CG) indicates variations in the range of 31 % up to 74%. There is not a determinant difference between the CG indicator for different sets of instances. For many applications, this gap can be already beyond what would be acceptable as an approximation. We can observe that savings of the *Indep* solution are always larger than savings of the *Dep* solution which is coherent with the fact that the independent marginals solution is a lower bound to the correlated marginals solution. We can observe also that the savings associated with Set A are always greater than the savings for all the other sets of instances. Set A is constructed in a way that the probability of outsourcing for customers that are distant from the Depot is higher.

6.4 Quality of recourse solution

In Table 3 we compare the solution of our recourse strategy, BPC, to solutions provided by the algorithm that implements reoptimization strategy, $REOPT$. The two solutions are given as a percentage of savings when compared to the deterministic approach, $DETM$, and were run for small instances only to be able to calculate exact reoptimization solutions.

For the instances that were run, the gaps between BPC and $REOPT$ solutions are very small. There is even no gap for the very small instances. We observe gaps larger than zero for the larger instance. Intuitively, we can see that for larger instances there is even more flexibility to rearrange the ordering of customers in a reoptimization strategy which can result in larger gaps.

■ **Table 3** Quality of recourse solution.
SolREOPT is the % savings average comparing to deterministic solution for *REOPT*.
SolBPC is the % savings average comparing to deterministic solution for *BPC*.

	Set A		Set B		Set C		Set D	
$|C|$	*SolREOPT*	*SolBPC*	*SolREOPT*	*SolBPC*	*SolREOPT*	*SolBPC*	*SolREOPT*	*SolBPC*
6	44.26	44.26	14.12	14.12	13.54	13.54	30.08	30.08
10	49.98	49.98	26.08	26.08	19.37	18.09	30.97	30.75
14	50.89	48.04	26.72	23.69	21.25	18.58	28.66	27.81

Based on the instances run, we conclude that our recourse strategy works as a good alternative to the more flexible reoptimization strategy.

7   Conclusion

We present a novel exact solution approach for the stochastic crowd shipping last-mile delivery problem where marginals are correlated, advancing the current state-of-the-art in this topic. In our approach, it is possible to capture customers that are absent or outsourced to ODs, providing a good tool to be used for a priori strategy planning solutions. We consider a worst-case joint uncertainty distribution.

We have analyzed under what conditions this approach can be relevant using the concept of the price of correlation and show that, in many cases of the instances, studied, the defined correlation gap is higher than what would be tolerated as an approximation of the problem.

Overall, we compare the solutions of the developed algorithm *BPC* against different exact solution algorithms using the same branch-and-bound method (e.g., one algorithm assuming independent marginals and another with an uncapacitated one vehicle with only one route). This comparison shows that the obtained solutions improve over the others, where more simplified assumptions are considered, and can help decision-makers in their work to obtain more competitive solutions.

References

1   Shipra Agrawal, Yichuan Ding, Amin Saberi, and Yinyu Ye. Price of correlations in stochastic optimization. *Operations Research*, 60(1):150–162, 2012.

2   Mohamed Abdellahi Amar, Walid Khaznaji, and Monia Bellalouna. An exact resolution for the probabilistic traveling salesman problem under the a priori strategy. *Procedia Computer Science*, 108:1414–1423, 2017. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

3   Mohamed Abdellahi Amar, Walid Khaznaji, and Monia Bellalouna. A parallel branch and bound algorithm for the probabilistic tsp. In Jaideep Vaidya and Jin Li, editors, *Algorithms and Architectures for Parallel Processing*, pages 437–448, Cham, 2018. Springer International Publishing.

4   Claudia Archetti, Martin W. P. Savelsbergh, and M. Grazia Speranza. The vehicle routing problem with occasional drivers. *European Journal of Operational Research*, 254(2):472–480, 2016.

5   Alp M. Arslan, Niels Agatz, Leo Kroon, and Rob Zuidwijk. Crowdsourced delivery—a dynamic pickup and delivery problem with ad hoc drivers. *Transportation Science*, 53(1):222–235, 2019.

6   Dimitris J. Bertsimas. A vehicle routing problem with stochastic demand. *Operations Research*, 40(3):574–585, 1992.

7   Xin Chen, Melvyn Sim, and Peng Sun. A robust optimization perspective on stochastic programming. *Operations Research*, 55(6):1058–1071, 2007.

8   Lars Dahle, Henrik Andersson, and Marielle Christiansen. The vehicle routing problem with dynamic occasional drivers. In Tolga Bektaş, Stefano Coniglio, Antonio Martinez-Sykora, and Stefan Voß, editors, *Computational Logistics*, pages 49–63, Cham, 2017. Springer International Publishing.

9   Lars Dahle, Henrik Andersson, Marielle Christiansen, and M. Grazia Speranza. The pickup and delivery problem with time windows and occasional drivers. *Computers & OR*, 109:122–133, 2019.

10   Iman Dayarian and Martin Savelsbergh. Crowdshipping and same-day delivery: Employing in-store customers to deliver online orders. *available in Optimization Online*, 2017.

11   Erick Delage and Yinyu Ye. Distributionally robust optimization under moment uncertainty with application to data-driven problems. *Operations Research*, 58(3):595–612, 2010.

12   Thai Dinh, Ricardo Fukasawa, and James Luedtke. Exact algorithms for the chance-constrained vehicle routing problem. *Mathematical Programming*, 172(1):105–138, November 2018.

13   Katarzyna Gdowska, Ana Viana, and João Pedro Pedroso. Stochastic last-mile delivery with crowdshipping. *Transportation Research Procedia*, 30:90–100, 2018.

14   Michel Gendreau, Gilbert Laporte, and René Séguin. An exact algorithm for the vehicle routing problem with stochastic demands and customers. *Transportation Science*, 29(2):143–155, 1995.

15   Sudipta Kumar Ghosal and W. Wiesemann. The distributionally robust chance constrained vehicle routing problem. *available at* `http://www.optimization-online.org/DB_FILE/2018/08/6759.pdf`, 2018.

16   Joel Goh and Melvyn Sim. Distributionally robust optimization and its tractable approximations. *Operations Research*, 58(4-part-1):902–917, 2010.

17   Patrick Jaillet. A priori solution of a traveling salesman problem in which a random subset of the customers are visited. *Operations Research*, 36:929–936, December 1988.

18   Felipe Lagos, Mathias Klapp, and Alejandro Toriello. Branch-and-price for probabilistic vehicle routing. *available at* `http://www.optimization-online.org/DB_HTML/2017/12/6364.html`, December 2017.

19   Gilbert Laporte, François V. Louveaux, and Hélène Mercure. A priori optimization of the probabilistic traveling salesman problem. *Operations Research*, 42(3):543–549, 1994.

20   Miles Lubin and Iain Dunning. Computing in Operations Research using Julia. *CoRR*, abs 1312.1431, 2013.

21   Peyman Mohajerin Esfahani and Daniel Kuhn. Data-driven distributionally robust optimization using the Wasserstein metric: performance guarantees and tractable reformulations. *Mathematical Programming*, 171(1):115–166, September 2018.

22   Alberto Santini, Ana Viana, Xenia Klimentova, and João Pedro Pedroso. Exact, heuristic and machine learning approaches to the probabilistic travelling salesman problem with crowdsourcing. *available at* `https://santini.in/files/papers/santini-viana-klimentova-pedroso-2020.pdf`, 2020.

23   Herbert Scarf. *A Min-Max Solution of an Inventory Problem*, pages 201–209. Stanford University Press, 1958.

24   Bart P. G. Van Parys, Paul J. Goulart, and Manfred Morari. Distributionally robust expectation inequalities for structured distributions. *Mathematical Programming*, December 2017. `doi:10.1007/s10107-017-1220-x`.

25   Christoph Weiler, Benjamin Biesinger, Bin Hu, and Günther R. Raidl. Heuristic approaches for the probabilistic traveling salesman problem. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *Computer Aided Systems Theory – EUROCAST 2015*, pages 342–349, Cham, 2015. Springer International Publishing.

26   Bo Zeng and Long Zhao. Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5):457–461, 2013.

A   Proof of Proposition 1

Our objective is to reformulate the inner maximization problem of our initial formulation

$$\min_{z \in Z} \max_{\mathbb{P} \in \mathcal{P}} \mathbb{E}_{\mathbb{P}}[h(z, \xi)], \tag{2}$$

given that our ambiguity set is defined as

$$\mathcal{P} = \{\mathbb{P} \mid \mathbb{P}\{\xi \in \Xi\} = 1; \text{ marginals } m_i \text{ for } \xi_i = 1, \, i \in C\}, \tag{3}$$

Based on the definitions (2) and (3) we can formulate the inner maximization problem of (2) as

$$\max \qquad \mathbb{E}_{\mathbb{P}}[h(z, \xi)] \tag{4}$$

$$s.t. \qquad \mathbb{P}\{\xi \in \Xi\} = 1 \tag{5}$$

$$\mathbb{P} \text{ has marginals } m_i \text{ for components } \xi_i = 1, \, i \in C \tag{6}$$

We now index the uncertainties ξ with indicators w and introduce variable $p_w \geq 0$ as the probability associated with each scenario in $\mathbb{P}$. We can then reformulate our inner problem as

$$\max \qquad \sum_{w \in W} p_w h(z, w) \tag{7}$$

$$s.t. \qquad \sum_{w \in W} p_w = 1 \qquad\qquad (s) \tag{8}$$

$$\sum_{w \in W} \xi_i^w p_w \geq m_i \qquad \forall i \in C \qquad (u_i) \tag{9}$$

where we introduce dual variables s and u_i. Note that we use sign $\geq$, instead of $=$, for constraints (9). It can be done since the solution to our problem will satisfy these constraints at equality. We then restrict to non positive dual variables u_i.

Since the formulation above is always feasible (the independent marginals joint distribution will always be a possible solution to this problem), we dualize and arrive to

$$\min \qquad s + \sum_{i \in C} m_i u_i \tag{10}$$

$$s.t. \qquad s + \sum_{i \in C} \xi_i^w u_i \geq h(z, w) \qquad \forall w \in W \tag{11}$$

$$s \geq 0, \, u_i \leq 0, \, \forall i \in C \tag{12}$$

where we can restrict $s \geq 0$ because the right side of (11) is always non negative and variables u_i are non positive.

We then merge our inner reformulation to the outer minimization problem. We arrive to the formulation of Proposition 1:

$$\min_{z \in Z} \qquad s - \sum_{i \in C} m_i u_i \tag{13}$$

$$s.t. \qquad s - \sum_{i \in C} \xi_i^w u_i \geq h(z, w) \qquad \forall w \in W \tag{14}$$

$$s \geq 0, \, u_i \geq 0, \, \forall i \in C \tag{15}$$

We note that one can easily verify that, at optimality, the duals of constraints (14) correspond to the worst-case probability associated to each scenario, since they reflect the probability of each scenario in the original formulation.

B   Development of complete reformulation in DROC

With first-stage variables defined, the constraints associated with the first stage are,

$$z_{i,j} + z_{j,i} = 1 \tag{16}$$

$$z_{i,j} + z_{j,r} + z_{r,i} \leq 2 \tag{17}$$

$$z^1_{i,j,r} \leq z_{i,r} \tag{18}$$

$$z^1_{i,j,r} \leq z_{r,j} \tag{19}$$

$$z^1_{i,j,r} \geq z_{i,r} + z_{r,j} - 1 \tag{20}$$

$$z^2_{i,j,r} \leq z_{r,i} \tag{21}$$

$$z^2_{i,j,r} \leq z_{r,j} \tag{22}$$

$$z^2_{i,j,r} \geq z_{r,i} + z_{r,j} - 1 \tag{23}$$

$$z^3_{i,j,r} \leq z_{i,r} \tag{24}$$

$$z^3_{i,j,r} \leq z_{j,r} \tag{25}$$

$$z^3_{i,j,r} \geq z_{i,r} + z_{j,r} - 1 \tag{26}$$

where all constraints are valid $\forall i, j, r \in C, i \neq j \neq r$.

Constraints (16) and (17) define the ordering feasible region for the first-stage binary variables $z_{i,j}$. Constraints (16) state that either customer i is served before j or the contrary. Constraints (17) are the so called 3-dicycle inequalities. They state that if customer i is served before j, and j is served before r, r cannot be served before i. Constraints (18) to (26) position customer r with relation to customer i and j. For example, constraints (18) to (20) state that customer r will only be served in between i and j if it is served after i, before j and only if these two conditions happen simultaneously. The other constraints have analogous purpose. Since we are concerned with a minimization problem, constraints (18), (19),(21),(22),(24) and (25) are redundant and can be eliminated from the final formulation. Also, variables $z^1_{i,j,r}$, $z^2_{i,j,r}$ and $z^3_{i,j,r}$ are naturally integer and integrality requirements for these can be relaxed.

The second-stage is defined in a way that we can calculate the cost of a route given the ordering of the first stage and the scenario to be considered. Due to the format of the resultant feasible second-stage region, where uncertainty parameters appear not only at the right hand side of constraints, but also as bilinear coefficients with first stage variables, we opt for equivalently defining the second stage with first-stage variables indexed by the indicator $w \in W = \{1, \ldots, |\Xi|\}$, meaning there is one variable for each possible scenario.

With the sets of main and auxiliary second-stage variables defined, we first define constraints relative to the auxiliary variables, valid $\forall i \in C$, $\forall w \in W$ and $\forall t \in \{0, \ldots, |C|-1\}$, when not stated otherwise:

$$\sum_{t \in \{0,\ldots,|C|-1\}} y^1_{w,i,t} = 1 - \xi^w_i \tag{27}$$

$$\sum_{t \in \{0,\ldots,|C|-1\}} t y^1_{w,i,t} \leq \sum_{j \in C} (1 - \xi^w_j) z_{j,i} \tag{28}$$

$$\sum_{i \in C} y^1_{w,i,t} \leq 1 \tag{29}$$

Constraints (27) state that a customer i can only be associated to one and only position t, if customer i is not outsourced in the referenced scenario. Otherwise there is no assigned position. Constraints (28) assigns of a position to each customer i based on the expression

$\sum_{j \in C}(1 - \xi_j^w)z_{j,i}$, that counts the number of customers before i in the referenced scenario (note the $\leq$ sign of the constraint to accommodate the case when i is outsourced). For this reason we add constraints (29) that assure a maximum of 1 customer for each position t and guarantee together with the other constraints a natural binary solution. We can therefore relax integrality requirement for $y_{w,i,t}^1$.

We next define constraints for the variables $y_{w,i,j}$. Note that $y_{w,i,j} = 1$ means that there is an arc linking customers or depot i and j in scenario ξ^w and this arc is part of a route defined in the second stage. The constraints below are valid $\forall w \in W$ and $\forall i, j, r \in C$, $i \neq j \neq r$.

$$y_{w,i,j} \leq 1 - \xi_i^w \tag{30}$$

$$y_{w,i,j} \leq 1 - \xi_j^w \tag{31}$$

$$y_{w,i,j} \leq z_{i,j} \tag{32}$$

$$y_{w,i,j} \leq \xi_r^w z_{i,j,r}^1 + z_{i,j,r}^2 + z_{i,j,r}^3 \tag{33}$$

$$y_{w,i,j} \geq (1 - \xi_i^w) + (1 - \xi_j^w) + z_{i,j}$$
$$+ \sum_r (\xi_r^w z_{i,j,r}^1 + z_{i,j,r}^2 + z_{i,j,r}^3) - |C| \tag{34}$$

$$y_{w,0,i} \leq 1 - \xi_i^w \tag{35}$$

$$y_{w,0,i} \leq \xi_j^w z_{j,i} + z_{i,j} \tag{36}$$

$$y_{w,0,i} \geq 1 - \xi_i^w + \sum_j (\xi_j^w z_{j,i} + z_{i,j}) - |C| + 1 \tag{37}$$

$$y_{w,i,0} \leq 1 - \xi_i^w \tag{38}$$

$$y_{w,i,0} \leq \xi_j^w z_{i,j} + z_{j,i} \tag{39}$$

$$y_{w,i,0} \geq 1 - \xi_i^w + \sum_j (\xi_j^w z_{i,j} + z_{j,i}) - |C| + 1 \tag{40}$$

Constraints (30) to (34) determine the condition for an arc (i, j) to exist in a second stage, if i and j are not the depot. Variable $y_{w,i,j} = 1$ only if 1) i is not outsourced (30), 2) if j is not outsourced (31), 3) if i is served before j (32) and, 4) for all other customers r, r is positioned before i and j ($z_{i,j,r}^2 = 1$) or after i and j ($z_{i,j,r}^3 = 1$) or, if positioned in between i and j ($z_{i,j,r}^1 = 1$), it is outsourced ($\xi_r^w = 1$). This is guaranteed by constraints (33). Constraints (34) guarantee that all these conditions have to happen simultaneously. Constraints (35) to (37) and constraints (38) to (40) work in an analogous form when one of the nodes of the arc is the depot (0). Because this is a minimization problem, constraints (30), (31), (32), (33), (35), (36), (38) and (39) are redundant and can be eliminated in the final formulation. Variables $y_{w,i,j}$ are naturally binary and the integrality requirement for these variables can be relaxed.

Constraints for variable $v_{w,i,j}$ are defined below. If variable $v_{w,i,j} = 1$, it means that the capacity of a vehicle is reached at customer i and, so, a detour should be performed by returning to the depot and coming back to customer j. This way, variable $v_{w,i,j}$ defines when one vehicle route reaches its ends and another vehicle route should be initiated. The constraints below are valid $\forall w \in W$ and $\forall i, j \in C$, $i \neq j$.

$$v_{w,i,j} \leq \sum_{k \in \{1, \ldots, \lfloor \frac{|C|}{Q} \rfloor\}} y^1_{w,i,kQ-1} \tag{41}$$

$$v_{w,i,j} \leq y_{w,i,j} \tag{42}$$

$$v_{w,i,j} \geq y_{w,i,j} + \sum_{k \in \{1, \ldots, \lfloor \frac{|C|}{Q} \rfloor\}} y^1_{w,i,kQ-1} - 1 \tag{43}$$

Constraints (41) guarantee that capacity is reached at customer i only if it occupies special positions in the ordering of customers relative to the scenario in reference. These positions are given by $kQ - 1$, where $k \in \{1, \ldots, \lfloor \frac{|C|}{Q} \rfloor\}$. Constraints (42) determine that the return is made to the next not outsourced customer, if it exists. Constraints (43) determine that all conditions should happen simultaneously. Again, because this is a minimization problem, constraints (41) and (42) are redundant and can be eliminated in the final formulation. Variables $v_{w,i,j}$ are naturally binary and the integrality requirement of these variables can be relaxed.

C   Detailed implementation of algorithm BPC

In the next subsections we detail the implementation of each feature of the Algorithm 1.

C.1   Branching

We create a search tree with no customers pre positioned at the root node. From the root node, $|C|$ branches lead to $|C|$ nodes on the first level, each of which corresponds to a particular customer being positioned in the first position. Generally, each node at level l in a tree corresponds to a set $J_l \subseteq \{1, \ldots, |C|\}$ filling the first l positions in a given order. By successively placing each customer j ($j \in C \backslash J_l$) in the $(|J_l| + 1)$-th position, $|C \backslash J_l|$ new nodes are created.

A node selection is done by use of a depth-first search strategy, i.e. the node selected is the one, among unprocessed nodes with maximum depth in the search tree. This way we navigate the tree prioritizing the search of new incumbent values. The scenarios accumulated in the solution of a parent node are transmitted to all downward children of the tree.

C.2   Independent marginals lower bound

The authors in [6, Theorem 1 Strategy b] present a closed expression to, given an ordered route, calculate the a priori expected cost under the recourse strategy we have defined for our problem, when marginals are independent. It can be calculated in polynomial time. Since an independent marginal distribution provides a lower bound to our case, we can use it as a means to prune the nodes of the branch-and-bound tree. Each node of our tree defines a partial ordering of the routes to undertake. To approximate the independent marginal expected cost from below we assume that all remaining customers not sequenced in the node ordering have same costs and probability, given by the best or minimum values among them. Since we run under a depth-first search strategy, each iteration of a same branch of the tree provides a better lower bound. Also, we do not have to recalculate the lower bound from the

beginning at each node, and can reuse partially the lower bound calculations of the parent's node. The last level of the tree provides an exact independent marginal expected cost for the respective route.

Let r be a route defined by an ordered sequence of visited customers, C, and let $r(i)$ represent the i-th planned visit in r (with $i = 0$, $i = N + 1$ meaning the Depot and $m_{r(0)} = m_{r(N+1)} = 0$, $c_{r(0),r(N+1)} = 0$). For completeness, and adapting to our case with compensation fees to be payed to ODs, the expression for the a priori independent marginals expected cost for a given route, $E(r)$, is given by

$$
\begin{aligned}
E(r) = \quad & \sum_{i=1}^{N} f_i m_i + \sum_{i=0}^{N} \sum_{j=i+1}^{N+1} \left((1 - m_{r(i)})(1 - m_{r(j)}) \prod_{l=i+1}^{j-1} m_{r(l)} \right) c_{r(i),r(j)} \\
& + \sum_{i=1}^{N} \sum_{j=i+1}^{N} (c_{r(i),r(0)} + c_{r(0),r(j)} - c_{r(i),r(j)}) \gamma_{r(i)} (1 - m_{r(j)}) \prod_{l=i+1}^{j-1} m_{r(l)}
\end{aligned}
\tag{44}
$$

where $\gamma_{r(i)} = 0$, $i \in \{1, \ldots, Q-1\}$, $\gamma_{r(i)} = (1 - m_{r(i)}) \sum_{k=1}^{\lfloor \frac{i}{Q} \rfloor} s(i - 1, kQ - 1)$, $i \geq Q$, and $s(b, r)$ expresses the probability of exactly r customers among the first b customers being not outsourced and is computed by recursion: For $b = 1, \ldots, N$, $r = 1, \ldots, b$, $s(b, r) = (1 - m_{r(b)}) s(b - 1, r - 1) + m_{r(b)} s(b - 1, r)$, with initial conditions $s(b, b) = \prod_{i=1}^{b} (1 - m_{r(i)})$, $s(b, 0) = \prod_{i=1}^{b} m_{r(i)}$.

C.3   Scenario separation problem

Formulation (DROC) can be understood as a two-stage robust optimization problem with exponential number of scenarios and second stage variables. To solve it, we adopt the algorithm developed in [26] where the authors present a constraint-and-column generation algorithm to solve two-stage robust optimization problems. They argue that enumerating all the possible uncertain scenarios is not feasible, but that not all scenarios (and their corresponding variables and constraints) are necessary in defining the optimal value. Probably only a few important scenarios play the significant role in the formulation. The authors emphasize that it is different from the 2-stage stochastic optimization model where every single scenario in the scenario set actually contributes to the optimal value through its realization probability. They also show that the algorithm converges in a finite number of iterations.

Let $\hat{s}$, $\hat{u}$, $\hat{z}$, $\hat{z^1}$, $\hat{z^2}$, $\hat{z^3}$ represent the values of variables s, u, z, z^1, z^2, z^3, respectively, after solving a node restricted problem with integer solution for these variables. The separation problem is given by

$$\min_{\xi,y,v,y^1} \quad \hat{s} + \sum_{i \in C} \hat{u}_i \xi_i - \sum_{i \in C} f_i \xi_i - \sum_{i,j \in V} c_{i,j} y_{i,j} - \sum_{i,j} (c_{i,0} + c_{0,j} - c_{i,j}) v_{i,j}$$

$$y_{i,j} \geq 1 - \xi_i + 1 - \xi_j + \hat{z}_{i,j}$$
$$+ \sum_r (\xi_r \hat{z^1}_{i,j,r} + \hat{z^2}_{i,j,r} + \hat{z^3}_{i,j,r}) - |C|$$

$$y_{i,j} \leq 1 - \xi_i$$
$$y_{i,j} \leq (1 - \xi_j)\hat{z}_{i,j}$$
$$y_{i,j} \leq \xi_r \hat{z^1}_{i,j,r} + \hat{z^2}_{i,j,r} + \hat{z^3}_{i,j,r}$$

$$y_{0,i} \geq 1 - \xi_i + \sum_j (\xi_j \hat{z}_{j,i} + \hat{z}_{i,j}) - |C| + 1$$

$$y_{0,i} \leq 1 - \xi_i$$
$$y_{0,i} \leq \xi_j \hat{z}_{j,i} + \hat{z}_{i,j}$$

$$y_{i,0} \geq 1 - \xi_i + \sum_j (\xi_j \hat{z}_{i,j} + \hat{z}_{j,i}) - |C| + 1$$

$$y_{i,0} \leq 1 - \xi_i \qquad\qquad\qquad\qquad \text{(SEP)}$$
$$y_{i,0} \leq \xi_j \hat{z}_{i,j} + \hat{z}_{j,i}$$

$$v_{i,j} \geq y_{i,j} + \sum_{k \in \{1,\ldots,\lfloor \frac{|C|}{Q} \rfloor\}} y^1_{i,kQ-1} - 1$$

$$v_{i,j} \leq \sum_{k \in \{1,\ldots,\lfloor \frac{|C|}{Q} \rfloor\}} y^1_{i,kQ-1}$$

$$v_{i,j} \leq y_{i,j}$$

$$\sum_{t \in \{0,\ldots,|C|-1\}} y^1_{i,t} = 1 - \xi_i$$

$$\sum_{t \in \{0,\ldots,|C|-1\}} t y^1_{i,t} \leq \sum_{j \in C} (1 - \xi_j) \hat{z}_{j,i}$$

$$\sum_{i \in C} y^1_{i,t} \leq 1$$

$$y_{i,j}, \ y_{0,i}, \ y_{i,0}, \ v_{i,j} \in [0,1], \xi_i \in \{0,1\},$$

where the constraints below are valid $\forall i, j, r \in C$, $i \neq j \neq r$, when not stated otherwise.

If the objective value of problem (SEP) is greater than a given tolerance value we insert the respective value of the scenario solution into our restricted node formulation, together with the respective new variables y, y^1 and v and new associated constraints of problem (DROC), and restart the node solving step of the algorithm .

C.4 Symmetry breaking implementation

The recourse strategy is composed by two components. The first one is defined by skipping the absent customers. The second one is defined by adding detours when a vehicle achieves its capacity at a customer position.

For the first component, there is clearly symmetry since, for any scenario, traversing the route in one direction and skipping absent customers has the same cost as traversing the route in opposite direction. If the recourse is to be defined only by the first component, we can implement a symmetry breaking strategy by ordering the customers lexicographically and filtering all branch nodes where first and last customer in the node ordering cannot be crescent (or decrescent). Since this is valid for all scenarios, it can be used by both the independent marginals and dependent marginals cases.

For the second component there is no symmetry, since the order of traversing the route will define different nodes where the vehicle will achieve capacity and, therefore, different detour costs. There is a lexicographical order of the customers that will lead to a better solution, but identifying that order while solving each branch-and-bound node of our algorithm can be time costly. On the other hand, there are calculations that can be shared by both lexicographical orders during the execution of the algorithm. For instance, calculations for the first component of the recourse, skipping absent customers, can be made only once since this cost is the same for both lexigraphical orders. To profit from the time saving incurred by sharing these calculations we adopt the lexicographical ordering branching filter also for the recourse with the two components. We use the same node to calculate lower bounds for the two orderings, by sharing possible calculations, and consider the minimum lower bound or feasible solution as a result for this node. Note that this adds to the possibility of sharing calculations between a parent and child in the depth-first branching strategy.

C.5   Initial Incumbent solution

For an initial incumbent solution we leverage the work done on heuristics for the probabilistic traveling salesman problem. We refer to the work of [25] where the authors consider different heuristic approaches for this problem. In particular, we adapt the Almost Nearest Neighbor Heuristic ([25]) to our case. By doing this, we attempt to find a solution with a maximum lower bound. Considering independent marginals, we search for an ordering of customers where we append the customer with the lowest change of expected length from the last inserted customer to the tour. For a given set T of customers already inserted in a tour, the cost of inserting customer j can be computed as

$$
\min_{j \in C \setminus T} \sum_{i=1}^{|T|} (1 - m_i)(1 - m_j) c_{i,j} \prod_{k=i+1}^{|T|} m_k,
$$

We solve problem (DROC) using the heuristic solution above and use its value as our first incumbent.

Locating Evacuation Centers Optimally in Path and Cycle Networks

Robert Benkoczi ✉
Department of Mathematics and Computer Science, University of Lethbridge, Canada

Binay Bhattacharya ✉
School of Computing Science, Simon Fraser University, Burnaby, Canada

Yuya Higashikawa ✉
Graduate School of Information Science, University of Hyogo, Kobe, Japan

Tsunehiko Kameda[1] ✉
School of Computing Science, Simon Fraser University, Burnaby, Canada

Naoki Katoh ✉
Graduate School of Information Science, University of Hyogo, Kobe, Japan

Junichi Teruyama ✉
Graduate School of Information Science, University of Hyogo, Kobe, Japan

──── **Abstract** ────

We present dynamic flow algorithms to solve the *k-sink problem* whose aim is to locate k sinks (evacuation centers) in such a way that the evacuation time of the last evacuee is minimized. In the *confluent model*, the evacuees originating from or passing through a vertex must evacuate to the same sink, and most known results on the k-sink problem adopt the confluent model. When the edge capacities are uniform (resp. general), our algorithms for non-confluent flow in the path networks run in $O(n+k^2 \log^2 n)$ (resp. $O(n \log n + k^2 \log^5 n)$) time, where n is the number of vertices. Our algorithms for cycle networks run in $O(k^2 n \log^2 n)$ (resp. $O(k^2 n \log^5 n)$) time, when the edge capacities are uniform (resp. general).

1 Introduction

Ford and Fulkerson [8] introduced the concept of *dynamic flow* which models movement of commodities in a network. Each vertex is assigned some initial amount of supply, and each edge has a capacity, which limits the rate of commodity flow into it, and the transit time to traverse it. Once on an edge, the flow front travels at a constant speed either to a sink, or to the vertex at the other end of the edge if there is no sink on the edge. *Congestion* is said to occur when supplies cannot flow continuously but must wait at some vertex to enter an outgoing edge, and congestion complicates the analysis.

---

[1] Corresponding author.

One variant of the dynamic flow problem is the *quickest transshipment* problem, where the source vertices have given amounts of supplies and the sink vertices have specified demands. The problem is to send exactly the right amount of commodity out of the sources into the sinks in minimum overall time. Hoppe and Tardos [13] provided a polynomial time algorithm for this problem in the case where the transit times are integral. However, the complexity of their algorithm is impractically high. Finding a practical polynomial time solution to this problem is still open. The interested reader is referred to a survey paper by Skutella [17].

The *k-sink problem* is another variant of the dynamic flow problem, whose aim is to locate a set of k sinks that accepts the *evacuees* in minimum time [10, 15]. Each source vertex has an initial number of evacuees, and each sink has infinite demand, namely it can receive an arbitrary number of evacuees with no capacity constraint. Evacuation starts simultaneously from all vertices. For general graphs, even the 1-sink problem is NP-hard [9, 14]. We thus address simple networks (path and cycle) in this paper.

The existing solutions to the evacuation problem impose the condition that all evacuees starting at or passing through a vertex must evacuate to the same sink. This can be justified by the fact that posting "This way out" signs at each vertex, directing the evacuees to a single exit, will avoid confusion. Such flow is called *confluent* flow. Adopting the confluence restriction, Arumugam et al. [2] showed that the k-sink problem for path networks with general edge capacities can be solved in $O(kn \log^2 n)$ time, where n is the number of vertices. A path network can model a corridor in a building, an aisle in an airplane, a street, etc. As for the uniform edge capacity model, Higashikawa et al. [11] then proposed an $O(kn)$ time algorithm. More recently, Bhattacharya et al. [4] improved it to $O(\min\{n+k^2 \log^2 n, n \log n\})$ time, and also presented an algorithm for the general edge capacity model that runs in $O(\min\{n \log n+k^2 \log^4 n, n \log^3 n\})$ time. These improvements were achieved by moving from dynamic programming based approach to parametric search based methods. A recent comprehensive survey on evacuation problems can be found in [12].

In this paper we consider *non-confluent* flow solution to the evacuation problem on path and cycle networks. This means that the evacuees from a vertex can move in two opposite directions. It can be practical, if each potential evacuee is given the exit number beforehand, so that he/she knows exactly which exit to take in case of emergency. We will treat each evacuee as if he/she was a tiny particle with a very small weight. This paper presents an algorithm that runs in $O(n+k^2 \log^2 n)$ (resp. $O(n \log n+k^2 \log^5 n)$) time for the uniform (resp. general) edge capacity model. Benkoczi and Das [3, 7] solve the k-sink problem for cycle networks for confluent flows, which run in time $O(n \log n)$ (resp. $O(n \log^3 n)$) when the edge capacities are uniform (resp. general).

This paper is organized as follows. After preliminaries in Sec. 2, we discuss the uniform capacity model in Secs. 3 and Sec. 4, where we deal with feasibility testing and optimization, respectively. Sec. 5 then discusses the general capacity model, and Sec. 6 extends the results to cycle networks.

2   Preliminaries

2.1   Definitions

Let $P(V, E)$ be a path network with the set of vertices $V = \{1, 2, \ldots, n\}$, arranged from left to right in this order. For each i ($1 \leq i \leq n-1$), there is an edge $e_i = (i, i+1) \in E$, which does not include its end vertices. For each vertex $i \in V$, let $w_i \in \mathbb{Z}_+$ denote its *weight*, which is the initial number of evacuees at vertex i, and for each $e_i \in E$, let $c(i, i+1)$ denote its *capacity*, which limits the number of evacuees who can enter e_i from i or $i+1$ per unit time.

By $a \in P$ we denote the fact that point a lies on P, either at a vertex or on an edge. For any two points $a, b \in P$, we write $a \prec b$ to mean that a lies to the left of b, and $a \preceq b$ means $a \prec b$ or $a = b$. The minimum capacity between a and b is denoted by $c(a, b)$. Let l_i denote the length of edge e_i. We use $d(a, b)$ to denote the distance between a and b, which is the sum of the edge lengths. If a and/or b lies on an edge, its prorated length is used.

For a pair of values or positions x and y, $[x, y]$ denotes the range or interval from x to y, inclusive, while $(x, y]$ (resp. $[x, y)$) denotes the range from x to y, excluding x (resp. y). For $a \preceq b$, $P[a, b]$ (resp. $P(a, b]$, $P[a, b)$) denotes the subpath of P from a to b with the above interpretation of the range. The set of vertices on $P[a, b]$ (resp. $P(a, b]$, $P[a, b)$) are denoted by $V[a, b]$ (resp. $V(a, b]$, $V[a, b)$). For a technical reason, we define v^+ (resp. v^-) to be an imaginary vertex such that $v \prec v^+$, $d(v, v^+)=0$, and $c(v, v^+)=c(v, v+1)$ (resp. $v^- \prec v$, $d(v^-, v)=0$, and $c(v^-, v)=c(v-1, v)$). Let τ denote the unit distance travel time, so that for a evacuee to travel from a to b, without encountering congestion, requires $d(a, b)\tau$ units of time.

Our model assumes that evacuation starts at the same time from every vertex.

▶ **Definition 1.** *Let* $W[i, j] \triangleq \sum_{h \in V[i,j]} w_h$, *and for* $h, i, j \in V$ *such that* $i \preceq h \preceq j$ *define*

$$f_L^{[i,\cdot]}(x, h) \triangleq \begin{cases} d(h, x)\tau + W[i, h]/c(h, x) & \text{for } x \succ h \\ 0 & \text{for } x \preceq h, \end{cases} \tag{1}$$

$$f_R^{[\cdot,j]}(x, h) \triangleq \begin{cases} d(x, h)\tau + W[h, j]/c(x, h) & \text{for } x \prec h \\ 0 & \text{for } x \succeq h. \end{cases} \tag{2}$$

Intuitively, $f_L^{[i,\cdot]}(x, h)$ is the evacuation time of all evacuees on $P[i, h]$ to point $x \succeq h$, assuming that all of them were at vertex h initially, and the flow from $P[h+1, x]$ to x does not interfere with it. Similarly $f_R^{[\cdot,j]}(x, h)$ is the evacuation time of all evacuees on $P[h, j]$ to $x \preceq h$, assuming that all of them were at vertex h initially, and the flow from $P[x, h-1]$ to x does not interfere with their flow. We now define their upper envelopes.

▶ **Definition 2.** *For* $i, j \in V$, *define*

$$\Theta_L^{[i,\cdot]}(x) \triangleq \max_{v \in V[i,x)} \left\{ f_L^{[i,\cdot]}(x, v) \right\} = f_L^{[i,\cdot]}(x, v_x^*), \tag{3}$$

$$\Theta_R^{[\cdot,j]}(x) \triangleq \max_{v \in V(x,j]} \left\{ f_R^{[\cdot,j]}(x, v) \right\} = f_R^{[\cdot,j]}(x, v_x^*). \tag{4}$$

The rightmost (resp. leftmost) vertex v_x^* *satisfying Eq. (3) (resp. Eq. (4)) is called the L-critical vertex (resp. R-critical vertex) for* $P[i, x]$ *(resp.* $P[x, j]$*) w.r.t.* x, *and is denoted by* $\rho_L^{[i,\cdot]}(x)$ *(resp.* $\rho_R^{[\cdot,j]}(x)$*).*

▶ **Lemma 3** ([12]). *For any point* $x \succ i$ *(resp.* $x \prec i$*),* $\Theta_L^{[i,\cdot]}(x)$ *(resp.* $\Theta_R^{[\cdot,j]}(x)$*) is the evacuation time for all evacuees on* $P[i, x)$ *(resp.* $P(x, j]$*) to* x.

We thus refer to $\Theta_L^{[i,\cdot]}(x)$ (resp. $\Theta_R^{[\cdot,j]}(x)$) as the *L-time* (resp. *R-time*) for $P[i, x]$ (resp. $P[x, j]$) at x.

▶ **Definition 4.** *An instance* P *of a path network is said to be* (λ, k)*-feasible or just* λ*-feasible, if* k *sinks can be placed on it so that every evacuee can evacuate to a sink within time* λ*. The* λ*-feasibility test decides if the given instance* P *is* (λ, k)*-feasible.*

2.2 Megiddo's lemma

We shall apply the following lemma implied by Megiddo's observation [16].

▶ **Lemma 5.** *Let $cmp(n)$ be the number of comparisons made with λ in a λ-feasibility test, $t(n)$ be the time needed to generate a λ value to be tested, $f(n)$ be the time complexity of the λ-feasibility test, and $h(n)$ be the time required by all other operations. Then the optimal solution to the k facility location problem can be found in time*

$$O(h(n) + cmp(n)\{f(n) + t(n)\}). \tag{5}$$

For a clear exposition of the idea behind this lemma, the reader is referred to Agarwal and Sharir [1]. Intuitively, we replace each comparison with λ in the λ-feasibility test by a comparison with λ^*, where λ^* is the optimal solution. Note that a feasibility test is actually a comparison of some value λ with λ^*, and it succeeds (resp. fails) if $\lambda \geq \lambda^*$ (resp. $\lambda < \lambda^*$). To determine λ^*, we perform successive λ-feasibility tests, using judiciously chosen λ values.

2.3   Review of CUE tree and CV tree

The *critical vertex tree* (*CV tree*) was introduced in [5], and the *capacity and upper envelope tree* (*CUE tree*) was introduced in [4]. They are balanced binary trees built over path P. Since they play an important role in this paper, we briefly review them for completeness.

The leaf nodes[2] of the CUE-tree, denoted by $\mathcal{T}$, are the vertices of P. See Fig. 1, for example. For node u of $\mathcal{T}$, let $l(u)$ (resp. $r(u)$) denote the leftmost (resp. rightmost) vertex
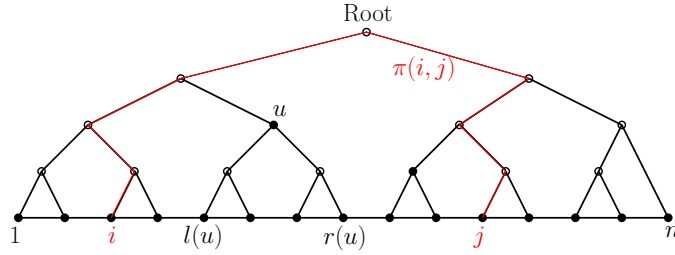
Figure 1 The structure of a CUE tree $\mathcal{T}$.

of P that belongs to subtree $\mathcal{T}(u)$, rooted at u. We say that u *spans* subpath $P[l(u), r(u)]$, whose vertex set is denoted by $V(u)$. Let $N[i, j]$ denote the set of nodes spanning the maximal subpaths of $P[i, j]$. Each node in $N[i, j]$ either lies on the path $\pi(i, j)$ from i to j or is a child of a node on $\pi(i, j)$. Each node u of $\mathcal{T}$ stores

  (i) $l(u)$ and $r(u)$,
 (ii) capacity $c(i, j)$,
(iii) four 1-dimensional arrays, which are described below.

Given $i, j \in V$, consider any node $u \in N[i, j{-}1]$. The L-time at $j \succ r(u)$ for the supplies from $V(u)$ is given by

$$\max_{h \in V(u)} \left\{ d(h, r(u))\tau + \frac{W[i, h]}{\min\{c(h, r(u)^+), c\}} \right\} + d(r(u), j)\tau, \tag{6}$$

where $c = c(r(u), j)$. We rewrite the first term of Eq. (6) as

$$\max_{h \in V(u)} \left\{ d(h, r(u))\tau + \max \left(\frac{W[i, h]}{c(h, r(u)^+)}, \frac{W[i, h]}{c} \right) \right\}, \tag{7}$$

---

[2] We use the term "node" here to distinguish it from the vertices on P.

which obviously equals

$$\max\{\Theta_L^w(W, u), \tilde{\Theta}_L^c(c, u) + W/c\}, \tag{8}$$

where

$$\Theta_L^w(W, u) \triangleq \max_{h \in V(u)} \left\{ d(h, r(u))\tau + \frac{W}{c(h, r(u)^+)} + \frac{W[l(u), h]}{c(h, r(u)^+)} \right\} \tag{9}$$

$$\tilde{\Theta}_L^c(c, u) \triangleq \max_{h \in V(u)} \left\{ d(h, r(u))\tau + \frac{W[l(u), h]}{c} \right\}. \tag{10}$$

Here W and c are unknowns at the time of constructing $\mathcal{T}$ during preprocessing. Note that $\Theta_L^w(W, u)$ is a piecewise linear function in W, consisting of $O(|V(u)|)$ linear segments. We need a method by which to compute the inflection points $(W, \Theta_L^w(W, u))$ of the upper envelope (9). Let $A_L^w(u)$ denote the 1-dimensional *left weight array* stored at node u of $\mathcal{T}$. It contains the W components of the inflection points $(W, \Theta_L^w(W, u))$ in the increasing order of W. For a technical reason, we consider $(0, \Theta_L^w(0, u))$ and (∞, ∞) as inflection points of $(W, \Theta_L^w(W, u))$. Let $A_L^c(u)$ denote the 1-dimensional *left capacity array* stored at node u. For a technical reason, we consider $(0, \infty)$ and $(\infty, \Theta_L^c(\infty, u))$ as inflection points of $\Theta_L^c(c, u)$. The size of $A_L^w(u)$ and $A_L^c(u)$ is clearly $O(|\mathcal{T}(u)|)$.

Suppose we want to compute the L-time of $V[i, j-1]$ at j. To this end, we calculate the contributions from subtree $\mathcal{T}(u)$, for each $u \in N[i, j-1]$ separately, and find their maximum. As we saw above, the contribution from each subtree $\mathcal{T}(u)$ is given by $\max\{\Theta_L^w(W, u), \tilde{\Theta}_L^c(c, u) + W/c\}$, where $W = W[i, l(u)-1]$ and $c = c(r(u), j)$.
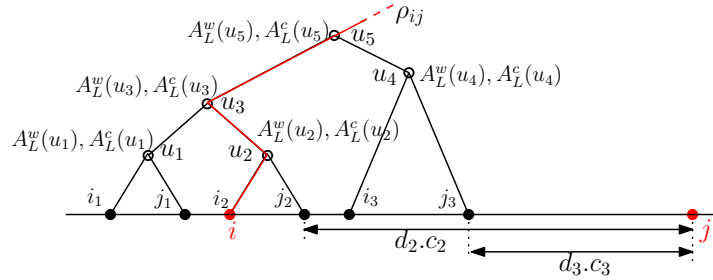
Figure 2 Weight arrays $A_L^w(u)$ and capacity arrays $A_L^c(u)$. $A_R^w(u)$ and $A_R^c(u)$ are not shown in this figure.

▶ **Example 6.** Let $W[u]$ denote the total weights of $V[u]$, and let $u = u_4$ in Fig. 2, for example. Then we search in array $A_L^w(u_4)$ with the search key $W = W[u_2]$, to find the two successive inflection points between which W falls, and obtain the time $\Theta_L^w(W[u_2], u_4)$ by interpolating between the two points. Adding $d_3\tau$ to it, we find the L-time of $V(i, r(u_4))$ at j, which is just one candidate for the true L-time of $V[i, j-1]$ at j. We repeat this for all $u \in N[i, j-1]$. ⌟

As for $\tilde{\Theta}_L^c(c, u)$, it is useful to consider it as a linear function in $1/c$, so that it is also piecewise linear, consisting of $O(|V(u)|)$ linear segments. At u, we store the 1-dimensional *capacity array* $A_L^c(u)$ in the increasing order of c, containing the c components of the inflection points $(c, \tilde{\Theta}_L^c(c, u))$. After searching for $c(r(u_4), j)$ in $A_L^c(u_4)$ in Fig. 2, for example, and interpolation, we need to compute $\tilde{\Theta}_L^c(c(r(u_4), j), u_4) + W[i_2, j_2]/c(r(u_4), j)$ to arrive at another candidate L-time at j.

Symmetrically to (9) and (10), we also define

$$\Theta_R^w(W, u) \triangleq \max_{h \in V(u)} \left\{ d(l(u), h)\tau + \frac{W}{c(l(u)^-, h)} + \frac{W[h, r(u)]}{c(l(u)^-, h)} \right\} \tag{11}$$

$$\tilde{\Theta}_R^c(c, u) \triangleq \max_{h \in V(u)} \left\{ d(l(u), h)\tau + \frac{W[h, r(u)]}{c} \right\}. \tag{12}$$

We construct and store at node u the *right weight array* $A_R^w(u)$ and *right capacity array* $A_R^c(u)$, based on (11) and (12), respectively. Thus $A_R^w(u)$ (resp. $A_R^c(u)$) is left-right symmetric to $A_L^w(u)$ (resp. $A_L^c(u)$).

The *CV tree* [5] is a simplified version of the CUE tree, which is useful for the uniform capacity model. Instead of data (ii) and (iii), node u stores the L-critical (resp. R-critical) vertex $\rho_L^{[l(u), \cdot]}(r(u)^+)$ (resp. $\rho_R^{[\cdot, r(u)]}(l(u)^-)$).

▶ **Lemma 7** ([4, 5]). *The CV tree (resp. CUE tree) can be constructed in $O(n)$ (resp. $O(n \log n)$) time.*

▶ **Lemma 8** ([5]). *Assume CV tree $\mathcal{T}$ is available, and let $i < j$. For the uniform edge capacity model, we can compute*
(a) $\rho_L^{[i, \cdot]}(j)$ *and* $\rho_R^{[\cdot, j]}(i)$ *in $O(\log n)$ time.*
(b) $\Theta_L^{[i, \cdot]}(x)$, $\Theta_R^{[\cdot, j]}(x)$, *and* $\Theta^{[i,j]}(x)$ *in $O(\log n)$ time for any point $x \in P[i, j]$.*

▶ **Lemma 9** ([4]). *Assume CUE tree $\mathcal{T}$ is available, and let $i < j$. For the general edge capacity model, we can compute*
**(a)** $\rho_L^{[i, \cdot]}(j)$ *and* $\rho_R^{[\cdot, j]}(i)$ *in $O(\log^2 n)$ time.*
(b) $\Theta_L^{[i, \cdot]}(x)$, $\Theta_R^{[\cdot, j]}(x)$, *and* $\Theta^{[i,j]}(x)$ *in $O(\log^2 n)$ time for any point $x \in P[i, j]$.*

Using fractional cascading, we can reduce the time complexity in Lemma 9 to $O(\log n)$.

2.4 Strategy

Let k (≥ 2) be the number of sinks to be placed, since if $k = 1$ the flows are always confluent. We want to compute $\{s_i, b_i, \alpha_i \mid 1 \leq i \leq k\}$ that minimize the evacuation time, where s_i is the location of the ith sink from the left, b_i is the rightmost vertex from which at least some evacuees move left to s_i, and is called the *boundary vertex* for s_i. α_i evacuees ($0 < \alpha_i \leq w_{b_i}$) evacuate left from b_i to s_i. Let $\bar{\alpha}_i \triangleq w_{b_i} - \alpha_i$, if $\alpha_i < w_{b_i}$.

Path partitioning idea: Imagine that we have performed the λ^*-feasibility test. The result may look like Fig. 3, where the dots and small circles represent vertices. Each triangle represents a sink, a red triangle represents a sink placed at a vertex, and a dot that is not a sink indicates a boundary vertex whose evacuees are not split. Let us remove the edges
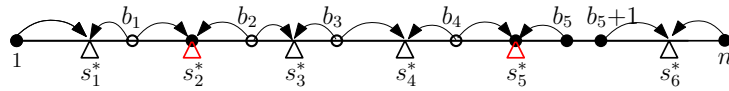
Figure 3 $\{s_i^*\}$ are optimal sinks and $\{b_i\}$ are boundary vertices between adjacent sinks.

carrying no flow, which are incident on non-split boundary vertices, if any, and then divide each sink that lies on a vertex into two sinks, one of which is attached to its left incident edge, and the other to its right incident edge. Then each connected subpath would be one of the four types shown in Fig. 4, where there is at least one sink in each subpath. For example,
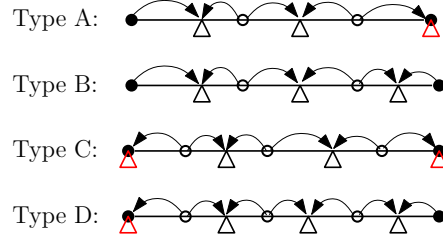
Figure 4 Four types of subpaths.

a subpath of Type A starts with vertex 1 or a vertex that is the right neighbor of a non-split boundary vertex and ends with a sink at a vertex. Our optimization algorithms locate sinks in each subpath in the optimal way. The optimal evacuation time is given by the subpath with the maximum evacuation time.

3 Feasibility test

Given a value (evacuation time) λ, starting from the left end of P, we identify the rightmost point s_1 such that all evacuees on $P[1, s_1)$ can evacuate right to s_1 within time λ. We then determine the *boundary vertex* for s_1, denoted by b_1, such that all evacuees on $P(s_1, b_1-1]$ and $\alpha_1 (0 < \alpha_1 \leq w_{b_1})$ evacuees from b_1 can evacuate left to s_1 within time λ, and b_1 is the rightmost such vertex. We repeat this for the remaining part $P[b_1, n]$ of P, with the weight of b_1 reduced to $\bar{\alpha}_1 = w_{b_1} - \alpha_1$, if $\alpha_1 < w_{b_1}$, and detaching $P[1, b_1]$ from P if $\alpha_1 = w_{b_1}$. It is clear that the given instance P is λ-feasible, if and only if the end vertex n of P is reached before no more than k sinks are introduced this way. We present this approach later as Algorithm **FTest**, after introducing its building blocks formally.

3.1 Finding maximal λ-covered subpaths

We say that vertex i is λ-*covered* by sink s, if $\Theta_L^{[i,\cdot]}(s) \leq \lambda$ or $\Theta_R^{[\cdot,i]}(s) \leq \lambda$. A subpath is said to be λ-*covered*, if every vertex on it is λ-covered by a sink.

3.1.1 Finding the next sink

Given a λ value, assume that we have introduced sinks, $\{s_1, \ldots, s_{i-1}\}$, and the associated boundary vertices, $\{b_1, b_2, \ldots, b_{i-1}\}$, so far for some $i \geq 1$, and we want to locate the new sink s_i. Assume further that each sink λ-covers a maximal subpath, and lies at the rightmost position possible. As the initial condition, we set $\bar{\alpha}_0 = w_1$, which implies that the amount w_1 must be sent to s_1. For $i \geq 2$, α_{i-1} $(0 < \alpha_{i-1} \leq w_{b_{i-1}})$ evacuees travel left to s_{i-1}. If $\alpha_{i-1} < w_{b_{i-1}}$,[3] then $\bar{\alpha}_{i-1} \triangleq w_{b_{i-1}} - \alpha_{i-1}$ (> 0) evacuees must travel right to s_i. Let $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(x)$,[4] where $x \succeq b_{i-1}$, denote the evacuation time for the evacuees from $P[b_{i-1}, x)$ at x, with the weight of b_{i-1} changed to $\bar{\alpha}_{i-1}$. For the purpose of testing λ-feasibility, we want to find a sink s_i, farthest from b_{i-1} on its right, such that $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(s_i) \leq \lambda$. Since $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(x)$ monotonically increases as we move x to the right, we can perform binary search to determine adjacent vertices $v, v+1 \in V$ such that $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v) < \lambda$ and $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v+1) \geq \lambda$. See Fig. 5. Then we can locate sink s_i on subpath $P[v, v+1]$ in constant time.[5]

---

[3] If not, then $b_{i-1}+1$ is like v_1. See b_5+1 in Fig. 3.

[4] It implicitly depends on λ, which indicated by the tilde on Θ.

[5] Note that v is included, because $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) \geq \lambda$ is possible. See Lemma 11.
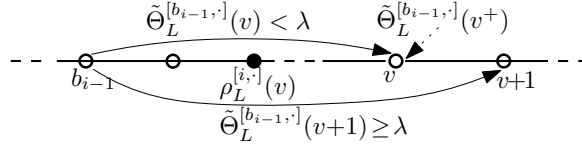
Figure 5 Looking for the next sink s_i.

▶ **Observation 10.** In the uniform edge capacity model, for a point $x \succ b_{i-1}$, the L-critical vertex $\rho_L^{[b_{i-1},\cdot]}(x)$ for $P[b_{i-1}, x]$ w.r.t. x does not depend on $\bar{\alpha}_{i-1}$ as long as $\bar{\alpha}_{i-1} > 0$, and $\rho_L^{[b_{i-1},\cdot]}(x)$ satisfies

$$\tilde{\Theta}_L^{[b_{i-1},\cdot]}(x) = d(\rho_L^{[b_{i-1},\cdot]}(x), x)\tau + \frac{W[b_{i-1}, \rho_L^{[b_{i-1},\cdot]}(x)] - \alpha_{i-1}}{c} \tag{13}$$

$$= d(\rho_L^{[b_{i-1},\cdot]}(x), x)\tau + \frac{W[b_{i-1}+1, \rho_L^{[b_{i-1},\cdot]}(x)] + \bar{\alpha}_{i-1}}{c}, \tag{14}$$

where we define $W[b_{i-1}+1, b_{i-1}] = 0$.   ⌟

▶ **Lemma 11.** *For a given λ, let $v \in V$ satisfy $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v) < \lambda$ and $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) = \tilde{\Theta}_L^{[b_{i-1},\cdot]}(v+1) - l_v\tau \geq \lambda$.[6] Then v is the rightmost possible position for sink s_i.*

Proof. It is clear that if $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) > \lambda$, then s_i must be placed at v. So consider the case where $\tilde{\Theta}_L^{[b_{i-1},\cdot]}(v^+) = \lambda$. In this case, the evacuation time will be $> \lambda$ at any location $\succ v^+$. We could place the sink at the imaginary location v^+, but we might as well place it at v, since there is no difference in the coverage on the left or right side, if s_i is placed at v^+ or at v.   ◀

When the condition of the above lemma holds, s_i is said to be *vertex-bound*, *VB* for short, to v for λ. The L-time jumps beyond λ if the sink is placed at any finite distance (> 0) to the right of v.

▶ **Lemma 12.** *Procedure* **NxtSink**$(\lambda, a, \overline{\alpha})$, *presented below, finds the next sink correctly, and if the CV tree $\mathcal{T}$ is available, then it runs in $O(\log n)$ time, performing $O(\log n)$ comparisons with λ.*

Proof. The correctness follows from the above discussions. The complexity follows from Lemma 8, since the portion of the split weight to be λ-covered by the next sink is known.   ◀

3.1.2   Finding the boundary vertex for sink s_i

We now look for s_i's boundary vertex b_i, as illustrated in Fig. 6.

▶ **Lemma 13.** *Given sink $s_i \in P(v, v+1]$, let b be the leftmost vertex such that $\Theta_R^{[\cdot,b]}(s_i) \geq \lambda$. If $d(s_i, b)\tau > \lambda$, then we must have $b_i = b-1$ and all the w_{b-1} evacuees can evacuate to s_i within time λ. Otherwise, we have $b_i = b$, and a portion of w_b can evacuate to s_i within time λ.*
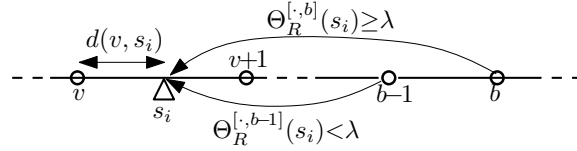
---

[6] This implies that $\rho_L^{[b_{i-1},\cdot]}(v^+) = v$.

■ **Procedure NxtSink**($\lambda, a, \overline{\alpha}$).

Input	: a (vertex with weight $\overline{\alpha}$ $(0 < \overline{\alpha} \le w_a)$), λ (limit on L-time)
Output	: v, $d(v, s)$, $s \in P[v, v+1]$ (next sink)

1 **if** $\tilde{\Theta}_L^{[a,\cdot]}(n) \le \lambda$ **then**
2 $\quad\mid\quad$ $s \leftarrow n$, $v \leftarrow n-1$, and $d(v,s) \leftarrow l_{n-1}$ **stop**.
3 **end**
4 Using binary search, find $h \in [a, \ldots, n-1]$ such that $\tilde{\Theta}_L^{[a,\cdot]}(h) < \lambda$ and
$\quad$ $\tilde{\Theta}_L^{[a,\cdot]}(h+1) \ge \lambda$;
5 **if** $\tilde{\Theta}_L^{[a,\cdot]}(h^+) = \tilde{\Theta}_L^{[a,\cdot]}(h+1) - l_h\tau \ge \lambda$ **then**
6 $\quad\mid\quad$ $d(h,s) \leftarrow 0$; $\qquad\qquad\qquad\qquad\qquad\qquad$ // s is VB to h.
7 **else**
8 $\quad\mid\quad$ $d(h,s) \leftarrow \{\lambda - \tilde{\Theta}_L^{[a,\cdot]}(h^+)\}/\tau$;
9 **end**
10 $s \leftarrow$ point at distance $d(h,s)$ to the right of v ;
11 $v \leftarrow h$.

Figure 6 Finding the boundary vertex b_i for sink s_i.

Proof. If $d(s_i, b)\tau > \lambda$, then clearly it takes more than λ time for the first evacuee from b to arrive at s.

Otherwise, there are two cases to consider. If $\frac{W[h, b_i-1]}{c} < d(h, b_i)\tau$, where $h = \rho_R^{[\cdot, b-1]}$, ($b_i$ is the R-critical vertex w.r.t. s_i), then at least one evacuee from b_i can arrive at s_i within λ. If $\frac{W[h, b_i-1]}{c} \ge d(h, b_i)\tau$ (b_i is not the R-critical vertex w.r.t. s_i), on the other hand, since $\Theta_R^{[\cdot, b-1]}(s_i) < \lambda$, the first arrival from b can reach s_i within λ, hence $b_i = b$. ◄

If $b_i = b-1$ in the above lemma, we call it the *separator vertex* for the current subpath. If $b_i = b$, on the other hand, we look for the split portion α_i $(0 < \alpha_i \le w_b)$ by setting

$$\Theta_R^{[\cdot, b-1]}(s_i) + \alpha_i/c = \lambda,$$

$$\text{or} \quad d(s_i, \rho_R^{[\cdot, b-1]}(s_i))\tau + \frac{W[\rho_R^{[\cdot, b-1]}(s_i), b-1] + \alpha_i}{c} = \lambda. \tag{15}$$

We now solve (15) for α_i, which yields

$$\alpha_i = \{\lambda - d(s_i, \rho_R^{[\cdot, b-1]}(s_i))\tau\}c - W[\rho_R^{[\cdot, b-1]}(s_i), b-1]. \tag{16}$$

If b is the R-critical vertex w.r.t. s_i, then we solve $d(s_i, b)\tau + \alpha_i/c = \lambda$, instead of (15).

Procedure **R-Bnd**(λ, s_i, d), given below, computes the boundary vertex b_i for s_i and also α_i.

Since Step 4 makes $O(\log n)$ probes, we have

▶ **Lemma 14.** *If the CV tree $\mathcal{T}$ is available and the position of s_i is known, **R-Bnd** computes the boundary vertex for a sink in $O(\log n)$ time by comparing $O(\log n)$ values with λ.*

◾ **Procedure R-Bnd**(λ, s_i, d).

Input	: λ, s_i, d // $s_i \in P[v, v{+}1]$ and $d = d(v, s_i)$.
Output	: b_i, α_i ($0 < \alpha_i \le w_{b_i}$)

1 **if** $\Theta_R^{[\cdot,n]}(s_i) \le \lambda$ **then**
2 $\quad$ Set $b_i \leftarrow n$ and $\alpha_i \leftarrow w_n$
3 **else**
4 $\quad$ Using binary search, find vertex b such that $\Theta_R^{[\cdot,b-1]}(s_i) < \lambda$ and $\Theta_R^{[\cdot,b]}(s_i) \ge \lambda$;
5 $\quad$ **if** $d(s_i, b)\tau > \lambda$ **then**
6 $\quad\quad$ $b_i \leftarrow b{-}1$, $\alpha_i \leftarrow w_{b-1}$; $\qquad\qquad\qquad$ // b_i is a separator vertex.
7 $\quad$ **else**
8 $\quad\quad$ $b_i \leftarrow b$, and compute α_i, using (16) with $b = b_i$
9 $\quad$ **end**
10 **end**

3.2 Feasibility test algorithm

Algorithm **FTest** below presents our feasibility test as a pseudo code. It makes $O(k)$ calls to **NxtSink** and **R-Bnd**, which are the most time consuming operations.

Lemmas 12 and 14 imply

▶ **Lemma 15.** *Algorithm* **FTest** *makes* $cmp(n) = O(k \log n)$ *comparisons with* λ.

▶ **Lemma 16.** *If the CV tree* $\mathcal{T}$ *is available, then Algorithm* **FTest** *decides* λ-*feasibility for a given* λ *in* $f(n) = O(k \log n)$ *time.*

Proof. When $i = 1$ in the **while** loop, **NxtSink**($\lambda, b_{i-1}, \bar{\alpha}_{i-1}$)=**NxtSink**($\lambda, 1, w_1$) in Step 3, and it generates the exact distance $d(v, s_1)$ in $O(\log n)$ time by Lemma 12. This distance is fed to **R-Bnd**(λ, s_1, d) in Step 5 as $d = d(v, s_1)$, and it generates the exact split portion α_1 in $O(\log n)$ time by Lemma 14. We can now compute $\bar{\alpha}_1 = w_{b_1} - \alpha_1$ as an input to the second invocation of **NxtSink**. The lemma follows by repeating this argument k times. ◀

4 Optimization for the uniform capacity model

In applying Lemma 5, we now know that $cmp(n) = O(k \log n)$ and $f(n) = O(k \log n)$ from Lemmas 15 and 16, respectively. Thus the remaining problem is to find $t(n)$, which is the time needed to identify the next λ value to be tested for feasibility. Suppose that we have located the first $i{-}1$ sinks $\{s_1, s_2, \ldots, s_{i-1}\}$ on edges, where $i \le k$, based on the current upper bound $\bar{\lambda}$. They are obtained as a result of the last successful feasibility test. Note that a $\bar{\lambda}$-feasibility test has already been performed, because it is how $\bar{\lambda}$ was updated to the current value.

4.1 Finding the next sink in optimal solution

As a result of the last successful feasibility test, the L-time and R-time of each sink s_h ($h \le i{-}1$) equal $\bar{\lambda}$, and we have the sink locations $\{s_h \mid 1 \le h \le i{-}1\}$ on edges and the split portions $\{\alpha_h \mid 1 \le h \le i{-}1\}$ of the boundary vertices $\{b_h \mid 1 \le h \le i{-}1\}$. See Fig. 7. Since we know $\bar{\alpha}_{i-1}$, based on it, we can compute the L-time $\lambda_v = \tilde{\Theta}_L^{[b_{i-1},\cdot]}(v)$ at vertex $v \succ b_{i-1}$, being probed using binary search as a candidate for sink s_i.

Algorithm 1 **FTest**.

Input	$: P, \lambda, k$
Output	$:$ `Feasible`/`Infeasible`, updated $\underline{\lambda}$ or $\overline{\lambda}$, $\{s_i, b_i \mid 1 \le i \le k\}$

1 $i \leftarrow 1; b_0 \leftarrow 1; \bar{\alpha}_0 \leftarrow w_1;$

2 **while** $[i < k] \wedge [\text{Vertex } n \text{ is not } \lambda\text{-}covered \text{ by a sink}]$ **do**

3 $\quad$ By invoking $\mathbf{NxtSink}(\lambda, b_{i-1}, \bar{\alpha}_{i-1})$, find the next sink s_i and $d(v, s_i)$, where
$\quad\quad v \preceq s_i \prec v{+}1;$

4 $\quad$ Set $d \leftarrow d(v, s_i)$; // If $d(v, s_i) = 0$, then s_i is VB to v w.r.t. λ.

5 $\quad$ Run Procedure $\mathbf{R\text{-}Bnd}(\lambda, s_i, d)$; // Generate b_i and α_i.

6 $\quad$ **if** $\alpha_i = w_{b_i}$ **then**

7 $\quad\quad$ $b_i \leftarrow b_i{+}1$; // The next subpath should start from updated b_i.

8 $\quad$ **else**

9 $\quad\quad$ $\bar{\alpha}_i \leftarrow w_{b_i} - \alpha_i;$

10 $\quad$ **end**

11 $\quad$ $i \leftarrow i{+}1$

12 **end**

13 **if** *vertex n is λ-covered by a sink* **then**

14 $\quad$ $\overline{\lambda} \leftarrow \lambda;$

15 $\quad$ Output `Feasible`

16 **else**

17 $\quad$ $\underline{\lambda} \leftarrow \lambda$;

18 $\quad$ Output `Infeasible`.

19 **end**

20 Output $\{s_i, b_i \mid 1 \le i \le k\}$, $\overline{\lambda}$ and $\underline{\lambda}$.
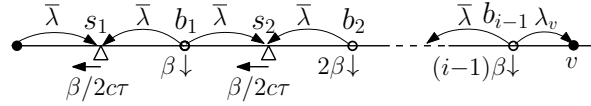
Figure 7 Equalization. Reduction in α_h is $h\beta$ for $1 \le h \le i{-}1$.

Whenever we probe such a vertex v, we need to compute the "equalized" evacuation time at all the sinks $\{s_h \mid 1 \le h \le i{-}1\}$,[7] including the candidate sink s_i that may be placed at v. We may assume that $\lambda_v < \overline{\lambda}$, since otherwise, the equalized time will be larger than $\overline{\lambda}$, and we would already know the outcome of its feasibility test. We thus have a "slack" of $\overline{\lambda}{-}\lambda_v$, which means that the L-time and R-time of each s_h ($1 \le h \le i{-}1$) can be reduced to remove this slack. For equalization, we move sinks $\{s_h \mid 1 \le h \le i{-}1\}$ left, and also reduce the split portions $\{\alpha_h \mid 1 \le h \le i{-}1\}$ to make all the L-time and R-time at all the sinks the same, which should equal the minimum evacuation time for subpath that we are processing, if s_i is placed at v (Type A or C).

▶ **Lemma 17.** *Let $\underline{\lambda}$ and $\overline{\lambda}$ be the current bounds, and let $\{s_1, \ldots, s_{i-1}; b_1, \ldots, b_{i-1}\}$ be the non-VB sinks and split boundary vertices, based on the current $\overline{\lambda}$ that resulted from the last successful feasibility test. If λ is moved within $(\underline{\lambda}, \overline{\lambda})$, then*
(a) *Each sink s_h ($1 \le h \le i{-}1$) moves on the same edge.*
(b) *Each boundary vertex b_h ($1 \le h \le i{-}1$) does not change.*

---

[7] If no sink has been introduced ($i{=}1$), this step is not needed.

Proof. (a) Let $s_h \in (u, u+1)$. In the process of determining that s_h is not VB, we would have tested the feasibility of $\tilde{\Theta}_L^{[b_h-1,\cdot]}(u)$ and $\tilde{\Theta}_L^{[b_h-1,\cdot]}(u+1)$, and the former (resp. latter) must have failed (resp. succeeded). Thus the bounds were updated by $\underline{\lambda} \leftarrow \max\{\underline{\lambda}, \tilde{\Theta}_L^{[b_h-1,\cdot]}(u)\}$ and $\overline{\lambda} \leftarrow \min\{\overline{\lambda}, \tilde{\Theta}_L^{[b_h-1,\cdot]}(u+1)\}$. This implies that s_h cannot reach u or $u+1$ if $\lambda \in (\underline{\lambda}, \overline{\lambda})$, and will stay on edge $(u, u+1)$.

(b) Assume that changing λ within $(\underline{\lambda}, \overline{\lambda})$ caused $\alpha_h = 0$ or $\alpha_h = w_{b_h}$. Let λ_0 (resp. λ_1) be the R-time for sink s_h with $\alpha_h = 0$ (resp. $\alpha_h = w_{b_h}$). In the process of identifying b_h, we would have done the λ_0-feasibility test and λ_1-feasibility test. Since the λ_1-feasibility test must have been successful, $\overline{\lambda}$ would have been set to λ_1 at that time. Moreover, $\overline{\lambda}$ could have been made smaller due to a later feasibility test. Thus changing λ with $\lambda \leq \overline{\lambda}$ cannot move α_h beyond w_{b_h}. Similar argument shows that $\alpha_h > 0$ for $\lambda > \underline{\lambda}$.   ◀

▶ **Corollary 18.** *Let $\underline{\lambda}$, $\overline{\lambda}$, and $\{s_1, \ldots, s_{i-1}; b_1, \ldots, b_{i-1}\}$ be as defined in Lemma 17. Then there is an optimal solution $\{\hat{s}_1, \ldots, \hat{s}_{k-1}; \hat{b}_1, \ldots, \hat{b}_{i-1}\}$ such that for $\forall h\ (1 \leq h \leq i-1)$, $\hat{s}_h$ and s_h lie on the same vertex or same edge, and $\forall h\ (1 \leq h \leq i-1) : \hat{b}_h = b_h$.*

In Fig. 7,[8] for $1 \leq h \leq i-1$, the value, $h\beta$, shown below b_h, indicates the amount by which α_h is reduced. Thus the difference between the values below b_h and b_{h+1} is β for all $1 \leq h \leq i-1$. To accommodate these changes in α_h and α_{h+1}, sink s_h must move to the left by the distance $\beta/2c\tau$ to balance its L-time and R-time. As a result, the evacuation time at each sink gets reduced to $\overline{\lambda} - \beta/2c$. Therefore, $(i-1)\beta$ is the increase in $\overline{\alpha}_{i-1}$ that must be sent to v. To make all the L-times and R-times the same, we should have $\lambda_v + (i-1)\beta/c = \overline{\lambda} - \beta/2c$, from which we get

$$\beta/2c = (\overline{\lambda} - \lambda_v)/(2i-1). \tag{17}$$

Then the L- and R-time of every sink equal

$$\overline{\lambda}_v = \overline{\lambda} - \beta/2c = \{(2(i-1)\overline{\lambda} + \lambda_v\}/(2i-1). \tag{18}$$

Clearly, this $\overline{\lambda}_v$ can be computed in $t(n) = O(1)$ time. We now perform the $\overline{\lambda}_v$-feasibility test to compare it with λ^*, which runs in $f(n) = O(k \log n)$ time by Lemma 16. If the $\overline{\lambda}_v$-feasibility test succeeds, then $\overline{\lambda}_v \geq \lambda^*$, which means that sink s_i needs to be placed at or to the left of v. If it fails, then $\overline{\lambda}_v < \lambda^*$, which means that sink s_i may be VB to v or it needs to be placed to the right of v. If the test is successful (resp. fails), we update $\overline{\lambda} \leftarrow \overline{\lambda}_v$ (resp. $\underline{\lambda} \leftarrow \overline{\lambda}_v$). This is repeated until we either locate a sink or reach vertex n.

The following lemma is the counterpart to Lemma 11.

▶ **Lemma 19.** *Suppose that the $\overline{\lambda}_v$-feasibility test failed, but the $\hat{\lambda}_{v+1}$-feasibility test succeeded. If the λ_v^+-feasibility test succeeds, where*

$$\lambda_v^+ = \{2(i-1)\overline{\lambda} + \lambda_{v+1} - l_v\tau\}/(2i-1), \tag{19}$$

then s_i is VB to v, otherwise, $s_i \in P(v, v+1]$.

---

[8] This is a special case of Fig. 8, and what is in the rest of this subsection follows from the analysis of the general edge capacity model in Sec. 5.1. But we present it here, since it is easier to understand the underlying idea with a simpler model.

Proof. λ_v^+ is obtained by replacing λ_v in (18) by $\lambda_{v+1} - l_v \tau$. Assume that the λ_v^+-feasibility test succeeds, i.e., $\tilde{\Theta}_L^{[b_{i-1}, \cdot]}(v^+) \geq \lambda^*$. In this case we could have either $s_i = v$ or $s_i = v^+$. But there is no advantage in placing it at v^+ over placing it at v, since they make no difference in the evacuation time of the overall solution.[9] If it fails, i.e., $\tilde{\Theta}_L^{[b_{i-1}, \cdot]}(v^+) < \lambda^*$, then we clearly have $s_i \succ v$.[10] ◀

As a result of the λ_v^+-feasibility test in Lemma 19, if s_i turns out to be VB to v, then we have identified a subpath of Type A or C. So we can isolate and discard the subpath ending at this sink s_i. But a copy of s_i should be made, because it is the start vertex of a subpath of Type C or D that comes next. If s_i is not VB, on the other hand, s_i lies on $(v, v+1)$, but we do not know exactly where: we only know that $d(v, s_i)$ depends linearly on λ, as implied by Lemma 17. We now proceed to determine the boundary vertex for s_i.

4.2   Finding next boundary vertex in optimal solution

We want to decide if boundary vertex b_i is a separator vertex, and once we have identified a separator vertex, we remove the edge incident to it from the right. Otherwise, we will only know that b_i will be split.

Assume that we have introduced i sinks so far in the current subpath, and as a result of the last successful $\overline{\lambda}$-feasibility test, the first $i-1$ sinks have the same L-time and R-time that are equal to $\overline{\lambda}$, which is the same as the L-time of s_i. We now reduce them by $\beta/2c$ from $\overline{\lambda}$ for some β to be determined below. Let λ_b' be the R-time at s_i for $P[s_i, b]$, where b is being tested as a possible boundary vertex for s_i. As we argued in Sec. 4.1, it should increase by $(2i-1)\beta/2c$, moving $\{s_h \mid 1 \leq h \leq i\}$ to the left by various distances. We thus have

$$\lambda_b' + (2i-1)\beta/2c = \overline{\lambda} - \beta/2c \Rightarrow \beta/2c = (\overline{\lambda} - \lambda_b')/2i. \tag{20}$$

We now run a $\overline{\lambda}_b$-feasibility test for

$$\overline{\lambda}_b \triangleq \overline{\lambda} - \beta/2c = \{(2i-1)\overline{\lambda} + \lambda_b'\}/2i. \tag{21}$$

This is analogous to how we identifed VB sinks in Sec. 4.1. Here is the counterpart to Lemma 19.

▶ **Lemma 20.** *Suppose that the $\hat{\lambda}_{b-1}$-feasibility test failed, but the $\overline{\lambda}_b$-feasibility test succeeded. If the λ_b^--feasibility test succeeds, where*

$$\lambda_b^- = \{(2i-1)\overline{\lambda} + w_{b-1}/c\}/2i, \tag{22}$$

then $b-1$ is the separator vertex for the current subpath. If it fails, $b_i = b$ and b is a split vertex.

Intuitively, if the λ_b^--feasibility test succeeds, then $b-1$ cannot accept any more evacuees within the equalized time $\overline{\lambda} - \beta/2c$. If $b-1$ is the separator vertex, we end up with Type B or D. If $b_i = b$ is a split vertex, we do not compute its split portion α_i at this time. The updated $\overline{\lambda}$ is used to find the next sink, using it in Eq. (17).

---

[9] Moreover, there is no physical point corresponding to v^+ other than v.

[10] Note that if the critical vertex for $P[b_{i-1}, v]$ w.r.t. v is the same as that w.r.t. v^+, then $\tilde{\Theta}_L^{[b_{i-1}, \cdot]}(v^+) = \tilde{\Theta}_L^{[b_{i-1}, \cdot]}(v) < \lambda^*$, so $\tilde{\Theta}_L^{[b_{i-1}, \cdot]}(v^+) \geq \lambda^*$ cannot happen.

▶ **Lemma 21.** *The total time to find all the λ values to be tested per sink and boundary vertex is $t(n) = O(k \log n)$, where $t(n)$ is defined in Lemma 5.*

Proof. We need to probe for the candidate vertex v for the next sink s_i (see Sec. 4.1) and boundary vertex b_i, and compute λ_v, $\overline{\lambda}_v$, and λ_v^+ (resp. λ_b', $\overline{\lambda}_b$, and λ_b^-). The dominant cost is incurred for finding λ_v and λ_b', which is $O(\log n)$ time, hence $t(n) = O(\log n)$.   ◀

4.3   Time complexity of the algorithm

The dominant component of $h(n)$ in Lemma 5 is the time used to construct the CV tree $\mathcal{T}$, which is $O(n)$ by Lemma 7. Lemmas 5, 15, 16, and 21 now imply

▶ **Theorem 22.** *We can find the k-sink in path networks with uniform edge capacities in $O(n+k^2 \log^2 n)$ time.*

5   General edge capacity model

We want to make Lemma 17 and Corollary 18 valid for the general edge capacity model, but different edge capacities introduce some complications. First of all, we need to use the CUE tree $\mathcal{T}$, instead of the CV tree, to find various L-time and R-time. The main issue is computing the next λ value to be tested in optimization, and finding the time $t(n)$ needed to carry out this operation.

Assume that the sinks and boundary vertices have been placed up to boundary vertex b, and each sink has L-time and R-time equal to $\overline{\lambda}$. To locate the next sink, we want to identify two adjacent vertices v and $v+1$ such that $\tilde{\Theta}_L^{[b,\cdot]}(v)$ (resp. $\tilde{\Theta}_L^{[b,\cdot]}(v+1)$) is infeasible (resp. feasible). Similar operations need to be performed to find the next boundary vertex.

5.1   Evacuation time equalization
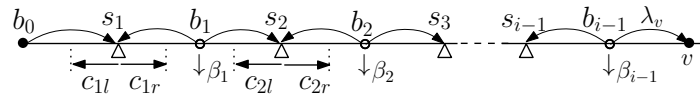
Figure 8 Evacuation time equalization.

5.1.1   Assuming the L- and R-critical vertices of each sink is unique

Fig. 8 corresponds to Fig. 7 in the uniform capacity model. Thus for $1 \le h \le i-1$, β_h is the reduction amount in α_h, and c_{hl} (resp. c_{hr}) is the capacity between sink s_h and its L-critical (resp. R-critical) vertex $\rho_L^{[b_{h-1},\cdot]}(s_h)$ (resp. $\rho_R^{[\cdot,b_h]}(s_h)$). If we start with the evacuation time based on the up-to-date $\overline{\lambda}$ and spread the slack to compute the equalized time λ for the feasibility test, then this λ value may not be the correct evacuation time at v, since λ was computed based on the (possibly wrong) assumption that $\rho_L^{[b_{h-1},\cdot]}(s_h)$ and $\rho_R^{[\cdot,b_h]}(s_h)$ would not change when λ is reduced from $\overline{\lambda}$. But in general, they may change. So the problem is how to ensure that critical vertices do not change. Our approach is to make the interval $(\underline{\lambda}, \overline{\lambda})$ sufficiently small so that as long as λ is varied within the constraint $\underline{\lambda} < \lambda < \overline{\lambda}$, they do not change We address this issue after presenting an evacuation time equalization method, assuming that the operations involved do not change critical vertices.

This assumption implies that capacities c_{hl} and c_{hr} do not change, as we move s_h to the left to equalize its L-time and R-time and reduce α_h by β_h, within the constraint that the value λ to be tested for feasibility satisfies $\underline{\lambda} < \lambda < \overline{\lambda}$.

Starting from the left end of the subpath of Type A that we are processing, if we reduce α_1 by β_1, the L-time and R-time of s_1 get reduced from $\overline{\lambda}$ to $\overline{\lambda} - \beta_1/2c_{1r}$. In general, reducing α_{h-1} by β_{h-1} (increasing $\overline{\alpha}_{h-1}$ by β_{h-1}) causes s_h to move to the left by some distance δ_h. Similarly, the decrease in the R-time at s_h is $\beta_h/c_{hr} - \delta_h\tau$. Equating these decreases, we get

$$\delta_h\tau - \beta_{h-1}/c_{hl} = \beta_h/c_{hr} - \delta_h\tau \Rightarrow \delta_h\tau = (\beta_{h-1}/c_{hl} + \beta_h/c_{hr})/2,$$

where $\beta_0 = 0$. We thus obtain

$$\delta_h\tau - \beta_{h-1}/c_{hl} = \beta_h/c_{hr} - \delta_h\tau = (\beta_h/c_{hr} - \beta_{h-1}/c_{hl})/2, \tag{23}$$

and the L-time and R-time get reduced from $\overline{\lambda}$ to

$$\overline{\lambda} - (\beta_h/c_{hr} - \beta_{h-1}/c_{hl})/2. \tag{24}$$

Equating the evacuation time reduction (23) for $h=1$ and $h=2$, we get

$$\beta_1/2c_{1r} = (\beta_2/c_{2r} - \beta_1/c_{2l})/2,$$

from which we can express β_2 as $\beta_2 = a_2\beta_1$ for constant $a_2 = c_{2r}(1/c_{2l} + 1/c_{1r})$. Equating the evacuation time reduction (23) for $h=1$ and $h=3$, we get

$$\beta_1/2c_{1r} = (\beta_3/c_{3r} - \beta_2/c_{3l})/2.$$

Substituting $\beta_2 = a_2\beta_1$ in this equality, we can express $\beta_3 = a_3\beta_1$ for some constant a_3. In general, we have $\beta_h = a_h\beta_1$ for a constant a_h, which is a function of capacities $\{c_{jl}, c_{jr} \mid 1 \leq j \leq h\}$. Moreover, a_h can be obtained in $O(1)$ time when h is incremented, since c_{jl} and c_{jr} ($j \leq h-1$) remain the same by our assumption, hence it is a linear function of $\overline{\lambda}$ whose coefficients are known.

We can now determine β_1 by equating the L-time at v and the L-time at s_1 as follows.

$$\lambda_v + \beta_{i-1}/c(b_{i-1}, v) = \overline{\lambda} - \beta_1/2c_{r1}. \tag{25}$$

With this β_1, the L-time and R-time of every sink equal

$$\overline{\lambda}_v = \overline{\lambda} - \beta_1/2c_{r1}. \tag{26}$$

This equalized time is used for $\overline{\lambda}_v$-feasibility testing. The other types of subpaths can be analyzed similarly.

5.1.2 Making the L- and R-critical vertices of each sink unique

As part of preprocessing, for each node u of CUE tree $\mathcal{T}$, we construct array $\Lambda_R^w(u)$ from $A_R^w(u)$ by replacing each element W_h in it by the corresponding time, $\lambda_u^h \triangleq \Theta_R^w(W_h, u)$, which is the R-time w.r.t. $l(u)^-$ for $V(u)$ plus W_h coming from the right side of $r(u)$. In other words, (W_h, λ_u^h) is an inflection point. We define array $\Lambda_L^w(u)$ symmetrically to $\Lambda_R^w(u)$.

To make our assumption about the uniqueness of c_{hl} and c_{hr} valid, we update $\underline{\lambda}$ and $\overline{\lambda}$ on additional occasions, which makes the interval $(\underline{\lambda}, \overline{\lambda})$ sufficiently narrow. Consider, for example, a subpath of Type A and, without loss of generality, let $b_0 = 1$ be its leftmost vertex. In finding the edge on which s_1 should lie, using binary search, we look for two

adjacent vertices v and $v+1$ such that the L-time $\Theta_L^{[1,\cdot]}(v)$ (resp. $\Theta_L^{[1,\cdot]}(v+1)$) is infeasible (resp. feasible). By Lemma 9(b), we can compute $\Theta_L^{[1,\cdot]}(v)$ in $O(\log^2 n)$ time, and by the definition of v, we have

$$\Theta_L^{[1,\cdot]}(v) < \lambda^* \leq \Theta_L^{[1,\cdot]}(v+1). \tag{27}$$

So we can update the bounds by $\underline{\lambda} \leftarrow \max\{\underline{\lambda}, \Theta_L^{[1,\cdot]}(v)\}$ and $\overline{\lambda} \leftarrow \min\{\overline{\lambda}, \Theta_L^{[1,\cdot]}(v+1)\}$. Thus the optimal s_1^* will lie on $P[v, v+1]$. Ignoring the special case of $s_1^* = v+1$, let $s_1^* \in e_v = (v, v+1)$. Then it is clear that the L-critical vertex w.r.t. s_1^* is the same as the L-critical vertex w.r.t. $v+1$, and we already know it is unique at this point.

Next, we look for the boundary vertex b_1 for s_1. Let $\overline{s}_1$ be the position of the first sink, whose L-time is $\overline{\lambda}$ that is the most up-to-date upper bound. Using binary search, we probe vertex $b \succ \overline{s}_1$, computing R-time $\lambda_b = \Theta_R^{[\cdot,b]}(\overline{s}_1) < \overline{\lambda}$.[11] Using this λ_b, we equalize the L-time and R-time of s_1, and test the equalized value, $\overline{\lambda}_b$, for feasibility. This way, after one feasibility test per probed vertex, we can identify two adjacent vertices b and $b+1$ such that $\Theta_R^{[\cdot,b]}(s_1)$ is infeasible and $\Theta_R^{[\cdot,b+1]}(s_1)$ is feasible, where s_1 is the sink with the equalized L-time and R-time. Note that these tests are counted in $cmp(n)$.

Before proceeding further, we want to make sure that any $s_1 \in (v, v+1)$ considered in the future will have a unique R-critical vertex, for $P[s_1, b+1]$, as long as $\lambda \in (\underline{\lambda}, \overline{\lambda})$ for the most up-to-date $\underline{\lambda}$ and $\overline{\lambda}$ and that $\alpha_{b+1} > 0$ holds. To this end, we need to narrow the interval $(\underline{\lambda}, \overline{\lambda})$ sufficiently. More generally, let $\underline{\lambda}$ and $\overline{\lambda}$ be the current bounds, and let $\{s_1, \ldots, s_{i-1}; b_1, \ldots, b_{i-1}\}$ be the non-VB sinks and split boundary vertices, based on the current $\overline{\lambda}$ that resulted from the last successful feasibility test. Suppose that $s_{i-1} \in (v, v+1)$ and $b_{i-1} = b+1$.

For each $u \in N[v+1, b]$, we do binary search in $\Lambda_R^w(u)$, and based on the probed value, we first equalize the L-time and R-time of $\{s_h \mid 1 \leq h \leq i-1\}$. We then perform $O(\log n)$ feasibility tests to identify two adjacent values, $\lambda_u^{g_u}, \lambda_u^{g_u+1} \in \Lambda_R^w(u)$ such that the feasibility test for the corresponding equalized value, $\overline{\lambda}_u^{g_u}$ (resp. $\overline{\lambda}_u^{g_u+1}$) fails (resp. succeeds). We now update the bounds by

$$\underline{\lambda} \leftarrow \max\left\{\underline{\lambda}, \max_{u \in N[v+1, b]}\{\overline{\lambda}_u^{g_u}\}\right\}, \tag{28}$$

$$\overline{\lambda} \leftarrow \min\left\{\overline{\lambda}, \min_{u \in N[v+1, b]}\{\overline{\lambda}_u^{g_u+1}\}\right\}. \tag{29}$$

What we have done so far essentially is to identify a unique critical vertex per node $u \in N[v+1, b]$. We need $O(\log n)$ more feasibility tests to make the critical vertex for $P[v+1, b]$ w.r.t. $s_i \in (v, v+1)$ unique.

▶ **Lemma 23.** *Let $\{s_1, \ldots, s_{i-1}; b_1, \ldots, b_{i-1}\}$ be the non-VB sinks and split boundary vertices, based on the current $\overline{\lambda}$ that resulted from the last successful feasibility test. For any equalized value $\lambda \in (\underline{\lambda}, \overline{\lambda})$, where $\underline{\lambda}$ (resp. $\overline{\lambda}$) is given by (28) (resp. (29)), the R-critical vertex for any $s_{i-1} \in (v, v+1)$ is unique, and we can compute (28) and (29) in $O(f(n) \log^2 n)$ time.*

Proof. For any $u \in N[v+1, b]$, we have $\underline{\lambda} \geq \overline{\lambda}_u^{g_u}$ and $\overline{\lambda} \leq \overline{\lambda}_u^{g_u+1}$. Therefore, $\lambda \in (\underline{\lambda}, \overline{\lambda})$ implies that λ lies in a unique position among the values in $\Lambda_R^w(u)$.

For each of the $O(\log n)$ probed values from $\Lambda_R^w(u)$, we can evaluate the equalized value $\overline{\lambda}_u^{g_u}$ in $O(k)$ time, and $\overline{\lambda}_u^{g_u}$ can be tested for feasibility in $f(n)$ time. This is repeated $O(\log n)$ times, resulting in $O(f(n) \log n)$ time. The total time for all nodes u in $N[v+1, b]$ is thus $O(f(n) \log^2 n)$. ◀

---

[11] Vertex b with $\Theta_R^{[\cdot,b]}(\overline{s}_1) \geq \overline{\lambda}$ is of no interest if $s_1^* \prec \overline{s}_1$.

This ensures that if any λ value satisfying $\underline{\lambda} < \lambda < \overline{\lambda}$ is tested for feasibility in the future, the unique critical vertex $\rho_R^{[\cdot,b+1]}(s_{i-1})$ is already known. Since we assume a subpath of Type A, $b+1$ is split in the optimal solution. Moreover, we can easily find the bounds on α_{b+1} corresponding to $\underline{\lambda}$ and $\overline{\lambda}$.

Let b_{i-1} be the last boundary vertex introduced, which is split, and consider vertex $v \succ b_{i-1}$ which is being probed as a candidate for s_i. See v in Fig. 8. We want to narrow the interval $(\underline{\lambda}, \overline{\lambda})$ sufficiently, so that the L-critical vertex for the subpath $P[b_{i-1}, s_i]$ w.r.t. s_i is unique when λ-feasibility is tested for any $\lambda \in (\underline{\lambda}, \overline{\lambda}]$. To this end, we do binary search in $\Lambda_L^w(u)$ for each $u \in N[b_{i-1}+1, v]$. Let $\lambda_u^{g_u}$ be the probed element. Using $\lambda_u^{g_u}$, we equalize the L-time and R-time of all s_h $(1 \le h \le i-1)$, and perform the feasibility test for the resulting equalized value $\overline{\lambda}_u^{g_u}$. This way we can identify two adjacent values $\lambda_u^{g_u}$ and $\lambda_u^{g_u+1}$ in $\Lambda_L^w(u)$ such that the feasibility test for the equalized value $\overline{\lambda}_u^{g_u}$ (resp. $\overline{\lambda}_u^{g_u+1}$) fails (resp. succeeds). The rest is similar to what we did for sink s_{i-1} above. This ensures that if any λ value satisfying $\underline{\lambda} < \lambda < \overline{\lambda}$ is tested for feasibility in the future, the L-critical vertex for the subpath $P[b_{i-1}, s_i]$ w.r.t. s_i is unique and already known. To summarize,

▶ **Lemma 24.** *Let $\{s_1, \ldots, s_i; b_1, \ldots, b_{i-1}\}$, where $s_i \in (v, v+1]$, be the sinks and split boundary vertices that resulted from the last successful feasibility test. In $O(f(n)\log^2 n)$ time, we can further reduce the interval $(\underline{\lambda}, \overline{\lambda})$ sufficiently, so that if any value $\lambda \in (\underline{\lambda}, \overline{\lambda})$ is tested for feasibility, the L-critical vertex for any s_i is unique.*

Clearly, the uniqueness of the critical vertices implies the uniqueness of the capacities $\{c_{hl}, c_{hr} \mid 1 \le h \le i-1\}$ in Fig. 8. Note that if the equalized λ lies outside the current interval $(\underline{\lambda}, \overline{\lambda})$, then we immediately know if it is feasible or not.

5.2 Complexity

▶ **Lemma 25.** *If the CUE tree $\mathcal{T}$ is available, then for any λ, we can decide λ-feasibility in $f(n) = O(k \log^2 n)$ time.*

Proof. The proof is similar to that of Lemma 16, except that it takes $O(\log^3 n)$ time to identify the maximal λ-interval by binary search, using the comment after Lemma 9. ◀

Since we already know $h(n)$, $cmp(n)$, and $f(n)$, the only remaining task is to find $t(n)$. Lemmas 23 and 24 imply

▶ **Lemma 26.** *If the CUE tree $\mathcal{T}$ is available, generating the values to be tested for feasibility takes $t(n) = O(f(n)\log^2 n) = O(k \log^4 n)$ time.*

The CUE tree can be constructed in $h(n) = O(n \log n)$ time by Lemma 7. We have $cmp(n) = O(k \log n)$, just as for the uniform capacity model. Finally, Lemmas 5, 25, and 26 imply our second main theorem.

▶ **Theorem 27.** *For the general edge capacity model, we can find the optimal k-sink in $O(n \log n + k^2 \log^5 n)$ time*

6 Cycle networks

We shall show that the feasibility test for non-confluent flows on cycle networks can be performed in time that is no more than n times the time needed by the feasibility test for path networks. It is known that [3, 7] the k-sink problem for cycle networks with confluent flows can be solved in $O(n \log n)$ (resp. $O(n \log^3 n)$) time when the edge capacities are uniform (resp. general). Given a cycle network $C = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the the vertex set, and the edge set E is given by $E = \{(i, i+1) \mid i = 1, 2, \ldots, i-1\} \cup \{(n, 1)\}$.

▶ **Lemma 28.** *There is always an optimal solution in which either at least one sink is located at a vertex or there is an edge that carries no flow.*

Proof. To prove the above claim, for a cycle C, let $s_1^*, s_2^*, \ldots, s_k^*$ be the locations of optimal sinks arranged clockwise, and b_i be the boundary vertices between s_i^* and s_{i+1}^*. If a sink is on a vertex, then we can divide it into two vertices, converting C into a path. If the boundary vertex b_i for sink s_i^* is not split, then cutting the edge (b_i, b_i+1) converts C into a path.

Therefore, assume that all the optimal sinks are on edges and that all boundary vertices are split. Pick a sink s_i^*, and gradually move it clockwise. We adjust the split portions of the boundary vertices to compensate for this move. Eventually one of the two things will occur. (i) some sink s_j^* reaches a vertex,[12] or (ii) a boundary vertex b_l is no longer split. In the former case, we cut the vertex into two vertices, which become the end vertices of the resulting path. In the latter case, the edge between b_l and b_l+1 will carry no flow any more, and it can be removed, resulting in a path from b_l+1 to b_l. ◀

This implies that we can solve the problem as follows. We create n paths by dividing each vertex into two vertices, and another n paths by removing each edge. We then solve the k-sink problem for each of these $2n$ paths. The solution with the minimum evacuation time is our overall solution. Let P_i denote the path that results by removing edge $(i, i+1)$, where vertex $n+1$ is interpreted as 1. We can solve the problem for each P_i in $O(n \log n + k^2 \log^5 n)$ time by Theorem 27. Thus the total time for all such paths is given by

$$O(n^2 \log n + k^2 n \log^5 n). \tag{30}$$

However, we can save time on the preprocessing time $h(n)$ as follows. We make a copy of path P_n, name its vertices $\{n+1, n+2, \ldots, 2n\}$, and connect P_n and its copy by introducing a new edge $(n, n+1)$. This results in a path P' of length $2n-1$. We now construct the CUE tree for this P'. In solving the problem for P_i for any i, whenever the value $\Theta_L^{[b_{i-1}, \cdot]}(v)$ or $\Theta_R^{[\cdot, b]}(s_i)$ is needed, we can use a portion of the CUE tree to compute it. Since we can construct this CUE tree in $O(n \log n)$ time, we have $h(n) = O(n \log n)$. This implies that we can replace the first term in (30) by $n \log n$, which is dominated by the second term.

▶ **Theorem 29.** *If the edge capacities are uniform (resp. general), we can find the optimal k-sink in cycle networks in $O(k^2 n \log^2 n)$ (resp. $O(k^2 n \log^5 n)$) time.*

7 Conclusion and discussion

We have presented algorithms to find a k-sink on path networks when the flow is non-confluent. For the uniform capacity model, the time complexity of our algorithm is asymptotically the same as the corresponding algorithm for confluent flow discussed in [4]. For the general capacity model, however, it takes longer than the corresponding algorithm in [4]. We showed that a similar approach can be used to find a k-sink on cycle networks, but the time complexity increases.

A model in which the sinks are constrained to be in a prescribed set of vertices might be more realistic. We can apply our methods developed in this paper with only small changes to find a solution in such a model.

---

[12] $j = i$ is possible.

There are a few open problems. First of all, can we improve the feasibility test for cycle networks? Tree networks appear much harder to deal with. For the confluent flow model, Chen and Golin [6] propose an $O((k+\log n)k^2 n \log^3 n)$ (resp. $O((k+\log n)k^2 n \log^4 n)$) time algorithm for finding a k-sink in the uniform (resp. general) capacity model. One of the difficulties in the non-confluent flow model for tree networks is that a split portion of a vertex cannot be represented by just one variable α_i per vertex, because a vertex may have many neighbors. Another serious problem is that the optimal split portion may be time-dependent.

─── **References** ───

1 P.K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Computing Surveys*, 30(4):412–458, 1998.

2 G.P. Arumugam, J. Augustine, M. Golin, Y. Higashikawa, N. Katoh, and P. Srikanthan. Optimal evacuation flows on dynamic paths with general edge capacities. *CoRR abs/1606.07208*, pages 1–37, 2016.

3 R. Benkoczi and R. Das. The min-max sink location problem on dynamic cycle networks. Unpublished manuscript, October 2018.

4 B. Bhattacharya, M. Golin, Y. Higashikawa, T. Kameda, and N. Katoh. Improved algorithms for computing k-sink on dynamic flow path networks. In *Proc. Algorithms and Data Structures Symp., Springer-Verlag, LNCS 10389*, pages 133–144, 2017.

5 B. Bhattacharya and T. Kameda. Improved algorithms for computing minmax regret sinks on path and tree networks. *Theoretical Computer Science*, 607:411–425, November 2015.

6 D. Chen and M.J. Golin. Minmax centered k-partitioning of trees and applications to sink evacuation with dynamic confluent flows. In *CoRR abs/1803.09289*, 2018.

7 R. Das. Minmax sink location problem on dynamic cycle networks. Master's thesis, University, of Lethbridge, Lethbridge, Canada, 2018.

8 L.R. Ford and A.D.R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6(3):419–433, 1958.

9 M.J. Golin, H. Khodabande, and B. Qin. Non-approximability and polylogarithmic approximations of the single-sink unsplittable and confluent dynamic flow problems. In *Proc. 28th Int'l Symp. on Algorithms and Computation (ISAAC)*, pages 41:1–41:13, 2017.

10 H.W. Hamacher and S.A. Tjandra. Mathematical modelling of evacuation problems: a state of the art. *in: Pedestrian and Evacuation Dynamics, Springer Verlag,*, pages 227–266, 2002.

11 Y. Higashikawa, M.J. Golin, and N. Katoh. Multiple sink location problems in dynamic path networks. *Theoretical Computer Science*, 607(1):2–15, 2015.

12 Y. Higashikawa and N. Katoh. A survey on facility location problems in dynamic networks. *The Review of Socionetwork Strategies*, 13:163–208, September 2019.

13 B. Hoppe and É. Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25(1):36–62, 2000.

14 N. Kamiyama. *Studies on Quickest Flow Problems in Dynamic Networks and Arborescence Problems in Directed Graphs: A Theoretical Approach to Evacuation Planning in Urban Areas.* PhD thesis, Kyoto University, 2005.

15 S. Mamada, K. Makino, and S. Fujishige. Optimal sink location problem for dynamic flows in a tree network. *IEICE Trans. Fundamentals*, E85-A:1020–1025, 2002.

16 N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4:414–424, 1979.

17 M. Skutella. An introduction to network flows over time. In *Research Trends in Combinatorial Optimization, W. Cook, L. Lovasz, and J.Vygen, Eds.*, pages 451–482. Springer Verlag, 2009.

Distance-Based Solution of Patrolling Problems with Individual Waiting Times

Peter Damaschke ✉ 🔾

Department of Computer Science and Engineering, Chalmers University, Göteborg, Sweden

—— **Abstract** ———————————————————————————

In patrolling problems, robots (or other vehicles) must perpetually visit certain points without exceeding given individual waiting times. Some obvious applications are monitoring, maintenance, and periodic fetching of resources. We propose a new generic formulation of the problem. As its main advantage, it enables a reduction of the multi-robot case to the one-robot case in a certain graph/hypergraph pair, which also relates the problem to some classic path problems in graphs: NP-hardness is shown by a reduction from the Hamiltonian cycle problem, and on the positive side, the formulation allows solution heuristics using distances in the mentioned graph. We demonstrate this approach for the case of two robots patrolling on a line, a problem whose complexity status is open, apart from approximation results. Specifically, we solve all instances with up to 6 equidistant points, and we find some surprising effects, e.g., critical problem instances (which are feasible instances that become infeasible when any waiting time is diminished) may contain rather large individual waiting times.

1 Introduction

Planning of periodically returning complex tasks for an unlimited time horizon and with different request frequencies is an abundant type of problems. For example, traffic companies want to offer clocked connections with different frequencies, and they have to construct timetables to serve these demands. Similarly, vehicles of shipping agencies may have to pick up, transport, and deliver goods from and to certain points periodically, and within prescribed maximum time intervals, and their routes must be planned. However, in the present paper we consider a type of problems that is conceptually somewhat simpler, in that fixed places rather than routes must be served, in a certain sense.

A set of distinguished points is given. Each of them must be perpetually visited by some vehicle, such that at most some prescribed waiting time elapses between any two consecutive visits of this point. These waiting times are individual, that is, they can differ for different points. As an application example, certain important places in some technical installation must be visited for monitoring and maintenance purposes, or for fetching some product or removing garbage. Some places need attention more frequently than others. If several identical vehicles are available, it does not matter which vehicle serves which point. Rather, every point must always be served within the prescribed waiting time by *some* of the vehicles. The problem is to plan a schedule for all these visits.

In a more general setting, a number of such tasks must be perpetually done within prescribed waiting times as described above, but we have some more freedom: Any task can be performed at several alternative places, and we can arbitrarily choose one of them. For instance, a pipe or lead or supply line may be checked at different points; consumable goods that must be renewed periodically may be fetched at various places to choose from, etc.

Visiting points periodically is called patrolling. In this paper we introduce a generic PATROLLING problem which encompasses some classic path problems in graphs as well as various patrolling problems from the literature [1, 3, 4, 6, 7, 8, 9] where also applications are discussed. The following part is more abstract and technical.

An instance of the PATROLLING problem, as we define it now, consists of an undirected finite graph $G = (V, E)$ named the *position graph*, whose n vertices are called *positions*, furthermore, $m + 1$ subsets $P_i \subset V$ $(i = 0, \ldots, m)$ called *properties*, and positive integers t_i $(i = 0, \ldots, m)$, where t_i is called the *waiting time* of property P_i. In other words, a problem instance is a graph and a hypergraph (on the same vertex set) with integer hyperedge weights.

We may say that $v \in V$ "has the property" P_i if $v \in P_i$. Nothing special is assumed about the hyperedges P_i, and a position may have several properties. Note that our indices begin with $i = 0$; the only reason is that this will yield more convenient expressions when we study a specific class of instances later on.

As usual, a *cycle C* in a graph G is a sequence $v_1, \ldots, v_k$ of vertices $v_j \in V$ such that $v_j v_{j+1}$ is an edge for every $j \in \{1, \ldots, k-1\}$, and $v_k v_1$ is an edge, too. It is important to notice that C may cross itself, that is, vertices may appear in C multiple times. The start vertex is immaterial, that is, any cyclic shift $v_{j+1}, \ldots, v_k, v_1, \ldots, v_j$ is the same cycle. We may *walk* a cycle C perpetually, which means to go round C infinitely often. A *round trip* on a path is the cycle obtained by travelling the path back and forth. Now we are ready to specify the problem.

PATROLLING:

Given a graph and a hypergraph on the same vertex set, and an individual waiting time t_i for every hyperedge (property) P_i, find a *solution cycle C* in G, satisfying the following condition for every i: When we walk C, there are never t_i consecutive vertices without property P_i.

Next we connect this formal definition to the above scenario of moving vehicles. From mow on we call them "robots" rather than vehicles, to comply with the terminology in earlier literature.

Imagine that some robot can move around in the graph. At every moment, the robot is at some position in V. Time is discrete, and in every time step the robot can move to some adjacent position. (We may also allow it to stay at its current position, but this option has no benefits, in terms of the problem.) For every index i, the robot must visit P_i at least once in every time interval of t_i steps, and in the case $|P_i| > 1$ it is immaterial which vertex with property P_i is chosen. The problem can also be formulated in directed graphs, but in this paper we consider only the undirected case where movements are reversible.

Most importantly, our formulation of PATROLLING also encloses cases where a fleet of $r > 1$ robots in a position graph H must perpetually visit every property P_i with waiting time at most t_i. (Note that it is not prescribed which of the robots visits P_i next. The only demand is that P_i must always be visited by *some* robot within the next t_i time units.) A reduction of this multi-robot version to PATROLLING is quite obvious: We define a position graph G whose vertices are the r-tuples of vertices of H, indicating all robots' positions. Hence, for constant r, the blow-up is polynomial. Two vertices $(u_1, \ldots, u_r) \neq (v_1, \ldots, v_r)$ are adjacent if and only if, for every index j, the vertices u_j and v_j are identical or adjacent in H. Moreover, to any r-tuple we assign all properties of its entries, and no further property. Now, the single robot in G represents r robots in H.

In more complicated applications of PATROLLING, vertices of the position graph may model states of a system rather than points in space. For instance, a small factory may want to regularly produce an assortment of diverse products, but it must adapt its machines each time when it switches to another product. Then we may use vertices and edges to represent arrangements of the machines and transitions between them, respectively.

We remark that, in PATROLLING, we are actually looking for an infinite path I in G where, for every index i, no subpath of t_i vertices is disjoint from P_i. But for simple reasons we can aim at a solution cycle instead: If I exists, we can divide I in subpaths of $s := \max_i t_i$ vertices. Since at most n^s different such subpaths can exist, by the pigeonhole principle, some subpath with at most $s(n^s + 1)$ vertices begins and ends with the same subpath of s vertices, which contains at least one vertex of each property. By identifying these two ends we get a solution cycle. This justifies our formulation with a cycle, and it also shows the existence of some solution cycle with length at most sn^s, if $(t_0, \ldots, t_m)$ has a solution at all. If G has maximum degree Δ, then similarly we get the existence of a solution cycle of length at most $ns\Delta^s$. A quite different argument yields another upper bound of ns^m. We mention these further bounds here without proof, as we will not further use them.

However, some order-theoretic concepts will be central. For any two vectors V and W of $m + 1$ waiting times we write $V \leq W$ if every waiting time in V is smaller than or equal to the corresponding waiting time in W, and $V < W$ if $V \leq W$ but $V \neq W$.

Given a position graph with a family of $m + 1$ properties, we call an integer vector $T = (t_0, \ldots, t_m)$ *feasible* if PATROLLING has a solution cycle with waiting time at most t_i for every property P_i, and we call a feasible vector T *critical* if no vector $T' < T$ is feasible. Also the solution cycle itself is called feasible or critical.

As we noticed in [5], for every fixed hypergraph, the number of critical vectors is finite, since the set of positive integer vectors of fixed length is well-quasi ordered (WQO). Hence also t is always bounded by some constant. However, WQO alone does not hint to specific bounds on the number of critical vectors, the maximum waiting times therein, and the lengths of solution cycles. Time bounds from naive exhaustive search would be prohibitive.

Therefore, the main goal of the present work is to provide heuristics for actually solving certain instances of PATROLLING, i.e., for constructing solution cycles or showing infeasibility. We will see that distances in the position graph are very informative for the problem.

A particularly intriguing case is the problem from [1] where two robots are patrolling on a line. This might appear to be a simple setting at first glance, but it is far from being simple. In [5] we have shown the existence of a PTAS: Any desired approximation ratio $1 + \varepsilon$ for the waiting times can be achieved by solving a discrete problem with $m + 1$ equidistant points, where m only depends on ε. But the weak spot is that just this discrete version is poorly understood, and even its complexity status (polynomial or NP-hard) is still open. (The existence of a PTAS does not hinge on this unknown complexity, as it requires polynomial time only for every fixed ε.) This is amazing, noticing that the one-robot problem on a line is trivial. Feasible two-robots instances can require some well choreographed "pas de deux". As pointed out in [5], the practical usability of the PTAS depends on exact solutions to discrete problem instances up to some size m. Therefore we use this particular problem as our playground and case study for our approach to PATROLLING, although the ideas are by no means limited to this case. We may also use them for more robots and other topologies (trees, cycles, grids, etc.). Since they contain *two robots on a line* as a special case, it is sensible to start with that. It seems worth considering also for other reasons: By way of contrast, in PATROLLING in a metric space where all distances are 1 (known as PINWHEEL

SCHEDULING), r robots do not add interesting aspects; just the waiting times are scaled by a factor r, because all robots can freely jump. Two robots on a line seems to be the "simplest" nontrivial case where the number of robots matters, due to the underlying metric.

We notice that the PERIODIC LATENCY problem in [2] (where also further related problems with many potential applications are mentioned) is very similar to PATROLLING, and the multiple robots version on the line can be solved in polynomial time by dynamic programming. However, a crucial difference is that every point must be assigned to a unique robot there, whereas in PATROLLING, robots can share their work arbitrarily.

We apply the above simple reduction from two robots to one robot. While it complicates the position graph and the family of properties (see Section 3.1), this is more than compensated by simpler descriptions of solutions, just by cycles traversed by *one* token in *one* graph. Moreover, we can take advantage of distances in the position graph.

Contributions

Our first contribution is the generic formulation of the PATROLLING problem itself, as motivated above. Then we relate it to some classic path problems in graphs, and we provide some simple but powerful distance-based lower bounds on the waiting times. Next we use them to solve instances of the two-robots-on-a-line problem. We can solve them completely and for arbitrary m, when some small waiting times (up to 3) are present. For larger waiting times, the problem becomes considerably more intricate. To our surprise, instances with $t_k = 2$ and $t_{k+1} = 3$ for some index k are already more peculiar, and there exist many critical vectors under this constraint, some with pretty large t_0 and t_m. This tempers our initial hopes that it could be a practical method to enumerate all critical vectors and a solution to each. (Such an enumeration would trivially solve all other instances as well.) But the results show that this innocently looking problem is surprisingly deep and structurally rich. Nevertheless, to show these effects and also the power of the distance approach, we try to enumerate the critical vectors for small sizes m. In the paper we do this completely for $m \leq 5$ (remember that despite WQO this is not trivial even for fixed m) and still partially for $m = 6$. We conclude with the general lessons and directions for further research. In the technical part, readers may skip many of the detailed case inspections without losing track.

2   Distances in the Position Graph

Let $d(u,v)$ denote the distance of the vertices u and v in $G = (V, E)$, or in any related graph when it is clear from context. The distance is the length, i.e., number of edges, of a shortest u-v-path. For $X, Y \subset V$, an X-Y-path is any x-y-path where $x \in X$ and $y \in Y$. We define $d(X, Y) = \min\{d(x,y) \mid x \in X, y \in Y\}$ and abbreviate $d(\{x\}, Y)$ by $d(x, Y)$.

We use $R \subset V$ to denote any set satisfying that every solution cycle must be entirely in $G[R]$, the subgraph of G induced by R, and equipped with the properties $P_i \cap R$. The *range* of a property P_i to be the set $R_i := \{v \in V \mid \exists u \in P_i : d(u,v) \leq \lfloor t_i/2 \rfloor\}$. Obviously, every solution cycle must be entirely in R_i. Thus we may initially set $R := \bigcap_i R_i$. By applying further necessary conditions we might then be able to restrict R further.

Similarly, we use $R' \subseteq E$ to denote any set of edges satisfying that every solution cycle can only traverse edges in R'. Initially we can make R' the set of all edges in $G[R]$. But we might also be able to restrict R' further. For instance, if $t_i = 2$, then edges not incident to P_i can be deleted from R'.

As simple as the following lemma is, it gives powerful lower bounds on the waiting times in feasible instances.

▶ **Lemma 1.** *Let $t > 0$ be an integer, and P_i and P_j any two properties, where $t_i \leq 2t + 1$. Then every solution cycle satisfies the following:*

- *Every visit of P_j is at the link of a P_i-P_j path and a P_j-P_i path, one of which has length at most t.*
- *If $t = d(P_i, P_j)$ and $t_i = 2t$, then both mentioned paths have length t.*
- *If $t = d(P_i, P_j)$ and $t_i = 2t + 1$, then one of the mentioned paths has length t, and the other path has length t or $t + 1$.*
- *Moreover, we cannot visit any vertex v with $d(v, P_i) \geq t + 1$.*

Proof. Consider the path between the last/first visit of P_i before/after any visit of P_j. Since its length is at most $2t + 1$, one of the two mentioned subpaths has a length at most t. The next two assertions follow instantly from the definition of $d(P_i, P_j)$. Finally, the length of the path between the last/first visit of P_i before/after some visit of v would be at least $2(t + 1)$, which contradicts $t_i \leq 2t + 1$. ◀

Some reformulations and special cases of Lemma 1 are also useful: Applying it to $t := d(P_i, P_j) - 1$ for any two properties P_i and P_j, we get by contradiction that $t_i \geq 2t + 2 = 2d(P_i, P_j)$. Hence

$$t_i \geq 2 \max_j d(P_i, P_j)$$

holds for every fixed j. By setting instead $t := d(P_i, P_j)$ we get assertions for every solution cycle for $t_i \leq 2d(P_i, P_j) + 1$: In this case the mentioned paths of length t must be shortest P_i-P_j-paths. Let D be the vector with components $t_i := 2 \max_j d(P_i, P_j)$ for $i = 0, \ldots, m$. The above inequality says that every feasible vector T satisfies $T \geq D$. If D itself is feasible, then D is therefore the only critical vector.

The following theorem is not difficult, however, it may be interesting to notice the connection to some classic path problems.

▶ **Theorem 2.** PATROLLING *with two properties is equivalent to finding some shortest path between the two hyperedges, and* HAMILTONIAN CYCLE *is polynomial-time reducible to* PATROLLING, *which is therefore NP-hard.*

Proof. Consider instances with only two properties X and Y. By Lemma 1, both waiting times must be at least $2d(X, Y)$. Conversely, if both waiting times are at least $2d(X, Y)$, then the round trip on any shortest X-Y-path is a solution cycle.

Next we present a reduction from HAMILTONIAN CYCLE. Given any graph with ν vertices, we declare every single vertex a property, and we set all waiting times equal to ν. Then any Hamiltonian cycle is obviously a solution cycle. Conversely, consider any subpath of ν vertices in a solution cycle for PATROLLING. Due to the waiting times ν, it must contain every vertex, hence it must contain every vertex exactly once. Furthermore, the next vertex in the cycle must equal the first vertex of this path. Thus we have identified a Hamiltonian cycle in the graph. ◀

We remark that membership in NP is unclear, because there may not exist a polynomial bound on the cycle length for PATROLLING in general, such that the standard way of verifying a solution in polynomial time is not available,

For the remainder of the paper we introduce some more terminology. A *constraint* is an inequality or equation of the form $t_k \leq c$ or $t_k = c$, with a constant c, or a conjunction of some of them. Given a graph and a hypergraph of properties, we call a constraint feasible if it can be satisfied by a feasible vector, and infeasible else. If a constraint contains equations, we call a vector *critical under the constraint* if it is feasible, but no waiting time outside the equations can be lowered. Such a vector is not necessarily critical, because it might still be possible to lower some of the waiting times that are fixed by the constraint.

3 Two Robots on a Line

3.1 The Position Graph

In the case of PATROLLING studied in [5], which is a discretized version of the problem from [1], two robots are patrolling on a line. To be precise, the position graph H is the path with $m + 1$ vertices indexed $0, \ldots, m$, and the $m + 1$ properties are the single vertices.

We set up the stage where the following study will take place. Let x and y be integer variables for the robots' positions, where $x \geq y$. (Whenever the robots meet, we can swap their roles.) Then the vertex set of the position graph G from the reduction in Section 1 consists of all points with integer coordinates (x, y), where $0 \leq y \leq m$, $0 \leq x \leq m$, and $x \geq y$. We informally call G "the triangle", since its convex hull forms a right triangle with one cathetus on the x-axis and the hypotenuse on the line $y = x$. We can identify the hypotenuse with H, as i is mapped to (i, i). Two vertices $(x, y) \neq (x', y')$ of G are adjacent if and only if both $|x - x'| \leq 1$ and $|y - y'| \leq 1$. Property P_i coming from vertex i of H is the union of the vertical line $x = i$ and the horizontal line $y = i$ which meet at point (i, i) on H. That is, every P_i is Γ-shaped, except for P_0 and P_m which are the catheti of the triangle.

As a detail, vertices on H are never needed in a solution cycle C: Assume that C contains a vertex (i, i) whose two neighbors in C are not in H. Then these two vertices are identical or adjacent. If both have the property P_i, we can simply remove (i, i) from C. If the neighbors are $(i, i - 1)$ and $(i + 1, i - 1)$, we can replace (i, i) in C with $(i + 1, i)$, and similarly in the symmetric case, or if both neighbors are $(i + 1, i - 1)$. If C contains a path of two or more vertices in H, then let $(i - 1, i - 1)$ and (i, i) be its end, hence the next vertex in C is $(i + 1, i)$ or $(i + 1, i - 1)$ or $(i, i - 1)$. In either case we can replace (i, i) in C with $(i, i - 1)$ or remove it. Thus we successively get rid of all vertices in H, hence it suffices to use the triangle of vertices (x, y) with $x > y$.

Defining $a_i := i - \lfloor t_i/2 \rfloor$ and $b_i := i + \lfloor t_i/2 \rfloor$, the range R_i of property P_i is the set of all vertices in the triangle that satisfy $a_i \leq x \leq b_i \vee a_i \leq y \leq b_i$. For the interval lengths we have $b_i - a_i = 2\lfloor t_i/2 \rfloor$, which equals t_i for even t_i, and $t_i - 1$ for odd t_i. Below we will describe the intersection $R := \bigcap_i R_i$ of ranges.

All critical solutions where the intervals $[0, m] \cap [a_i, b_i]$ in H have an empty intersection are obtained as follows [1]: For any fixed $d \in [1, m - 2]$, split H into $[0, d]$ and $[d + 1, m]$, and let one robot zigzag in each of these two parts. We rephrase this known result:

▶ **Proposition 3.** *For $m \geq 3$, all critical vectors with $\forall i : t_i \geq 2$ and $\bigcap_i [a_i, b_i] = \emptyset$ are given by $t_i = 2 \max\{i, d - i\}$ for all $i \leq d$, and $t_j = 2 \max\{j - d - 1, m - j\}$ for all $j \geq d + 1$, where d is any fixed integer with $1 \leq d \leq m - 2$.*

By the informal notion of a *BB path* (abbreviation of "billiard ball path") we mean a path consisting of straight line segments with slope $+1$ or -1 that changes direction only by reflection at the border of R. The solution cycles to the critical vectors in Proposition 3 are then exactly the BB paths in $[d + 1, m] \times [0, d]$.

▶ **Definition 4.** *Let $\mathcal{M}_0$ denote the set of all critical vectors from Proposition 3.*

From now on we assume for the waiting times that the intervals $[0, m] \cap [a_i, b_i]$ in H have a nonempty intersection, which is denoted $[a, b] := [0, m] \cap \bigcap_i [a_i, b_i]$. With $a' := \max_i a_i$ and $b' := \min_i b_i$ we have $a = \max\{a', 0\}$ and $b = \min\{b', m\}$. These two numbers are cornerstones in the characterization of the intersection R of ranges given below.

R contains the *stripes* $a \le x \le b$ and $a \le y \le b$. Furthermore, R cannot contain vertices with $x < a$ or $y > b$, but R may intersect the rectangle $Q := [b+1, m] \times [0, a-1]$. For any vertex $(x, y) \in Q$, the following statements are equivalent: $(x, y) \in R \iff \forall i : (x, y) \in R_i \iff \forall i : y \ge a_i \lor x \le b_i \iff \not\exists i : y < a_i \land x > b_i$. Geometrically this means that $Q \cap R$ is obtained from Q by cutting out all quadrants with upper left corner of the form $(b_i + 1, a_i - 1)$, hence $Q \cap R$ is the region above some increasing staircase curve. Now we also characterize which of these quadrants intersect Q at all.

▶ **Lemma 5.** *Every feasible vector satisfies $t_i \ge 2i$ for $i < a$, and $t_j \ge 2(m - j)$ for $j > b$. Furthermore, if all these inequalities are satisfied, then $Q \cap R$ is obtained from Q by cutting out all quadrants with upper left corner of the form $(b_k + 1, a_k - 1)$, for all $k \in [a, b]$. In fact, we have $a_i \le 0$ for $i < a$, and $b_j \ge m$ for $j > b$.*

Proof. As argued above, we must cut out exactly the mentioned quadrants; it remains to show that only the indices $k \in [a, b]$ are needed. Thus, consider any $i < a$. (For $j > b$ we proceed similarly.) From the definition of a_i and b_i we conclude $b_i - a_i + 1 \ge t_i$. Since no vertices (x, y) with $x < a$ are in R, only the horizontal line of P_i crosses R, which implies $2d(P_0, P_i) = 2i$ in R. Lemma 1 implies $t_i \ge 2d(P_0, P_i) = 2i$, or the instance is not feasible. Stacking these inequalities together, we obtain $b_i - a_i + 1 \ge 2i$. Since $b_i - a_i$ is even, this further implies $b_i - a_i \ge 2i$. Since also $(a_i + b_i)/2 = i$, it follows $a_i \le 0$, thus the quadrant with upper left corner $(b_i + 1, a_i - 1)$ does not intersect Q. ◀

The following lemma only rephrases some inequalities known from the definition of $[a, b]$ and from Lemma 5, and presents lower bounds on the waiting times for every fixed $[a, b]$.

▶ **Lemma 6.** *Every feasible vector satisfies $t_i \ge 2i$ and $t_i \ge 2(b - i)$ for all $i < a$, and similarly, $t_j \ge 2(m - j)$ and $t_j \ge 2(j - a)$ for all $j > b$.*

Another general remark is: Due to the inherent symmetry of the problem, every statement about a vector $(t_0, \ldots, t_m)$ holds also true for its reversal. To avoid many tiresome repetitions of this fact, from now on, every vector we talk about can also mean its reversal. In other words, we do not distinguish between $(t_0, \ldots, t_m)$ and $(t_m, \ldots, t_0)$.

3.2 Short Waiting Times

In this section we study the consequences of the presence of the smallest possible waiting times t_k in a critical vector $(t_0, \ldots, t_m) \notin \mathcal{M}_0$. The motivation is twofold. Firstly, R is then mainly composed of two narrow stripes, which should make the solution cycles relatively simple. Secondly, given some vector of waiting times $(t_0, \ldots, t_m)$, even if we only aim at solutions with good approximation ratios rather than exact solutions, the smallest t_k could not be relaxed. The following theorem collects some cases of instances with small waiting times that can be completely solved. Quite surprisingly, the constraint $t_k = 2 \land t_{k+1} = 3$ turned out to be a much more subtle case (expect if $k = 1$), therefore it is not listed in the following result.

▶ **Theorem 7.** PATROLLING *for two robots on a line is solvable in $O(m)$ time when $(t_0, \ldots, t_m)$ contains some 1, or two neighbored $2s$, or two neighbored $3s$, or one 2 neighbored by two $4s$. Moreover, each of the following constraints yields exactly one critical vector under the respective constraint:*

- $t_k = 1$ *for any m and k;*
- $t_k = t_{k+1} = 2$ *for $m \geq 3$ and $1 \leq k < k+1 \leq m-1$;*
- $t_1 = 2 \wedge t_2 = 3$ *for $m \geq 4$;*
- $t_k = t_{k+1} = 3$ *for $m \geq 5$ and $2 \leq k < k+1 \leq m-2$.*

Furthermore, no critical vector has $t_0 = 3$ or $t_m = 3$, and every critical vector not captured by the above cases satisfies $T \geq (4, 4, 2, \ldots, 2, 4, 4)$.

▶ **Definition 8.** *We define $\mathcal{M}_1$ to be the set of the critical vectors under the constraints in Theorem 7.*

The remainder of this section is devoted to the proof of Theorem 7. The scheme is as follows. We consider some constraint with some small waiting time(s) and the resulting position graph with vertex set R and edge set R' as defined in Section 3.1. Recall that R is the union of stripes $a \leq x \leq b$ and $a \leq y \leq b$ plus some subset of Q, and that $d(P_i, P_0) = i$ and $d(P_j, P_m) = m - j$ holds for all $i < a$ and $j > b$, implying $t_i \geq 2i$ and $t_j \geq 2(m - j)$. Note that we will define the position graph and interval $[a, b]$ using the considered constraints only. (They might further shrink due to other waiting times, but this does not affect the following conclusions.) In this position graph we will observe that $d(P_i, P_m) = m - i - c$ and $d(P_0, P_j) = j - c$ holds for some fixed number c and for certain (maybe all) indices $i < a$ and $j > b$. Then Lemma 1 also yields $t_i \geq 2(m - i - c)$ and $t_j \geq 2(j - c)$ for these indices. If we can construct a solution cycle that matches all lower bounds, it follows that the obtained vector of waiting times is the unique critical vector under the constraint.

Let $T = (t_0, \ldots, t_m)$ always denote some critical vector. Wildcard symbol $*$ may be used for unspecified coordinates. We will frequently apply Lemma 5 to obtain the position graphs, and Lemma 1 and its consequences to obtain lower bounds, but without explicitly citing the lemmas, for the sake of brevity.

Constraint $t_0 = 1$ implies that R is the line $y = 0$ from $(1, 0)$ to $(m, 0)$. Note that $d(P_1, P_j) = j - 1$ and $d(P_j, P_j) = m - j$ for all j. The round trip on R yields optimal waiting times $t_j = 2 \max\{j - 1, m - j\}$ for all $j \geq 1$.

Constraint $t_k = 1$, for some k with $1 \leq k \leq m - 1$, implies $a = b = k$, hence R consists of the lines $x = k$ and $y = k$. That is, R is merely a path, and we have $c = 1$, as we can skip (k, k). The round trip on R yields $t_i = 2 \max\{i, m - i - 1\}$ for all $i \leq k - 1$, and $t_j = 2 \max\{j - 1, m - j\}$ for all $j \geq k + 1$.

Every further critical vector T satisfies $T \geq (2, \ldots, 2)$.

For $m \geq 3$, the set $\mathcal{M}_0$ contains the feasible vector given by the waiting times $t_0 = t_1 = 2$ and $t_j = 2 \max\{j - 2, m - j\}$ for all $j \geq 2$.

Constraint $t_0 \leq 3$ implies $b = 1$. Hence all P_j with $j \geq 2$ are vertical lines, thus their waiting times cannot be smaller than in the above solution above from $\mathcal{M}_0$ It follows that this solution is the only critical vector with $t_0 \leq 3$. In particular, no critical vector with $t_0 = 3$ exists, and similarly for t_m.

Every further critical vector T satisfies $T \geq (4, 2, \ldots, 2, 4)$.

Constraint $t_k = t_{k+1} = 2$, for some k with $1 \leq k < k+1 \leq m-1$, implies $[a, b] = [k, k+1]$ and $c = 2$. See Figure 1. The round trip on the path consisting of the edge $(k+1, k-1)(k+2, k)$ and BB paths in the two stripes yields $t_i = 2 \max\{i, m-i-2\}$ for all $i \leq k-1$, $t_k = t_{k+1} = 2$, and $t_j = 2 \max\{j - 2, m - j\}$ for all $j \geq k + 1$. Hence this constraint admits exactly one critical vector.
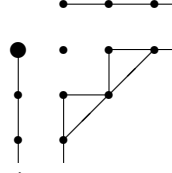
Figure 1 Position graph for the constraint $t_k = t_{k+1} = 2$. The picture illustrates the vertices in R and, for simplicity, the edges that do *not* belong to R'. We use the same convention also in all subsequent pictures. Vertex (k, k) is highlighted here.
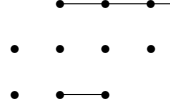
Figure 2 Position graph for the constraint $t_1 = 2 \land t_2 = 3$. The vertex in the lower left corner is $(1, 0)$.

Constraint $t_1 = 2 \land t_2 = 3$ implies $[a, b] = [1, 2]$. See Figure 2. Since $d(P_0, P_m) = m - 3$, we have $t_0 \geq 2(m - 3)$.

Constraint $t_1 = 2 \land t_2 = 3 \land t_0 = 2(m - 3)$ forces every solution cycle to include some shortest P_0-P_m-path, and thus the edge $(3, 0)(4, 1)$, if $m \geq 4$. Since $t_2 \leq 3$, the previous vertex must be $(2, 1)$, whose distance to P_j $(j \geq 3)$ is $j - 2$. This shows $t_j \geq 2(j - 2)$ for every $j \geq 3$. For $m \geq 5$, the round trip on the path that begins with $(2, 1)(3, 0)(4, 1)$ and continues as BB path in the horizontal stripe attains these bounds: $t_0 = 2(m - 3)$, $t_1 = 2$, $t_2 = 3$, and $t_j = 2 \max\{j - 2, m - j\}$ for all $j \geq 3$. This shows that this constraint admits exactly one critical vector.

For $t_0 = 2(m - 3) + 1$, the conclusions in the previous paragraph still hold true. Since the vector with $t_0 = 2(m - 2)$, $t_1 = t_2 = 2$, and the same values t_j for all $j \geq 3$ is feasible (as seen earlier), we conclude that a larger t_0 does not yield further critical vectors. Thus, already the constraint $t_1 = 2 \land t_2 = 3$ admits only one critical vector.

Constraint $t_1 \leq 3 \land t_2 \leq 3$ implies that the solution cycle from $t_1 = 2 \land t_2 = 3$ still has optimal waiting times for the same reasons, hence raising t_1 from 2 to 3 does not provide a new critical vector.

Constraint $t_2 = t_3 = 3$ implies $[a, b] = [2, 3]$. See Figure 3. The round trip on $(2, 1)(3, 0)(4, 1)(5, 2) \ldots (m, *)$, where the last part is a BB path in the horizontal stripe, achieves $t_0 = 2(m - 3)$, $t_1 = 2(m - 4)$, $t_2 = t_3 = 3$, and $t_j = 2 \max\{j - 2, m - j\}$ for all $j \geq 4$. From $d(P_0, P_m) = m - 3$ and $d(P_1, P_m) = m - 4$ we see that t_0 and t_1 are optimal. We claim that all t_j, $j \geq 4$, are optimal, too. In fact, since $d(P_j, P_0) = j - 3$, every solution cycle with $t_j \leq 2(j - 3) + 1$ would have to contain some shortest P_0-P_j-path. But every such path contains the edge $(3, 0)(4, 1)$, and since $t_2 = 3$, the previous vertex must have $x = 2$. Hence every solution cycle touches the line $x = 2$ whose distance to P_j is $j - 2$.
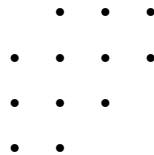
Figure 3 Position graph for the constraint $t_1 = 2 \land t_2 = 3$. The vertex in the lower left corner is $(2, 0)$. The horizontal stripe can be longer than displayed here.
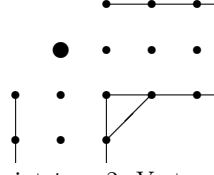
■ **Figure 4** Position graph for the constraint $t_k = 2$. Vertex (k, k) is highlighted.

Constraint $t_k = t_{k+1} = 3$, for some k with $3 \leq k < k+1 \leq m-3$, implies $[a, b] = [k, k+1]$ and $c = 3$. The round trip on the BB path going with slope $+1$ through $(k+2, k-1)$ achieves $t_i = 2\max\{i, m-i-3\}$ for all $i \leq k-1$, $t_k = t_{k+1} = 3$, and $t_j = 2\max\{j-3, m-j\}$ for all $j \geq k+2$, and these waiting times are optimal under the mentioned constraint.

At this point, remember that every further critical vector T satisfies $T \geq (4, 2, \ldots, 2, 4)$.

Constraint $t_1 = 2$ alone implies $a \leq b \leq 2$. Thus $t_0 \geq 4$, $t_1 = 2$, $t_2 \geq 4$, and $t_j \geq 2\max\{j-3, m-j\}$ for all $j \geq 3$. Here, $t_2 \geq 4$ holds since $t_2 \leq 3$ was already treated earlier, and $t_j \geq 2(j-3)$ comes from $d(P_3, P_j) = j - 3$. But these waiting times constitute some smaller vector in $\mathcal{M}_0$. The conclusion is literally the same for $t_1 = 3$. It follows that every further critical vector T even satisfies $T \geq (4, 4, 2, \ldots, 2, 4, 4)$.

Constraint $t_k = 2$, for some k with $2 \leq k \leq m-2$, implies $[a, b] = [k-1, k+1]$ and $c = 2$. See Figure 4. Here is a solution cycle that matches the resulting lower bounds: We traverse the path $Z = (k, k-2)(k+1, k-1)(k+2, k)$, then some BB path to P_m that starts and ends in $(k+2, k)$, then Z backwards, then some BB path to P_0 that starts and ends in $(k, k-2)$, and so forth. Regardless of the parities of k and $m-k$ and of the choice of the BB path in the case of even lengths, the waiting times are $t_i = 2\max\{i, m-i-2\}$ for all $i \leq k-2$, and $t_j = 2\max\{j-2, m-j\}$ for all $j \geq k+2$. The values of t_{k-1} and t_{k+1} achieved by this path depend on k and $m-k$, but they are at most 4.

3.3 The Smallest Instances

Using the general results on critical vectors with the smallest waiting times, we can now demonstrate how to solve the smallest instances for all or "most" vectors of waiting times. Cases with $m \leq 4$ are easy to settle and therefore omitted. Case $m = 5$ is still simple:

▶ **Proposition 9.** *For $m = 5$ there is no critical vector $T \notin \mathcal{M}_0 \cup \mathcal{M}_1$.*

Proof. Let $T = (t_0, \ldots, t_5)$. Note that $T \geq (4, 4, 2, 2, 4, 4)$. If $t_2 \leq 3$ then $d(P_0, P_5) \geq 3$, hence $T \geq (6, 4, 2, 2, 4, 6) \in \mathcal{M}_1$. Similarly we can rule out $t_3 \leq 3$. It follows $T \geq (4, 4, 4, 4, 4, 4) > (4, 2, 4, 4, 2, 4) \in \mathcal{M}_0$. ◀

But already the case $m = 6$ reveals the subtlety of the problem.

▶ **Proposition 10.** *For $m = 6$, all critical vectors $T \notin \mathcal{M}_0 \cup \mathcal{M}_1$ are $(6, 4, 4, 3, 4, 4, 6)$ and $(8, 4, 2, 4, 4, 6, 8)$, and several vectors $T \geq (8, 4, 2, 3, 4, 6, 8)$ that are critical under the constraint $t_2 = 2 \wedge t_3 = 3$.*

Proof. Let $T = (t_0, \ldots, t_6)$. Note that $T \geq (4, 4, 2, 2, 2, 4, 4)$.

If $t_3 = 2$ then $t_2 \geq 3$ and $t_4 \geq 3$. If now $t_2 \geq 4$ and $t_4 \geq 4$ then (see the end of Section 3.2) we have $T \geq (8, 6, 4, 2, 4, 6, 8) > (8, 6, 2, 2, 4, 6, 8) \in \mathcal{M}_1$. Hence $t_2 = 3$ or $t_4 = 3$. It suffices to consider one of these symmetric cases. Constraint $t_3 = 2 \wedge t_4 = 3$ (see Figure 5) implies, by applying Lemma 1, that $T \geq (8, 6, 4, 2, 3, 6, 8) > (8, 6, 4, 2, 2, 6, 8) \in \mathcal{M}_1$. This excludes $t_3 = 2$ and shows $T \geq (4, 4, 2, 3, 2, 4, 4)$.
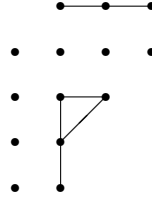
Figure 5 Position graph for the constraint $t_3 = 2 \wedge t_4 = 3$. The leftmost line is $x = 3$.
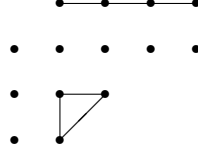
Figure 6 Position graph for $m = 6$ and the constraint $t_2 = 2 \wedge t_3 = 3$. The leftmost line is $x = 2$.

Let $t_3 = 3$. If some of the neighbors equals 3 as well, then we have a constraint from Theorem 7. Hence both neighbors are at least 4, or some equals 2. We also have $2 \leq a \leq b \leq 4$. Now $b = 2$ would imply $t_3 \geq 6$ by Lemma 6. Hence $b \geq 3$, therefore $t_0 \geq 6$, again by Lemma 6. By symmetry this also holds on the other side. Therefore $t_3 = 3$ implies $T \geq (6, 4, 2, 3, 2, 4, 6)$.

Constraint $t_2 = 4 \wedge t_3 = 3 \wedge t_4 = 4$ now yields $T \geq (6, 4, 4, 3, 4, 4, 6)$, which is achieved by the round trip on $(3, 0)(4, 1)(5, 2)(6, 3)$. Hence this is the only critical vector under this constraint.

It remains to study the constraint $t_2 = 2 \wedge t_3 = 3$, and vectors with $t_3 \geq 4$.

Assume that both $t_2 \geq 4$ and $t_4 \geq 4$, in other words, $T \geq (4, 4, 4, 3, 4, 4, 4)$. Assume that also $t_0 \leq 5$. Then $a \leq b \leq 2$. Hence, by Lemma 6, we even have $T \geq (4, 4, 4, 6, 4, 6, 8) > (4, 2, 4, 6, 4, 4, 6) \in \mathcal{M}_0$. This shows $t_0 \geq 6$ and by symmetry also $t_m \geq 6$, hence $T \geq (6, 4, 4, 3, 4, 4, 6)$, which was already feasible. It follows that $t_2 \leq 3$ or $t_4 \leq 3$. By symmetry we can suppose $t_2 \leq 3$.

This yields $1 \leq a \leq 2 \leq b \leq 3$, thus $T \geq (4, 4, 2, 3, 4, 6, 8)$ by Lemma 6. From $d(P_0, P_6) \geq 3$ we also get $t_0 \geq 6$, thus $T \geq (6, 4, 2, 3, 4, 6, 8)$. Now, if $t_2 = 3$ then $T \geq (6, 4, 3, 3, 4, 6, 8) \in \mathcal{M}_1$. This finally shows $t_2 = 2$. Since now the edge $(3, 0)(4, 1)$ is no longer in R', the bounds are further raised to $d(P_0, P_6) \geq 4$ and $T \geq (8, 4, 2, 3, 4, 6, 8)$. If $t_3 = 4$ then we get the critical vector $(8, 4, 2, 4, 4, 6, 8)$, achieved by the cycle $(2, 0)(3, 1)(4, 2)(5, 3)(6, 2)(5, 1)(4, 2)(3, 1)$. So there only remains the constraint $t_2 = 2 \wedge t_3 = 3$. ◄

Figure 6 shows the position graph for the constraint $t_2 = 2 \wedge t_3 = 3$ that we discuss a bit further now.

Recall that critical vectors T under this constraint satisfy $T \geq (8, 4, 2, 3, 4, 6, 8)$. Since $(8, 6, 2, 2, 4, 6, 8) \in \mathcal{M}_1$, every such critical vector has $t_1 \leq 5$. Hence every solution cycle must contain some P_1-P_6 path of length 2, and therefore the path $(3, 2)(4, 1)(5, 2)(6, 3)$.

Constraint $t_1 = 4 \wedge t_2 = 2 \wedge t_3 = 3$ requires this path to appear on both sides of $(6, 3)$, which enforces $(3, 2)(4, 1)(5, 2)(6, 3)(5, 2)(4, 1)(3, 2)$.

Constraint $t_2 = 2 \wedge t_3 = 3 \wedge t_4 = 4$ enforces, by similar arguments, every solution cycle to contain a path $(4, 2)(3, 1)(2, 0)(3, 1)(4, 2)$.

Moreover, every visit of P_6 and P_0, respectively, necessarily happens within such a path.

Constraint $t_2 = 2 \wedge t_3 = 3 \wedge t_4 = 4 \wedge t_5 = 6$ enforces $(5, *)(4, 2)(3, 1)(2, 0)(3, 1)(4, 2)(5, *)$.

Constraint $t_1 = 4 \wedge t_2 = 2 \wedge t_3 = 3 \wedge t_4 = 4 \wedge t_5 = 6$ now obviously enforces both $(3, 2)(4, 1)(5, 2)(6, 3)(5, 2)(4, 1)(3, 2)$ and $(5, *)(4, 2)(3, 1)(2, 0)(3, 1)(4, 2)(5, *)$, and since these paths "disagree", they must occupy disjoint subpaths of any solution cycle. Finally, consider any "consecutive" visits of P_6 and P_0, that is, without other such visits in

between. They are surrounded by the mentioned paths, and furthermore, at least some $(4, *)$ must exist between them. However, a single vertex $(4, *)$ is not enough, because then P_1, P_2, P_3 must all be visited by $(4, *)(5, *)$, which is obviously impossible. Hence we must place at least two vertices there. On the other hand, the round trip on, for instance, $(6, 3)(5, 2)(4, 1)(3, 2)(3, 1)(4, 2)(5, 3)(4, 2)(3, 1)(2, 0)$ has waiting times $(18, 4, 2, 3, 4, 6, 18)$, and this vector is critical.

This example illustrates two aspects: Small waiting times in the middle can cause very large waiting times at the ends, and the distance lower bounds are strong enough to uniquely identify large parts of the solution cycles, which makes their construction quite efficient. Systematic search with the help of a computer program[1] produced, e.g., as many as 10 critical vectors for $m = 6$ under the constraint $t_2 = 2 \wedge t_3 = 3$, and the waiting time 18 in the example above is the highest one appearing in them. Slightly relaxed waiting times at inner points allow smaller waiting times at the ends, and some combinations of times can be chosen independently, which explains the exploding number of critical vectors. Enumeration for slightly larger m gave a similar picture, after considerably larger computation time, and with growing numbers of critical vectors.

4   Discussion and Conclusions

We have formulated the Patrolling problem, even with several robots, as a problem dealing with only one vehicle that has to visit "properties" in one so-called position graph. (This plays a bit with the ambiguity of the word property which can also mean an object located somewhere.) The main advantage is that we can use the distances in this graph for solving instances of the problem: They yield simple lower bounds on the waiting times, moreover, waiting times in critical vectors are often equal to (or close to) these lower bounds. Hence their solution cycles must traverse shortest paths between the respective properties, and they are sometimes even unique. Moreover, certain combinations of (e.g., small) waiting times make the resulting position graphs simple. All this facilitates the construction of solution cycles or the verification of infeasible instances.

We have mainly studied the case of two robots patrolling on a line, whose complexity is open. As argued in Section 1, this problem is not as narrow as one might think, rather, it is the natural case to study first. The above ideas are used to determine all critical vectors for the smallest m, partly manually (as shown here), and partly supported by an implementation using further pruning techniques that are hard to summarize here. (However, a small side remark is that, due to the shape of the position graphs, shortest paths can be computed very quickly by a greedy algorithm.)

We found surprisingly many critical vectors, mainly caused by a certain combination of small waiting times in the interior of the line segment. Although pre-computation and plain enumeration of the critical instances is apparently not the method of choice for solving *given* single instances (as initially hoped), this study gives useful pointers to solution techniques. Still, vectors being far from criticality seem to be harder to solve. The ultimate goal would be to generalize these observations, in order to derive either a polynomial algorithm, or at least a better practical approximation algorithm than in [1, 5], or to identify gadgets for an NP-hardness proof. (We remark that unresolved complexities are typical in this field, e.g., the complexity status of Pinwheel Scheduling [6, 8, 9] is notoriously open.)

----

[1]   provided by Anton Gustafsson and Iman Radjavi

Note that a superpolynomial number of critical vectors does not yet rule out a polynomial algorithm. Furthermore, the methods we have demonstrated here to construct *all* critical vectors may similarly be used heuristically to find *some* critical vector being smaller than any given input vector T, or proving T infeasible. Some hope for fast algorithms comes from a certain "stratification" of waiting time vectors: We observed that instances containing some small waiting times behave differently than instances where all waiting times are large in relation to m, and also the size of the intersection $[a, b]$ of ranges plays some role.

Other open problems for two robots on a line, besides the complexity, have arisen: Is the number of critical vectors actually exponential in m? Are the largest waiting times linearly bounded in m? How long can their solution cycles be, in the worst case?

Moreover, since two robots on a line is only a special case and a first testbed, we would like to apply our insights also to more robots and more general topologies, such as trees and grids. But it seems to be a reasonable procedure to first aim for a thorough understanding of the "smallest" non-trivial case. The heuristics developed here should extend more or less straightforwardly to more general cases of Patrolling. But we do not expect to transform results to seemingly similar problems, e.g., as mentioned earlier, Periodic Latency behaves differently and seems to be simpler from the outset.

References

1   Huda Chuangpishit, Jurek Czyzowicz, Leszek Gasieniec, Konstantinos Georgiou, Tomasz Jurdzinski, and Evangelos Kranakis. Patrolling a path connecting a set of points with unbalanced frequencies of visits. In A Min Tjoa, Ladjel Bellatreche, Stefan Biffl, Jan van Leeuwen, and Jirí Wiedermann, editors, *SOFSEM 2018: Theory and Practice of Computer Science – 44th International Conference on Current Trends in Theory and Practice of Computer Science, Krems, Austria, January 29 – February 2, 2018, Proceedings*, volume 10706 of *Lecture Notes in Computer Science*, pages 367–380. Springer, 2018. `doi:10.1007/978-3-319-73117-9_26`.

2   Sofie Coene, Frits C. R. Spieksma, and Gerhard J. Woeginger. Charlemagne's challenge: The periodic latency problem. *Oper. Res.*, 59(3):674–683, 2011. `doi:10.1287/opre.1110.0919`.

3   Jurek Czyzowicz, Kostantinos Georgiou, and Evangelos Kranakis. Patrolling. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 371–400. Springer, 2019. `doi:10.1007/978-3-030-11072-7_15`.

4   Jurek Czyzowicz, Adrian Kosowski, Evangelos Kranakis, and Najmeh Taleb. Patrolling trees with mobile robots. In Frédéric Cuppens, Lingyu Wang, Nora Cuppens-Boulahia, Nadia Tawbi, and Joaquín García-Alfaro, editors, *Foundations and Practice of Security – 9th International Symposium, FPS 2016, Québec City, QC, Canada, October 24-25, 2016, Revised Selected Papers*, volume 10128 of *Lecture Notes in Computer Science*, pages 331–344. Springer, 2016. `doi:10.1007/978-3-319-51966-1_22`.

5   Peter Damaschke. Two robots patrolling on a line: Integer version and approximability. In Leszek Gasieniec, Ralf Klasing, and Tomasz Radzik, editors, *Combinatorial Algorithms – 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8-10, 2020, Proceedings*, volume 12126 of *Lecture Notes in Computer Science*, pages 211–223. Springer, 2020. `doi:10.1007/978-3-030-48966-3_16`.

6   Peter C. Fishburn and J. C. Lagarias. Pinwheel scheduling: Achievable densities. *Algorithmica*, 34(1):14–38, 2002. `doi:10.1007/s00453-002-0938-9`.

7 Leszek Gasieniec, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In Bernhard Steffen, Christel Baier, Mark van den Brand, Johann Eder, Mike Hinchey, and Tiziana Margaria, editors, *SOFSEM 2017: Theory and Practice of Computer Science – 43rd International Conference on Current Trends in Theory and Practice of Computer Science, Limerick, Ireland, January 16-20, 2017, Proceedings*, volume 10139 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2017. `doi: 10.1007/978-3-319-51963-0_18`.

8 Robert Holte, Louis E. Rosier, Igor Tulchinsky, and Donald A. Varvel. Pinwheel scheduling with two distinct numbers. *Theor. Comput. Sci.*, 100(1):105–135, 1992. `doi:10.1016/ 0304-3975(92)90365-M`.

9 Shun-Shii Lin and Kwei-Jay Lin. A pinwheel scheduler for three distinct numbers with a tight schedulability bound. *Algorithmica*, 19(4):411–426, 1997. `doi:10.1007/PL00009181`.

Transfer Customization with the Trip-Based Public Transit Routing Algorithm

Vassilissa Lehoux-Lebacque[1] ✉
NAVER LABS Europe, Meylan, France

Christelle Loiodice ✉
NAVER LABS Europe, Meylan, France

──── **Abstract** ────

In the context of routing in public transit networks, we consider the issue of the customization of walking transfer times, which is incompatible with the preprocessing required by many state-of-the-art algorithms. We propose to extend one of those, the Trip-Based Public Transit Routing algorithm, to take into account *at query time* user defined transfer speed and maximum transfer duration. The obtained algorithm is optimal for the bicriteria problem of optimizing minimum arrival time and number of transfers. It is tested on two large data sets and the query times are compatible with real-time queries in a production context.

1 Introduction

In mobility applications or websites, finding paths between an origin and a destination is a classical problem. In public transit networks, those paths can combine public transit modes and walking between the stations. In this paper, we are interested in building sets of alternative paths according to user specified transfer speed and/or maximum transfer duration. Customization of transfer times is an important feature for routing applications, as in many contexts, users have an a priori idea of the maximum duration they wish to spend on a transfer, or the speed at which they will perform it. The speed or maximum duration can be related to weather (not walking too much under the rain or walking more slowly in hot weather), to trip aim (travelling with a heavy luggage, taking small kids to an activity) or simply to the physical condition of the user. In some other contexts, a user might wish to set a large maximum duration and a high speed, for instance if long transfers at a brisk pace are perceived as an opportunity to keep fit. In addition, some modes that can be carried by the user in the public transports, like kick scooter or roller blades, have a network very similar to the walking network and can be modelled by faster walking transfers in the routing algorithm. In the following, we will refer to walking transfers for simplicity, but transfers could be done using those modes if speed customization is added.

Many efficient algorithms have been developed over the last years for mono- or bicriteria routing in public transit network (e.g. [1, 4, 5, 15, 3]). However, they mostly consider fixed transfer speed and sometimes rely on long preprocessing depending on fixed transfer times. When transitive closure of transfers is not required by the algorithm, there usually exists a limit at the application level of the duration of a transfer. This reduces the size of the routing problem and is acceptable for many users who would like to avoid long transfers. In [13], which extends the RAPTOR [4] algorithm, the authors consider higher maximum

---

[1] Corresponding author

transfer duration at the application level, and user customizable transfer speed and maximum transfer duration. In [14], the authors consider unrestricted walking transfers (no maximum duration). They find that the earliest arrival time is improved for long distance queries by allowing more walking in 75 percent of the times, compared to a limit of 8 min walking for Germany and a limit of 15 min for Switzerland. This issue of unlimited transfer times has then been studied in several recent publications [11, 2]. However, as their transfer graphs can count transfers of several hours, it is not clear that users will be willing to perform the optimal itineraries when they involve too much walking. Especially if alternatives exist with less walking, and even if those alternatives are significantly slower. We hence consider that the possibility to customize the transfer speed and maximum duration at query time will provide users with itineraries better adapted to their context and preferences.

In this article, we want to extend the Trip-Based Public Transit Routing algorithm [15] to user customized transfer speed and maximum duration at query time, while preserving the optimality of the algorithm. The outline is as follows. Section 2 describes the principle of the Trip-Based Public Transit Routing algorithm and the notations used in the paper. The proposed extension is explained in Section 3 and a proof of optimality is given. Tests on two large size data sets are discussed in Section 4. Section 5 concludes the article.

2   Notations and principle of the Trip-Based Public Transit Routing

In this section, we describe public transit networks using notations similar to that of [15], for easier reference. Public transit information contains the schedules of the transit vehicles. For each vehicle, it defines the passage times of the vehicle at the stations (also called *stops*) where its passengers can board and alight. A *trip t* corresponds to a vehicle following its *sequence of stops* $\overrightarrow{p}(t) = \langle t@0, t@1, \dots \rangle$. We denote by $\tau_{arr}(t, i)$ (resp. $\tau_{dep}(t, i)$) the *arrival time* (resp. *departure time*) of t at the ith stop of $\overrightarrow{p}(t)$. We group trips of identical stop sequences that do not overtake each other into *lines*. The lines hence do not exactly represent the routes of the public transport network. Similarly to trip notations, we denote by $\overrightarrow{p}(L) = \langle L@0, L@1, \dots \rangle$ the stop sequence of line L. Note first that the partition of the trips into lines is not unique. Second, as the trips of a line do not overtake each other, they form a totally ordered set with the relation $\preceq$ and a partial order with $\prec$ defined for two trips t and u having the same sequence by:

$$t \preceq u \Longleftrightarrow \forall i \in \{0, 1, \dots, |\overrightarrow{p}(t)| - 1\}, \quad \tau_{arr}(t, i) \leq \tau_{arr}(u, i)$$
$$t \prec u \Longleftrightarrow t \preceq u \text{ and } \exists i \in \{0, 1, \dots, |\overrightarrow{p}(t)| - 1\}, \quad \tau_{arr}(t, i) < \tau_{arr}(u, i)$$

$t@i \to t@j$ denotes a displacement between the ith and the jth stops of trip t using trip t and similarly, a transfer between trip t at the ith station and trip u at the jth station is denoted $t@i \to u@j$. For a given stop s, $\mathbf{L}(s)$ is the set of all line-index pairs (L, i) such that $s = L@i$. Information about the transfers between the stops of network is usually represented directly by *walking transfer times* $\Delta\tau_{\text{fp}}(p, q)$ defined for any pair of stops (p, q), $p \neq q$ that are close enough from one another. When transferring between two trips at a given station $(t@i = u@j = p)$, a *minimum change time* $\Delta\tau_{\text{fp}}(p, p)$ can also be defined to represent the time needed to move within the station.

The **Trip-Based Public Transit Routing (TB) algorithm** [15] is an exact state-of-the-art algorithm for routing in public transit networks. A *bicriteria earliest arrival time query* (EAT) takes as inputs an origin, a destination and a start time. The two criteria minimized are *arrival time* and *number of transfers*.

If a set of criteria $(c_1, c_2, \ldots, c_n)$ is to be minimized, a solution s with value $(v_1, v_2, \ldots, v_n)$ is *non-dominated in the Pareto sense* if there is no other solution s' with values $(v'_1, v'_2, \ldots, v'_n)$ such that for all $i \in \{1, 2, \ldots, n\}$, $v'_i \leq v_i$ and $\exists i \in \{1, 2, \ldots, n\}$ such that $v'_i < v_i$. Non-dominated solutions are called *optimal* and the maximum cardinality set of non-dominated solutions is denoted *Pareto set*. The *Pareto front* is the image of the Pareto set in the criterion space. As most routing algorithms, the TB algorithm doesn't compute the complete Pareto set but only one solution with this value per element in the Pareto front. As in [12], we call this family of sets *complete sets*. The TB algorithm builds a complete set of solutions for minimum arrival time and number of transfers in polynomial time. It uses a specific graph representation based on trips as vertices and feasible transfers as arcs. For each trip, a neighbourhood of reachable trips is built by a preprocessing step and is pruned while ensuring that a complete set of solutions can still be obtained. We call a preprocessing that ensure the optimality of the algorithm a *correct* preprocessing. Such a preprocessing for the TB algorithm ensures that for any optimal value, there exists an optimal solution with this value whose transfers are all in the pruned transfer set. In the search graph, an EAT query consists in a breadth-first search like exploration. Trip segments reached from the origin given the departure time form the initial current queue Q, while for every stop p from which destination can be reached and any trip t such that $p = t@j$, trips segments $t@i \rightarrow t@k$ with $i < j \leq k$ are the targets of the algorithm. Those targets can be represented by the set $\mathcal{L}$ of triplets $(L, i, \Delta\tau)$ where s is a stop from which destination can be reached, $\Delta\tau$ is the duration of walking from s to destination and $(L, i) \in \mathbf{L}(s)$. During an iteration, all the trip segments of the current queue are processed. If a trip segment is a target, best arrival time can be improved. Transfers are performed to add the reached trip segments to the queue for the next iteration.

The TB algorithm can also be used with slight modifications to compute *profile queries*, where all the optimal paths must be found for a given starting time range.

Pruning phase. Given an origin trip t and a destination trip u, transfer $t@i \rightarrow u@j$ is *feasible* if and only if

$$\tau_{arr}(t, i) + \Delta\tau_{\mathrm{fp}}(t@i, u@j) \leq \tau_{dep}(u, j)$$

When considering the set of feasible transfers between a trip t at its ith stop and a line L at its jth stop, the order on the trips of L implies that this set is either empty or has a minimum element according to $\preceq$ and $\prec$. This element is the earliest trip such that the transfer is feasible. To construct a complete solution set for minimum arrival time and number of transfers, it is sufficient to add only this earliest transfer to the search graph.

The preprocessing phase of the TB algorithm as described in [15] first computes the set of all earliest feasible transfers and then prune the neighbourhood of each trip based on stop labels those values are the earliest arrival times at stops when transferring from the trip.

In [7], the authors modify the preprocessing to make it faster than in the original version. The key idea is to perform an additional pruning step based on trip-to-line transfers before the arrival time based pruning. The transfers between a trip t and a line L are compared using the following dominance relation. If $u, u' \in L$, a transfer $t@i \rightarrow u@j$ *is dominated* by a transfer $t@i' \rightarrow u'@j'$ if and only if

$$i \leq i' \quad \text{and} \quad u' \leq u \quad \text{and} \quad j' \leq j \quad \text{and}$$
$$(i < i' \quad \text{or} \quad u' < u \quad \text{or} \quad j' < j)$$

We will extend both preprocessing steps to take into account transfer time customization.

3   Customization of transfers

Now we want to enable customization of transfer speed and maximum transfer duration at query time. First note that the values chosen by the user need to be bounded between realistic values defined at the application level. To modify the transfer speed, we consider that the public transit information contains transfer times for some constant chosen speed s_{std}. We can define for each query a *duration coefficient* σ corresponding to the user chosen speed s: $\sigma = s_{std}/s$. If the standard duration of a transfer is $\Delta\tau$, the application of a duration coefficient σ will result in a duration $\sigma\Delta\tau$. To avoid unrealistic fast transfer time values, a minimum application level duration coefficient can be chosen with $1 \geq \varsigma_{\min} > 0$. Similarly, a maximum transfer duration coefficient at the application level $\varsigma_{\max} \geq 1$ can be set.

3.1   Modifications of the query phase

Suppose that we obtain after preprocessing a transfer set correct for any user defined transfer duration coefficient $\sigma \in [\varsigma_{\min}, \varsigma_{\max}]$ and maximum transfer duration $\Delta\tau_{\max} \geq 0$. To avoid performing any transfer longer than $\Delta\tau_{\max}$, we can prune at query time the transfers whose duration exceeds the bound. For faster computations, and unlike in the standard algorithm, the duration must be an attribute of the transfer. Saving the maximum transfer duration coefficient for which the transfer is feasible will similarly enable faster pruning during the search phase. It would also be possible to add a minimum duration coefficient for which the transfer can be useful if above that speed the previous destination trip of the same line can be taken instead. In the case where possible speed values are from a discrete set, a speed mask can be added to transfers in order to keep only the right ones for each speed during the query phase.

Now each transfer of the transfer set is a triplet $(t@i \rightarrow u@j, \Delta\tau, \sigma_{\max})$ where $\Delta\tau$ is the standard transfer duration and $\sigma_{\max}$ the maximum duration coefficient for which the transfer is feasible. The user gives as additional inputs a maximum transfer duration $\Delta\tau_{\max}$ and transfer duration coefficient σ. They are used to prune the transfers during the search phase when exploring the neighbourhood of the trips. If a transfer $(t@i \rightarrow u@j, \Delta\tau, \sigma_{\max})$ is such that $\Delta\tau_{\max} > \sigma\,\Delta\tau$ or $\sigma > \sigma_{\max}$, it can be pruned. For concision, the pseudo-code of the modified query algorithm can be found in appendix in Algorithm 2.

Note that it would also be possible to bound the travel duration from origin to the first stop or from the last stop to destination by pruning the initial queue Q_0 and the target set $\mathcal{L}$ according to a user defined value (possibly different of $\Delta\tau_{\max}$).

It has been proven in [6] that a correct transfer set for EAT queries is correct for latest departure time queries (LDT). LDT queries could hence be modified similarly to integrate maximum duration and variable transfer speed.

3.2   Preprocessing phase

When considering multiple possible speeds, there might be several transfers of interest for a given origin trip t at stop $t@i$ toward a given destination line L' at $L'@j$, instead of a single one. The smallest destination trip to consider is the earliest trip such that the transfer is feasible with the fastest possible speed:

$$u_{\min} = \min\{u \in L' \,|\, \tau_{dep}(u, j) \geq \tau_{arr}(t, i) + \varsigma_{\min}\,\Delta\tau_{\mathrm{fp}}(t@i, L'@j)\}$$

The latest corresponds the slowest speed:

$$u_{\max} = \min\{u \in L' \,|\, \tau_{dep}(u, j) \geq \tau_{arr}(t, i) + \varsigma_{\max}\,\Delta\tau_{\mathrm{fp}}(t@i, L'@j)\}$$

And all the trips of L' in between could be taken, depending on the user chosen transfer speed, each corresponding to a transfer to an earliest trip for a given speed range. We call *trips of interest* of the transfer $t@i \rightarrow L'@j$ the destination trips of L' in $\{u_{\min}, \ldots, u_{\max}\}$. When the set of possible speed values is finite, not all the trips of the range $\{u_{\min}, \ldots, u_{\max}\}$ might be relevant, and we will consider only the earliest for each speed of the set.

For each feasible transfer described above, we save in the transfer set the t-uple $(t@i \rightarrow u@j, \Delta\tau_{\mathrm{fp}}(t@i, u@j), \sigma_{\max})$ where $\sigma_{\max}$ is the maximum duration coefficient such that the transfer is feasible.

Note that the obtained transfer set is a correct transfer set. However, as explained, the query times would be impacted by the unnecessary transfers. We will hence consider both the line-based and the arrival time-based prunings and explain how to modify them to take into account a customizable transfer speed and maximum transfer duration.

3.2.1 Pruning based on lines

The line-based pruning is based on a dominance relation between transfers. Since we want to customize the maximum transfer duration, a transfer $(t@i \rightarrow u@j, \Delta\tau, \sigma_{\max})$ cannot be dominated by a transfer $(t@i \rightarrow v@j, \Delta\tau', \sigma'_{\max})$ such that $\Delta\tau' > \Delta\tau$. Indeed, the second transfer could be forbidden by the custom maximum transfer duration, while the first is not. Similarly, as $\sigma_{\max}$ is the maximum duration coefficient such that the first transfer is feasible, if $\sigma'_{\max} < \sigma_{\max}$, the second transfer cannot dominate the first. We hence obtain the following dominance relation. A transfer $(t@i \rightarrow u@j, \Delta\tau, \sigma_{\max})$ *is dominated* by a transfer $(t@i' \rightarrow u'@j', \Delta\tau', \sigma'_{\max})$ if and only if

$$i \leq i' \quad \text{and} \quad u' \leq u \quad \text{and} \quad j' \leq j \quad \text{and} \quad \Delta\tau' \leq \Delta\tau \quad \text{and} \quad \sigma_{\max} \leq \sigma'_{\max} \quad \text{and}$$
$$(i < i' \quad \text{or} \quad u' < u \quad \text{or} \quad j' < j \quad \text{or} \quad \Delta\tau' < \Delta\tau \quad \text{or} \quad \sigma_{\max} < \sigma'_{\max})$$

Using this condition, it is possible to prune the transfer set as before. However, it is expected that the percentage of pruned transfers will be lower, as the dominance condition is stronger and that preprocessing will be longer, as additional comparisons need to be performed. Corresponding pseudo-code can be found in appendix in Algorithm 4 describing the modified preprocessing that builds the search graph arc set.

3.2.2 Pruning based on arrival times

Remember that in the original TB algorithm, a transfer is removed from the set of possible transfers if previously scanned transfers allow for reaching the same stops at the same or an earlier time. As the transfers are scanned starting from the end of the origin line, later transfers are kept in case of identical arrival times. Now, we want to consider the possibility to disable some transfers at query time according to maximal duration or if speed customization makes the transfer time too long to reach the destination trip before it leaves. Applying the same pruning will not be correct, as a transfer can be removed because of previously checked transfers with longer duration. As a consequence, we consider for each tentative arrival time at a stop the transfer time and the maximum duration coefficient for which the transfer is feasible. Also, comparing arrival times is made more difficult by the speed variability. All arrival times, in this preprocessing, have a speed independent component corresponding to the arrival time of a trip at one stop of its sequence. Then, when reaching additional stations by footpaths, the duration is dependent of speed. Obviously, simply comparing the sum of the two is not correct, as the variable part will be multiplied by a duration coefficient.

We hence label the stops with a bag of t-uples instead of a single value. Each t-uple indicates arrival time, fixed and variable parts, standard transfer duration and maximum duration coefficient for the transfer to be feasible. We denote $(arr_f, arr_v, \Delta\tau, \sigma_{\max})$ such a label, with arr_f the fixed arrival time part, arr_v the variable arrival time part with standard speed, $\Delta\tau$ the standard duration, and $\sigma_{\max}$ the maximum duration coefficient. A transfer is removed from the set if it doesn't improve any of the label bags of the reached stops (i.e. its labels are dominated at each stop). If we compare the labels $(arr_f, arr_v, \Delta\tau, \sigma_{\max})$ and $(arr'_f, arr'_v, \Delta\tau', \sigma'_{\max})$, $(arr_f, arr_v, \Delta\tau, \sigma_{\max})$ is dominated if and only if:

(a) $\sigma_{\max} \leq \sigma'_{\max}$

(b) $\Delta\tau' \leq \Delta\tau$

(c) $\forall \sigma \in [\varsigma_{\min}, \varsigma_{\max}], \quad arr'_f + arr'_v \times \sigma \leq arr_f + arr_v \times \sigma$

Conditions (a) and (b) correspond to classical Pareto dominance between criterion values. Condition (c) corresponds to arrival time dominance, but must be true for all possible speeds. It is equivalent to:

$$\forall \sigma \in [\varsigma_{\min}, \varsigma_{\max}], \quad \frac{arr'_f - arr_f}{\sigma} \leq arr_v - arr'_v \tag{1}$$

In particular, inequation (1) is true if it is true for the minimum value $\varsigma_{\min}$ that duration coefficient σ can take, obtaining the following conditions for label dominance:

(a) $\sigma_{\max} \leq \sigma'_{\max}$

(b) $\Delta\tau' \leq \Delta\tau$

(c) $\frac{arr'_f - arr_f}{\varsigma_{\min}} \leq arr_v - arr'_v$

We maintain for each stop a bag of all the non-dominated labels to compare with new entries. We keep a transfer when it updates at least one label bag.

Note that in the case where possible speeds are only within a small discrete set (for instance slow, standard, fast), it is possible to save one label bag per stop and speed and use simpler labels with arrival time (computed for the given speed) and standard transfer duration (or transfer duration computed for the given speed). Each transfer is feasible for a subset of the speeds and can hence update label bags for each of those speeds.
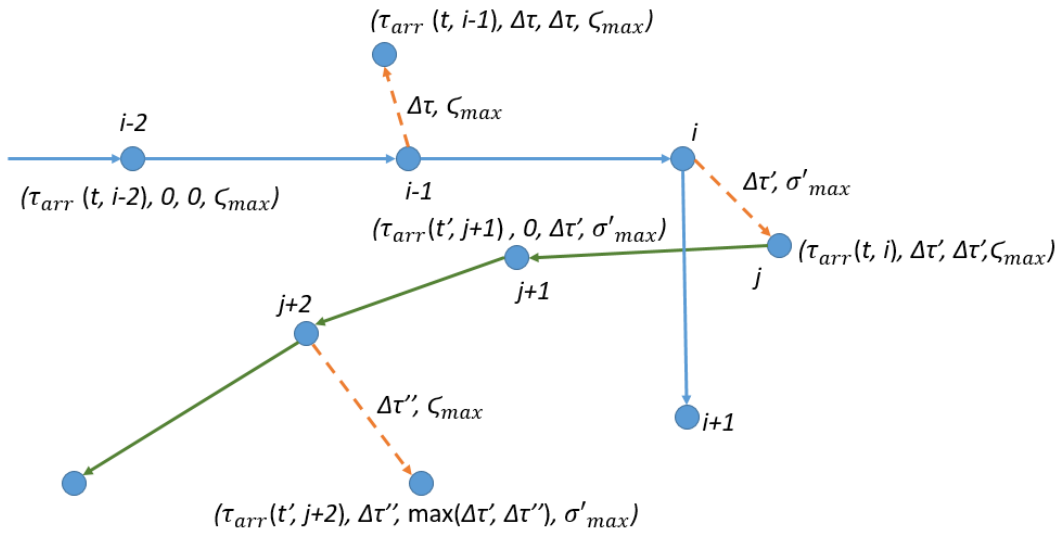
Figure 1 Arrival time labels for a given transfer $t@i \rightarrow t'@j$.

Figure 1 shows some tentative labels for a single transfer between a trip t (above) and a trip t' (below). First the stops of $\overrightarrow{p}(t)$ are marked with a fixed part equal to the trip arrival time, and as no transfer is performed, a null variable part and maximum transfer time and maximum duration coefficient $\varsigma_{\max}$. After transferring from t to t', the stops of $\overrightarrow{p}(t')$ are marked by the arrival time of t', a null variable arrival time part, standard transfer duration and maximum duration coefficient from the transfer between t and t'. When performing transfers from the stops of the trips' stop sequences to reach additional stops, we do not know which trips will be taken later. As a consequence, we use $\varsigma_{\max}$ as a bound. A path with several transfers is feasible for a given duration coefficient only if all its transfers' maximum coefficients are higher. Similarly, if a maximum transfer duration value is provided, the path is feasible only if all the transfer times are bellow the given bound. We hence take the maximum of the successive transfer durations and the minimum of the maximum transfer duration coefficient to mark additional stops reached from t'. As in [15], we check also minimum change times (giving them similar labels and multiplying them by a duration coefficient when speed varies). Algorithm 1 describes this pruning phase.

3.3 Correctness

To prove that the preprocessing steps build correct transfer sets, we need to prove that for any value in the Pareto front, there is an optimal solution with this value such that all its transfers are in the computed transfer set. For each preprocessing step, we prove it by constructing such a solution from any optimal solution.

▶ **Proposition 1.** *The modified line-based preprocessing (Algorithm 4) computes a correct set $\mathcal{T}$ of transfers for earliest arrival time and minimum number of transfers.*

Proof. Consider an optimal solution s for a given duration coefficient σ and a maximum transfer duration $\Delta\tau_{\max}$ with at least one transfer. It can be described by its trip segment sequence: $s = \langle t_1@j_1 \rightarrow t_1@i_1, t_2@j_2 \rightarrow t_2@i_2 \ldots, t_{k+1}@j_{k+1} \rightarrow t_{k+1}@i_{k+1} \rangle$
with L_i the line of the trip t_i, for $i \in \{1, \ldots, k+1\}$. Consider the first transfer $t_1@i_1 \rightarrow t_2@j_2$ of s. If it belongs to the transfer set $\mathcal{T}(t_1, L_2)$ of t_1 to L_2 obtained at the end of the pruning, we can move to the next transfer.

Otherwise, if t_2 is not in the set T_2 of trips of interest of transfer $t_1@i_1 \rightarrow L_2@i_2$, we can replace it with a transfer to the maximum trip u of T_2 such that $u \leq t_2$ as it can only improve arrival time at $L_2@i_2$ while keeping the transfer feasible for the same speed range, including duration coefficient σ.

Now, we suppose that $t_2 \in T_2$ but that $(t_1@i_1 \rightarrow t_2@j_2, \Delta\tau_1, \sigma^1_{\max}) \notin \mathcal{T}(t_1, L_2)$, which means that it has been pruned. Since pruned transfers are dominated, there exists a transfer $(t_1@i \rightarrow t@j, \Delta\tau, \sigma_{\max})$ of $\mathcal{T}(t_1, L_2)$ such that $i \geq i_1$, $j \leq j_2$, $t \leq t_2$, $\Delta\tau \leq \Delta\tau_1$ and $\sigma^1_{\max} \leq \sigma_{\max}$. If $k > 1$, transfer $t@i_2 \rightarrow t_3@j_3$ is feasible, since transfer $t_2@i_2 \rightarrow t_3@j_3$ is feasible and $t \leq t_2$. In solution s, we can hence replace $t_1@j_1 \rightarrow t_1@i_1$ by $t_1@j_1 \rightarrow t_1@i$, and $t_2@j_2 \rightarrow t_2@i_2$ by $t@j \rightarrow t@i_2$ to obtain a new solution s'.

As the new solution uses a transfer $(t_1@i \rightarrow t@j, \Delta\tau, \sigma_{\max})$ such that $\Delta\tau \leq \Delta\tau_1 \leq \Delta\tau_{\max}$ and $\sigma \leq \sigma^1_{\max} \leq \sigma_{\max}$, it is feasible for custom speed and custom maximum transfer time. It also has an at least as good arrival time as s, and the same number of transfers. They are hence both optimal with the same value.

Processing the transfers of s one after the other, we iteratively replace all the transfers that do not belong to the pruned transfer set $\mathcal{T}$ by transfers belonging to it. The optimal solution obtained is equivalent to s while using only transfers of $\mathcal{T}$, which completes the proof. ◀

■ **Algorithm 1** Modifications of arrival time based pruning.

Input: Timetable data, footpath data, transfer set $\mathcal{T}$
Input: System maximum duration coefficient $\varsigma_{\max}$
Output: Reduced transfer set $\mathcal{T}$
for each trip t **do**
    $\tau_A(.) \leftarrow \emptyset$ ▷ Label bag with earliest arrival time at stops
    $\tau_C(.) \leftarrow \emptyset$ ▷ Label bag with earliest change time at stops
    for $i \leftarrow |\overrightarrow{p}(t)| - 1, \ldots, 1$ **do**
        Update $\tau_A(t@i)$ with label $(\tau_{arr}(t,i), 0, 0, \varsigma_{\max})$
        Update $\tau_C(t@i)$ with label $(\tau_{arr}(t,i), \Delta\tau_{\mathrm{fp}}(t@i, t@i), \tau_{\mathrm{fp}}(t@i, t@i), \varsigma_{\max})$
        for each stop $q \neq t@i$ such that $\Delta\tau_{\mathrm{fp}}(t@i, q)$ is defined **do**
            Update $\tau_A(q)$ with label $(\tau_{arr}(t,i), \Delta\tau_{\mathrm{fp}}(t@i, q), \Delta\tau_{\mathrm{fp}}(t@i, q), \varsigma_{\max})$
            Update $\tau_C(q)$ with label $(\tau_{arr}(t,i), \Delta\tau_{\mathrm{fp}}(t@i, q), \Delta\tau_{\mathrm{fp}}(t@i, q), \varsigma_{\max})$
        end for
        for each transfer $(t@i \rightarrow u@j, \Delta\tau, \sigma_{\max}) \in \mathcal{T}$ **do**
            $keep \leftarrow$ **false**
            for each stop $u@k$ on trip u with $k > j$ **do**
                if $(\tau_{arr}(u,k), 0, \Delta\tau, \sigma_{\max})$ is not dominated in $\tau_A(u@k)$ **then**
                    Update $\tau_A(u@k)$ with $(\tau_{arr}(u,k), 0, \Delta\tau, \sigma_{\max})$
                    $keep \leftarrow$ **true**
                end if
                $lab_C \leftarrow (\tau_{arr}(u,k), \Delta\tau_{\mathrm{fp}}(u@k, u@k), \max(\Delta\tau, \Delta\tau_{\mathrm{fp}}(u@k, u@k)), \sigma_{\max})$
                if lab_C is not dominated in $\tau_C(u@k)$ **then**
                    Update $\tau_C(u@k)$ with lab_C
                    $keep \leftarrow$ **true**
                end if
                for each stop $q \neq u@k$ such that $\Delta\tau_{\mathrm{fp}}(u@k, q)$ is defined **do**
                    $lab \leftarrow (\tau_{arr}(u,k), \Delta\tau_{\mathrm{fp}}(u@k, q), \max(\Delta\tau, \Delta\tau_{\mathrm{fp}}(u@k, q)), \sigma_{\max})$
                    if lab is not dominated in $\tau_A(q)$ **then**
                      Update $\tau_A(q)$ with lab
                      $keep \leftarrow$ **true**
                    end if
                    if lab is not dominated in $\tau_C(q)$ **then**
                      Update $\tau_C(q)$ with lab
                      $keep \leftarrow$ **true**
                    end if
                end for
            end for
            if $\neg keep$ **then**
                $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(t@i \rightarrow u@j, \Delta\tau, \sigma_{\max}\}$ ▷ No improvement: remove the transfer
            end if
        end for
    end for
end for

▶ **Proposition 2.** *The modified arrival time based preprocessing (Algorithm 1) computes a correct set $\mathcal{T}$ of transfers for earliest arrival time and minimum number of transfers.*

Proof. Consider again an optimal solution s' with at least one transfer for an origin stop org, a target stop tgt, a duration coefficient σ and a maximum transfer duration $\Delta\tau_{\max}$. We consider both the cases where line-based pruning is applied to the set of transfers of interest and where the set of transfers of interest is pruned directly without line-based pruning. From the proof of Proposition 1, we can construct in both cases another optimal solution s from s' (possibly equal to s') such that all its transfers are in the input transfer set given to the arrival time based pruning as input.

We again describe s by its trip segment sequence, but we add the origin and target stops at the beginning and the end of the sequence:
$s = \langle org, t_1@j_1 \to t_1@i_1, t_2@j_2 \to t_2@i_2 \ldots, t_{k+1}@j_{k+1} \to t_{k+1}@i_{k+1}, tgt\rangle$
with L_i the line of the trip t_i, for $i \in \{1, \ldots, k+1\}$.

Suppose that the first transfer $(t_1@i_1 \to t_2@j_2, \Delta\tau, \sigma_{\max})$ of s is not in $\mathcal{T}$. If it is the last transfer ($k = 1$), it means that there exists a transfer $(t_1@i_1' \to t_2'@j_2', \Delta\tau', \sigma_{\max}')$ of $\mathcal{T}$ such that $i_1' \geq i_1$ (as later transfers are scanned first) and target is reachable from $t_2'@i_2'$ for an index $i_2' > j_2'$ and $l' = (\tau_{arr}(t_2', j_2'), \Delta\tau_{\mathrm{fp}}(t_2'@j_2', tgt), \max\{\Delta\tau', \Delta\tau_{\mathrm{fp}}(t_2'@j_2', tgt)\}, \sigma_{\max}')$ is dominating $l = (\tau_{arr}(t_2, j_2), \Delta\tau_{\mathrm{fp}}(t_2@j_2, tgt), \max\{\Delta\tau, \Delta\tau_{\mathrm{fp}}(t_2@j_2, tgt)\}, \sigma_{\max})$ for the arrival time label bag $\tau_A(tgt)$. As $\Delta\tau' \leq \max\{\Delta\tau, \Delta\tau_{\mathrm{fp}}(t_2@j_2, tgt)\} \leq \Delta\tau_{\max}$ and $\sigma \leq \sigma_{\max} \leq \sigma_{\max}'$, this transfer is feasible for custom parameters $\Delta\tau_{\max}$ and σ. Note that target could not be reached directly from one of the stops of t and arrival time be at least as good as that of s since s is optimal and has hence the minimum number of trips for its arrival time. The solution $\hat{s} = \langle org, t_1@j_1 \to t_1@i_1', t_2'@j_2' \to t_2'@i_2', tgt\rangle$ has hence the same arrival time as s but its transfers belong to $\mathcal{T}$.

Now, consider the case where transfer $t_1@i_1 \to t_2@j_2$ is not the last transfer of s. As transfer $(t_1@i_1 \to t_2@j_2, \Delta\tau, \sigma_{\max})$ has been pruned, there exist a transfer $t_1@i_1' \to t_2'@j_2'$ of $\mathcal{T}$ such that $i_1' \geq i_1$, $t_3@j_3$ can be reached from the trip segment $t_2'@j_2' \to t_2'@i_2'$ and the label $l' = (\tau_{arr}(t_2', j_2'), \Delta\tau_{\mathrm{fp}}(t_2'@j_2', t_3@j_3), \max\{\Delta\tau', \Delta\tau_{\mathrm{fp}}(t_2'@j_2', t_3@j_3)\}, \sigma_{\max}')$ is dominating $l = (\tau_{arr}(t_2, j_2), \Delta\tau_{\mathrm{fp}}(t_2@j_2, t_3@j_3), \max\{\Delta\tau, \Delta\tau_{\mathrm{fp}}(t_2@j_2, t_3@j_3)\}, \sigma_{\max})$ for the change time label bag $\tau_C(t_3@j_3)$. As previously, this transfer exists since arriving at $t_3@j_3$ directly from t_1 at a time at least as good as that of s without performing a transfer would mean that s is not optimal. The transfer is also feasible for custom parameters $\Delta\tau_{\max}$ and σ. From dominance condition (c), the change time at $t_3@j_3$ is identical or improved for all duration coefficients, including σ. It will hence be possible to board trip t_3 at index j_3 after performing the transfer. We can hence replace $t_1@i_1 \to t_2@j_2$ by $t_1@i_1' \to t_2'@j_2'$ in solution s.

Repeating this procedure for the transfers of s in order leads to build a solution $\hat{s}$ with the same number of transfers as s, the same arrival time and all its transfers in $\mathcal{T}$.   ◀

4   Experiments

To evaluate the computation time performances, we implemented the proposed algorithms in rust and ran our experiments on a 64 2.7 GHz CPU Intel(R) Xeon(R) CPU E5-4650 server with 20 M of L3 cache and 504 GB of RAM. We used two large size data sets. The first covers the Région Île-De-France and is provided by IDFM [8] (Île-De-France Mobilités). The footpaths are computed with an OSRM [10] monomodal routing server using OSM [9] road data with a standard speed of 4 kph. To compare the impact of different maximum transfer times, two footpath sets are generated: one with a maximum of 10 min between two adjacent stops and one with a maximum of 30 min. The second data set is provided by Naver Map

Table 1 Data sets.

Data set	Nb stops	Nb trips	Nb lines	Nb connections	Nb footpaths (10 min)	Nb footpaths (30 min)
IDFM	42.3 K	319.2 K	1.9 K	103.8 M	846.2 K	7.186 M
Korea	180.9 K	446.7 K	31.7 K	241.9 M	4.196 M	−

Table 2 Preprocessing for IDFM with maximum 10 min and 30 min transfer time.

Version	IDFM (10 min)			IDFM (30 min)		
	# kept transfers	# removed transfers	Mean duration (s)	# kept transfers	# removed transfers	Mean duration (s)
Standard	98.1 M	1 314.8 M	44	135.9 M	8 382.9 M	692
Variable speed	153.0 M	1 443.9 M	94	320.0 M	11 786.6 M	1 373
Max. duration and var. speed	242.9 M	1 353.9 M	1 892	732.8 M	11 374 M	96 616

and contains public transit information for Korea and footpaths those maximum value is 10 min. We illustrate on this one the impact of the arrival time based preprocessing compare to line-based pruning only. Table 1 gives the respective sizes of the two networks.

To test the proposed algorithms in a standard context, we allow for 3 different speeds (slow: 2 kph, standard: 4 kph and fast: 6 kph). We hence have $\varsigma_{max} = 2$ and $\varsigma_{min} = 2/3$. We compare 3 versions of the code: the standard version without customization, a version with speed customization and a version with speed and maximum transfer duration customization.

4.1 Preprocessing

As explained, with speed customization, there might be several transfers of interest from each origin trip-index pair to each reachable line-index pair. The total number of feasible transfers before pruning is hence increased (see Table 2 and Table 3) and the preprocessing is more computationally expensive. Enabling the maximum transfer duration constraint also increases the number of kept transfers as conditions for removal are harder to fulfil. The final number of transfers for each speed is indicated in appendix (see Table 6 and Table 7).

The preprocessing times for maximum duration and variable speed are considerably increased compare to the standard version, while variable speed only multiply them by 2.33. Indeed, label bag updating is much more expensive than taking the minimum between two arrival times. As we use a straightforward implementation for those label bag updates and as the number of labels can be large for one stop, the computation times are significantly impacted for arrival time based pruning. However, they remain in an acceptable range for public transit data update made every two or three days, which is often the case. On the

Table 3 Preprocessing for Korea with maximum 10 min transfer time.

Version	Line based pruning			All prunings		
	nb kept transfers	nb removed transfers	Mean duration (s)	nb kept transfers	nb removed transfers	Mean duration (s)
Standard	608.6 M	2192.2 M	89	238.1 M	3 251.9 M	170
Variable speed	1 085.5 M	2773.9 M	116	463.8 M	4 106.3 M	490
Max. duration and var. speed	1 520.1 M	2.339.2 M	140	658.1 M	3 912.1 M	14 773

other hand, line-based pruning is less impacted in terms of computation times since less transfers are compared at once (only those to the same line) and the (c) condition of arrival time based pruning is not necessary. It can hence be considered as an alternative when more frequent updates are needed, at the price of slower query times.

4.2 Query phase

For each data set, we generated uniformly at random 100 origin-destination pairs from stop to stop. We run earliest arrival time queries and one-hour profile queries starting at 8.30 am (rush hour is the densest in term of number of trips and transfers).

Table 4 presents the mean execution times and number of solutions for EAT queries with the different versions of the algorithm given a selected speed. As expected, using appropriate transfer structure with speed mask, the execution times are not much impacted by the existence of several speeds instead of one. They are increased compare to that of the standard code without any modifications as the number of transfers is larger, but not much. Remember that to divide by 3 the execution time, the number of transfers removed is 9 out of 10 in the standard version [15]. Here when the number of transfers is multiplied by 2.35 for IDFM 30 min, the mean query duration is multiplied by 1.49 compared to standard version while it includes additional transfer checking. The query times of the different speed values are similar.

When adding the possibility to set maximum transfer duration (see Table 5), the number of transfers is multiplied by 5.39 for IDFM 30 min and the computation times are multiplied by 2.04 for standard speed compare to standard version. Different values of maximum transfer time hardly impact the query times with only a few milliseconds difference between 20 min, 10 min, 5 min and no restriction.

The results are similar for the other two networks and we can conclude that although the modification does increase the query times, those remain sufficiently low for interactive queries in a production application, with at most half a second of execution time for the Korean network.

Numerical results for profile queries can be found in appendix in Tables 8 and 9, and are similar to that of EAT queries.

5 Conclusion

In this article, we extend the Trip-Based Public Transit Routing algorithm, to take into account *at query time* user defined transfer speed and maximum transfer duration, while keeping the optimality for the bicriteria problem of optimizing minimum arrival time and number of transfers. The tests on two large scale data sets show that the preprocessing steps are significantly slower, but the query times are much less increased and still compatible with real-time queries in a production context. Many other algorithms of the literature rely on preprocessing steps using fixed sets of transfers of immutable duration. It would hence be interesting to design similar extensions for those algorithms, in particular for the ones relying on unbounded transfer duration, where transfers in an optimal solution can be very long without customization.

■ **Table 4** EAT queries at 8.30 am.

Data set	Version	Speed	Mean query time (ms)	Mean nb solutions
IDFM 30 min	Standard	-	75	1.86
IDFM 30 min	Variable speed	Standard	112	1.86
IDFM 30 min	Variable speed	Slow	108	1.76
IDFM 30 min	Variable speed	Fast	108	2.03
IDFM 30 min	Max. duration - var. speed	Standard	153	1.86
IDFM 30 min	Max. duration - var. speed	Slow	134	1.76
IDFM 30 min	Max. duration - var. speed	Fast	130	2.03
IDFM 10 min	Standard	-	91	1.71
IDFM 10 min	Variable speed	Standard	98	1.71
IDFM 10 min	Variable speed	Slow	94	1.72
IDFM 10 min	Variable speed	Fast	100	1.76
IDFM 10 min	Max. duration - var. speed	Standard	117	1.71
IDFM 10 min	Max. duration - var. speed	Slow	108	1.72
IDFM 10 min	Max. duration - var. speed	Fast	107	1.76
Korea	Standard	-	316	2.00
Korea	Variable speed	Standard	418	2.00
Korea	Variable speed	Slow	356	1.93
Korea	Variable speed	Fast	374	2.12
Korea	Max. duration - var. speed	Standard	583	2.00
Korea	Max. duration - var. speed	Slow	531	1.93
Korea	Max. duration - var. speed	Fast	543	2.12

■ **Table 5** EAT queries at with user defined maximum transfer time and speed customization, standard speed.

Data set	Max transfer time (min)	Mean query time (ms)	Mean nb solutions
IDFM 30 min	-	153	1.86
IDFM 30 min	20	150	1.83
IDFM 30 min	10	158	1.87
IDFM 30 min	5	155	2.02
IDFM 10 min	-	145	1.71
IDFM 10 min	5	150	1.96
Korea	-	565	2.00
Korea	5	538	1.97

────── **References** ──────

1 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *Proceedings of the 18th Annual European Conference on Algorithms: Part I*, ESA'10, pages 290–301, Berlin, Heidelberg, 2010. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1888935.1888969`.

2 Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ESA.2019.14`.

3 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Renato F. Werneck. Public Transit Labeling. In Evripidis Bampis, editor, *Experimental Algorithms*, pages 273–285. Springer International Publishing, 2015. `doi:10.1007/978-3-319-20086-6_21`.

4 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. In *Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 130–140, 2012. `doi:10.1137/1.9781611972924.13`.

5 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Experimental Algorithms. SEA 2013*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54, Berlin, Heidelberg, 2013. Springer. `doi:10.1007/978-3-642-38527-8_6`.

6 Vassilissa Lehoux and Darko Drakulic. Mode Personalization in Trip-Based Transit Routing. In Valentina Cacchiani and Alberto Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASIcs)*, pages 13:1–13:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.ATMOS.2019.13`.

7 Vassilissa Lehoux and Christelle Loiodice. Faster Preprocessing for the Trip-Based Public Transit Routing Algorithm. In Dennis Huisman and Christos D. Zaroliagis, editors, *20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020)*, volume 85 of *OpenAccess Series in Informatics (OASIcs)*, pages 3:1–3:12, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ATMOS.2020.3`.

8 Île De France Mobilités. Open data. URL: `https://www.iledefrance-mobilites.fr`.

9 OSM. Open street map. URL: `https://www.openstreetmap.org`.

10 OSRM. Open source routing machine. URL: `http://project-osrm.org/`.

11 Duc-Minh Phan and Laurent Viennot. Fast public transitrouting with unrestrictedwalking through hub labeling. In *Proceedings of the Special Event on Analysis of Experimental Algorithms (SEA2019)*, volume 11544 of *Lecture Notes in Computer Science*. Springer, Cham, 2019. `doi:10.1007/978-3-030-34029-2_16`.

12 Andrea Raith, Marie Schmidt, Anita Schöbel, and Lisa Thom. Extensions of labeling algorithms for multi-objective uncertain shortest path problems. *Networks*, 72(1):84–127, 2018. `doi:10.1002/net.21815`.

13 Luis Ulloa, Vassilissa Lehoux, and Frédéric Roulland. Trip Planning Within a Multimodal Urban Mobility. *IET Intelligent Transport Systems*, 12(2):87–92, 2018. `doi:10.1049/iet-its.2016.0265`.

14 Dorothea Wagner and Tobias Zündorf. Public Transit Routing with Unrestricted Walking. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 7:1–7:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.ATMOS.2017.7`.

15 Sacha Witt. Trip-based public transit routing. In N. Bansal and I. Finocchi, editors, *ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2015. Springer. `doi:10.1007/978-3-662-48350-3_85`.

A   Algorithms

■ **Algorithm 2** Earliest arrival query.

---
input Timetable data, transfer set $\mathcal{T}$
input Source stop p_{src}, destination stop p_{tgt}, start time τ
input Maximum transfer duration $\Delta\tau_{\max}$, transfer duration coefficient σ
output Result set J
$J \leftarrow \emptyset$, $\mathcal{L} \leftarrow \emptyset$
$Q_n \leftarrow \emptyset$ for $n = 0, 1, \ldots$
$R(.) \leftarrow \infty$ for all trips t

INITIALIZATION()
$\tau_{\min} \leftarrow \infty$ ▷ The current minimum arrival time at target
$n \leftarrow 0$
while $Q_n \neq \emptyset$ **do**
   for each $t@b \to t@e \in Q_n$ **do**
                                  ▷ Checking if a target is reached
      for each $(L_t, i, \Delta\tau) \in \mathcal{L}$ with $b < i$ and $\tau_{arr}(t, i) + \Delta\tau < \tau_{\min}$ **do**
         $\tau_{\min} \leftarrow \tau_{arr}(t, i) + \Delta\tau$
         $J \leftarrow J \cup \{(\tau_{\min}, n)\}$, removing dominated entries
      end for

      if $\tau_{arr}(t, b+1) < \tau_{\min}$ **then**   ▷ Filling the queue for the next round
         for each transfer $(t@i \to u@j, \Delta\tau, \sigma_{\max}) \in \mathcal{T}$ with $b < i \leq e$ and
              $\sigma \times \Delta\tau \leq \Delta\tau_{\max}$ and $\sigma \leq \sigma_{\max}$ **do**
            ENQUEUE$(u, j, n+1)$
         end for
      end if
   end for
   $n \leftarrow n + 1$
end while

■ **Algorithm 3** Auxiliary procedures.

---
procedure INITIALIZATION
   for each stop q such that $\Delta\tau_{\mathrm{fp}}(q, p_{tgt})$ is defined **do**
      $\Delta\tau \leftarrow 0$ if $p_{tgt} = q$, else $\Delta\tau = \sigma \times \Delta\tau_{\mathrm{fp}}(q, p_{tgt})$
      for each $(L, i) \in \mathbf{L}(q)$ **do**
         $\mathcal{L} \leftarrow \mathcal{L} \cup \{(L, i, \Delta\tau\}$
      end for
   end for

   for each stop q such that $\Delta\tau_{\mathrm{fp}}(p_{src}, q)$ is defined **do**
      $\Delta\tau = 0$ if $p_{src} = q$, else $\Delta\tau = \sigma \times \Delta\tau_{\mathrm{fp}}(p_{src}, q)$
      for each $(L, i) \in \mathbf{L}(q)$ **do**
         $t \leftarrow$ earliest trip of L such that $\tau + \Delta\tau \leq \tau_{dep}(t, i)$
         ENQUEUE$(t, i, 0)$
      end for
   end for
end procedure
procedure ENQUEUE(trip t, index i, nb transfers n)
   if $i < R(t)$ **then**
      $Q_n \leftarrow Q_n \cup \{t@i \to t@R(t)\}$
      for each trip u with $t \leq u$ and $L_t = L_u$ **do**
         $R(u) \leftarrow \min(R(u), i)$
      end for
   end if
end procedure

Algorithm 4 Modification of transfer set building.

Input: Timetable data, footpath data
Input: Maximum and minimum transfer duration coefficients $\varsigma_{\max}$ and $\varsigma_{\min}$
Output: Reduced transfer set $\mathcal{T}$
$\mathcal{T} \leftarrow \emptyset$
for each line L **do**
   $\mathcal{T}(L) \leftarrow LINE\_TRANSFERS(L)$
   for each trip t of L **do**
      $T \leftarrow \emptyset$ $\triangleright$ Transfer set for each target line
      $L_{prev} \leftarrow$ **null**
      for each transfer $(i, L'@j, \Delta\tau)$ of $\mathcal{T}(L)$ **do**
         if $L_{prev} \neq L'$ **then**
            $\mathcal{T} \leftarrow \mathcal{T} \cup T$
            $T \leftarrow \emptyset,\ L_{prev} = L'$
         end if
         $t'_{\min} \leftarrow$ earliest trip of L' at j such that $\tau_{dep}(t'_{\min}, j) \geq \tau_{arr}(t, i) + \Delta\tau \times \varsigma_{\min}$
         $t'_{\max} \leftarrow$ earliest trip of L' at j such that $\tau_{dep}(t'_{\max}, j) \geq \tau_{arr}(t, i) + \Delta\tau \times \varsigma_{\max}$
         $Labs \leftarrow \emptyset$
         for each trip t', $t'_{\min} \leq t' \leq t'_{\max}$ **do**
            $\sigma_{\max} \leftarrow$ maximum value $\sigma \leq \varsigma_{\max}$ such that $\tau_{dep}(t', j) \geq \tau_{arr}(t, i) + \Delta\tau \times \sigma$
            $Labs \leftarrow Labs \cup \{(t@i \to t'@j, \Delta\tau, \sigma_{\max})\}$
         end for
         if $T = \emptyset$ **then**
            $T(L') \leftarrow Labs$
         else
            for each $lab \in Labs$ **do**
               if lab is not dominated by an element of T **then**
                  Update T with lab
               end if
            end for
         end if
      end for
      $\mathcal{T} \leftarrow \mathcal{T} \cup T$
   end for
end for
return $\mathcal{T}$

procedure LINE_TRANSFERS(line L, footpath data) $\triangleright$ Builds the line neighbourhood
   for $i \leftarrow |\overrightarrow{p}(L)| - 1, \ldots, 1$ **do**
      for each stop q such that $\Delta\tau_{\text{fp}}(L@i, q)$ is defined **do**
         for each (L', j) such that $q = L'@j$ **do**
            $\mathcal{T} \leftarrow \mathcal{T} \cup (i, L'@j, \Delta\tau_{\text{fp}}(L@i,\ L'@j))$
         end for
      end for
   end for
   Sort $\mathcal{T}$ first by target line, then by decreasing origin line index, then by increasing target line
index, then by chosen sorting in case of tides
   return $\mathcal{T}$
end procedure

B   Experiments

Tables 6 and 7 describe the number of transfers for each speed level for speed customization only and for maximum transfer duration and speed customization. We can observe that the number of transfers are similar for each speed in all configurations.

Table 6 Preprocessing for IDFM with maximum 10 min and 30 min transfer time - Number of transfers for each speed level in millions.

	IDFM (10 min)			IDFM (30 min)		
Version	Fast	Standard	Slow	Fast	Standard	Slow
Variable speed	126.6	114.5	99.5	623.2	639.6	699.0
Max. duration and var. speed	209.4	196.9	179.0	2 364.9	2 328.5	2 268.4

Table 7 Preprocessing for Korea with maximum 10 min transfer time - Number of transfers for each speed level in millions.

	Line based pruning			All prunings		
Version	Fast	Standard	Slow	Fast	Standard	Slow
Variable speed	608.7	631.6	678.9	379.8	324.8	248.1
Max duration and variable speed	1 126.0	1 107.5	1 067.2	548.4	486.7	392.0

Tables 8 and 9 describe the performances of profile queries.

Table 8 One-hour profile queries at 8.30 am with user defined speed.

Data set	Version	Speed	Mean query time (ms)	Mean nb solutions
IDFM 30 min	Standard	-	125	5.26
IDFM 30 min	Variable speed	Standard	202	5.26
IDFM 30 min	Variable speed	Slow	210	4.91
IDFM 30 min	Variable speed	Fast	155	5.6
IDFM 30 min	Max. duration - var. speed	Standard	347	5.26
IDFM 30 min	Max. duration - var. speed	Slow	229	4.91
IDFM 30 min	Max. duration - var. speed	Fast	237	5.6
IDFM 10 min	Standard	-	139	2.1
IDFM 10 min	Variable speed	Standard	146	2.1
IDFM 10 min	Variable speed	Slow	118	2.04
IDFM 10 min	Variable speed	Fast	126	2.28
IDFM 10 min	Max. duration - var. speed	Standard	144	2.1
IDFM 10 min	Max. duration - var. speed	Slow	137	2.04
IDFM 10 min	Max. duration - var. speed	Fast	126	2.28
Korea	Standard	-	586	3.96
Korea	Variable speed	Standard	698	3.96
Korea	Variable speed	Slow	672	3.83
Korea	Variable speed	Fast	682	4.25
Korea	Max. duration - var. speed	Standard	976	3.96
Korea	Max. duration - var. speed	Slow	941	3.83
Korea	Max. duration - var. speed	Fast	950	4.25

Table 9 One-hour profile queries at 8.30 am with user defined maximum transfer time and speed.

Data set	Speed	Max transfer time (min)	Mean query time (ms)	Mean nb solutions
IDFM 30 min	Standard	-	125	5.26
IDFM 30 min	Standard	20	347	5.26
IDFM 30 min	Standard	10	250	5.15
IDFM 30 min	Standard	5	262	5.33
IDFM 10 min	Standard	-	140	2.1
IDFM 10 min	Standard	5	136	2.14
Korea	Standard	-	976	3.96
Korea	Standard	5	989	3.9

An Improved Scheduling Algorithm for Traveling Tournament Problem with Maximum Trip Length Two

Diptendu Chatterjee[1] ✉ 📧

Applied Statistics Unit, Indian Statistical Institute, Kolkata, India

Bimal Kumar Roy ✉ 🏠

Applied Statistics Unit, Indian Statistical Institute, Kolkata, India

──── **Abstract** ────

The Traveling Tournament Problem(TTP) is a combinatorial optimization problem where we have to give a scheduling algorithm which minimizes the total distance traveled by all the participating teams of a double round-robin tournament maintaining given constraints. Most of the instances of this problem with more than ten teams are still unsolved. By definition of the problem the number of teams participating has to be even. There are different variants of this problem depending on the constraints. In this problem, we consider the case where number of teams is a multiple of four and a team can not play more than two consecutive home or away matches. Our scheduling algorithm gives better result than the existing best result for number of teams less or equal to 32.

1 Introduction

Double Round-robin tournament is one of the most unbiased way of evaluating teams participating in a competition. In this kind of tournament each of the participating team plays with every other team twice, i.e. one game in its home and another game in the home of the other team. This nullifies the effect of home ground and support. So, in this kind of tournament each team is tested in all the venues and in all the conditions. If there are n teams participating, then each team will play $2(n-1)$ games and total number of games played will be $n(n-1)$. After all the matches are played, the team with highest point wins the tournament. Traveling Tournament Problem is inspired by *Major League Baseball*. The general form of constrained Traveling Tournament Problem, i.e. $TTP - k$ for some natural number k, given participating teams and all the mutual distances between their home grounds is defined as follows.

▶ **Definition 1.** *TTP-k is scheduling of a double round-robin tournament where total travel distance by all the participating teams is minimized given the following constraints:*
1. *Each pair of participating team play exactly two matches with each other once in each of their home venues.*
2. *No pair of teams play consecutive matches with each other.*
3. *In an away tour a visiting team travels directly from the home of one opponent to home of the next opponent without returning to its own home.*
4. *The lengths of the home stands and away tours for any participating team is not more than k.*

For odd number of teams scheduling of a Traveling Tournament Problem is not possible as in a match day every team should participate.

Like its benefits, Traveling Tournament Problem has some drawbacks also. The main drawbacks are huge number of matches and scheduling complexity. We can not decrease the number of matches, but we can lower the complexity of the scheduling. But with imposed constraints on scheduling the complexity increases. For a small number of teams the scheduling is simpler and the complexity increases with number of teams and imposed constraints. TTP-∞ and and TTP-3 has been proven to be NP-hard in [2] and [18] respectively. TTP-1 is impossible to schedule [5]. So, the only possible case where complete solution may be possible is TTP-2. The complexity of TTP-2 is still not settled. The existing best result on approximating TTP-2 is given by Xiao and Kou [23]. They gave an approximation factor of $(1 + \frac{2}{n} + \frac{2}{n-2})$ for TTP-2 with n divisible by 4, where n is the number of participating teams. We work on a similar setup, where we schedule a TTP-2 on n teams with n divisible by 4 and our schedule improves the result for $n \leq 32$.

Formal definition of the problem, some useful definitions, notations and some well known results related to Traveling Tournament Problem are given here.

1.1   Problem Definition

TTP-2.   Traveling Tournament Problem-2 is scheduling of a double round-robin tournament where total travel distance by all the participating teams is minimized maintaining the following constraints:

Constraint 1: Each pair of participating team play exactly two matches with each other once in each of their home venues.

Constraint 2: No pair of teams play consecutive matches with each other.

Constraint 3: In an away tour a visiting team travels directly from the home of one opponent to home of the next opponent without returning to its own home.

Constraint 4: The lengths of the home stands and away tours for any participating team is not more than 2.

1.2   Previous Work

Traveling Tournament Problem(TTP) is a special variant of the Traveling Salesman Problem. The Traveling Tournament Problem was first introduced by Easton, Nemhauser, and Trick [7]. In a TTP, when there is no constraint on home stands or away trip length, it becomes a problem of scheduling n Traveling Salesman Problem synchronously. It has been shown that, TTP-k i.e. Traveling Tournament Problem with not more than k home stands or away matches is NP-Hard when $K = \infty$ [2] or $k = 3$ [18]. Relationship of some variants of round-robin tournaments with the planar three-index assignment problem has been analyzed and complexity of scheduling a minimum cost round-robin tournament has been established using the same [3]. They also showed the applicability of some techniques for planar three-index assignment problem to solve a sub-problem of scheduling a minimum cost round-robin tournament . A large amount of work has been done towards the approximation algorithms [23, 13, 12, 16, 22, 24]. A large amount of work on heuristic algorithms also has been done [1, 6, 8, 10, 15]. Many offline and online set of benchmark data set can be found for TTP-3 [7, 20]. For many benchmark result on improvements and complete solutions, work of high performance computers for more than a week is required [21]. But with that also most of the instances of TTP-k on more 10 teams are not completely solvable [20]. They worked on a basketball tournament with ten teams where the away trip for any team consists of one or

two matches. It is also ben shown that TTP-1 is impossible to schedule [5]. A survey on round-robin tournament scheduling has been done by Rasmessen and Trick [17]. Work has also been done on complexity of TTP-k [9, 11, 14].

Our main focus is on TTP-2 which was first introduced by Campbell and Chen [4].Thielen and Westphal [19] has contributed towards approximation factor for TTP-2 and later gave an approximation factor of $(1 + \frac{16}{n})$ for all $n \geq 12$ and n divisible by 4. Their result has been improved by Xiao and Kou [23]. They gave an approximation factor of $(1 + \frac{2}{n-2} + \frac{2}{n})$ where n is divisible by 4. Our scheduling algorithm give better result than this for $n \leq 32$.

1.3 Our Result

We propose a scheduling algorithm for TTP-2 which yields an approximation factor of $\left(1 + \frac{\lceil \log_2 \frac{n}{4} \rceil + 4}{2(n-2)}\right)$. For number of participating teams less or equal to 32, this gives a better result than existing best result, with approximation factor of $(1 + \frac{2}{n-2} + \frac{2}{n})$ in [23].

2 Preliminaries

2.1 Definitions and Notations

In this paper, for getting better approximation factor for TTP-2, graph theoretic approach has been followed. Due to this, teams are invariably referred as vertices and distances between home locations of teams are referred as weights of edges of the graph.

▶ **Definition 2.** *Matching Graph: A matching graph $G(V, E)$ is a graph where no two edges have a common vertex. So, for a matching graph, $|V| = n \Rightarrow |E| \leq \frac{n}{2}$. The pair of vertices connected through an edge in a matching graph is called matched vertices of the matching graph.*

▶ **Definition 3.** *Maximal Matching of a Graph: Maximal matching of a graph $G(V, E)$ is a matching of G, which is not subset of any other matching of G. It may not be unique for a given graph.*

▶ **Definition 4.** *Minimum Maximal Matching of an Undirected Weighted Graph: Minimum Maximal Matching of an Undirected Weighted Graph $G(V, E)$ is a maximal matching of G with sum of all the weights of its edges is the smallest among that of all the maximal matching subgraphs of G. For a minimum maximal matching of an undirected weighted complete graph with n vertices, the number of edges of the matching will be $\frac{n}{2}$.*

In this work, an edge between two vertices is represented as a match between the teams corresponding to the vertices. Now a *super-match* is defined as follows:

▶ **Definition 5.** *Super-match: A super-match between two pairs of matched vertices M_i and M_j is a set of edges $\{(u, w), (u, x), (v, w), (v, x)\}$ where $M_i = \{u, v\}$ and $M_j = \{w, x\}$.*

2.2 A Simple Lower Bound for TTP-2

Let, there are n teams participating in TTP-2. Distances between the home locations of each pair of teams are given. Let, d_{ij} be the distance between home locations of ith and j^{th} team. Now we construct an undirected weighted complete graph with all the n home locations as vertices with weights of the edges as the physical distances between the home locations of teams corresponding to the vertices connected through it and call it $G(V, E)$.

As G is a complete graph and $|V| = n =$ even, we get a minimum maximal matching in G and call it G_m. Let, sum of the weights of all the edges of G_m be W_m, sum of the weights of all the edges in G be W_t and sum of the weights of all the edges from a vertex i in G be W_i.

So, for an optimized schedule with the given constraints it is natural for a team to travel to two matched teams in G_m in an away trip. But for the vertex matched with itself in G_m, it will make a to and fro journey. In that case, the total travel by i^{th} team is $W_i + W_m$. This gives a minimum travel by ith team given the constraints.

Now, if it is possible to synchronously fit this above mentioned minimum travel by each participating team in the schedule then the total traveled distance by all the teams in the tournament will be,

$$\sum_{i \in V}(W_i + W_m) = 2W_t + nW_m$$

This gives a lower bound to TTP-2. But due to the imposed constraints on scheduling and number of teams, it is not always possible to synchronously fit the minimum travel schedule of each participating teams in the schedule and here comes the optimization and hardness of the problem and makes this problem interesting.

3   Design of Schedule

Suppose there are n teams participating in a Double Round-robin Tournament where n is divisible by 4. We construct the undirected weighted graph $G(V, E)$ as described in **Section 2** and also find the minimum maximal matching G_m in G. Now we number the vertices and the matched pairs such that matched pair M_i consist of vertices $2i-1$ and $2i$, $\forall i \in \{1, \dots, \frac{n}{2}\}$. Now, we design the schedule in $\lceil \log_2 \frac{n}{2} \rceil$ rounds and $(\frac{n}{2} - 1)$ levels such that i^{th} round is consist of $\lceil \frac{1}{2}(\frac{n}{2^i} - 1) \rceil$ levels and each level consists of $\frac{n}{4}$ *super-matches*. A *super-match* is played between two different matched pairs where both the teams in a matched pair plays home and away matches with both the teams in the other matched pair. In every level each matched pair plays a *super-match*. We have designed three types of *super-matches* which are used in our schedule. Suppose, there are two pairs of matched vertices A_1, A_2 and B_1, B_2 in G_m described in the previous section. We give three types of *super-match* namely Type-1, Type-2, Type-3 which are the building blocks of our schedule.

Type-1.   This consists of four match days namely T_1, T_2, T_3 and T_4 and the matches on this match days are given below:

$$
\begin{aligned}
T_1 &: \quad A_1 \to B_1, A_2 \to B_2.\\
T_2 &: \quad A_1 \to B_2, A_2 \to B_1.\\
T_3 &: \quad B_1 \to A_1, B_2 \to A_2.\\
T_4 &: \quad B_1 \to A_2, B_2 \to A_1.
\end{aligned}
$$

where $u \to v$ means u is playing an away match with v in the home of v.
The home-away match sequence of the participating teams become the following:
$A_1 : aahh : A_2 : aahh : B_1 : hhaa : B_2 : hhaa.$ where a means away match and h means home match.
Type-1 *super-match* does not violate minimum travel of any of its participating teams.
This way we can simultaneously schedule $\frac{n}{4}$ Type-1 *super-matches* in a level but then we can not schedule matches between the teams with same home away match sequences due to **constraint:4** of the problem definition. So we need a different kind of *super-match* like Type-1 and hence comes the need of Type-2 *super-match*.

Type-2. This consists of four match days namely T_1, T_2, T_3 and T_4 and the matches on this match days are given below:

T_1 : $A_1 \rightarrow B_1, A_2 \rightarrow B_2$.
T_2 : $B_2 \rightarrow A_1, B_1 \rightarrow A_2$.
T_3 : $B_1 \rightarrow A_1, B_2 \rightarrow A_2$.
T_4 : $A_1 \rightarrow B_2, A_2 \rightarrow B_1$.

 The home-away match sequence of the participating teams become the following:
$A_1 : ahha : A_2 : ahha : B_1 : haah : B_2 : haah$. Where a means away match and h means home match.
Type-2 *super-match* violates minimum travel of all of its participating teams but helps to schedule matches of all the teams according to their minimum travel schedule in the next level. We may refer the Type-2 *super-match* as *flip* in future. But after this modification also it is not possible to schedule home and away matches between two matched teams in G_m maintaining their minimum travel schedule. So there comes the need of Type-3 schedule block.

Type-3. This consists of six match days namely T_1, T_2, T_3, T_4, T_5 and T_6 and the matches on this match days are given below:

T_1 : $A_1 \rightarrow B_1, A_2 \rightarrow B_2$.
T_2 : $A_1 \rightarrow A_2, B_2 \rightarrow B_1$.
T_3 : $B_2 \rightarrow A_1, B_1 \rightarrow A_2$.
T_4 : $A_2 \rightarrow A_1, B_1 \rightarrow B_2$.
T_5 : $A_1 \rightarrow B_2, A_2 \rightarrow B_1$.
T_6 : $B_1 \rightarrow A_1, B_2 \rightarrow A_2$.

 The home-away match sequence of the participating teams become the following:
$A_1 : aahhah : A_2 : ahhaah : B_1 : hhaaha : B_2 : haahha$.
where a means away match and h means home match.

Although Type-1 *super-match* does not violate the minimum travel schedule for the teams, we can not schedule a double round robin tournament only with Type-1 *super-matches*. We need Type-2 and Type-3 *super-matches*. Now, $\frac{n}{4}$ number of Type-3 *super-matches* are unavoidable for any TTP-2 scheduling as each Type-3 *super-match* involves home and away matches between matched vertices for two pairs of matched vertices of G_m described in the previous section. So, for n participating teams at least $\frac{n}{4}$ number of Type-3 *super-matches* are required and our algorithm uses exactly $\frac{n}{4}$ numbers of Type-3 schedule blocks. Now, the only scope of improvement is reduction in numbers of Type-2 *super-matches*. So our main aim to keep the the number of Type-2 *super-matches* or *flips* as low as possible.

4 Our Algorithm

Following algorithm gives a improved schedule in terms of total distance traveled by all the teams than the existing best result [23] for TTP-2 when, $n \leq 32$ where the number of **Type-2** super matches are bounded by $\left(\frac{n}{8} * \left\lceil \log_2 \frac{n}{4} \right\rceil \right)$ for all $n \in \mathbb{N}$. In next section, few schedules are given as examples using our algorithm.

In the above pseudo code for TTP-2 of n teams using our technique, first we find the Minimum Maximal Matching in the complete graph on all the vertices or teams. Let the set of matched pair of vertices be $\{M_1, \ldots, M_{n/2}\}$. Then we consider each M_i's as a team situated at the mid point of the locations of its constituent vertices. Then for a complete graph on these M_i's as vertices, we again find the minimum maximal matching and let the set of matched vertices be $\{N_1, \ldots, N_{n/4}\}$.

■ **Algorithm 1** Schedule TTP-2.

1: **INPUT**: $G(V, E)$ with $|V| = n, |E| = \binom{n}{2}, W = \{w_e | e \in E\}$.
2: Identify the minimum maximal matching, $G_m(V, E_m)$, of G.
3: $\forall i \in \{1, \ldots, \frac{n}{2}\}$, define $M_i = \{(u, v) | u, v \in V$ & $\text{Edge}(u, v) \in E_m\}$.
4: $\forall v \in V$, allot a number to v such that $(u, v) \in M_i \implies \#u = (2i - 1)$ & $\#v = 2i \; \forall i \in \{1, \ldots, \frac{n}{2}\}$.
5: Define $X = \{x_i | \text{location of } x_i \text{ is in the midpoint of } u$ & $v \text{ where } (u, v) \in M_i \; \forall i \in \{1, \ldots, \frac{n}{2}\}\}$.
6: Define a complete graph $H(X, E') | \forall e \in E'$, weight of the edge $e, W_e = \text{dist}(x_m, x_n)$ where e is the edge between x_m & x_n.
7: Identify the minimum maximal matching, $H_m(X, E'_m)$, of H.
8: $\forall i \in \{1, \ldots, \frac{n}{4}\}$, define$N_i = \{(M_m, M_n) | x_m, x_n \in X$ & $\text{Edge}(x_m, x_n) \in E'_m\}$.
9: **for** $i = 1 : 1 : \lceil \log_2 \frac{n}{2} \rceil$ **do**
10:     **while** $2^{i+1} < n$ **do**
11:         **if** $2^{i+2} | n$ **then**
12:           Schedule first $\lceil \frac{1}{2} \times (\frac{n}{2^i} - 1) \rceil - 1$ levels of ith round each with $\frac{n}{4}$ Type-1 *super-matches* and last level with $\frac{n}{8}$ Type-1 and $\frac{n}{8}$ Type-2 *super-matches*.
13:         **else**
14:           Schedule the $\left\lfloor \frac{n}{2} \sum_1^i 2^{-k} - 1 \right\rfloor^{th}$ match days with $\lfloor \frac{n}{8} \rfloor$ Type-2 *super-matches* for $i \in \{1, 2, \ldots, \log_2 n\}$ and rest of the super-matches as Type-1. For all other match days except the last one schedule all super-matches as Type-1.
15:         **end if**
16:         Schedule this last level of the tournament with $\frac{n}{4}$ Type-3 *super-matches* where $\forall i \in \{1, \ldots, \frac{n}{4}\}, M_p$ plays with $M_q | M_p, M_q \in N_i$.
17:     **end while**
18: **end for**

Now, we schedule the Type-2 super-matches in the different levels of different rounds according to the rule described in line 12 or line 14 of the algorithm depending on the value of n. We schedule all the Type-3 super-matches in the last level of the last round of the tournament between matched pairs of M_i's, i.e. between the elements of N_i's to minimize the total travel distance.

5   Examples of Scheduling with Our Algorithm

For better understanding of our scheduling algorithm we give two examples of schedule for $n = 12, 16$ here and $n = 20, 24, 28$ in the Appendix-A. An improved schedule of *Indian Premier League*, where $n=8$, is presented in Appendix-B. Let,

$$F_n = \frac{n}{8} * \left\lceil \log_2 \frac{n}{4} \right\rceil \quad \text{for } n \in \mathbb{N}. \tag{1}$$

5.1   Schedule for $n = 12$

For designing a Traveling Tournament Problem of 12 teams using our technique, first we number the teams or the vertices with natural numbers as follows.

Vertex Set=\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}.

Then we find the Minimum Maximal Matching in the complete graph containing the vertices in the above mentioned vertex set. Let the set of matched pair of vertices be $\{M_1, M_2, M_3, M_4, M_5, M_6\}$ and without loss of generality we can say that

M_1={1,2}, M_2={3,4}, M_3={5,6}, M_4={7,8}, M_5={9,10}, M_6={11,12}

Then we consider each M_i's as a team situated at the mid point of the locations of its constituent vertices for $i \in \{1,2,3,4,5,6\}$. Then for a complete graph on these M_i's as vertices, we find the minimum maximal matching and let the set of matched vertices be

$\{N_1, N_2, N_3\}$ such that N_1={M_1, M_5}, N_2={M_2, M_3}, N_3={M_4, M_6}.

Now, we describe the fixture of super-matches in Table 1 to be scheduled in all the levels of all the rounds according to our scheduling technique in a tabular form. We can observe that the super-matches scheduled in the last level of the last round of the tournament are between matched pairs of M_i's, i.e. between the elements of N_i's.

Table 1 Fixture of Super-Matches for $n = 12$

Round:1, Level:1	**Round:1, Level:2**	**Round:1, Level:3**
$M_1 \xrightarrow{Type-1} M_2$	$M_1 \xrightarrow{Type-1} M_4$	$M_1 \xrightarrow{Type-1} M_3$
$M_3 \xrightarrow{Type-1} M_4$	$M_3 \xrightarrow{Type-2} M_6$	$M_6 \xrightarrow{Type-2} M_2$
$M_5 \xrightarrow{Type-1} M_6$	$M_5 \xrightarrow{Type-1} M_2$	$M_5 \xrightarrow{Type-1} M_4$

Round:2, Level:1	**Round:3, Level:1**
$M_1 \xrightarrow{Type-2} M_6$	$M_6 \xrightarrow{Type-3} M_4$
$M_2 \xrightarrow{Type-1} M_4$	$M_2 \xrightarrow{Type-3} M_3$
$M_5 \xrightarrow{Type-1} M_3$	$M_5 \xrightarrow{Type-3} M_1$

Number of *Flips*$= 3 = F_{12}$.

5.2 Schedule for $n = 16$

Now for designing a Traveling Tournament Problem of 16 teams using our technique, first we number the teams or the vertices with natural numbers in a similar fashion as follows.

Vertex Set={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}.

Then we find the Minimum Maximal Matching in the complete graph containing the vertices in the above mentioned vertex set. Let the set of matched pair of vertices be $\{M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8\}$ and without loss of generality we can say that

M_1={1,2}, M_2={3,4}, M_3={5,6}, M_4={7,8}, M_5={9,10}, M_6={11,12}, M_7={13,14}, M_8={15,16}

Then we consider each M_i's as a team situated at the mid point of the locations of its constituent vertices for $i \in \{1,2,3,4,5,6,7,8\}$. Then for a complete graph on these M_i's as vertices, we find the minimum maximal matching and let the set of matched vertices be $\{N_1, N_2, N_3, N_4\}$ where

N_1={M_1, M_5}, N_2={M_2, M_6}, N_3={M_3, M_7}, N_4={M_4, M_8}.

Now, we describe the fixture of super-matches in Table 2 to be scheduled in all the levels of all the rounds according to our scheduling technique in a tabular form. We can observe that the super-matches scheduled in the last level of the last round of the tournament are between matched pairs of M_i's, i.e. between the elements of N_i's. Also as 8 is a power of 2, we exactly know the super-matches which are *flips* in the different levels of all the rounds of the tournament according to our scheduling technique.

Correctness of this algorithm is assured by the structures of Type-1, Type-2 and Type-3 *super-matches*. As all three of these structures do not violate any of the constraints in the problem definition, so our schedule also does not violate any of the constraints. Which proves the correctness of our algorithm.

■ **Table 2** Fixture of Super-Matches for $n = 16$

Round:1, Level:1	**Round:1, Level:2**	**Round:1, Level:3**	**Round:1, Level:4**
$M_1 \xrightarrow{Type-1} M_2$	$M_1 \xrightarrow{Type-1} M_4$	$M_1 \xrightarrow{Type-1} M_6$	$M_1 \xrightarrow{Type-1} M_8$
$M_3 \xrightarrow{Type-1} M_4$	$M_3 \xrightarrow{Type-1} M_6$	$M_3 \xrightarrow{Type-1} M_8$	$M_3 \xrightarrow{Type-2} M_2$
$M_5 \xrightarrow{Type-1} M_6$	$M_5 \xrightarrow{Type-1} M_8$	$M_5 \xrightarrow{Type-1} M_2$	$M_5 \xrightarrow{Type-1} M_4$
$M_7 \xrightarrow{Type-1} M_8$	$M_7 \xrightarrow{Type-1} M_2$	$M_7 \xrightarrow{Type-1} M_4$	$M_7 \xrightarrow{Type-2} M_6$

Round:2, Level:1	**Round:2, Level:2**	**Round:3, Level:1**
$M_1 \xrightarrow{Type-1} M_3$	$M_1 \xrightarrow{Type-1} M_7$	$M_1 \xrightarrow{Type-3} M_5$
$M_5 \xrightarrow{Type-1} M_7$	$M_5 \xrightarrow{Type-2} M_3$	$M_3 \xrightarrow{Type-3} M_7$
$M_2 \xrightarrow{Type-1} M_8$	$M_2 \xrightarrow{Type-1} M_4$	$M_2 \xrightarrow{Type-3} M_6$
$M_6 \xrightarrow{Type-1} M_4$	$M_6 \xrightarrow{Type-2} M_8$	$M_8 \xrightarrow{Type-3} M_4$

Number of *Flips*= $4 = F_{16}$.

6   Proof of Results

Theorems related to the analysis of the proposed algorithm along with their proofs are presented in this section.

▶ **Theorem 6.** *All the Type-3 schedule blocks together introduce a relative error at most $\frac{2}{n-2}$ times of the Lower Bound of TTP-2.*

Proof. Suppose for some $i \in \{1, \ldots, \frac{n}{4}\}$, N_i includes 4 vertices of G i.e. A_1, A_2, B_1, B_2 where A_1 and A_2 are matched pairs in G_m and so are B_1 and B_2. For a Type-3 schedule in between them, travel for each team are given below:

$$A_1 \quad : \quad A_1 \to B_1 \to A_2 \to A_1 \to B_2 \to A_1.$$
$$A_2 \quad : \quad A_2 \to B_2 \to A_2 \to A_1 \to B_1 \to A_2.$$
$$B_1 \quad : \quad B_1 \to A_2 \to B_2 \to B_1 \to A_1 \to B_1.$$
$$B_2 \quad : \quad B_2 \to B_1 \to A_1 \to B_2 \to A_2 \to B_2.$$

So the total distance traveled is,

$$5*dist(A_1, B_1)+3*dist(A_2, B_1)+2*dist(A_1, A_2)+3*dist(A_1, B_2)+5*dist(A_2, B_2)+2*dist(B_1, B_2)$$

For the minimum travel schedule the value is,

$$2*dist(A_1, B_1)+2*dist(A_2, B_1)+6*dist(A_1, A_2)+2*dist(A_1, B_2)+2*dist(A_2, B_2)+6*dist(B_1, B_2)$$

So the extra amount of travel is,

$$3*dist(A_1, B_1)+1*dist(A_2, B_1)-4*dist(A_1, A_2)+1*dist(A_1, B_2)+3*dist(A_2, B_2)-4*dist(B_1, B_2)$$

Using triangle inequality, the above expression is upper bounded by,

$$2 * dist(A_1, B_1) + 2 * dist(A_2, B_2) + 2 * dist(A_1, B_2) + 2 * dist(A_2, B_1)$$

Let us denote, *super-edge D_{ij}* between pairs A_1, A_2 and B_1, B_2 as,

$$dist(A_1, B_1) + dist(A_2, B_2) + dist(A_1, B_2) + dist(A_2, B_1)$$

where

$$A_1, A_2 \in M_i \text{ and } B_1, B_2 \in M_j \text{ for some } i, j \in \{1, \ldots, \frac{n}{2}\}$$

Now, there are $\frac{n}{2}$ numbers of pair of vertices like A_1, A_2. If we consider all pairwise distances between all these $\frac{n}{2}$ pairs, then we get all the edges of the complete graph G but the edges of the matching G_m. But among all these $\binom{n/2}{2}$ pairwise distances, we are interested in $\frac{n}{4}$ matched pairwise distances as described in line 16 of algorithm 1, while calculating the error due to all Type-3 schedule blocks. So, the total error due to Type-3 schedule blocks is bounded by

$$2 * \frac{n/4}{\binom{n/2}{2}} * (W_t - W_m) < \frac{2}{n-2} * (\text{Lower Bound of TTP-2}). \qquad \blacktriangleleft$$

▶ **Theorem 7.** *All the Type-2 and Type-3 schedule blocks together introduces relative error at most* $\dfrac{\lceil \log_2 \frac{n}{4} \rceil + 4}{2(n-2)}$ *times of the Lower Bound of TTP-2.*

Proof. Suppose a Type-2 schedule block is designed among 4 vertices of G i.e. A_1, A_2, B_1, B_2 where A_1 and A_2 are matched pairs in G_m and so are B_1 and B_2. For a Type-2 schedule in between them, travel for each team are given below:

$A_1 \quad : \quad A_1 \to B_1 \to A_1 \to B_2 \to A_1.$

$A_2 \quad : \quad A_2 \to B_2 \to A_2 \to B_1 \to A_2.$

$B_1 \quad : \quad B_1 \to A_2 \to A_1 \to B_1.$

$B_2 \quad : \quad B_2 \to A_1 \to A_2 \to B_2.$

So the total distance traveled is,

$$3 * dist(A_1, B_1) + 3 * dist(A_2, B_1) + 2 * dist(A_1, A_2) + 3 * dist(A_1, B_2) + 3 * dist(A_2, B_2)$$

For the minimum travel schedule the value is,

$$2 * dist(A_1, B_1) + 2 * dist(A_2, B_1) + 2 * dist(A_1, A_2) + 2 * dist(A_1, B_2) + 2 * dist(A_2, B_2) + 2 * dist(B_1, B_2)$$

So the extra amount of travel is,

$$dist(A_1, B_1) + dist(A_2, B_1) + dist(A_1, B_2) + dist(A_2, B_2) - 2 * dist(B_1, B_2)$$

Which is upper bounded by,

$$dist(A_1, B_1) + dist(A_2, B_2) + dist(A_1, B_2) + dist(A_2, B_1)$$

Let us denote the pairwise distance $D_P(A, B)$ between pairs A_1, A_2 and B_1, B_2 as,

$$dist(A_1, B_1) + dist(A_2, B_2) + dist(A_1, B_2) + dist(A_2, B_1)$$

Now, there are $\frac{n}{2}$ numbers of pair of vertices like A_1, A_2. If we consider all pairwise distances between all these $\frac{n}{2}$ pairs, then we get all the edges of the complete graph G but the edges of the matching G_m. But among all these $\binom{n/2}{2}$ pairwise distances, we have already selected $\frac{n}{4}$ pairwise distances as described in the proof of Theorem 6 and now we are interested in at most F_n, given in equation 1, pairwise distances as per line 12 or 14 of algorithm 1, while calculating the error due to all Type-2 schedule blocks. So, the total error due to Type-2 and Type-3 schedule blocks is bounded by,

$$\frac{\frac{n}{8} * \lceil \log_2 \frac{n}{4} \rceil + \frac{n}{2}}{\binom{n/2}{2}} * (W_t - W_m) < \frac{\lceil \log_2 \frac{n}{4} \rceil + 4}{2(n-2)} * (\text{Lower Bound of TTP-2}). \qquad \blacktriangleleft$$

▶ **Theorem 8.** *Our algorithm gives better approximation than existing best result for number of participating teams less than or equal to 32.*

Proof. From the last two theorems we can see that the approximation factor in our algorithm is $1 + \frac{\left\lceil \log_2 \frac{n}{4} \right\rceil + 4}{2(n-2)}$ and in the existing best result the approximation factor is $1 + \frac{2}{n-2} + \frac{2}{n}$ [23]. So for $n \leq 32$,

$$\frac{8}{n} \leq \left\lfloor \log_2 \frac{64}{n} \right\rfloor \iff \frac{8}{n} \leq 4 - \left\lceil \log_2 \frac{n}{4} \right\rceil \iff \left\lceil \log_2 \frac{n}{4} \right\rceil \leq 4 - \frac{8}{n} \iff \left\lceil \log_2 \frac{n}{4} \right\rceil \leq \frac{4}{n}(n-2)$$

$$\iff \frac{\left\lceil \log_2 \frac{n}{4} \right\rceil}{(n-2)} \leq \frac{4}{n} \iff \frac{\left\lceil \log_2 \frac{n}{4} \right\rceil}{2(n-2)} \leq \frac{2}{n} \iff \frac{4 + \left\lceil \log_2 \frac{n}{4} \right\rceil}{2(n-2)} \leq \frac{2}{n-2} + \frac{2}{n}$$

This proves the theorem.   ◀

7   Conclusion

In this work, a better approximation factor than the existing best result has been achieved for Traveling Tournament Problem with maximum trip length two with our scheduling algorithm when the number of participating team is less or equal to 32. Due to time constraints and other factors, most of the tournaments involving number of teams more than 32 are not Round-Robin tournaments. For example a round-robin tournament with 40 teams will require 78 match days, 1560 matches and 40 grounds which demand lots of time, human support and a very long season. That is why most of the Round-Robin tournaments are conducted with less than 32 teams. Therefore, it can be said that for almost all practical cases the proposed scheduling algorithm would produce better result than the existing best result. One of the popular double round-robin tournament in India is **Indian Premier League(IPL)** and the number of teams involved in this tournament is 8. This tournament is not in TTP-2 structure now. But, if it is scheduled in TTP-2 structure, the proposed algorithm will significantly lower the total travel distance. An improved schedule of **IPL** using the proposed scheduling algorithm is presented in Appendix-B. It shows a 15% decrease in total travel distance in comparison with the actual **IPL-2019** schedule.

8   Scope of Future Work

As described in our algorithm, we know the specific match days of the schedule where the **Type-2** super matches or *Flips* are to be incorporated. But as we have specified the pairs of teams between whom the **Type-3** super matches are to be played to minimize the total travel distance due to the **Type-3** super matches, nothing of this kind is done for the *Flips*. So, a revisit in this topic can give some idea about the specific pairs of teams for minimizing the distance due to the *Flips*.

References

1   Aris Anagnostopoulos, Laurent Michel, Pascal Van Hentenryck, and Yannis Vergados. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9(2):177–193, 2006.

2   Rishiraj Bhattacharyya. Complexity of the unconstrained traveling tournament problem. *Operations Research Letters*, 44(5):649–654, 2016.

3   Dirk Briskorn, Andreas Drexl, and Frits CR Spieksma. Round robin tournaments and three index assignments. *4OR*, 8(4):365–374, 2010.

4   Robert Thomas Campbell and DS Chen. A minimum distance basketball scheduling problem. *Management science in sports*, 4:15–26, 1976.

5   Dominique De Werra. Some models of graphs for scheduling sports competitions. *Discrete Applied Mathematics*, 21(1):47–65, 1988.

6   Luca Di Gaspero and Andrea Schaerf. A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics*, 13(2):189–207, 2007.

7   Kelly Easton, George Nemhauser, and Michael Trick. The traveling tournament problem description and benchmarks. In *International Conference on Principles and Practice of Constraint Programming*, pages 580–584. Springer, 2001.

8   Kelly Easton, George Nemhauser, and Michael Trick. Solving the travelling tournament problem: A combined integer programming and constraint programming approach. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 100–109. Springer, 2002.

9   Nobutomo Fujiwara, Shinji Imahori, Tomomi Matsui, and Ryuhei Miyashiro. Constructive algorithms for the constant distance traveling tournament problem. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 135–146. Springer, 2006.

10   Marc Goerigk, Richard Hoshino, Ken-ichi Kawarabayashi, and Stephan Westphal. Solving the traveling tournament problem by packing three-vertex paths. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

11   Richard Hoshino and Ken-ichi Kawarabayashi. The linear distance traveling tournament problem. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

12   Richard Hoshino and Ken-ichi Kawarabayashi. An approximation algorithm for the bipartite traveling tournament problem. *Mathematics of Operations Research*, 38(4):720–728, 2013.

13   Shinji Imahori, Tomomi Matsui, and Ryuhei Miyashiro. A 2.75-approximation algorithm for the unconstrained traveling tournament problem. *Annals of Operations Research*, 218(1):237–247, 2014.

14   Graham Kendall, Sigrid Knust, Celso C Ribeiro, and Sebastián Urrutia. Scheduling in sports: An annotated bibliography. *Computers & Operations Research*, 37(1):1–19, 2010.

15   Andrew Lim, Brian Rodrigues, and Xingwen Zhang. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operational Research*, 174(3):1459–1478, 2006.

16   Ryuhei Miyashiro, Tomomi Matsui, and Shinji Imahori. An approximation algorithm for the traveling tournament problem. *Annals of Operations Research*, 194(1):317–324, 2012.

17   Rasmus V Rasmussen and Michael A Trick. Round robin scheduling–a survey. *European Journal of Operational Research*, 188(3):617–636, 2008.

18   Clemens Thielen and Stephan Westphal. Complexity of the traveling tournament problem. *Theoretical Computer Science*, 412(4-5):345–351, 2011.

19   Clemens Thielen and Stephan Westphal. Approximation algorithms for ttp (2). *Mathematical Methods of Operations Research*, 76(1):1–20, 2012.

20   MA Trick. Challenge traveling tournament instances. *http://mat. tepper. cmu. edu/TOURN/*, 1999.

21   Pascal Van Hentenryck and Yannis Vergados. Population-based simulated annealing for traveling tournaments. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 267. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

22   Stephan Westphal and Karl Noparlik. A 5.875-approximation for the traveling tournament problem. *Annals of Operations Research*, 218(1):347–360, 2014.

23   Mingyu Xiao and Shaowei Kou. An improved approximation algorithm for the traveling tournament problem with maximum trip length two. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

24   Daisuke Yamaguchi, Shinji Imahori, Ryuhei Miyashiro, and Tomomi Matsui. An improved approximation algorithm for the traveling tournament problem. *Algorithmica*, 61(4):1077, 2011.

A   More Examples of Schedule

Schedules for $n = 20, 24, 28$ are given below for a better insight of our algorithm.

A.1   Schedule for $n = 20$

Vertex Set $= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\}$.
Set of Pair of vertices $= \{M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8, M_9, M_{10}\}$;
where $M_1 = \{1, 2\}, M_2 = \{3, 4\}, M_3 = \{5, 6\}, M_4 = \{7, 8\}, M_5 = \{9, 10\}, M_6 = \{11, 12\}$,
$M_7 = \{13, 14\}, M_8 = \{15, 16\}, M_9 = \{17, 18\}, M_{10} = \{19, 20\}$;

Then we consider each M_i's as a team situated at the mid point of the locations of its constituent vertices for $i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Then for a complete graph on these M_i's as vertices, we find the minimum maximal matching and let the set of matched vertices be $\{N_1, N_2, N_3, N_4, N_5\}$ where

$N_1 = \{M_1, M_5\}, N_2 = \{M_2, M_{10}\}, N_3 = \{M_3, M_9\}, N_4 = \{M_4, M_7\}, N_5 = \{M_6, M_8\}$.

The fixture of super-matches for $n = 20$ is described in Table 3.

Table 3 Fixture of Super-Matches for $n = 20$

Round:1, Level:1	**Round:1, Level:2**	**Round:1, Level:3**
$M_1 \xrightarrow{Type-1} M_2$	$M_1 \xrightarrow{Type-1} M_4$	$M_1 \xrightarrow{Type-1} M_6$
$M_3 \xrightarrow{Type-1} M_4$	$M_3 \xrightarrow{Type-1} M_6$	$M_3 \xrightarrow{Type-2} M_8$
$M_5 \xrightarrow{Type-1} M_6$	$M_5 \xrightarrow{Type-1} M_8$	$M_5 \xrightarrow{Type-1} M_{10}$
$M_7 \xrightarrow{Type-1} M_8$	$M_7 \xrightarrow{Type-1} M_{10}$	$M_7 \xrightarrow{Type-2} M_2$
$M_9 \xrightarrow{Type-1} M_{10}$	$M_9 \xrightarrow{Type-1} M_2$	$M_9 \xrightarrow{Type-1} M_4$

Round:1, Level:4	**Round:1, Level:5**	**Round:2, Level:1**
$M_1 \xrightarrow{Type-1} M_3$	$M_1 \xrightarrow{Type-1} M_8$	$M_1 \xrightarrow{Type-2} M_7$
$M_8 \xrightarrow{Type-2} M_{10}$	$M_{10} \xrightarrow{Type-1} M_4$	$M_{10} \xrightarrow{Type-1} M_6$
$M_5 \xrightarrow{Type-1} M_7$	$M_5 \xrightarrow{Type-1} M_3$	$M_5 \xrightarrow{Type-2} M_4$
$M_2 \xrightarrow{Type-1} M_4$	$M_2 \xrightarrow{Type-1} M_6$	$M_2 \xrightarrow{Type-1} M_3$
$M_9 \xrightarrow{Type-1} M_6$	$M_9 \xrightarrow{Type-1} M_7$	$M_9 \xrightarrow{Type-1} M_8$

Round:2, Level:2	**Round:3, Level:1**	**Round:4, Level:1**
$M_7 \xrightarrow{Type-1} M_3$	$M_7 \xrightarrow{Type-1} M_6$	$M_7 \xrightarrow{Type-3} M_4$
$M_{10} \xrightarrow{Type-1} M_1$	$M_{10} \xrightarrow{Type-1} M_3$	$M_{10} \xrightarrow{Type-3} M_2$
$M_4 \xrightarrow{Type-1} M_6$	$M_4 \xrightarrow{Type-2} M_8$	$M_8 \xrightarrow{Type-3} M_6$
$M_2 \xrightarrow{Type-1} M_8$	$M_2 \xrightarrow{Type-2} M_5$	$M_5 \xrightarrow{Type-3} M_1$
$M_9 \xrightarrow{Type-1} M_5$	$M_9 \xrightarrow{Type-1} M_1$	$M_9 \xrightarrow{Type-3} M_3$

Number of *Flips*$= 7 = \lfloor F_{20} \rfloor$.

A.2   Schedule for $n = 24$

Vertex Set $= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24\}$.
Set of Pair of vertices $= \{M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8, M_9, M_{10}, M_{11}, M_{12}\}$;
where $M_1=\{1,2\}, M_2=\{3,4\}, M_3=\{5,6\}, M_4=\{7,8\}, M_5=\{9,10\}, M_6=\{11,12\}, M_7=\{13,14\}$,
$M_8=\{15,16\}, M_9=\{17,18\}, M_{10}=\{19,20\}, M_{11}=\{21,22\}, M_{12}=\{23,24\}$;

Then we consider each M_i's as a team situated at the mid point of the locations of its constituent vertices for $i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. Then for a complete graph on these M_i's as vertices, we find the minimum maximal matching and let the set of matched vertices be $\{N_1, N_2, N_3, N_4, N_5, N_6\}$ where

$N_1 = \{M_9, M_5\}$, $N_2 = \{M_1, M_7\}$, $N_3 = \{M_{11}, M_3\}$, $N_4 = \{M_{10}, M_6\}$, $N_5 = \{M_2, M_4\}$, $N_6 = \{M_8, M_{12}\}$.

The fixture of super-matches for $n = 24$ is given in Table 4.

Table 4 Fixture of Super-Matches for $n = 24$

Round:1, Level:1	**Round:1, Level:2**	**Round:1, Level:3**	**Round:1, Level:4**
$M_1 \xrightarrow{Type-1} M_2$	$M_1 \xrightarrow{Type-1} M_4$	$M_1 \xrightarrow{Type-1} M_6$	$M_1 \xrightarrow{Type-1} M_8$
$M_3 \xrightarrow{Type-1} M_4$	$M_3 \xrightarrow{Type-1} M_6$	$M_3 \xrightarrow{Type-1} M_8$	$M_3 \xrightarrow{Type-1} M_{10}$
$M_5 \xrightarrow{Type-1} M_6$	$M_5 \xrightarrow{Type-1} M_8$	$M_5 \xrightarrow{Type-1} M_{10}$	$M_5 \xrightarrow{Type-1} M_{12}$
$M_7 \xrightarrow{Type-1} M_8$	$M_7 \xrightarrow{Type-1} M_{10}$	$M_7 \xrightarrow{Type-1} M_{12}$	$M_7 \xrightarrow{Type-1} M_2$
$M_9 \xrightarrow{Type-1} M_{10}$	$M_9 \xrightarrow{Type-1} M_{12}$	$M_9 \xrightarrow{Type-1} M_2$	$M_9 \xrightarrow{Type-1} M_4$
$M_{11} \xrightarrow{Type-1} M_{12}$	$M_{11} \xrightarrow{Type-1} M_2$	$M_{11} \xrightarrow{Type-1} M_4$	$M_{11} \xrightarrow{Type-1} M_6$

Round:1, Level:5	**Round:1, Level:6**	**Round:2, Level:1**	**Round:2, Level:2**
$M_1 \xrightarrow{Type-1} M_{10}$	$M_1 \xrightarrow{Type-2} M_{12}$	$M_{12} \xrightarrow{Type-1} M_2$	$M_{12} \xrightarrow{Type-1} M_6$
$M_3 \xrightarrow{Type-1} M_{12}$	$M_3 \xrightarrow{Type-1} M_2$	$M_3 \xrightarrow{Type-1} M_1$	$M_3 \xrightarrow{Type-1} M_5$
$M_5 \xrightarrow{Type-1} M_2$	$M_5 \xrightarrow{Type-2} M_4$	$M_4 \xrightarrow{Type-1} M_6$	$M_4 \xrightarrow{Type-2} M_{10}$
$M_7 \xrightarrow{Type-1} M_4$	$M_7 \xrightarrow{Type-1} M_6$	$M_7 \xrightarrow{Type-1} M_5$	$M_7 \xrightarrow{Type-2} M_9$
$M_9 \xrightarrow{Type-1} M_6$	$M_9 \xrightarrow{Type-2} M_8$	$M_8 \xrightarrow{Type-1} M_{10}$	$M_8 \xrightarrow{Type-1} M_2$
$M_{11} \xrightarrow{Type-1} M_8$	$M_{11} \xrightarrow{Type-1} M_{10}$	$M_{11} \xrightarrow{Type-1} M_9$	$M_{11} \xrightarrow{Type-1} M_1$

Round:2, Level:3	**Round:3, Level:1**	**Round:4, Level:1**	
$M_{12} \xrightarrow{Type-1} M_4$	$M_{12} \xrightarrow{Type-2} M_{10}$	$M_9 \xrightarrow{Type-3} M_5$	
$M_3 \xrightarrow{Type-1} M_7$	$M_3 \xrightarrow{Type-2} M_9$	$M_1 \xrightarrow{Type-3} M_7$	
$M_{10} \xrightarrow{Type-2} M_2$	$M_2 \xrightarrow{Type-1} M_6$	$M_{11} \xrightarrow{Type-3} M_3$	
$M_9 \xrightarrow{Type-2} M_1$	$M_1 \xrightarrow{Type-1} M_5$	$M_{10} \xrightarrow{Type-3} M_6$	
$M_8 \xrightarrow{Type-1} M_6$	$M_8 \xrightarrow{Type-1} M_4$	$M_2 \xrightarrow{Type-3} M_4$	
$M_{11} \xrightarrow{Type-1} M_5$	$M_{11} \xrightarrow{Type-1} M_7$	$M_8 \xrightarrow{Type-3} M_{12}$	

Number of *Flips* $= 9 = F_{24}$.

A.3 Schedule for $n = 28$

Vertex Set $= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28\}$.
Set of Pair of vertices $= \{M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8, M_9, M_{10}, M_{11}, M_{12}, M_{13}, M_{14}\}$;
where $M_1 = \{1,2\}$, $M_2 = \{3,4\}$, $M_3 = \{5,6\}$, $M_4 = \{7,8\}$, $M_5 = \{9,10\}$, $M_6 = \{11,12\}$, $M_7 = \{13,14\}$, $M_8 = \{15,16\}$, $M_9 = \{17,18\}$, $M_{10} = \{19,20\}$, $M_{11} = \{21,22\}$, $M_{12} = \{23,24\}$, $M_{13} = \{25,26\}$, $M_{14} = \{27,28\}$;

Then we consider each M_i's as a team situated at the mid point of the locations of its constituent vertices for $i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$. Then for a complete graph on these M_i's as vertices, we find the minimum maximal matching and let the set of matched vertices be $\{N_1, N_2, N_3, N_4, N_5, N_6, N_7\}$ where

$N_1 = \{M_{14}, M_4\}$, $N_2 = \{M_{12}, M_6\}$, $N_3 = \{M_3, M_2\}$, $N_4 = \{M_8, M_{10}\}$, $N_5 = \{M_9, M_5\}$, $N_6 = \{M_7, M_{11}\}$, $N_7 = \{M_1, M_{13}\}$.

The fixture of super-matches for $n = 28$ is presented in Table 5.

Table 5 Fixture of Super-Matches for $n = 28$

Round:1, Level:1	**Round:1, Level:2**	**Round:1, Level:3**	**Round:1, Level:4**
$M_1 \xrightarrow{Type-1} M_2$	$M_1 \xrightarrow{Type-1} M_4$	$M_1 \xrightarrow{Type-1} M_6$	$M_1 \xrightarrow{Type-1} M_8$
$M_3 \xrightarrow{Type-1} M_4$	$M_3 \xrightarrow{Type-1} M_6$	$M_3 \xrightarrow{Type-1} M_8$	$M_3 \xrightarrow{Type-1} M_{10}$
$M_5 \xrightarrow{Type-1} M_6$	$M_5 \xrightarrow{Type-1} M_8$	$M_5 \xrightarrow{Type-1} M_{10}$	$M_5 \xrightarrow{Type-1} M_{12}$
$M_7 \xrightarrow{Type-1} M_8$	$M_7 \xrightarrow{Type-1} M_{10}$	$M_7 \xrightarrow{Type-1} M_{12}$	$M_7 \xrightarrow{Type-1} M_{14}$
$M_9 \xrightarrow{Type-1} M_{10}$	$M_9 \xrightarrow{Type-1} M_{12}$	$M_9 \xrightarrow{Type-1} M_{14}$	$M_9 \xrightarrow{Type-1} M_2$
$M_{11} \xrightarrow{Type-1} M_{12}$	$M_{11} \xrightarrow{Type-1} M_{14}$	$M_{11} \xrightarrow{Type-1} M_2$	$M_{11} \xrightarrow{Type-1} M_4$
$M_{13} \xrightarrow{Type-1} M_{14}$	$M_{13} \xrightarrow{Type-1} M_2$	$M_{13} \xrightarrow{Type-1} M_4$	$M_{13} \xrightarrow{Type-1} M_6$

Round:1, Level:5	**Round:1, Level:6**	**Round:1, Level:7**
$M_1 \xrightarrow{Type-1} M_{12}$	$M_1 \xrightarrow{Type-1} M_{10}$	$M_1 \xrightarrow{Type-1} M_{14}$
$M_3 \xrightarrow{Type-1} M_{14}$	$M_3 \xrightarrow{Type-2} M_{12}$	$M_{12} \xrightarrow{Type-1} M_{10}$
$M_5 \xrightarrow{Type-1} M_2$	$M_5 \xrightarrow{Type-1} M_{14}$	$M_5 \xrightarrow{Type-1} M_7$
$M_7 \xrightarrow{Type-1} M_4$	$M_7 \xrightarrow{Type-2} M_2$	$M_2 \xrightarrow{Type-1} M_4$
$M_9 \xrightarrow{Type-1} M_6$	$M_9 \xrightarrow{Type-1} M_4$	$M_9 \xrightarrow{Type-1} M_{11}$
$M_{11} \xrightarrow{Type-1} M_8$	$M_{11} \xrightarrow{Type-2} M_6$	$M_6 \xrightarrow{Type-1} M_8$
$M_{13} \xrightarrow{Type-1} M_{10}$	$M_{13} \xrightarrow{Type-1} M_8$	$M_{13} \xrightarrow{Type-1} M_3$

Round:2, Level:1	**Round:2, Level:2**	**Round:2, Level:3**
$M_1 \xrightarrow{Type-1} M_7$	$M_1 \xrightarrow{Type-2} M_{11}$	$M_{11} \xrightarrow{Type-2} M_{10}$
$M_{12} \xrightarrow{Type-1} M_8$	$M_{12} \xrightarrow{Type-1} M_4$	$M_{12} \xrightarrow{Type-1} M_{14}$
$M_5 \xrightarrow{Type-1} M_4$	$M_5 \xrightarrow{Type-2} M_3$	$M_3 \xrightarrow{Type-1} M_1$
$M_2 \xrightarrow{Type-1} M_{10}$	$M_2 \xrightarrow{Type-1} M_{14}$	$M_2 \xrightarrow{Type-2} M_8$
$M_9 \xrightarrow{Type-1} M_3$	$M_9 \xrightarrow{Type-1} M_8$	$M_9 \xrightarrow{Type-1} M_7$
$M_6 \xrightarrow{Type-1} M_{14}$	$M_6 \xrightarrow{Type-1} M_{10}$	$M_6 \xrightarrow{Type-1} M_4$
$M_{13} \xrightarrow{Type-1} M_{11}$	$M_{13} \xrightarrow{Type-1} M_7$	$M_{13} \xrightarrow{Type-2} M_5$

Round:3, Level:1	**Round:3, Level:2**	**Round:4, Level:1**
$M_{10} \xrightarrow{Type-1} M_4$	$M_{10} \xrightarrow{Type-2} M_{14}$	$M_{14} \xrightarrow{Type-3} M_4$
$M_{12} \xrightarrow{Type-1} M_{13}$	$M_{12} \xrightarrow{Type-1} M_2$	$M_{12} \xrightarrow{Type-3} M_6$
$M_3 \xrightarrow{Type-1} M_7$	$M_3 \xrightarrow{Type-1} M_{11}$	$M_3 \xrightarrow{Type-3} M_2$
$M_8 \xrightarrow{Type-1} M_{14}$	$M_8 \xrightarrow{Type-1} M_4$	$M_8 \xrightarrow{Type-3} M_{10}$
$M_9 \xrightarrow{Type-1} M_1$	$M_9 \xrightarrow{Type-1} M_{13}$	$M_9 \xrightarrow{Type-3} M_5$
$M_6 \xrightarrow{Type-1} M_2$	$M_6 \xrightarrow{Type-2} M_7$	$M_7 \xrightarrow{Type-3} M_{11}$
$M_5 \xrightarrow{Type-1} M_1$	$M_5 \xrightarrow{Type-2} M_1$	$M_1 \xrightarrow{Type-3} M_{13}$

Number of *Flips*$= 11 = \lceil F_{28} \rceil$.

B   Tabular IPL Schedule

In this section, we present a schedule of **Indian Premier League(IPL)** using our algorithm1. IPL is a Double Round-robin Tournament of eight teams. The proposed schedule is presented in Table-1 where the teams are represented as the following:

KOL → Kolkata Knight Riders   **MUM** → Mumbai Indians
CHE → Chennai Super Kings   **BANG** → Royal Challengers Bangalore
RAJ → Rajasthan Royals   **DEL** → Delhi Capitals
HYD → Sunrisers Hyderabad   **PUN** → Kings XI Punjab

■ **Table 6** *Proposed Indian Premier League Schedule*.

Match Day	1
Away	*Home*
MUM	KOL
HYD	RAJ
CHE	DEL
BANG	PUN

Match Day	2
Away	*Home*
MUM	RAJ
HYD	KOL
CHE	PUN
BANG	DEL

Match Day	3
Away	*Home*
RAJ	HYD
KOL	MUM
DEL	CHE
PUN	BANG

Match Day	4
Away	*Home*
RAJ	MUM
KOL	HYD
DEL	BANG
PUN	CHE

Match Day	5
Away	*Home*
MUM	DEL
HYD	PUN
CHE	KOL
BANG	RAJ

Match Day	6
Away	*Home*
MUM	PUN
HYD	DEL
RAJ	CHE
KOL	BANG

Match Day	7
Away	*Home*
PUN	HYD
DEL	MUM
RAJ	BANG
KOL	CHE

Match Day	8
Away	*Home*
DEL	HYD
PUN	MUM
CHE	RAJ
BANG	KOL

Match Day	9
Away	*Home*
MUM	CHE
HYD	BANG
KOL	DEL
RAJ	PUN

Match Day	10
Away	*Home*
MUM	HYD
KOL	RAJ
BAG	CHE
PUN	DEL

Match Day	11
Away	*Home*
BANG	MUM
CHE	HYD
PUN	KOL
DEL	RAJ

Match Day	12
Away	*Home*
HYD	MUM
RAJ	KOL
CHE	BANG
DEL	PUN

Match Day	13
Away	*Home*
MUM	BANG
HYD	CHE
KOL	PUN
RAJ	DEL

Match Day	14
Away	*Home*
CHE	MUM
BANG	HYD
DEL	KOL
PUN	RAJ

This schedule gives 15% better result than actual **IPL-2019** schedule in terms of total distance traveled by all the teams.

Efficient Algorithms for the Multi-Period Line Planning Problem in Public Transportation

Güvenç Şahin[1] ✉ 🆔
Industrial Engineering,Sabanci University, Istanbul, Turkey
Zuse Institut Berlin, Germany

Amin Ahmadi Digehsara ✉ 🆔
Industrial Engineering, Sabanci University, Istanbul, Turkey

Ralf Borndörfer ✉ 🆔
Zuse Institut Berlin, Germany

──────── **Abstract** ────────

In order to plan and schedule a demand-responsive public transportation system, both temporal and spatial changes in demand should be taken into account even at the line planning stage. We study the multi-period line planning problem with integrated decisions regarding dynamic allocation of vehicles among the lines. Given the NP-hard nature of the line planning problem, the multi-period version is clearly difficult to solve for large public transit networks even with advanced solvers. It becomes necessary to develop algorithms that are capable of solving even the very-large instances in reasonable time. For instances which belong to real public transit networks, we present results of a heuristic local branching algorithm and an exact approach based on constraint propagation.

1 Introduction

Responsive and flexible public transportation services become more indispensable as private services are under the radar for their detrimental effect on the environment. Earlier research on public transportation planning focused on fundamental issues such as identifying and framing the problems and constructing accurate models for these problems while integration of various problems associated with different planning stages has come forward [7] more recently along with advancement in research and computational power. While on-demand services, acclaimed for their responsiveness and utmost flexibility, are considered as the potential future of public transportation, it is accepted that they cannot replace the traditional public transit services. Yet, responsiveness of public services could be improved without sacrificing efficiency and effectiveness. In this respect, transit demand as the main driver should be pivotal in developing the plans and constructing the schedules for these services.

Traditional planning approaches consider demand as a static component particularly at higher levels of decision making. This is natural and plausible for the network planning and development stage. On the other hand, concurrent spatial and temporal changes in the

──────────────

[1] Corresponding author.

demand are critical for operational schedules. Recently, [6] propose a novel multi-period approach for the line planning stage supposing that line plans are the main link between strategic and operational plans and should be made with careful consideration of operational issues at the strategic level. With an effort to improve the demand-responsiveness of line plans, their primary contribution is the development of a multi-period line planning model which considers the changes in transit demand over time. Concurrently, they integrate tactical resource allocation constraints in order to ensure feasibility of multi-period line plans and exemplify this integration with dynamic allocation and assignment of vehicles to lines.

In their cost oriented multi-period approach with fixed costs of line selection and variable costs of service frequency on lines, the planning horizon is divided into discrete time periods each of which is associated with a different demand pattern. While the level of service on each line is determined for each period with the corresponding demand pattern, the periods are not independent from each other as they are coupled through the line selection decisions [6]. When compared to traditional static counterparts, allocation and assignment of resources to activities throughout the planning horizon are crucial in the case of multi-period planning [8]. As the activity levels (line frequencies) change from one period to the next, the resources (vehicles) are to be reallocated or reassigned. For vehicle scheduling and assignment, this can be achieved by discretizing the planning horizon as in [1] and [4]. Accordingly, a vehicle service (or a trip) is completed in one period; it can then be used on the same line or transferred to another. In the case of the latter, consideration of appropriate transfer time (i.e. the time it takes for the vehicle to travel from the ending station of one line to the starting station of another) is necessary.

In [6], it is shown that a multi-period approach is necessary when demand variation in time is a significant issue and also superior to a traditional approach that would combine line planning solutions of independent individual periods. However, computational challenges persist even at a higher level in comparison to single-period static line planning problems not only because of the convoluted structure of the multi-period line planning problem but also due to integration of vehicle transfer constraints. Out of the three PTN examples, finding optimal solutions for the largest one, namely the Quito Trolebus system, is not possible with a commercial solver. In this work, we discuss possible approaches that can be scaled to solve multi-period line planning problems with vehicle transfers even for a very-large PTN.

2   Problem Setting

An instance of the multi-period line planning problem presented in [6] is denoted by a public transportation network $PTN = (S, E)$ defined by a set of stations S and set of edges E connecting the stations, a set of time intervals T representing the planning horizon, transit demand d_e^t over the edges $e \in E$ in each period $t \in T$, and a set of potential lines L. A line $l \in L$ can be described as a path with a starting station and an ending station along with a subset of the edges to represent the path. Given the length of a discrete time period along with the starting and ending stations of lines, the transfer time from line l to line k is denoted by ρ_{lk} which should be calculated in multiples of time periods. In order to account for idle vehicles during a time period, an artificial line l_0 is used to represent a depot while $L_0 = L \cup \{l_0\}$.

The mathematical model in the form of a mixed integer programming problem formulation for the multi-period line planning problem with vehicle transfers (MPLPP-VT) includes a binary variable $y_l \in \{0, 1\}$ that takes value 1 if line l is selected, a non-negative integer variable v_l^t denoting the service level (and also corresponding to the number of vehicles

dispatched) on line l in period t, and w_{lk}^{st} denoting the number of vehicles from line l used in period s to line k to be used in period t. Given that c_l^f and c_l^o denote respectively the fixed cost of selecting a line charged for the complete planning horizon and the operational cost on a line for each service in a period, the resulting formulation becomes

$$\min \quad \sum_{l \in L} c_l^f y_l + \sum_{l \in L} \sum_{t \in t} c_l^o v_l^t \tag{1}$$

$$\text{s.t.} \quad \sum_{l \in L_e} \mathcal{K} v_l^t \geq d_e^t \qquad \forall\, e \in E, \forall\, t \in T \tag{2}$$

$$\mathcal{W} y_l - v_l^t \geq 0 \qquad \forall\, l \in L, \forall\, t \in T \tag{3}$$

$$\sum_{\substack{k \in L_0 \\ t - \rho_{kl} \geq 0}} w_{kl}^{t-\rho_{kl},t} = v_l^t \qquad \forall\, l \in L, \forall\, t \in T \tag{4}$$

$$\sum_{\substack{k \in L_0 \\ t - \rho_{kl} \geq 0}} w_{kl}^{t-\rho_{kl},t} - \sum_{\substack{k \in L_0 \\ t + \rho_{kl} \leq |T|+1}} w_{lk}^{t,t+\rho_{lk}} = 0 \qquad \forall\, l \in L_0, \forall\, t \in T \tag{5}$$

$$\sum_{l \in L_0} \sum_{t \in T} w_{l_0 l}^{0t} = \mathcal{U} \tag{6}$$

$$\sum_{l \in L_0} \sum_{t \in T} w_{ll_0}^{t,|T|+1} = \mathcal{U} \tag{7}$$

$$y_l \in \{0, 1\} \qquad \forall\, l \in L \tag{8}$$

$$v_l^t \in N \qquad \forall\, l \in L, \forall\, t \in T \tag{9}$$

$$w_{lk}^{st} \in N \qquad \forall\, l, k \in L_0, \forall\, s \in \{0\} \cup T,$$
$$\forall\, t \in T \cup \{|T|+1\}, s < t. \tag{10}$$

The objective function (1) is to minimize the sum of total fixed costs for selecting lines and variable costs for providing service. Constraints (2) ensure that the demand on an edge in a period is covered by sufficient number of services with $\mathcal{K}$ denoting the capacity of a vehicle and L_e denoting the lines containing edge e in their path. Constraints (3) associate the line selections with service level decisions and put an upper bound $\mathcal{W}$ on the service level of a line in a period. Constraints (4) provide required number of vehicles to a line in each period considering all transfers including the vehicles that are already on the line (self-transfer represented with $w_{ll}^{t-1,t}$) and are to be retrieved from the depot. In each period, constraints (5) balance the vehicles transferred to and transferred from the line, again including self-transfers. Fleet size is controlled by constraints (6) and (7) ensuring that $\mathcal{U}$ vehicles are released from the depot at the beginning of the planning horizon, period 0, and all $\mathcal{U}$ vehicles are transferred back to the depot at the end of the planning horizon, period $|T| + 1$.

Line planning problem is known to be NP-Hard, even for many special cases as shown in [9]. Therefore, computational challenges are expected to increase when many line planning problems are coupled with each other along with the addition of resource related constraints as exemplified for a very large instance of a real PTN in [6] which cannot be solved to optimality in reasonable time. Hence, it should be worthwhile to work on both heuristic and efficient exact algorithms.

3 Algorithms

Our earlier attempts focused on methods that rely on Benders' decomposition and Lagrangean relaxation. However, both methods failed to produce reliable algorithms. As a heuristic which still relies on solving the problem formulation (1)–(10), we use a local branching algorithm in its traditional form. As an exact solution approach, we propose an algorithm that solves a part of the problem formulation and adds missing constraints iteratively when they are violated only.

3.1 Local Branching

Local branching is an iterative method which may provide a high-quality incumbent solution within an acceptable computational time [3]. At each iteration, the original problem is divided into two sufficiently smaller sub-problems by generating so-called local branching cuts. The sub-problems include the feasible solutions of the original problem satisfying the additional local branching cuts. The algorithm may either identify a better feasible solution by solving the sub-problems within a short time or change the search region by a diversification mechanism. The algorithm terminates when some stopping criteria, i.e., the total time limit or the maximum number of diversifications, are reached. In the case of the MPLPP-VT, binary decision variables for line selection are used to partition the original solution space.

### 3.2 Logic-based Decomposition with Constraint Propagation

The spirit of our exact solution approach dates back to the original ideas in [2] for the TSP in the sense that we first eliminate a subset of the constraints, find a feasible solution with respect to the remaining constraints and identify which of the relaxed constraints are violated by this solution, and add the violated constraints to the problem formulation.

The algorithm iteratively continues in this fashion until no constraint violation is detected at an iteration. A critical feature of our algorithm is to explore only integer feasible solutions; hence, the integrality constraints are not relaxed. This idea of generating integer solutions for a relaxation of an integer programming problem formulation has been explored several times, particularly for the TSP. However, a straightforward implementation of such a scheme has only been presented recently in [5].

We adapt this idea to the MPLPP-VT and refer to this algorithm as logic-based decomposition with constraint propagation (LbDwCP). When constraints (5) are eliminated, the remaining problem is called the line planning subproblem (LPsP). The LPsP is solved to optimality. Given the line selection and service level decisions from the optimal LPsP solution, we check if the eliminated constraints associated with transfer of vehicles are satisfied; it is called the vehicle transfer feasibility problem (VTfP). If all constraints are satisfied, the solution of LPsP is also optimal for MPLPP-VT; otherwise, LPsP is extended with the selected violated constraints of VTfP and resolved to optimality. Figure 1 illustrates the mechanics of the algorithm.

4 Computational Results

We use the real PTN data from [6]; it includes the Istanbul Metrobus as the smaller problem with 44 stations and 9 lines (with three variants of the demand data), the Athens Metro as the medium-size problem with 51 stations and 59 lines and the Quito Trolebus as the
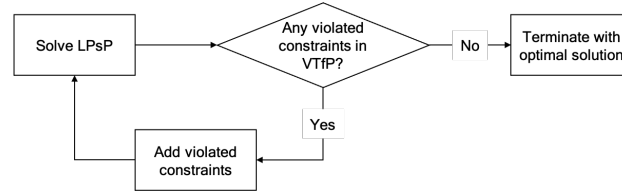
Figure 1 Average utilization and distribution of inbound truck sizes.

large-scale problem with 278 stations and 318 lines (Quito-318). We also generate a smaller
version of the Quito Trolebus problem with the same network but only 122 lines (Quito-122)
by eliminating some of the lines whose paths are already included in longer lines. We compare
the performance of four alternative solution methods with the following settings:

- The commercial solver Gurobi is run on its default integer programming solver settings
  with a CPU time limit of 86400 seconds (1 day) for Quito-318.
- The local branching algorithm is limited with 20, 420, 1800 and 3600 seconds to solve a
  node problem and 60, 3600, 18000, and 86400 seconds for the total CPU time.
- Although the LbDwCP algorithm is designed to solve the LPsP problem to optimality in
  each iteration, the optimal LPsP solution cannot be found for the Quito-318 instance
  even at the first iteration. Therefore, a time limit of 3600 seconds is set to solve the LPsP
  in each iteration, and the best feasible solution found within this limit is used to check
  for the violated constraints.
- We also use Gurobi's built-in lazy constraints functionality as a benchmark approach for
  the LbDwCP algorithm; the violated constraints for every new integer incumbent solution
  are added within the branch-and-bound procedure employing the callback function. A
  CPU time limit of 86400 seconds (1 day) is set for Quito-318.

The results are shown in Table 1 where the Cost column shows the best feasible solution
found while the Time columns shows the CPU time in seconds.

Table 1 Performance of alternative solution approaches.

	Gurobi		Local branching		LbDwCP		Lazy constraints	
Instance	Cost	Time	Cost	Time	Cost	Time	Cost	Time
Istanbul-1	61334.60	<1	61334.60	2	61334.60	6	61334.60	<1
Istanbul-2	48807.00	<1	48807.00	<1	48807.00	5	48807.00	1
Istanbul-3	35377.80	<1	35377.80	<1	35377.80	6	35377.80	1
Athens	68030.58	2070	68030.58	1050	68030.58	417	68030.58	286
Quito-122	21690.17	59134	21691.65	18000	21690.17	9509	21690.17	10512
Quito-318	21611.21	86400	21618.39	86400	21569.29	86400	21543.51	86400

The results with Istanbul instances do not help to distinguish between the alternative
approaches since the corresponding problems are already small enough to be solved to
optimality in less than 1 second while we verify that even the heuristic local branching may
reach optimality. Looking at the results for Athens and Quito-122, we observe that both the
LbDwCP algorithm and using the lazy constraints with Gurobi improve the performance of the
commercial solver significantly. The local branching heuristic also provides quite satisfactory
performance as it finds the optimal solution for Athens and almost optimal solutions for
Quito-122 within the CPU time limit of 18000 seconds, one third of the CPU time required
to find the optimal solution. With the largest instance, Quito-318, the solver terminates

with an optimality gap of 4.39%. Given that the local branching finds solutions almost as good as the solver, and both LbDwCP algorithm and the lazy constraints implementation find even better solutions all within the same CPU time limit, it seems plausible to employ either the LbDwCP or the lazy constraints implementation both of which use constraint propagation also for large-scale instances. We conduct further experiments on instances of the same problem set with different demand patterns and varying the problem parameters such as the capacity of the vehicles, the size of the fleet and line service capacities.

5   Conclusion and Outlook

The multi-period version of the line planning problem in public transportation targets a more demand-responsive underlying line plan considering the sufficiency and timeliness of services on the PTN. We follow the footsteps of the developments in [6]; we present and discuss computational results for solution approaches that can be considered as alternatives to solving the problem directly with commercial solvers. Results show that it is still challenging to obtain optimal solutions for very-large instances but good-quality solutions can be obtained within reasonable time.

Further and ongoing research focuses on two challenges. First, solving LPsP to optimality requires more effort . Secondly, the accuracy of the multi-period approach can be further improved by avoiding approximations due to time-discretization.

References

1   Stefan Bunte and Natalia Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4):299–317, 2009.

2   George B. Dantzig, Delbert R. Fulkerson, and Selmer M. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

3   Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1):23–47, 2003.

4   Johann Hartleb and Marie Schmidt. A rolling horizon heuristic with optimality guarantee for an on-demand vehicle scheduling problem. In Dennis Huisman and Christos D. Zaroliagis, editors, *20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020)*, volume 85 of *OpenAccess Series in Informatics (OASIcs)*, pages 15:1–15:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

5   Ulrich Pferschy and Rostislav Staněk. Generating subtour elimination constraints for the TSP from pure integer solutions. *Central European Journal of Operations Research*, 25:231–260, 2017.

6   Güvenç Şahin, Amin Ahmadi Digehsara, Ralf Borndörfer, and Thomas Schlechte. Multi-period line planning with resource transfers. *Transportation Research Part C: Emerging Technologies*, 119:102726, 2020.

7   Anita Schöbel. An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation. *Transportation Research Part C: Emerging Technologies*, 74:348–365, 2017.

8   Linus Schrage. *A Guide to Optimization-Based Multiperiod Planning*, chapter 3, pages 50–63. INFORMS, 2018.

9   Luis M. Torres, Ramiro Torres, Ralf Borndörfer, and Marc E. Pfetsch. Line planning on tree networks with applications to the quito trolebús system. *International Transactions in Operational Research*, 18(4):455–472, 2011.

An Integrated Model for Rapid and Slow Transit Network Design

Natividad González-Blanco ✉ 📵
Department of Applied Mathematics II, University of Sevilla, Spain
IMUS, Sevilla, Spain

Antonio J. Lozano ✉ 📵
Department of Integrated Sciences, University of Huelva, Spain

Vladimir Marianov ✉ 📵
Pontificia Universidad Católica de Chile, Santiago, Chile
Complex Engineering Systems Institute (ISCI), Santiago, Chile

Juan A. Mesa ✉ 📵
Department of Applied Mathematics II, University of Sevilla, Spain
IMUS, Sevilla, Spain

Abstract

Usually, when a rapid transit line is planned a less efficient system already partially covers the demand of the new line. Thus, when the rapid transit starts its regular services, the slow mode (e.g. bus lines) have to be cancelled or their routes modified. Usually this process is planned according to a sequential way. Firstly, the rapid transit line is designed taking into account private and public flows, and possibly surveys on mobility in order to predict the future utilization of the new infrastructure and/or other criteria. Then, in a second stage, the bus route network is redesigned. However, this sequential process can lead to a suboptimal solution, for which reason in this paper a cooperative model for rapid and slow transit network design is studied. The aim is to design simultaneously both networks and the objective is to maximize the number of passengers captured by both public modes against the private mode. We present a mathematical programming formulation and solve the problem by an improved Benders decomposition approach.

1 Location of a rapid transit line along with improving the feeder bus system: definitions

In this section we assume that changes in the bus routes can be done. Rerouting bus lines is very common when a rapid transit line starts functioning. During the last five years more than 80 new metro lines have been added to metro networks around the world, and 27 new metro systems have been inaugurated. Therefore, more than one hundred new lines have become operating. Many other existing lines have been extended or upgraded. Moreover, new modern trams, train-trams, and commuter lines have also started their operation. In almost all the cases, bus lines were (partially) doing the service before, and when a rapid transit line is put in service bus routes could become totally or partially useless. One typical example is the adaptation of the Bus Rapid Transit TranSantiago when Metro Line 7 will

start operation. Another example is bus line 3 of TUSSAM (Urban Transport of Seville) with planned Line 3 of Metro de Sevilla. Usually, the metro planning projects do not take into account the bus system because often they depend on a different agency, and after the introduction of the rapid transit service the bus system is reorganized. However, this procedure could lead to suboptimal solutions. The feeder buses planning problem has been researched to some extent ([3]) and models and algorithms for the Rapid transit network design problem have been recently revised ([4]), but as far as the authors are aware for the cooperative slow and rapid transit network design problem, no research has been done. For this purpose, in this section, an integer mathematical programming program is presented.

1.1   Data

In order to describe the problem we need to define the following elements.

1. We consider the network $\mathcal{N} = (N, E)$ used by the private mode, where N and E is the set of nodes and edges. For homogeneity purposes the rapid transit line R and the slow line S will be selected from this network.
2. The network (N_R, E_R) is the subgraph of $\mathcal{N}$ where the rapid transit line can be selected, thus $N_R \subset N$ and $E_R \subset E$.
3. The network (N_S, E_S) is the subgraph of $\mathcal{N}$ where the slow line S can be selected, thus $N_S \subset N$ and $E_S \subset E$.
4. For the rapid transit line R, there exists a maximum number of edges E^R_{max} to build. For the slow line S, bounds E^S_{max} and E^S_{id} are given to limit the number of edges to build and the minimum number of edges that must be coincident between the old and the modified line S (i.e. the number of edges not relocated). For that, vector v^S_e, $e \in E_S$ denotes the current path of the slow line S.
5. For each $e = \{k, l\} \in E$, we define two arcs: $a = (k, l)$ and $\hat{a} = (k, l)$. The resulting set of arcs is denoted by A. With respect to each mode of transport we refer the set of arcs by A_R and A_S, respectively. We use notation $\delta^+_w(k)$ ($\delta^-_w(k)$ respectively) to denote the set of arcs going out (in respectively) of node $k \in N_R$. In the same way, we use notation $\gamma^+_w(k)$ ($\gamma^-_w(k)$ respectively) to denote the set of arcs going out (in respectively) of node $k \in N_S$.
6. For each mode of transport, we assume that there is a set of possible starting points, O_R and O_S, of the lines. In the same way, sets containing possible end points D_R and D_S.
7. The set of demands W is a subset of $N \times N$. The mobility pattern is given by a matrix $G = (g^w)$, where $g^w, w = (w^s, w^t)$, denotes the number of passengers going from w^s to w^t, $(w^s, w^t) \in W$. The fixed cost of going from node w^s to node w^t using the private network is denoted by u^w_{priv}.
8. The set of possible transfer nodes is denoted by $N_{trans} = N_R \cap N_S$.
9. Other costs are those of traversing arc a in the rapid and slow mode, t^R_a and t^S_a, respectively. The transfer cost at station k from S to R and from R to S are t^{SR}_k and t^{RS}_k, respectively. The dwell time costs (stops) are t^R_{stop} are t^S_{stop}, which will be assumed independent from nodes. The waiting time at stations/stops, t_{wait}, is usually set as a half of the headway.

1.2   Variables

1. $x^R_e = 1$ if edge $e = \{k, l\} \in E_R$ is included in the rapid public line R; 0 otherwise. Analogously, $x^S_e = 1$ if edge $e = \{k, l\} \in E_S$ is included in the slow public line S; 0 otherwise.
2. $y^R_i = 1$ if node $i \in N$ is included in the alignment of the rapid system R, but it does not stop on it; 0 otherwise.

3. $z_i^R = 1$ if R stops at i; 0 otherwise. Analogously, $z_k^S = 1$ if k is a stop of mode S; 0 otherwise.

4. $f_a^{wR} = 1$ if demand w traverses arc $a \in A_R$, 0 otherwise.

5. $f_a^{wS} = 1$ if demand w traverses arc $a \in A_S$; 0 otherwise.

6. $f_k^{wSR} = 1$ if demand w transfers from S to R at node $k \in N_{trans}$; 0 if there is no transfer of w from S to R at k.

7. $f_k^{wRS} = 1$ if demand w transfers from R to S at node $k \in N_{trans}$; 0 if there is no transfer of w from S to R at k.

8. $f^w = 1$, if demand w uses S, R, or the combined modes RS and SR.

1.3 Objective and constraints

The aim of the problem in to design line R and to re-design line S so that the trip coverage of both public modes would be maximized, thus minimizing the private traffic:

$$\max_{\boldsymbol{x,y,z,f}} \sum_{w \in W} g^w f^w \tag{1}$$

- Budget constraints: Impose upper bounds on the budget and/or on the number of edges for both modes of transport.

$$\sum_{e \in E_R} x_e^R \leq E_{max}^R, \tag{2}$$

$$\sum_{e \in E_S} x_e^S \leq E_{max}^S. \tag{3}$$

- Design constraints: Among them are the following: If an edge is constructed for the rapid system its endpoints are either a station or a non-stop node. At least one node has to be selected from the sets of origins and destinations of the rapid and slow lines. The lines must be chain graphs. If an edge is selected to be in the rapid or slow line its endpoints are nodes of the line.

$$x_e^R \leq z_i^R + y_i^R, \quad e \in E_R, i \in e, \tag{4}$$

$$\sum_{o \in O_R} \sum_{e \in \delta(o)} x_e^R = 1, \tag{5}$$

$$\sum_{d \in D_R} \sum_{e \in \delta(d)} x_e^R = 1, \tag{6}$$

$$\sum_{o \in O_R} z_o^R = 1, \tag{7}$$

$$\sum_{d \in D_R} z_d^R = 1, \tag{8}$$

$$z_i^R + y_i^R \leq 1, \quad i \in N_R, \tag{9}$$

$$\sum_{e \in E_R} x_e^R + 1 = \sum_{i \in N_R} (y_i^R + z_i^R), \tag{10}$$

$$\sum_{e \in \delta(k)} x_e^R \leq 2(z_k^R + y_k^R), \quad k \in N_R \setminus (O_R \cup D_R), \tag{11}$$

$$x_e^S \leq z_i^S, \quad e \in E_S, i \in e, \tag{12}$$

$$\sum_{o \in O_S} z_o^S = 1, \tag{13}$$

$$\sum_{d \in D_S} z_d^S = 1, \tag{14}$$

$$\sum_{e \in E_S} v_e^S x_e^S \geq E_{id}^S, \tag{15}$$

$$\sum_{e \in E_S} x_e^S + 1 = \sum_{i \in N_S} z_i^S, \tag{16}$$

$$\sum_{e \in \gamma(k)} x_e^S \leq 2z_k^S, \quad k \in N_S \setminus (O_S \cup D_S), \tag{17}$$

$$f^w \leq 1 - y_{w^s}^R, \quad \text{if } w^s \in N_R, \tag{18}$$

$$f^w \leq 1 - y_{w^t}^R, \quad \text{if } w^t \in N_R, \tag{19}$$

- Relation between variables f^w, z_k^R and z_k^S.

$$f^w \leq \begin{cases} z_k^R + z_k^S, & \text{if } k \in N_R \cap N_S, \\ z_k^R, & \text{if } k \in N_R \text{ and } k \notin N_S, \quad w \in W, k \in \{w^s, w^t\}, \\ z_k^S, & \text{if } k \in N_S \text{ and } k \notin N_R, \end{cases} \tag{20}$$

- Flow conservation constraints. Flows have to be maintained either by slow or rapid modes.

$$\sum_{a \in \delta_w^+(k)} f_a^{wR} + \sum_{a \in \gamma_w^+(k)} f_a^{wS} - \left(\sum_{a \in \delta_w^-(k)} f_a^{wR} + \sum_{a \in \gamma_w^-(k)} f_a^{wS} \right) =$$

$$\begin{cases} f^w, & \text{if } k = w^s, \\ -f^w, & \text{if } k = w^t, \qquad w \in W, k \in N_R \cup N_S, \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

- Transfer constraints. Only one transfer from slow to rapid mode and from rapid to slow is allowed.

$$\sum_{k \in N_{trans} \setminus \{w^s, w^t\}} f_k^{wSR} \leq 1, \quad w \in W, \tag{22}$$

$$\sum_{k \in N_{trans} \setminus \{w^s, w^t\}} f_k^{wRS} \leq 1, \quad w \in W, \tag{23}$$

$$\sum_{a \in \delta_w^-(k)} f_a^{wR} + f_k^{wSR} - \left(\sum_{a \in \delta_w^+(k)} f_a^{wR} + f_k^{wRS} \right) = 0, \quad w \in W, k \in N_{trans} \setminus \{w^s, w^t\}, \tag{24}$$

$$\sum_{a \in \gamma_w^-(k)} f_a^{wS} + f_k^{wRS} - \left(\sum_{a \in \gamma_w^+(k)} f_a^{wS} + f_k^{wSR} \right) = 0, \quad w \in W, k \in N_{trans} \setminus \{w^s, w^t\}, \tag{25}$$

- Location-allocation constraints. Link design and flow variables.

$$f_a^{wR} + f_{\hat{a}}^{wR} \leq x_e^R, \quad w \in W, e = \{i, j\} \in E_R : a = (i, j), \hat{a} = (j, i), \tag{26}$$

$$f_a^{wS} + f_{\hat{a}}^{wS} \leq x_e^S, \quad w \in W, e = \{i, j\} \in E_S : a = (i, j), \hat{a} = (j, i), \tag{27}$$

- Alignment stop constraints. Stated conditions on the construction of a node regarding in- or out-flows.

$$f_k^{wSR} + f_k^{wRS} \leq z_k^R, \quad w \in W, k \in N_{trans} \setminus \{w^s, w^t\}, \tag{28}$$

$$f_k^{wSR} + f_k^{wRS} \leq z_k^S, \quad w \in W, k \in N_{trans} \setminus \{w^s, w^t\}, \tag{29}$$

$$\sum_{a \in \delta^+(w^s)} f_a^{wR} \leq z_{w^s}^R, \quad w = (w^s, w^t) \in W, \text{ if } w^s \in N_R, \tag{30}$$

$$\sum_{a \in \delta^-(w^t)} f_a^{wR} \leq z_{w^t}^R, \quad w = (w^s, w^t) \in W, \text{ if } w^t \in N_R, \tag{31}$$

- Mode choice. Assign the demand either to the public modes or to the private one depending of the total time of the trip.

$$\sum_{a \in A_R} t_a^R f_a^{wR} + \sum_{a \in A_S} t_a^S f_a^{wS} + \sum_{k \in N_{trans}} t_k^{RS} f_k^{wRS} + \sum_{k \in N_{trans}} t_k^{SR} f_k^{wSR} +$$
$$+ t_{stop}^R \sum_{k \in N^R} z_k^R \sum_{a \in \delta^+(k)} f_a^{wR} + t_{stop}^S \sum_{k \in N^R} z_k^S \sum_{a \in \gamma^+(k)} f_a^{wS} + f^w \left(t_{wait}^R - \frac{1}{2} t_{stop}^R \right) \leq u_{priv}^w, \tag{32}$$

- Binary constraints. All the variables are assumed to be in $\{0, 1\}$.

$$x_e^R, x_e^S, y_k^R, z_k^R, z_k^S, f_a^{wR}, f_a^{wS}, f_k^{wRS}, f_k^{wSR}, f^w \in \{0, 1\}. \tag{33}$$

2 Solving the problem

Since the problem is NP-hard, we use a Benders decomposition approach to exactly solve it (see [1]). With this exact procedure we pretend to improve the computational time. Actually, our Benders implementation is used as a sub-routine in a Branch-and-Benders-cut scheme. We use the ideas exposed in [2] in order to get stronger cuts than the standard ones.

Our computational experiments were performed on a computer equipped with a Intel Core i5-7300 CPU processor, with 2.50 gigahertz 4-core, and 16 gigabytes of RAM memory. The operating system is 64-bit Windows 10. Codes were implemented in Python 3.8. These experiments have been carried out through `CPLEX 12.10` solver, named `CPLEX`, using its Python interface. `CPLEX` parameters were set to their default values and the model was optimized in a single threaded mode.

The tested instance is composed by 64 nodes and 128 edges. The W set is formed by all possible O/D pairs. The new slow line S must coincide with the old one on at least 3 edges and can consist of a maximum of 6 edges in total. With respect to the rapid transit line, it must be composed by 9 nodes and 9 edges at most.

After two hours, the optimal solution is obtained using the implemented routine Branch-and-Benders-cut scheme ad-hoc to the problem, an hour and a half earlier than if we directly solve the MIP formulation with `CPLEX`. It should be noted that the Benders decomposition algorithm existing in `CPLEX` is not competitive with the two named methods.

This integrated model results in an optimum design with respect to the maximization of the coverage for the whole public transport (composed by the rapid and slow modes). Locating each line independently without taking into account the influence that may exist between them, or even locate them in a sequential way can result in suboptimal solutions. The sequential design method is the one used in practice. That is, currently the rapid transit line R is located first and then the slow line S is relocated. For example, considering the tested instance, the optimal objective value for the integrated model is 35.8% bigger than that of the independently localization.

────── **References** ──────

1   J Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

2   M Conforti and L. A Wolsey. "Facet" separation with one linear program. *Mathematical Programming*, 178(1):361–380, 2019.

3   Lian-bo Deng, Wei Gao, Wen-liang Zhou, and Tian-zhen Lai. Optimal design of feeder-bus network related to urban rail line based on transfer system. *Procedia-Social and Behavioral Sciences*, 96:2383–2394, 2013.

4   Gilbert Laporte and Juan A Mesa. The design of rapid transit networks. In G. Laporte, S. Nickel, and F. Saldanha da Gama, editors, *Location science*, chapter 24, pages 685–701. Springer, 2020.

A Column Generation-Based Heuristic for the Line Planning Problem with Service Levels

Hector Gatt ✉
IMT Atlantique, LS2N, Nantes France
Lumiplan, Saint-Herblain, France

Jean-Marie Freche ✉
Lumiplan, Saint-Herblain, France

Fabien Lehuédé ✉
IMT Atlantique, LS2N, Nantes, France

Thomas G. Yeung ✉
IMT Atlantique, LS2N, Nantes, France

─── **Abstract** ───

This paper addresses the line planning problem by the combination of existing models reinforced with realistic characteristics like lines frequencies intervals or maximum number of lines, useful for public transportation companies. The problem is solved by an innovative, easily implementable, heuristic combining column generation and elementary column enumeration methods. In this paper, the operator's exploitation costs are minimized while respecting new quality of service parameters addressed to passengers. Furthermore, a case study based on a real network is performed and described in this paper to prove the efficiency of our method.

1   Introduction

Offering a high-quality bus network at a reasonable cost is the main objective of many public transportation companies. Known as the *Line Planning Problem*, this problem is included at a strategic level in a global planning process of a public transportation system first introduced by Ceder and Wilson in 1986 [6]. It consists of designing a set of lines and frequencies such that a given demand can be transported. The two major challenges of this problem are cost and quality of service. While the capital and operating expense of the network is the main challenge of the operator, the passenger seeks a fast, reliable, and convenient network. Hence, the existent line planning models can be classified in two types: with cost-oriented or with passenger oriented objective functions. For each of these two objectives, many mathematical approaches have been developed, see Schöbel [8] for an overview.

According to Schöbel [8] and Karbstein [7], there are two approaches to consider passengers choices in the line planning problem: the fixed passenger routing approach or the integrated passenger routing approach. The former assumes that the number of passengers traveling along each path is known, while the second approach assumes the passengers' path choice depends on the lines proposed. The first approach is nowadays somewhat abandoned in favor of the second one. A change & go graph, as proposed by Schöbel and Scholl in 2005 [9] can be a method to deal with the integrated passenger routing approach for solving the Line Planning problem. This method consists of duplicating each node each time a transfer from one line to another on this node is done, and to connect them with a new edge. The

advantage of this method is that it models transfers with great accuracy. However, the size of the graph grows exponentially with the size of the network, which prevents the approach from being used on real-world cases. Another method to deal with the integrated passenger routing approach can be to use a bi-level method. This method separates the problem into two levels, the upper one to determine the lines and their frequencies and the lower one to define the passengers choices on the network. Szeto and Jiang [12] propose a mixed-integer, non-linear, bi-level formulation for the Line Planning problem. Their upper level aims to define a line set minimizing the number of transfers with respect to constraints on fleet size or lines frequencies, and their lower level aims to solve an assignment model with respect to bus capacity. In this paper, we focus on the cost-oriented problem associated with the integrated passenger routing approach to better match the motivating companies expectations. In our case, these are to offer a network with a guaranteed service level at minimum cost. Inspired by the direct connection approach formulated by Borndörfer and Karbstein [5], we focus on network creation with novel service level constraints and solved by an innovative heuristic combining column generation and elementary column enumeration methods.

2   The Line Planning Problem with Service Levels

2.1   Problem Description

The Line Planning Problem (LPP) seeks to define a system of lines with associated operation frequencies satisfying passengers demands. In our case, we decide to focus on the cost oriented Line Planning Problem in order to minimize the operation costs. Operation costs can be characterised as the sum of the products between the total lines lengths and a kilometric cost c. The total length of a line $l \in \mathcal{L}$ ($\mathcal{L}$ being a set of bus lines) is defined by the product of its outward and return length d_l and its frequency f_l (number of bus passages on the line l during the time period considered). Therefore, we can define the cost of a line plan $(\mathcal{L}, f)$ as $c(\mathcal{L}, f) = \sum\limits_{l \in \mathcal{L}} d_l \times f_l \times c$.

We consider a public transportation network composed of main stations linked together by edges that have travel times, distances, and maximum capacities. The maximum capacity of an edge refers to the maximum number of buses allowed to circulate on it. It is used to avoid the saturation of the network on an edge. The public transportation network is represented as an undirected graph $G = (V, E)$ where V are the main stations and E the edges. Infrastructure parameters (number of buses available, bus capacity, kilometric cost) and user-specified parameters such as the lines authorized frequency, minimum and maximum lengths, maximum total number are also taken into account. Finally, passengers demands are considered, represented as an origin-destination (OD) matrix where each $(i - j)$ coefficient represents the number of passengers desiring to go from point i to j during a given time period.

Our problem being an extension of the Line Planning Problem that incorporates service levels, we decide to name it the Line Planing Problem with Service Levels (LPP-SL).

2.2   Passenger Service Levels

In 1995, Baaj et al. [1] introduced the notion of a time deviation threshold for passengers compared to their shortest path. More recently, in 2018, Suman et al. [11] analyze the perception of potential users about existing bus services in Delhi, India, and conclude most of people avoid using buses due to overloading, excessive travel time compared with a personal

vehicle, the need to make a transfer, and lack of punctuality. Suman reuses his study to propose with Bolia [10] in 2019 a model maximizing the directness of their network, defined as total passenger kilometers without transfers.

Borndörfer et al. [4] (2007) offer a method to generate bus lines and passenger paths. However, they do not differentiate direct and non-direct paths. Borndörfer and Karbstein [5] (2012) introduce the direct connection approach, that distinguishes passenger paths without transfers and passenger paths with one or more transfers. This method loses accuracy compared to the change-and-go method [9] by not determining the exact number of transfers, but conversely this method can be used on large graphs and thus exploited on real size networks. With this approach, Borndörfer and Karbstein propose a method to generate direct and non-direct passenger paths. However they do not generate bus lines, only selecting them from a pre-defined pool. Finally, Bertsimas et al. [3] (2021) offer a method to model direct and non-direct passenger paths, while generating bus lines. However, they only constrain exploitation costs to a maximum threshold and choose to maximize the demand that is served by the lines chosen instead. Furthermore, as passengers transfers are modeled by analyzing the exact transfer node and line used on each part of the path, they chose to limit the passenger paths to a maximum of one transfer to avoid the saturation of their model.

Inspired by these articles, we propose two conditions to model the quality of service of a network: (1) A passenger must have a path allowing him to go from his origin to his destination by not deviating by a pre-defined threshold from its shortest path time (2) A minimum percentage of passengers must be able to go from their origin to final destination without making a transfer. These two conditions named *maximum SPT deviation* and *minimum direct percentage* are respectively defined by the two quality of service parameters α and β, $\alpha \geq 1$ and $\beta \in [0,1]$. The first condition can thus be defined as $t_p \leq \alpha \times SPT_{st} \quad \forall p \in P_{st}$; where $p \in P_{st}$ symbolizes the paths to go from s to t, t_p the time of the path p, and SPT_{st} the shortest time to go from s to t based on the arcs in the current network.

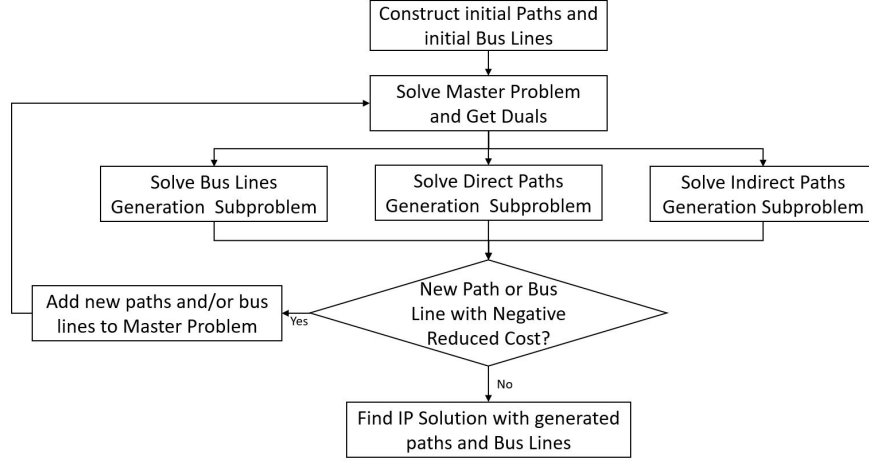## 3   Column Generation-based Heuristic

3.1   Modeling Approach

To integrate a maximum SPT deviation constraint, we adopt a path-based formulation to model passenger flows. Furthermore, to integrate a minimum direct percentage in a realistic size bus network, we extend the approach of Borndörfer and Karbstein [5] by dividing traveler paths in two pools, one for direct and one for non-direct, and using a column generation approach for the design of bus lines. Our Line Planning Problem is thus composed of four decision variables: a binary variable z_l that represents the opening of a bus line, a positive integer variable f_l that represents the bus line frequency, and two continuous variables, y_p^{0+} and $y_{p'}^1$, that track the number of passengers respectively traveling on direct path p and non-direct path p'. Hence, the integration of a minimum direct percentage leads to a more realistic network than Borndörfer et al. [4] where an uncontrolled number of passengers can be linked with non-direct paths.

3.2   The Heuristic

The number of potentially attractive bus lines and passenger paths increase at an exponential rate as the network size increases. Complete enumeration is not possible for large instances and commercial solvers also become quickly overwhelmed. Hence, using a column generation method can be a good choice to solve this problem by selecting the pool of best bus lines and passenger paths prior to using a commercial solver to solve the integer problem.

We propose a heuristic which consists of a column generation step, followed by a column enumeration method based on [2]. The general process of this heuristic is presented in Figure 1.

■ **Figure 1** General overview of column generation heuristic for the considered Line Planning Problem.

By deciding to generate direct paths, non-direct paths and lines during the column generation process and not only direct and non-direct paths, we avoid needing to compute a line pool beforehand as Borndörfer and Karbstein do [5]. This yields a final line pool of smaller size and thus a smaller resolution time of the Line Planning Problem.

In the first step, some initial sets of bus lines and passenger paths are generated while the LPP-SL has no solution. This gives rise to a restricted formulation of the LPP-SL (denoted rLPP-SL). The rLPP-SL is linearized using the variable substitution of [4] to model the use of a line and its frequency as a single variable. This forms the column generation master problem. Bus lines, direct paths and non-direct paths are then generated in the column generation step with two dedicated labelling algorithms, in which every label dominated by another label during the labeling algorithms execution is detected and removed, leading to time savings. This process terminates when the optimum of the Master problem has been found, providing a lower bound to the LPP-SL. All generated columns are then used in the rLPP-SL to solve the MIP with a commercial solver, thus providing an upper bound to the problem.

In this column generation heuristic, columns leading to an optimal solution of the LPP-SL may not be generated because column generation is not used at each node of the branch-and-bound algorithm that solves the MIP. To consolidate the sets of traveler paths and bus lines, we add an elementary column enumeration step as introduced by Baldacci et al. [2] for the VRP. For a given upper bound z_{IP} and a lower bound z_{MP} of the problem, this technique consists of enumerating all possible lines and paths with reduced cost c_l and c_p such that $\bar{c}_l \leq z_{IP} - z_{MP}$ and $\bar{c}_p \leq z_{IP} - z_{MP}$. All these line and path variables are then added to the rLPP-SL, which is solved with the MIP solver, using the previous upper bound as an initial solution. This yields the final line plan composed of lines and frequencies.

4 Numerical Results

A case study has been chosen on the city of Poitiers, France with over 200 000 residents. Data is obtained from the collaboration between the local public transport operator and Lumiplan, a private company that offers services to optimize public transportation networks. To carry out this study, a graph composed of 78 nodes and 106 edges has been defined, based on the existing network. The network and the graph are presented in Figure 2.
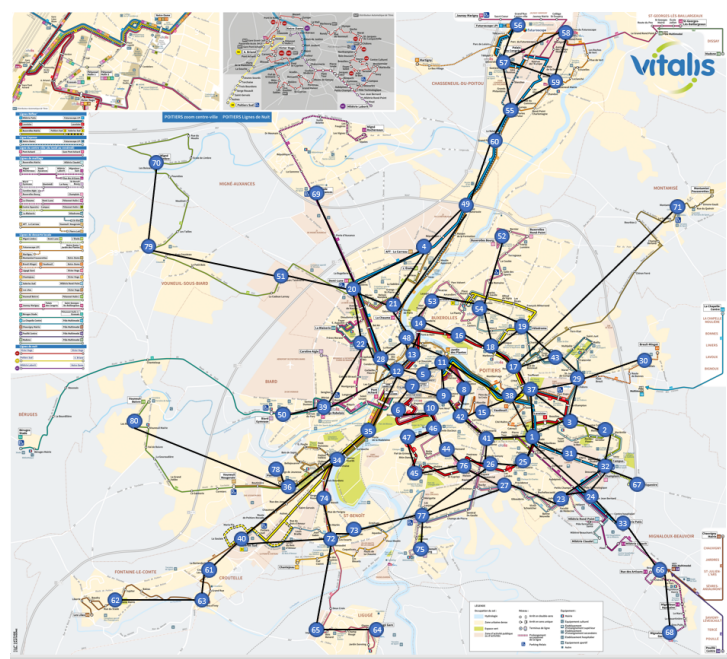

Figure 2 Poitiers Bus Network – Graph Model.

This case study is separated in two experiments, the first aims to redesign the network by minimizing cost with the same quality of services parameters for users. The second experiment analyzes the sensitivity of various parameters such as the service level, the maximum number of lines, or the line interval frequency on the cost of the network. The following table presents objective values details on the Poitiers instance for the first experiment. After both the Column-Generation and Column-Enumeration steps, the LPP-SL is solved using CPLEX with a computation time limit of 10 hours.

Table 1 Characteristics of current and redesigned line plans for Poitiers. The columns list objective value and service level parameters (maximum SPT deviation and minimum direct percentage).

Network	Objective Value	Max. SPT Dev.	Min. Dir. Perc.
Poitiers - Current Network	18 300		
Poitiers - Redesigned Network	13 971	1.75	0.7

We calculated the maximum SPT deviation and minimum direct percentage values (1) for the current network and we used the same values as constraints for optimizing the redesign, which our approach was able to reduce the objective value by more than 23%. Concerning the second experiment, we were able to make some observations on the results obtained. For the same service level, the redesigned network with 22 lines had an objective function

value 2% lower than one with only 21 lines. Furthermore, it was noticed that the number of non-direct paths generated with our heuristic increases as the Maximum Time Deviation parameter increases. Indeed, an increase of this parameter of 0.25 units lead to an increase of the generated non-direct paths of 100%. On the contrary, the number of generated lines decreased as this parameter increased.

5 Conclusion

We propose a heuristic for solving the line planning problem. In this formulation, we set the operator cost as a minimization objective while specifying quality of service parameters for the passengers. This heuristic, composed of column generation and column enumeration methods aims thus to minimize operator exploitation costs, depending on the lines defined and their associated frequencies. The quality of service has been defined according to two parameters, the first referring to the travel times and the second one referring to the number of direct passengers. Computational results for an instance based on a real city were obtained and showed the relevance of our heuristic for public transport companies to define a high quality network at a reasonable cost. Work will be pursued in the coming months on existing instances to evaluate the efficiency of our method on optimal known solution. Furthermore, a line typology study will be integrated into our heuristic to evaluate the conformity of the generated lines.

We are currently continuing this work at a more tactical level to define the line frequencies at different, smaller time periods. To this aim, a transit assignment model will need to be defined to determine which lines are attractive for every passenger, followed by a frequency setting model to adjust the line frequencies at a minimized cost while respecting quality of service parameters.

── **References** ──

1 M. H. Baaj and H. S. Mahmassani. Hybrid route generation heuristic algorithm for the design of transit networks. *Transportation Research Part C: Emerging Technologies*, 1995.

2 R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vrp based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 2008.

3 D. Bertsimas, Y. S. Ng, and J. Yan. Data-Driven Transit Network Design at Scale. *Operations Research*, 2021.

4 R. Borndörfer, M. Grötschel, and M. E. Pfetsch. A Column-Generation Approach to Line Planning in Public Transport. *Transportation Science*, 2007.

5 R. Borndörfer and M. Karbstein. A Direct Connection Approach to Integrated Line Planning and Passenger Routing. *Discrete Optimization*, 2012.

6 A. Ceder and N. H. M. Wilson. Bus network design. *Transportation Research Part B: Methodological*, 1986.

7 M. Karbstein. *Line Planning and Connectivity*. PhD thesis, Mathematik und Naturwissenschaften der Technischen Universität Berlin, 2013.

8 A. Schöbel. Line planning in public transportation: models and methods. *Springer*, 2012.

9 A. Schöbel and S. Scholl. Line planning with minimal traveling time. *ATMOS*, 2005.

10 H. K. Suman and N. B. Bolia. Improvement in direct bus services through route planning. *Transport Policy*, 2019.

11 H. K. Suman, N. B. Bolia, and G. Tiwari. Perception of potential bus users and impact of feasible interventions to improve quality of bus services in Delhi. *Case Studies on Transport Policy*, 2018.

12 W.Y. Szeto and Y. Jiang. Transit route and frequency design: Bi-level modeling and hybrid artificial bee colony algorithm approach. *Transportation Research Part B*, 2014.