Diagnosing Non-Intermittent Anomalies in Reinforcement Learning Policy Executions

Avraham Natan
Ben-Gurion University of the Negev, Beersheba, Israel

Roni Stern
Ben-Gurion University of the Negev, Beersheba, Israel

Meir Kalech
Ben-Gurion University of the Negev, Beersheba, Israel

Abstract

Reinforcement learning (RL) algorithms output policies specifying which action an agent should take in a given state. However, faults can sometimes arise during policy execution due to internal faults in the agent. As a result, actions may have unexpected effects. In this work, we aim to diagnose such faults and infer their root cause. We consider two types of diagnosis problems. In the first, which we call RLDXw, we assume we only know what a normal execution looks like. In the second, called RLDXs, we assume we have models for the faulty behavior of a component, which we call fault modes. The solution to RLDXw is a time step at which a fault occurred for the first time. The solution to RLDXs is more informative, represented as a fault mode according to which the RL task was executed. Solving those problems is useful in practice to facilitate efficient repair of faulty agents, since it can focus the repair efforts on specific actions. We formally define RLDXw and RLDXs and design two algorithms called WFMa and SFMa for solving them. We evaluate our algorithms on a benchmark of RL domains and discuss their strengths and limitations. When the number of the observed states increases, both WFMa and SFMa report a decrease in runtime (up to significantly 6.5 times faster). Additionally, the runtime of SFMa increases linearly with the increase in candidate fault modes.

2012 ACM Subject Classification Hardware → Bug detection, localization and diagnosis; Theory of computation → Reinforcement learning

Keywords and phrases Diagnosis, Reinforcement Learning, Autonomous Systems

Digital Object Identifier 10.4230/OASIcs.DX.2024.23

Category Short Paper

Funding *Roni Stern*: Israeli Science Foundation (ISF) grant #1238/23

1 Introduction

As the use of autonomous systems increases, there is a need for approaches to address challenges in different aspects of the agent's task execution, such as planning and fault diagnosis. In this work, we ask what made an agent diverge from its expected course of action. We focus on a setting where the environment is dynamic and sometimes even unknown. In that case, the expected actions of an agent are defined by the so-called *Policy*, which is the product of *Reinforcement Learning* based planning approaches.

Reinforcement Learning (RL) is a technique for guiding an agent through a dynamic environment such that the agent maximizes the rewards it collects. To achieve this, RL models train on observation data available through simulations or actual interaction with the environment. The output of the RL model is referred to as *policy*. The policy guides the agent on what action to perform in every state until the agent's goal is reached.

Sometimes, internal faults within the agent may occur. Such faults may change the agent's ability to perform its actions normally. This, in turn, may lead to some actions having unexpected effects. When this happens, we want to understand what caused the agent to deviate from its policy, that is a *Diagnosis*. Inferring the root cause can help repair the components responsible for executing faulty actions faster. In a scenario where repair is not possible, this can improve future planning if the faulty components can be assumed faulty, therefore computing suitable policies.

Diagnosis outlines a range of AI techniques that try to understand the root causes of failures in systems. Given a system, its expected behavior, and an observation, the diagnosis problem tries to infer what caused the observation to be different than expected. Previous work asks this question in a variety of settings such as Software engineering [1], Multi-Agent Systems [10], and more, and proposes different approaches such as Spectrum-based Fault Localization [1] and Model-Based Diagnosis (MBD) [15]. One difference between works in MBD is what can be assumed about the behavior mode of faulty system components. One approach assumes no knowledge on the faulty behaviour mode and is called Weak Fault Model (WFM) [3]. The other approach assumes a number of modes in which a faulty system component can be and is called Strong Fault Model (SFM) [21].

In this work, we solve the problem of diagnosing faulty RL policy executions. We use MBD concepts to define the problem for WFM (RLDXw), where the only assumption about the actions is whether they are faulty or healthy, and for SFM (RLDXs), where we assume a number of candidate fault modes that can explain the faulty execution, and where one of them is the correct one. In addition, we assume that non-intermittent faults are present in this work. This means that if an action of some type (for example, left acceleration) is faulty, then every time this action type is executed, it will execute in a faulty manner. Moreover, we assume that faulty actions behave in a single faulty manner, i.e., if the left acceleration results in a right acceleration, then whenever a left acceleration is attempted, the result will always be a right acceleration. The contributions of this work are as follows:

1. We define formally the problems RLDXw and RLDXs.
2. We propose two algorithms, WFMa and SFMa to solve RLDXw and RLDXs, respectively.
3. We evaluate the algorithms theoretically and empirically and discuss their strengths and weaknesses.

In particular, we ask the following two questions:

Q1 How does the number of candidate fault modes impact the performance of SFMa?
Q2 How does the number of visible states affect the performance of WFMa and SFMa?

We test WFMa and SFMa on the domains *Acrobot*, *CartPole*, *MountainCar* and *Taxi*, all of which can be found in the Gymnasium website[1]. We found out that the runtime of SFMa increases linearly with the increase in the number of candidate fault modes, and that the runtime of both algorithms increases with the decrease in percentage of observed states.

2 Background and Related Work

2.1 Markov Decision Problem

A *Markov Decision Problem* (MDP) is a sequential decision-making problem in which the cost and transition functions depend only on the current state of the system and the current action [14]. Formally, an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, P_a(S, S'), R_a(S, S') \rangle$ where $\mathcal{S}$ is a set of environment and agent states, $\mathcal{A}$ is the set of actions an agent can perform,

[1] `https://gymnasium.farama.org/`

$P_a(S, S') = Pr(S_{t+1} = S'|S_t = S, a_t = a)$ is the probability of reaching state S' given that the current state at time t is S and the agent executes action a. This is also known as the transition function and $R_a(S, S')$ is the immediate reward gained by executing action a from S.

A solution to an MDP is a *policy* that guides the agent to choose actions that maximize its cumulative reward. In this work, we consider deterministic policies. This means a policy π can be defined as a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a well-studied branch of Machine Learning designed for solving MDPs and other sequential decision-making problems [22]. Some RL algorithms, such as Deep Q-Networks (DQN), learn a function that estimates the value of performing actions in states [9]. The policy is computed implicitly by choosing in every state the action that maximizes this value function. Other RL algorithms, such as Asynchronous Advantage Actor-Critic [8] and Proximal Policy Optimization (PPO) [20], learn a policy directly.

The agent applies its actions, based on the learned policy or value function, on an *environment* that defines what the outcomes of those actions are. The environment defines the set of states an agent can be in and the transition function between the states. When an agent in state S executes action a, its next state is determined by the environment, which is specified by the probability $P_a(S, S')$ that is defined for the environment.

2.3 Automated Diagnosis

Automated diagnosis (DX) delves into the question of finding the root cause of a system failure. There are many approaches to formalize a diagnosis problem, and they are all essentially similar in the way they generally define a problem – when a system is not behaving the way it is supposed to behave, we say there is a diagnosis problem [15]. Diagnosis methods take two distinct approaches when modeling the system in question. The first approach is called *Weak fault model* (WFM). In WFM, there is no assumption about what is the behaviour of faulty parts in the system [3]. The parts are either assumed normal or not normal. Although simple, this approach provides diagnoses that are not very informative, As it can not provide further information about the behaviour of faulty parts. To complement this approach, *Strong fault model* (SFM) assumes known information about the faulty operation modes of the system parts [21].

In this work, we address faulty executions of actions in RL environments. Related work focused on diagnosing plan steps that were executed abnormally [18, 7, 10] while others try to diagnose faulty plan executions that happen due to disagreements between the agents' beliefs [17, 2, 5]. Some works use a single observation [4], while others use many [23]. More recent work uses approaches from software diagnosis to diagnose multi-agent systems [11, 12]. Recent surveys survey other advancements in diagnosis [6, 16].

3 Problem Definition

In this section, we define our problem. Let Π be an MDP and π a policy created for it, e.g., using some RL algorithm. For simplicity, we assume that the transition function in Π is deterministic. This allows us to define a *simulator* for Π as a function $\chi : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ such that $\chi(S, a)$ is the state the agent is expected to reach when executing a in state S. We extend the definition of χ to address the normal execution of π for i steps starting from S, denoted as $\chi(S, \pi, i)$. This extension is defined as a function $\chi : \mathcal{S} \times \mathcal{A} \times \mathbb{N} \rightarrow \mathcal{T}$ that given a state, an action, and a number of steps returns a trajectory T, where the trajectory is an alternating sequence of actions and states, simulated by the normal execution of the policy π starting from state S.

In this work, we assume executing an action may fail, and there are several *fault modes* that specify different ways in which an action behaves when faulty.

▶ **Definition 1** (Fault Mode). *A fault mode is a function $f : \mathcal{A} \rightarrow \mathcal{A}'$ that takes an action $a \in \mathcal{A}$ and returns an action a' from a potentially different set of actions $\mathcal{A}'$. If $\forall a \in \mathcal{A}$ it holds that $f(a) = a$, we say that f is the healthy fault mode, and we denote it as h.*

In this work, we make the common single-fault assumption [19, 13], i.e., we assume that only one fault mode is present at any given time. However, a fault mode may affect multiple actions.

We extend the definition of χ to include simulating a policy together with a fault mode. That means that for a fault mode f, the simulator $\chi(S_0, \pi, i, f)$ outputs a trajectory $T = (S_0, \hat{a}_1, \hat{S}_1, ..., \hat{a}_i, \hat{S}_i)$ that corresponds to executing the policy from the state S_0 for i steps, where the faulty actions according to f fail. To support the normal definition defined previously as $\chi(S_0, \pi, i)$, we define it now as $\chi(S_0, \pi, i, h)$, where $h : \mathcal{A} \rightarrow \mathcal{A}$ is the model of the normal behavior, and is defined as $h(a) = a$.

▶ **Example 2.** Consider the *MountainCar* environment, a standard RL domain [24]. Figure 1 provides a visual representation. In this environment, a car is located between two hills. The car can only accelerate to the left, perform no action, and accelerate to the right. We denote these actions as L, N and R. The task of the car is to climb the right hill by strategically accelerating while using gravity to build its speed. Let us assume that in a normal execution,

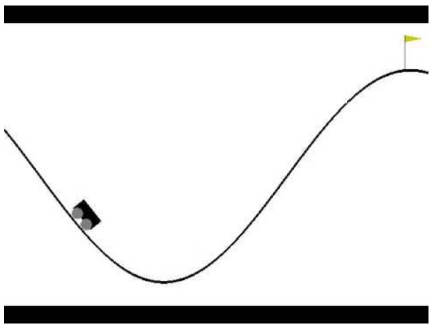

Figure 1 The MountainCar environment.

the car would reach its goal and the corresponding trajectory is $(a_1 = R, S_1, a_2 = R, S_2, a_3 = R, S_3, a_4 = N, S_4, a_5 = N, S_5, a_6 = L, S_6, a_7 = L, S_7, a_8 = L, S_8, a_9 = R, S_9, a_{10} = R, S_{10})$. In addition there are 3 candidate fault modes f_1, f_2, f_3 defined as:

- $\mathbf{f_1(L) = N}, f_1(N) = N, f_1(R) = R$
- $f_2(L) = L, f_2(N) = N, \mathbf{f_2(R) = N}$
- $\mathbf{f_3(L) = R}, f_3(N) = N, f_3(R) = R$

Assume the car is faulty and behaves according to fault mode f_1. As a result, actions a_6, a_7, and a_8 failed, and the car did not reach its goal. Also, suppose the car observed the states $\mathcal{O} = (S_0, S_5, S_7, S_{10})$. Because of the failed actions, the states $\hat{S}_7$ and $\hat{S}_{10}$ from the trajectory, returned by simulating $\chi(S_0, \pi, i, h)$, do not match the observed states S_7 and S_{10}.

In this work we aim to find the cause of the inconsistency between the expected healthy execution and the observed faulty execution of a policy π. We define this problem for WFM and for SFM separately. We begin by formally defining the problem for WFM, which we call RLDXw:

▶ **Definition 3** (RLDXw problem)**.** *An RLDXw problem is defined as a tuple $\langle \mathcal{O}, \chi, \pi \rangle$ where π is a policy, χ is a simulator and $\mathcal{O} = (S_0, S_1, \ldots, S_n)$ is an observed series of states. An RLDXw problem arises when there exists an observed state S_i s.t. $S_i \neq \hat{S}_i$ where $\hat{S}_i$ is the state at step i of the trajectory generated by $\chi(S_0, \pi, i, h)$.*

A solution to an RLDXw problem is a *diagnosis*, which in the WFM setting is a time-step i, where the first fault occurred. Formally:

▶ **Definition 4** (RLDXw Diagnosis)**.** *Given an RLDXw problem $\langle \mathcal{O}, \chi, \pi \rangle$, a diagnosis is a time-step i such that $\chi(S_0, \pi, i, h) \neq S_i$ and $\forall i' < i$ it holds that $S_{i'} \in \mathcal{O} \Rightarrow \chi(S_0, \pi, i', h) = S_{i'}$.*

Since RLDXw assumes WFM, it only knows how the effects of actions that were executed normally. Because of that, when two states $S_i, S_{i'}$ are observed, such that S_i is expected, but $S_{i'}$ is not, we can not reconstruct the actual execution. Alternatively, we define the problem for SFW, where we assume several candidate fault modes, out of which one is correct. We call this problem RLDXs. Formally:

▶ **Definition 5** (RLDXs problem)**.** *An RLDXs problem is defined as a tuple $\langle \mathcal{O}, \chi, \pi, \mathcal{F} \rangle$ where π is a policy, χ is a simulator, $\mathcal{O} = (S_0, S_1, \ldots, S_n)$ is an observed series of states, and $\mathcal{F}$ is a set of fault modes. An RLDXs problem arises when there exists an observed state S_i s.t. $S_i \neq \hat{S}_i$ where $\hat{S}_i$ is the state at step i of the trajectory generated by $\chi(S_0, \pi, i, h)$.*

A solution to an RLDXs problem is a *diagnosis*, which in the SFM setting is a fault mode f for which there exists a trajectory generated by the execution of $\chi(S_0, \pi, i, f)$ that is consistent with the observations. Formally:

▶ **Definition 6** (RLDXs Diagnosis)**.** *Given an RLDXs problem $\langle \mathcal{O}, \chi, \pi, \mathcal{F} \rangle$, a diagnosis is a fault mode f_j such that executing $\chi(S_0, \pi, n, f_j)$ can result in a trajectory $T = (S_0, \hat{a}_1, \hat{S}_1, ..., \hat{a}_n, \hat{S}_n)$ where for each observed state $S_i \in \mathcal{O}$ it holds that $S_i = \hat{S}_i$.*

4 Method Description

Next, we present the algorithms for solving the RLDXw and RLDXs.

4.1 Solving RLDXw

We present the algorithm called WFMa which stands for "Weak", since it assumes WFM. It shown in Algorithm 1. WFMa keeps track of the last index where a simulated state matched an observed state as i (lines 1,8), and when it first encounters mismatching states at step i', it returns a set of diagnoses $\mathcal{D} = \{i, i+1, ..., i'\}$ indicating the possible time steps at which an action could have failed for the first time.

▶ **Example 7.** Refer to Example 2. WFMa starts by keeping track of index 0, the index of the first state, as the variable i. It keeps looping throughout the states, updating i every time an observation is available. When WFMa compares the observed state S_7 with the simulated state $\hat{S}_7$, the states are not the same. At this stage, i has the value 5, which is the value of the last matching state. The algorithm will return the diagnoses $\mathcal{D} = \{5, 6, 7\}$, which are all possible diagnoses. This is because WFMa can not determine which of the 3 time steps is the time step of the fault's first occurence.

Algorithm 1 WFMa.

```
Input: $\pi$ - The policy.
Input: $\chi$ - The simulator.
Input: $\mathcal{O} = (S_0, ..., S_n)$ - The observed states.
Result: $\mathcal{D} = \{d_1, d_2, ...\}$ - The set of diagnoses.
1  $i \leftarrow 0$
2  $\hat{S}_0 \leftarrow S_0$
3  for $i' = 1$ to $n$ do
4      $\hat{a}_{i'} \leftarrow \pi(\hat{S}_{i'-1})$
5      $\hat{S}_{i'} \leftarrow \chi(\hat{S}_{i'-1}, \hat{a}_{i'})$
6      if $S_{i'} \in \mathcal{O}$ then
7          if $S_{i'} = \hat{S}_{i'}$ then
8              $i \leftarrow i'$
9          else
10             return $\mathcal{D} = \{i, i+1, ..., i'\}$
```

4.2 Solving RLDXs

To address this setting, we present the algorithm we call SFMa, which is detailed in Algorithm 2. The input to SFMa is the policy π, the simulator χ, the observation $\mathcal{O}$, and also a set of candidate fault modes $\mathcal{F}$. It returns a set of diagnoses $\mathcal{D} = \{d_1, d_2, ...\}$ where each diagnosis is a fault mode that is consistent with the observed states.

SFMa starts by creating an empty trajectory that includes S_0 for each of the k candidate fault modes (Lines 1-4). Then, it goes over the n steps in $\mathcal{O}$ until the candidate set of fault modes becomes size 1 (Lines 5,14). In each step and for each fault mode, SFMa simulates the next state $\hat{S}_{i,j}$ based on the previous state $\hat{S}_{i-1,j}$ and the influence of the fault mode on the predicted action $\hat{a}_{i,j}$ while adding $\hat{a}_{i,j}$ and $\hat{S}_{i,j}$ to the fault mode's trajectory (Lines 6-11). Next, SFMa checks if $\hat{S}_{i,j}$ and S_i are equal, and if they are not, SFMa removes $\langle f_j, \mathcal{T}_{f_j} \rangle$ from the candidate fault modes (Lines 12-13). Finally, in every step $1 < i \leq n$, SFMa checks the size of $\mathcal{G}$, and halts if the size becomes 1, finding the correct fault mode. In case SFMa did not halt, it means there is more than 1 candidate fault mode that is consistent with the observation.

▶ **Example 8.** Refer to Example 2. After the initialization, SFMa simulates the next states for every fault model. Since f_2 changes R actions, the trajectory T_{f_2} deviates from the normal execution as early as state $\hat{S}_2$, meaning that by the time SFMa reaches time step $i = 5$, the state $\hat{S}_5$ returned by simulation does not match the observed state S_5. Consequently, the tuple $\langle f_2, \mathcal{T}_{f_2} \rangle$ will be removed from $\mathcal{G}$. SFMa will then continue running until reaching time step 7, at which point SFMa will find that the state $\hat{S}_{7,3}$ that is part of $\mathcal{T}_{f_3}$ is not equal to S_7. This is because f_3 changes R actions differently than f_1. At this point, SFMa will remove $\langle f_3, \mathcal{T}_{f_3} \rangle$ from $\mathcal{G}$, and will return the last and only fault mode, f_1.

5 Theoretic Analysis

In the following two sections we provide theoretic evaluation, that includes runtime analysis and some correctness proofs.

Algorithm 2 SFMa.

```
Input: $\pi$ - The policy.
Input: $\chi$ - The simulator.
Input: $\mathcal{O} = (S_0, ..., S_n)$ - The observed states.
Input: $\mathcal{F} = \{f_1, f_2, ...f_k\}$ - Candidate fault modes.
Result: $\mathcal{D} = \{d_1, d_2, ...\}$ - The set of diagnoses.
1  $\mathcal{G} \leftarrow \emptyset$
2  for $j = 1$ to $k$ do
3      $\mathcal{T}_{f_j} \leftarrow (S_0)$
4      $\mathcal{G} \leftarrow \mathcal{G} \cup \langle f_j, \mathcal{T}_{f_j} \rangle$
5  for $i = 1$ to $n$ do
6      for $j = 1$ to $|\mathcal{G}|$ do
7          $\hat{a}_i \leftarrow \pi(\hat{S}_{i-1,j})$
8          $\hat{a}_{i,j} \leftarrow f_j(\hat{a}_i)$
9          $\hat{S}_{i,j} \leftarrow \chi(\hat{S}_{i-1,j}, \hat{a}_{i,j})$
10         insert $\hat{a}_{i,j}$ to $\mathcal{T}_{f_j}$
11         insert $\hat{S}_{i,j}$ to $\mathcal{T}_{f_j}$
12         if $S_i \in \mathcal{O} \wedge S_i \neq \hat{S}_{i,j}$ then
13             $\mathcal{G} = \mathcal{G} \setminus \langle f_j, \mathcal{T}_{f_j} \rangle$
14     if $|\mathcal{G}| = 1$ then
15         $\mathcal{D} \leftarrow \{f_j \ s.t. \ \langle f_j, \mathcal{T}_{f_j} \rangle \in \mathcal{G}\}$
16         return $\mathcal{D}$
17 $\mathcal{D} \leftarrow \{f_j \ s.t. \ \langle f_j, \mathcal{T}_{f_j} \rangle \in \mathcal{G}\}$
18 return $\mathcal{D}$
```

5.1 Runtime analysis

The components that influence the run-time are the length between the first and last observed states n, the number of candidate fault modes k (in the case of SFMa) and three domain-dependent components: action prediction, action simulation, and state comparison. We refer to those domain dependent components using the constant $R(domain)$. In the analysis we will focus on the runtime as a function of n and k.

WFMa (Algorithm 1) is straightforward - it goes over the number of states n, and compares each state until it finds two states that are not matching. This gives a total runtime of $O(n \cdot R(domain))$.

SFMa (Algorithm 2) goes over the states where for each state it executes the policy, and then goes over the fault modes, where it does a finite number of simulation and prediction actions. This gives a runtime of $O(n \cdot k \cdot R(domain))$.

5.2 Correctness Proofs

▶ **Theorem 9.** *The index i_e of the first failure in the observed execution holds $i < i_e \leq i'$ where i, i' are the first and last diagnoses returned by WFMa.*

Proof. Assume, by contradiction, that the index of the first failure in the observed execution i_e does not hold $i < i_e \leq i'$. This means that $i_e \leq i \vee i' < i_e$.

Case 1 ($i_e \leq i$). In this case the algorithm still keeps the index $l < i_e$ of the last matching state. The algorithm will reach a step i'_e that holds $i_e \leq i'_e \leq i$, for which an observation exists. The observation $S_{i'_e}$ will not match the simulated state $\hat{S}_{i'_e}$, and the algorithm will halt, returning $l, .., i'_e$ in contradiction to the returned diagnoses $i, .., i'$.

Case 2 ($i' < i_e$). In that case, all of the observable states until step i_e are matching the simulation. That includes state $S_{i'}$. Consequently, when the algorithm checks $S_{i'} = \hat{S}_{i'}$ in line 7, it will set the beginning index to be some $b > i$. As a result, the returned index in the first diagnosis will be b that holds $b \geq i' > i$, in contradiction to the returned indices $i, i+1, ..., i'$. ◀

▶ **Theorem 10.** *SFMa is sound.*

Proof. To prove this, we show that for every fault mode f_j returned by SFMa as diagnosis, it follows that $\forall S_i \in \mathcal{O},\ S_i = \hat{S}_i$, where $\hat{S}_i$ is the i-th state in the trajectory simulated by $\chi(S_0, \pi, n, f_j)$.

Let f_j be a fault mode returned by SFMa. Assume by contradiction that $\exists S_i \in \mathcal{O}$ such that $S_i \neq \hat{S}_i$. Because of that, SFMa will remove the tuple $\langle f_j, \mathcal{T}_{f_j} \rangle$ from $\mathcal{G}$ at the i-th iteration in line 13. Following that, f_j will not be returned as a diagnosis, in contradiction to our initial assumption. ◀

▶ **Theorem 11.** *SFMa is complete.*

Proof. To prove that we show that every fault mode f_j for which it holds that $\forall S_i \in \mathcal{O},\ S_i = \hat{S}_i$, where $\hat{S}_i$ is the i-th state in the trajectory simulated by $\chi(S_0, \pi, n, f_j)$, will be returned by SFMa.

Let f_j be a fault mode as such, and assume by contradiction that it was not returned by SFMa. This means that for some time-step i SFMa removed the tuple $\langle f_j, \mathcal{T}_{f_j} \rangle$ from $\mathcal{G}$. This means that in line 12 SFMa determined that $S_i \neq \hat{S}_i$, by contradiction to f_j being defined as above. ◀

6 Evaluation

In the next two sections we outline the experimental setup and show empirical results.

6.1 Domains

We experimented with the domains *Acrobot*, *CartPole*, *MountainCar* and *Taxi*, all of which can be found in the Gymnasium website[2]. Table 1 shows general statistics about each domain. The "State Vars" row in Table 1 displays the type of variables describing the state – continuous or discrete. The "Env. Type" row displays whether the environment in each domain is dynamic or static, where "dynamic" here means the state may change even if the agent does not do any action, e.g., due to external forces that operate over the agent. The "Avg. Traj." row displays the average length of the trajectories in this domain.

Table 1 General statistics of our benchmark domains.

Domain	**Acrobot**	**CartPole**	**MountainCar**	**Taxi**
# actions	3	2	3	6
State Vars	Continuous	Continuous	Continuous	Discrete
Env. Type	Dynamic	Dynamic	Dynamic	Static
Avg. Traj.	148	54	183	63

[2] `https://gymnasium.farama.org/`

6.2 Experimental Setup

Before generating the problems, we obtained a policy for each domain, either by training or by using pre-trained policies available online. The generation of the problems consisted of the following steps:

1. Initialize a set $\mathcal{F}$ of k fault modes.
2. Choose a fault mode $f_e \in \mathcal{F}$ to be used during policy execution.
3. Execute the policy with the chosen fault mode for up to 200 steps, 10 times. We deemed 200 steps a sufficient trajectory length for the purpose of our experiments, during which at least one fault will occur.
4. Set the percent of visible states v. In an execution with n states, the states that must be observable are states S_0 and S_n. However, execution lengths can vary since we do not control them. Because of that we experiment with the percent of observable states rather than the number. We set $v = 0\%$ for an execution where only S_0 and S_n are observable, and $v = 100\%$ for an execution where every state is observable. We then experiment with different values of v.

6.3 Parameters

For our experiments, we considered fault modes that change some of the actions to behave like other actions. We use this type of fault modes for convenience. For each domain except CartPole, we tested 10 different fault modes f_e. For CartPole, we could only test 3 fault modes due to the number of different actions being only two (moving to the left and moving to the right) and the fact that for the experiments we defined fault modes to be *Many-to-one* or *Onto* functions $f : \mathcal{A} \rightarrow \mathcal{A}$.

We ask the following questions:

Q1 How does the number of fault modes impact the performance of SFMa?

Q2 How does the number of visible states affect the performance of WFMa and SFMa?

To answer **Q1** we experimented with $k \in \{2, 4, 6, 8, 10\}$ for SFMa in domains *Acrobot*, *MountainCar* and *Taxi* and with $k \in \{2, 3\}$ for SFMa in the *CartPole* domain. To answer **Q2** we experimented with $v \in \{0, 5, 10, 15, 20, 30, 40, 60, 80, 100\}$ for both WFMa and SFMa. Lastly, we set fixed parameter of $k = 0$ for WFMa since WFM assumes no fault modes.

6.4 Results

In this section, we present the results that answer our research questions. The Y-axis shows the runtime, and the X-axis shows the different parameter values we experiment with. The X-axis parameters differ from figure to figure. In addition, we crop the values to show whole numbers only, since the runtime is shown in milliseconds.

Figure 2 shows the runtime of SFMa with the increase in the number of candidate fault modes k. We can observe a linear increase in the runtime with the increase in fault modes number. This is expected since an increase in fault modes number has a linear effect on Algorithm 2.

Figure 3 shows the runtime of WFMa with the increase of the percent of visible states v. We observe that the runtime increases significantly from 5% to 0% visibility. We explain this result as follows. At 5% visibility the visible states include states that are not the initial or the last. Because of that, WFMa has more chances to determine whether the agent experienced a fault earlier in the diagnosis process when it encounters a state that does not match its counterpart's simulated state. The one domain that stands out is *CartPole*, where there is no significant increase in runtime for any value of v and where the runtime is very

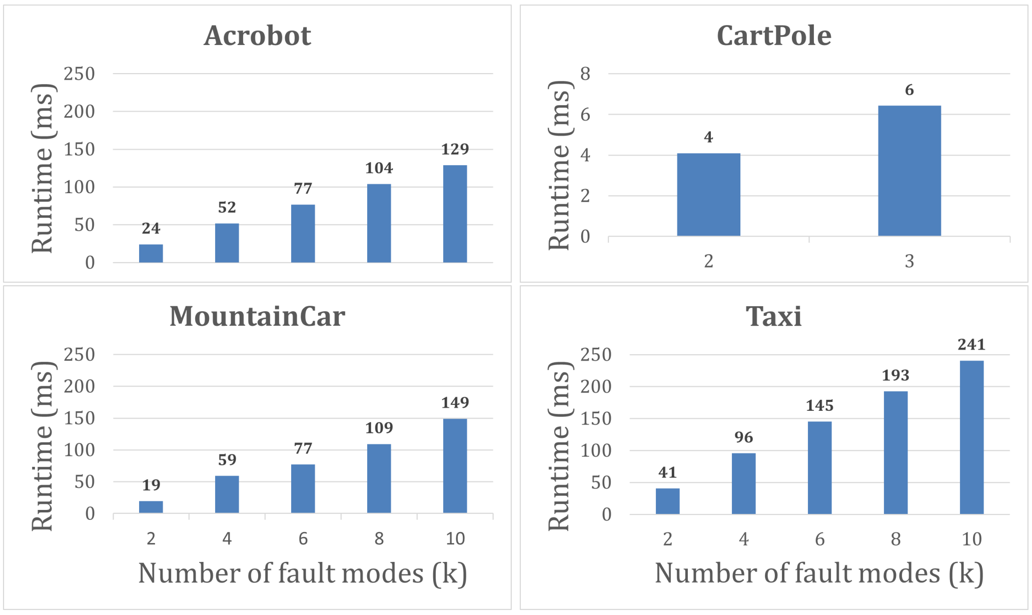

Figure 2 Runtime in milliseconds of SFMa with increase of k.

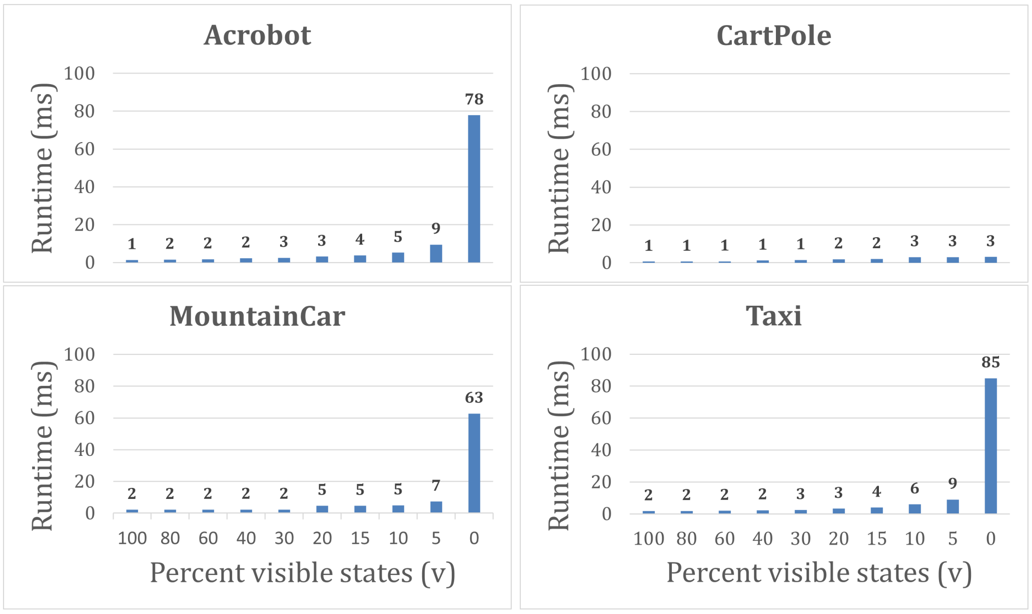

Figure 3 Runtime in milliseconds of WFMa with increase of v.

low. To explain this, we remind that the number of actions in this domain is 2. This means that only up to 3 fault modes could be tested (as explained in Section 6.3) and shown in Figure 2. For such a few actions and candidate fault modes, the runtime stays low and is not influenced by the different percentages of visible states.

Figure 4 shows the runtime of SFMa with the increase of the percent of visible states v. Here we also observe a significant increase between 5% and 0% visible states. The only exception is the *CartPole* domain, where the runtime increases steadily across all values of v. This increase, however, is not significant. We explain this case the same way we explained it in the previous paragraph for Figure 3.

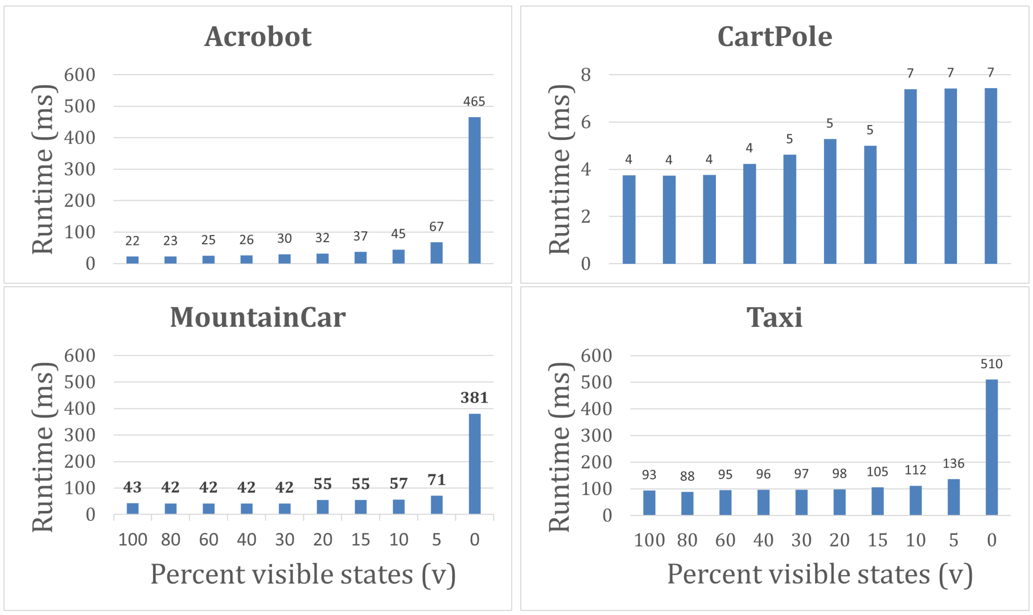

Figure 4 Runtime in milliseconds of SFMa with increase of v.

7 Discussion

In this section we discuss the strengths and weaknesses of the proposed algorithms, as well as some limitations.

The first thing to notice is the significant difference between weak and strong fault model assumptions. While WFMa is faster in terms of runtime, it outputs a less informative diagnosis than SFMa.

Another limiting assumption is the deterministic nature of the policy we use. This allows the reconstruction of a trajectory. For WFMa, it means that given a state S it can reconstruct the healthy execution. For SFMa, it means that given a state S and a fault mode f it can reconstruct an execution where the faulty actions fail according to f. This assumption is important for our algorithms. However, it limits them to cases where the policy is deterministic.

Last but not least, we assume non-intermittent faults. This significantly effects the runtime of SFMa since it means there is only one fauty trajectory, in which every action that should fail according to the fault mode, fails. Although the assumption of non-intermittent faults is a limiting assumption, cases where such faults occur can be thought of. For example, a flying device where one of its engines short-circuited will always have that engine not working, when the commands for that engine are sent.

8 Conclusions and Future Work

In this work, we addressed the problem of diagnosing faulty executions of policies generated by reinforcement learning algorithms. In particular, we focused on non-intermittent faults. We presented two algorithms that address this problem. The first, called WFMa, addresses the problem while assuming *weak fault model*. In this setting a diagnosis is a time-step where a faulty action failed for the first time. The second algorithm, called SFMa, addresses the problem while assuming *strong fault model*. In this setting, a diagnosis is a fault mode that can explain the observation. We tested the algorithms on four well known Reinforcement Learning domains from the Gym benchmarks.

We found out that the runtime of both algorithms increases with the decrease in percentage of observed states. In particular for both algorithms and for every domain except *CartPole*, the runtime increase is the highest between 5% and 0% observability. Apart from that, we found that the runtime of SFMa increases linearly with the increase of the number of candidate fault modes.

As future work, we plan to address intermittent faults setting, and particularly for strong fault model. In this setting, the complexity of the problem grows by a factor of $O(2^n)$. That is because, in the worst case, for a fault mode that affects every one of the n actions, and where only S_0 and S_n are observed, there is a need to address in every time step the possibility of a faulty and a healthy execution of the faulty actions. Apart from that, we plan to experiment with the algorithms on other domains from the Gymnasium benchmark domains and, possibly, test our algorithms on real-world problems. Lastly, we plan to extend this research direction by introducing problems involving multiple agents.

References

1 Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*, pages 89–98. IEEE, 2007.

2 Matthew Daigle, Xenofon Koutsoukos, and Gautam Biswas. Distributed diagnosis of coupled mobile robots. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 3787–3794. IEEE, 2006. doi:10.1109/ROBOT.2006.1642281.

3 Johan De Kleer, Alan K Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial intelligence*, 56(2-3):197–222, 1992. doi:10.1016/0004-3702(92)90027-U.

4 Orel Elimelech, Roni Stern, Meir Kalech, and Yedidya Bar-Zeev. Diagnosing resource usage failures in multi-agent systems. *Expert Systems with Applications*, 77:44–56, 2017. doi:10.1016/J.ESWA.2017.01.047.

5 Meir Kalech and Gal A Kaminka. Coordination diagnostic algorithms for teams of situated agents: Scaling up. *Computational Intelligence*, 27(3):393–421, 2011. doi:10.1111/J.1467-8640.2011.00386.X.

6 Meir Kalech and Avraham Natan. Model-based diagnosis of multi-agent systems: A survey. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 12334–12341, 2022. doi:10.1609/AAAI.V36I11.21498.

7 Roberto Micalizio and Pietro Torasso. Cooperative monitoring to diagnose multiagent plans. *Journal of Artificial Intelligence Research*, 51:1–70, 2014. doi:10.1613/JAIR.4339.

8 Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

9 Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. doi:10.1038/NATURE14236.

10 Avraham Natan and Meir Kalech. Privacy-aware distributed diagnosis of multi-agent plans. *Expert Systems with Applications*, 192:116313, 2022. doi:10.1016/J.ESWA.2021.116313.

11 Avraham Natan, Meir Kalech, and Roman Barták. Diagnosis of intermittent faults in multi-agent systems: An sfl approach. *Artificial Intelligence*, 324:103994, 2023. doi:10.1016/J.ARTINT.2023.103994.

12 Avraham Natan, Roni Stern, and Meir Kalech. Distributed spectrum-based fault localization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6491–6498, 2023. doi:10.1609/AAAI.V37I5.25798.

13 Alexandre Perez, Rui Abreu, and Marcelo d'Amorim. Prevalence of single-fault fixes and its impact on fault localization. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 12–22, 2017. doi:10.1109/ICST.2017.9.

14 Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

15 Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

16 Patrick Rodler. How should i compute my candidates? a taxonomy and classification of diagnosis computation algorithms. *arXiv preprint arXiv:2207.12583*, 2022. doi:10.48550/arXiv.2207.12583.

17 Nico Roos, Annette Ten Teije, and Cees Witteveen. A protocol for multi-agent diagnosis with spatially distributed knowledge. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 655–661, 2003. doi:10.1145/860575.860681.

18 Nico Roos and Cees Witteveen. Models and methods for plan diagnosis. *Autonomous Agents and Multi-Agent Systems*, 19:30–52, 2009. doi:10.1007/S10458-007-9017-6.

19 Michael Schmid, Emanuel Gebauer, Christian Hanzl, and Christian Endisch. Active model-based fault diagnosis in reconfigurable battery systems. *IEEE Transactions on Power Electronics*, 36(3):2584–2597, 2020.

20 John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. arXiv:1707.06347.

21 Peter Struss and Oskar Dressler. " physical negation" integrating fault models into the general diagnostic engine. In *IJCAI*, volume 89, pages 1318–1323, 1989. URL: http://ijcai.org/Proceedings/89-2/Papers/075.pdf.

22 Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

23 Gianluca Torta and Roberto Micalizio. Smt-based diagnosis of multi-agent temporal plans. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2097–2099, 2018. URL: http://dl.acm.org/citation.cfm?id=3238084.

24 Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. doi:10.5281/zenodo.8127026.