ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230462

Blame Attribution for Multi-Agent Path Finding Execution Failures

Avraham Natan*, Roni Stern and Meir Kalech

Ben-Gurion University of the Negev
ORCiD ID: Avraham Natan https://orcid.org/0000-0002-3600-6813

Abstract. In Multi-Agent Systems (MAS), Multi-Agent Path Finding (MAPF) is the problem of finding a conflict-free plan for a group of agents from a set of starting points to a set of target points. Deviations from this plan are standard in real-world applications and may decrease overall system efficiency and even lead to accidents and deadlocks. In large MAS scenarios with physical robots, multiple faulty events occur over time, contributing to the overall degraded system performance. This raises the main problem we address in this work: how to attribute blame for a degraded MAS performance over a set of faulty events. We formally define this problem and propose using the Shapley values to solve it. Then, we propose an algorithm that efficiently approximates Shapley values by considering only some subsets of faulty events set. We analyze this algorithm theoretically and experimentally and demonstrate that it enables effectively trading off runtime for error.

1 Introduction

Multi-Agent Planing (MAP) is the problem of finding a set of plan series for a group of agents known as Multi-Agent System (MAS) to execute in order to achieve a goal defined by the system [48]. MAP is a known problem and has been studied and applied in many domains, and in particular in the Multi-Agent Path Finding (MAPF) domain [49, 32, 46].

The execution of such plans often deviates from the plan. Such deviations may occur for a variety of reasons, including internal reasons (i.e., failures in the navigation systems of an agent [37]), external reasons (difficult environmental conditions such as sandstorms [11]), or due to imprecise assumptions about the world [36]. The impact of these deviations may lead to system failures, e.g., accidents and deadlocks, or unacceptable degradation in overall system throughput. For example, consider an automatic warehouse where robots are tasked to move items [47]. A delay in one of the robots may cause it to interfere with another robot which in turn will interfere later with other robots, and so on. Eventually, this can cause a significant and unacceptable delay in moving the items. Another example comes from the domain of multi-agent logistics [32], where agents collaborate to move packages from one place to the other by passing the packages between each other. A faulty execution of an action may cause a chain reaction which will lead to package delivery failure.

An important question to ask when a multi-agent system fails is "what is the root cause of the failure?". Previously proposed diagnosis algorithms for MAS [29, 28] were designed to answer this question and localize the responsible faulty events. However, they do not account for the possibility that each agent in the MAS deviates a little bit. In this work we ask the complementing question: "how much did each faulty event contribute to the system failure?" We denote this question as *Blame Attribution*. To motivate answering the blame attribution question, consider an automated warehousing company. The agents continuously move items, and unavoidably, small deviations in the agent's movement speed are constantly occurring. On a small scale, this is acceptable, but with time those deviations pile up and cause jams that lead to an unacceptable decrease in throughput. Assigning blame to individual faults is helpful since it allows the planning team to plan more robustly for future tasks. The QA team logging the faults can forward the faults that matter, helping the planning team replan more effectively. For instance, blame attribution might conclude that a specific agent that rotates slowly should be given tasks that involve fewer rotations or that only a selected number of robots should traverse a key area (i.e., a narrow corridor) at any given time. This is very useful for companies that wish to cut costs and use their agents to the fullest. In another scenario of autonomous vehicles, diagnosis algorithms may infer which defective vehicles are to blame for an accident, but will not determine how much each vehicle contributed to it. This could be achieved by blame attribution algorithms.

The first contribution of our paper is to formally define the blame attribution problem in the context of MAPF execution failures. We call this the **Blame Attribution for Multi-Agent Path Finding Execution Failures (BAMPEF)**. There may be multiple ways to attribute blame, but in this work, we propose to use the well-known Shapley Values [43]. Shapley values have been used in game theory [39, 38], in moral philosophy [53, 33], law [10], politics [21, 9], and other areas [14, 13]. Also, it has several desirable properties that are suitable for BAMPEF.

Unfortunately, the calculation of Shapley values is exponential in the number of members (in our case, fault events), as it iterates over all subsets of them. For instance, the execution in our domain took 36 seconds on average when executing with 13 fault events that the agents were involved in. To address this gap, there are approaches that improve run-time by approximating the real Shapley values, for example by sampling subsets of the fault events [4]. **The second contribution** of this paper is a fast method to approximate the Shapley values that we call **Diagnosis-Directed Blame Attribution (DDBA)**. DDBA uses concepts from the field of Model-Based Diagnosis [40] to identify which subsets of fault events are sufficient to obtain an effective approximation for the shapely values, and which

* Corresponding Author. Email: natanavr@post.bgu.ac.il

will not contribute as much. Limiting this calculation to consider only these subsets significantly reduces the run-time. For instance, in the same example of 13 fault events it took 1.62 seconds on average, while the mentioned sampling-based approach ran for 3.54 seconds. Experiments show that our proposed method outperforms the sampling-based calculation in terms of run-time, and still provides a better approximation of the Shapley values.

2 Background and Related Work

In this section, we present the necessary background for our work, including Multi-Agent Planning, Multi-Agent Path Finding and Shapley values, and survey previous work on robust path finding, MAS diagnosis, and blame attribution.

2.1 Multi-Agent Planning and Path Finding

Multi-Agent Planning (MAP), is a well-studied AI domain [49]. The domain's main problem is finding a series of actions that allows a group of agents to achieve their goals while minimizing costs. The problem for n agents is formally defined as a tuple $\mathcal{T}_{MAP} = \langle \mathcal{AG}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{AG} = \{1, ..., n\}$ is a finite non-empty set of agents, $\mathcal{V} = \bigcup_{i \in \mathcal{AG}} \mathcal{V}^i$ where $\mathcal{V}^i$ is the set of state variables known to agent i, $\mathcal{I} = \bigcup_{i \in \mathcal{AG}} \mathcal{I}^i$ is a state of fluents that defines the initial state, $\mathcal{G}$ is the set of goal states, and $\mathcal{A} = \bigcup_{i \in \mathcal{AG}} \mathcal{A}^i$ is the set of actions. [49]. A solution to this problem is an ordered set of actions whose application achieves the system's goals. A specific but widespread use of MAP is the field of Multi-Agent Path Finding.

Multi-Agent Path Finding (MAPF), is also a well-studied AI domain [46]. The main problem in this domain is finding paths from starting points to target points for a group of agents while minimizing costs. The problem for n agents is formally defined as a tuple $\langle G, s, t \rangle$ where $G = (V, E)$ is a graph representation of a set of locations (V) and the roads between them (E), and $s : [1, ..., n] \rightarrow V$, $t : [1, ..., n] \rightarrow V$ maps each agent to its start and target vertices on the graph, respectively. A solution to this problem is a set of n single-agent paths, one for each agent.

2.2 Shapley Values

Calculation of Shapley Values is a concept from the game theory domain, first introduced in 1953 [43]. The goal of Shapley values calculation is to determine the division of power among a group of members. This is done by using the *marginal contribution* of each member to the various subsets of the member group. A formal definition can be found in many forms in the literature [54, 42, 51]. We give here one of those forms.

Definition 1 (**Shapley Value**). *Given a group N of n members and a cost function $v : 2^N \rightarrow \mathbb{R}$, the Shapley Value for member i is defined as follows:*

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} \big[v(S \cup i) - v(S) \big]$$

We next present related work on Diagnosis and Shapley values.

2.3 Related Work

A common approach to address MAS failures, particularly in the MAPF domain, is to run a diagnosis process to identify the faulty agents that caused the failure. Our work addresses a different aspect - given a system failure, attributing the blame among the fault events that caused the failure. To that end, we survey works in three research fields: MAS Diagnosis, MAPF and Blame Attribution in AI.

MAPF algorithms address MAPF execution failures in a variety of ways. One work addresses delay and acceleration faults in the MAPF execution by postprocessing the output of a MAPF solver [20]. Their approach considers the robots' velocities and creates a safe distance between them. Another work proposes pathfinding that considers delay probabilities (MAPF-DP) [31]. The authors propose a solver for generating MAPF-DP solutions and policies for robust plan execution. Addressing a more general problem of MAPF under uncertainty was done in a later work [52]. The authors develop a multi-agent path planner that considers uncertainty. The planner plans through the belief space of the agents and coordinates agents that are likely to collide. Another work [19] presents a robust framework for multi-agent plan execution that deals with uncertainty and unpredictable speed changes and obstacle appearances. The authors demonstrate their method on warehouse simulations and mixed reality simulations using physical robots. Lastly, a recent work applied MAPF plan execution techniques to a realistic railway scheduling problem with uncertainty [30]. By incorporating existing state-of-the-art MAPF techniques, the method is capable of planning collision-free paths and deadlock-free actions in real time.

The above work shows how planning before or during execution helps build robust systems. In the case where systems fail, diagnosis methods are used. Diagnosis of MAS has been studied in various settings w.r.t. different criteria [28]. Attributes like fault types, centralized/decentralized methods of diagnosis, number of observations, and temporality are some of the settings that previous work addresses. Fault type is a notable point of the difference between previous work. Some studies aim to diagnose faulty agents when a fault in the execution of the plan occurs [8, 41, 7, 34, 35, 50, 37]. Others aim to diagnose failures in the cooperation between agents [6, 24, 27, 25, 26, 23]. We refer to [28] for a comprehensive survey of previous work in this field. In all the mentioned work, the assumption is that agents are either "faulty" or "not faulty", with no intermediary values. One can imagine a system where many of the agents fail a bit (e.g. lags in its movement in the MAPF domain). In such cases, diagnosing the faulty agents is not sufficient, but rather blame attribution is necessary.

Blame Attribution has been addressed previously in the context of responsibility and blame in AI [5, 18, 17, 12]. These works present definitions for the concept of blame attribution, while considering environments where a number of members participate. They also point out challenges in blame attribution in systems where an action of one member influences the actions of another [18], which is common in multi-agent systems where the action of one agent depends on or influences other actions. Building on the above work, a recent work [51] addresses the task of allocating blame to agents for causing system inefficiency [51]. They particularly focus on cooperative decision-making formalized by Multi-Agent Markov Decision Processes (MMDPs). They present criteria of desirable properties for blame attribution, inspired by game theory literature [22, 45, 3]. An example of such property is the *Efficiency* which states that the value of the grand coalition is distributed among the members. They study some of the known blame attribution methods with relation to this criteria: *core* [15], *Shapley Values* [43, 44] and *Banzhaf Index* [1, 2], and also introduce a novel blame attribution method. Although presenting important results, the run-time of the various methods is not presented. In our work, we focus on the Shapley values to attribute

the blame among fault events in a multi-agent system, while improving the run-time of the baseline Shapley values by using a diagnosis-based method. In another work [4], the authors present an approximation to the Shapley values that does not consider the complete subset of the members (in our work – faults), but rather rely on a random sampling of subsets. This sampling provides an approximation to the Shapley values and by doing so, reduces the run-time. Our method also uses select subsets of the faults set but chooses the specific subsets which are connected to the success of the system, thus guiding the entire process to a closer approximation of the real values.

3 Methodology

In this section, we define the problem of Blame Attribution in Multi-Agent Plan Execution Failures (BAMPEF) and propose an algorithm for solving it. Our algorithm is general and requires a plan as an input. The algorithm can easily be configured to conform to different inputs of different domains. For simplicity, we define and demonstrate our method on the well-studied domain of Multi-Agent Path Finding (MAPF) that we surveyed in the previous sections [46]. Notable application instances are automated warehousing [47] and automated parking [55]. We first define domain-specific preliminaries for MAPF and then present our domain-agnostic approach.

3.1 MAPF related preliminary definitions

A solution to the MAPF problem is a set of single-agent plans (paths), one for each agent. Here we describe the plan:

Definition 2 (Plan). *Given a set of agents A, time-steps $T \subseteq \mathbb{Z}^+$ and locations V, a plan $\pi : A \times \mathbb{Z}^+ \to V$ is a mapping of (a,t) to the location agent a is planned to hold at time-step t, $\forall a \in A, t \in T$.*

Example 1. *Table 1 shows an example of a plan π, and Figure 1 shows a visualization of the plan. We will use them as a running example throughout the paper.*

π	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
a_1	(7,3)	(6,3)	(5,3)	(4,3)	(3,3)	(2,3)	(1,3)	(0,3)
a_2	(2,2)	(2,3)	(2,4)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)
a_3	(1,7)	(1,6)	(1,5)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)

Table 1. A running example showing a plan π for 8 steps.

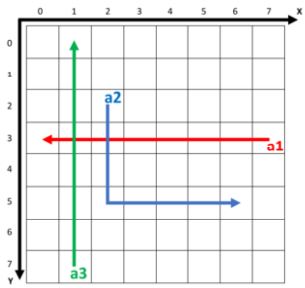

Figure 1. Visual representation of the plan presented in Table 1. The three arrows represent the plans of three agents.

Once a plan is generated, the system is executed. Each agent follows its plan, and an observation is recorded.

Definition 3 (Observation). *An observation $o : A \times \mathbb{Z}^+ \to V$ is a mapping of (a,t) to the location agent a occupied at time-step t.*

Another important representation of the observation is called Plan-Step. Plan-Step is a mapping between the observed locations to their ordinal time-step according to the plan, that is generated during the plan execution. Formally:

Definition 4 (Plan-Step). *A plan-step $\tau_\pi : A \times \mathbb{Z}^+ \to \mathbb{Z}^+$ is a mapping between the wall clock time and the time-step at which the agent was planned to occupy its current location.*

In a non-interrupted plan execution, where each agent follows its plan, the following corollary stands as a direct result of the above definitions:

Corollary 1. *Given that no faults occur during the execution, it follows that $\forall a,t : o(a,t) = \pi(a,t)$, and $\tau_\pi(a,t) = t$.*

Faulty steps, however, can happen for a variety of reasons and manifest in a variety of forms. A realistic assumption is that agents might not accelerate or slow down as planned due to physical reasons, leading to undesired accelerations or delays. When a fault occurs, Corollary 1 is no longer true. When this happens, we are interested to compute the time offset between the agent's current plan step and the planned plan step. We define the Plan-Offset of an agent as the difference in time between the plan steps of the agent and the wall clock. Formally:

Definition 5 (Plan-Offset). *A plan-offset $\Delta_{\tau,\pi} : A \times \mathbb{Z}^+ \to \mathbb{Z}$ is defined as $\Delta_{\tau,\pi}(a,z) = z - \tau_\pi(a,z)$, where z is the wall clock.*

If no faults occur, then it follows that $\forall a,t : \Delta_{\tau,\pi}(a,t) = 0$, and when a fault occurs, $\exists a,t : \Delta_{\tau,\pi}(a,t) \neq 0$. Our work deals with attributing the blame among the different faulty steps.

In our work, we focus on two types of faults: *Delay Fault* and *Acceleration fault*. Both fault types, change the speed of the agent. We generalize those faults as *Speed Change Fault*. Formally:

Definition 6 (Speed Change Fault). *A Speed Change Fault for agent a at time t that follows plan π is a function $\chi : A \times \mathbb{Z}^+ \times \mathbb{Z} \times \mathbb{F}^\tau \to \mathbb{F}^\tau$ that does the following: $\chi(a,t,z,\tau_\pi) = \tau'_\pi$ s.t : $\forall t' \geq t, \tau'_\pi(a,t') = \tau_\pi(a,t') + z$.*

Intuitively, this definition states that a speed change fault changes the plan-step mapping τ to consider the speed change fault that happened. For positive values of z, we denote the fault as *Acceleration fault* and for negative as *Delay Fault*.

It is logical to assume that due to faults, agents will sometimes interrupt other agents. In that case, we say that a *conflict* has occurred between two agents. More specifically, we say that agent a is in conflict at time t with agent a' when a' occupies the location agent a tries to move to at time t. In many applications, conflicted agents are instructed to stay in place, adjusting their plan offset accordingly: $\big(\pi(a,\tau_\pi(a,t)+1) = o(a',t) \vee \pi(a,\tau_\pi(a,t)+1) = o(a',t-1)\big) \to \forall t' \geq t : \chi(a,t,1,\tau_\pi)$

Example 2. *Tables 2, 3 and 4 show an example of an execution (observation, plan-step, plan-offset) of the plans in Table 1 that introduced some faults. The execution is also presented in Figure 2. Agent a_1 experiences two speed change faults of type acceleration fault during time-steps one and two. This is expressed as position skipping in Table 2, in higher time-step in Table 3 and in negative values in Table 4 (see columns t_1 and t_2). During that time, a_3 advances normally, and this is reflected by the first two green arrows in Figure 1, in observation similar to the plan, shown in Table 2, and in the expected values in columns t_1 and t_2 of Tables 3 and 4. Agent*

a_2 experiences speed change fault of type delay fault at t_2, which causes it to remain in its position, as shown by the circle arrow and the corresponding values in the tables. Next, in time-step 3, agent a_1 tries to advance to position $(2,3)$, but since a_2 is still there, there is a conflict between the agents. a_1 remains in its position and its plan offset increases accordingly. The other two agents proceed with their plans. Specifically agent a_2 reaches its next position with a delay. This example continues until the agents reach their targets, and during which agent a_1 is further delayed by agent a_3. This delay is also shown in the corresponding tables.

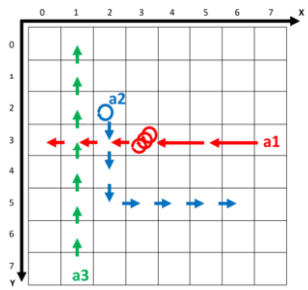

Figure 2. Visual representation of a faulty execution of the plans in Figure 1. The accelerations and delays are indicated by the round arrows.

o	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
a_1	(7,3)	(5,3)	(3,3)	(3,3)	(3,3)	(3,3)	(2,3)	(1,3)	(0,3)
a_2	(2,2)	(2,2)	(2,3)	(2,4)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)
a_3	(1,7)	(1,6)	(1,5)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)	-

Table 2. The observation o of the faulty execution in Figure 2.

τ_π	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
a_1	1	3	5	5	5	5	6	7	8
a_2	1	1	2	3	4	5	6	7	8
a_3	1	2	3	4	5	6	7	8	-

Table 3. The Plan-Step τ_π of the faulty execution in Figure 2.

$\Delta_{\tau,\pi}$	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
a_1	0	-1	-2	-1	0	1	1	1	1
a_2	0	1	1	1	1	1	1	1	1
a_3	0	0	0	0	0	0	0	0	-

Table 4. The Plan-Offset $\Delta_{\tau,\pi}$ of the faulty execution in Figure 2.

Considering Corollary 1, when $o(a,t) \neq \pi(a,t)$, we call it an *Inconsistent Event*. We define three types of inconsistent events: (1) *Expected Event* is an inconsistent event that can be explained by a previous fault event. (2) A *Conflict Event* is an inconsistent event where agent a' was occupying the position to which a is planned to move at t, or occupied it at step t. (3) A *Fault Event* is an inconsistent event that is neither of the first two. Formally:

Definition 7 (Inconsistent Event). *An Inconsistent Event is a tuple $\langle \pi, o, a, t \rangle$ where $o(a,t) \neq \pi(a,t)$. We separate such events into three types:*

- *Expected Event: if $o(a,t) = \pi(a, \tau_\pi(a,t))$.*
- *Conflict Event: if $\exists a' : \big(\pi(a, \tau_\pi(a,t)+1) = o(a',t) \vee \pi(a, \tau_\pi(a,t)+1) = o(a', t-1)\big)$.*
- *Fault Event: if it is neither a conflict event nor an expected event.*

We omit π, o when the context is clear. In addition, we abuse the notation of inconsistent event to include the change in speed. A fault event, in that case, given that we remove π, o will have the form: $\langle a, t, z \rangle$ when $z \in \mathbb{Z}$ is the acceleration amount (positive or negative) for the fault.

Using the above definitions, we can derive the fault events set, denoted as E once a system is being executed.

Example 3. *Consider Definition 7. We compute E for our running example presented in Figure 2. The resulting set is: $E = \Big\{\langle a_1, 2, 1\rangle, \langle a_1, 3, 1\rangle, \langle a_1, 5, -2\rangle, \langle a_2, 2, -1\rangle\Big\}$. For instance $\langle a_1, 2, 1\rangle$ denotes that agent a_1 had an acceleration fault at time step 2 - it accelerated by 1 step in executing its plan. An example of conflict event is $\langle a_1, 4, -1\rangle$, since $\pi(a_1, \tau_\pi(a_1, 4)+1) = o(a_2, 3)$, i.e., agent a_2 holds the position that agent a_1 tries to occupy at time step 4. An example of an expected event is $\langle a_1, 7, 0\rangle$, since $o(a_1, 7) = \pi(a_1, \tau_\pi(a_1, 7))$, i.e., considering the previously executed steps, agent a_1 should advance to the position observed at time-step 7, although it is not originally planned.*

Note that an agent that already halted can still cause a conflict. This can happen if the goal position of that agent is on the path of another agent, and when due to accelerations or delays, the agent reaches the goal position before the other agent passed it.

Fault events may cause the multi-agent system degradation in its performance. We hence define a *Value Function* to evaluate the execution of a plan π with observations o and the set E. There may be different ways to define this function, in this paper, we define it as the maximal plan offset at the wall clock time at which the system halted. Formally:

Definition 8 (Value Function). *A Value Function is a function $v : \mathbb{F}^\pi \times \mathbb{F}^o \times \mathbb{F}^E \rightarrow \mathbb{R}$ that returns the cost of simulating π, given the fault events E derived from the observation o. We define v as: $v(\pi, o, E) = \max_{a \in A} \Delta_{\tau,\pi}(a, t_{last})$ where t_{last} is the wall clock time when the system simulation halted.*

Example 4. *In our example, the cost value is: $v(\pi, o, E) = \max\{\Delta_{\tau,\pi}(a_1, t_9), \Delta_{\tau,\pi}(a_2, t_9), \Delta_{\tau,\pi}(a_3, t_8)\} = \max\{1, 1, 0\} = 1$*

Now that we defined the domain-specific preliminaries, we can use the value function v, together with the plans π, observations o, and fault events E to evaluate simulations of plan executions, which is the focus of the rest of the paper. As mentioned in the beginning of Section 2.1, the preliminary definitions are domain-specific, and should be defined separately for each domain.

3.2 The Blame Attribution problem

It is obvious that if $E = \emptyset$ then $v(\pi, o, E) = 0$. In case that $E \neq \emptyset$, we are interested to compute the amount of contribution of each one of the fault events in E to the degradation of the system. We thus define the problem of **Blame Attribution for Multi-Agent Plan Execution Failures (BAMPEF)**. Formally,

Definition 9 (BAMPEF). *Given a tuple $\langle \pi, o, E, v \rangle$ the objective is to compute the blame of each fault event $e \in E$.*

A solution to the BAMPEF problem is a vector $\vec{S}$ of the form $\mathbb{R}^{|E|}$, with each value in the vector representing the blame value of a fault event in E. In this work, we approach this problem using Shapley values calculation. Next, we elaborate on the way we apply it.

3.3 Shapley Values for Blame Attribution

Shapley values have been used in a wide variety of research domains. The Shapley values have many desirable properties. Specifically, the properties that are relevant to our domain are Efficiency, Symmetry, Additivity, and Null-member. We present each of them with its relevance to our domain:

- *Efficiency* states that the value of the grand coalition is distributed among the members. Its relevance to our domain is trivial because we assume the fault events are the sole reason for the system failure, and as such we are interested to distribute the blame solely among them.
- *Symmetry* states that for any two equal members, the distributed value is equal. In our domain, it means that two fault events that have an equal marginal contribution to the system's failure will get the same blame.
- *Additivity* states that if two coalition games, described by gain functions v and w, are combined, then the distributed gains should correspond to the gains derived from v and the gains derived from w. Mathematically: $\phi_e(v+w) = \phi_e(v) + \phi_e(w)$. This is important since it allows us to aggregate Shapley values, which we do in our approach, as we describe in the next section.
- *Null-member* states that the value of a null actor (an actor that does not contribute to any coalition) is zero. It is relevant to our domain since we assume that some of the fault events are not the cause of the system failure, and in that case, they will receive no blame.

Those properties are desirable to our domain and specifically to our approach. For that reason, we chose the Shapley values as a baseline for attributing the blame among the fault events.

Recall the Shapley values formula for a member as shown in Definition 1, applying it to solve the BAMPEF problem will be formally defined as follows:

Definition 10 (Shapley Values for BAMPEF). *Given a set E of n fault events and a value function $v : 2^E \to \mathbb{R}$, the Shapley Value for fault event e is:*

$$\phi_e(v) = \sum_{E' \subseteq E \setminus \{e\}} \frac{|E'|!(n-|E'|-1)!}{n!} \left[v(E \setminus (E' \cup \{e\})) - v(E \setminus E')\right]$$

We compute this formula for every fault event $e \in E$, and get the vector of Shapley values $\vec{S}$. This vector is normalized and the resulting vector has values ranging between 0 and 1, when their total sum is 1. Example 5 demonstrates this calculation for our running example.

Example 5. *Given the set E from Example 3, the Shapley values vector that corresponds to the fault set E is: $\vec{S} = [0.5, 0.5, 0.0, 0.0]$. This means that the first two fault events have equal blame for the degradation of the system.*

Calculating the Shapley values is exponential in the number of the fault events, due to the iteration over all subsets of E. To address this, we propose an approach that approximates the Shapley values, by considering only part of the fault events in E, selected by a diagnosis process. We elaborate on it in the next section.

3.4 Diagnosis-Directed Blame Attribution

Diagnosis processes aim to identify the root cause of a failure in a system. In our domain, the root cause is the fault events that caused the degradation in the system's performance. To this end, we define the concept of *Useful Repair*. A *Useful Repair* is a subset of the fault events set E, that reduces the cost of the value function when the system is simulated without having those faults. Formally:

Definition 11 (Useful Repair). *Given a set of fault events E and a cost function $v : 2^E \to \mathbb{R}$, the Useful Repair is:*

$$\Omega = \{E' \subseteq E : v(\pi, o, E \setminus E') < v(\pi, o, E)\}$$

Once the set Ω is calculated, we calculate the Shapley values of the fault events with respect to each $\omega \in \Omega$. We call this approach **Diagnosis Directed Blame Attribution (DDBA)**. To that end, we extend definition 1 to consider the set ω:

Definition 12 (Shapley Value for BAMPEF w.r.t. ω). *Given a set $\omega \subseteq E$ of n fault events and a cost function $v : 2^E \to \mathbb{R}$, the Shapley value w.r.t. ω for fault event e is defined as follows:* $\phi_e^{\omega}(v) = \sum_{\omega' \subseteq \omega \setminus \{e\}} \frac{|\omega'|!(n-|\omega'|-1)!}{n!} \left[v(E \setminus (\omega' \cup \{e\})) - v(E \setminus \omega')\right]$

At this point Ω may still be big - there may be a lot of useful repair sets. The calculation would run on all $\omega \in \Omega$, and this might lead to long run times. In order to reduce this run-time, we propose to decrease the size of Ω by considering only $\omega \in \Omega$ with cardinality up to a number k. We denote the resulting smaller set as Ω'. To that end, we iterate over the different cardinalities $i \in [1, ..., k]$, and for each cardinality, we compute Shapley values of $\omega \in \Omega'$ with cardinality i.

Finally, we aggregate those values over all the fault events in Ω' to achieve an approximation to the Shapley values presented in the previous section. Note, that the approximation does not converge to the real value, since we consider only subsets of E that repaired the system. However, their corresponding counterfactuals have a non-negligible contribution to the Shapley values by definition (as they correspond to returning to a nominal state). Thus, the motivation for using our approximation is that it guides the approximation toward subsets that have a larger impact on the Shapley values.

Algorithm 1: Diagnosis Directed Blame Attribution

Input: E - a set of fault events
Input: k - iterations number
Result: $\vec{S}$ - Shapley values corresponding to E

1 $\vec{S} \leftarrow [0, ..., 0]$
2 **for** $i \in [1, ..., k]$ **do**
3 $\quad \Omega' \leftarrow \{E' \subseteq E : |E'| = i \wedge v(\pi, o, E \setminus E') < v(\pi, o, E)\}$
4 $\quad$ **for** $\omega \in \Omega'$ **do**
5 $\quad\quad \vec{S} \leftarrow \vec{S} + normalize([\phi_{e_1}^{\omega}(v), ..., \phi_{e_{|E|}}^{\omega}(v)])$
6 **return** $normalize(\vec{S})$

Algorithm 1 formalizes this approach. First, the solution vector $\vec{S}$ is initialized (Line 1). Then, the algorithm runs for k iterations ($i \in 1, ..., k$). In each iteration, it extracts all the subsets of E that are of size i and are useful repairs (Line 3). Then, for each useful repair that was extracted, the algorithm computes a vector of Shapley values, normalizes it, and adds it to $\vec{S}$ (Lines 4-6). Finally, $\vec{S}$ is normalized and returned (Lines 8-9).

Example 6. *Consider E from Example 3. For $k = 2$, the subset $\Omega' \subseteq E$ includes all single and double fault event sets, that are useful repairs:*

$$\Omega' = \Big\{\{\langle a_1, 2, 1\rangle\}, \{\langle a_1, 3, 1\rangle\}, \{\langle a_1, 5, -2\rangle\}, \{\langle a_2, 2, -1\rangle\}, \{\langle a_1, 2, 1\rangle, \langle a_1, 3, 1\rangle\}, \{\langle a_1, 2, 1\rangle, \langle a_1, 5, -2\rangle\}, \{\langle a_1, 2, 1\rangle, \langle a_2, 2, -1\rangle\}, \{\langle a_1, 3, 1\rangle, \langle a_1, 5, -2\rangle\}, \{\langle a_1, 3, 1\rangle, \langle a_2, 2, -1\rangle\}, \{\langle a_1, 5, -2\rangle, \langle a_2, 2, -1\rangle\}\Big\}$$

The final aggregated set is $\vec{S} = [0.4, 0.4, 0.0, 0.2]$. It is worth noting that in this case, we saved the computation of the Shapley values for the sets that include 3 and 4 fault events.

We note to the reader that for the sake of simplicity in our running example all fault events with cardinally 1 and 2 are useful repairs. This is not the case for more complex systems with higher numbers of agents and with longer plans.

3.5 Time complexity analysis

In this section, we provide theoretical analysis to the run-time of DDBA. The theoretical evaluation of the random sampling approach can by found in [4]. Recall that Shapley's calculation is exponential in the size of the set on which the calculation is applied. In the case of the standard calculation, as defined in Definition 10, if it is applied on the set E, then the complexity is $2^{|E|}$. DDBA on the other hand, considers only the useful repair sets of cardinality up to some k. For each $1 \leq i \leq k$, the number of those sets is bounded by $\binom{|E|}{i}$, which is tighter bound than $|E|^i$. For each such set, we apply the traditional Shapley calculation. This gives theoretical runtime of 2^i. Putting all together, the theoretical bound to the runtime for DDBA with a cardinality of useful repairs up to k is: $\sum_{i=1}^{k} \binom{|E|}{i} \cdot 2^i$. Moreover, we use dynamic programming approach by memorizing the results of all subsets's Shapley values. This lowers the bound to $\binom{|E|}{k} \cdot 2^k$. Finally, since not every set is a useful repair, the actual number is significantly lower.

4 Evaluation

In this section, we experimentally compare DDBA with a baseline approach and a complete calculation of the Shapley values. Specifically, we compare the following three algorithms:

- ***Gold*** - the Shapley Values calculation as presented in Definition 10, that takes into account the full set of fault events E for its calculation.
- ***Random*** - the algorithm based on random sampling, which we mentioned in Section 2 [4].
- **DDBA** - our algorithm which selects all the subsets of the useful repair set (Ω') with cardinality up to k.

4.1 Experimental Setup

All our experiments were performed on a 12x12 empty grid. The independent variables in our experiment were the number agents x, the plan length y, the number of faulty agents f, the probability that a faulty agent experiences a fault p, and the failure threshold *th*. In each experiment, we generated a BAMPEF problem $\langle \pi, o, E, v \rangle$ by performing the following process. First, we create a plan π by randomly placing each of the x agents in the grid and moving them randomly for y time steps, ensuring that they do not collide. Then, we randomly select f of the x agents to be faulty. Next, we simulate the execution of π in a step-by-step manner. In every time step, a healthy agent follows its plan unless the next location it is planned to occupy is blocked by some other agent, in which case it waits in its place. A faulty agent behaves similarly, except that with probability p it experiences a fault. A fault is either a delay or an acceleration of 1-3 time steps, selected randomly according to a uniform distribution. If according to the value function, the cost of the current execution exceeds the failure threshold *th*, we create a BAMPEF problem with the corresponding plan, observations, and faulty events. We generated BAMPEF instances with plan lengths $y \in \{8, 10, 12\}$, agents $x \in \{8, 10, 12\}$, faulty agents $f \in \{3, 4, 5\}$, fault probabilities $p \in \{0.5, 0.7, 0.9\}$ and thresholds *th* $\in \{2, 3, 4\}$. An important factor that we wanted to evaluate is the influence of the number of fault events on the Shapley values algorithms. However, the simulation could not directly control the number of fault events. As a result, we repeated the above process for each configuration, until we achieved 30 BAMPEF instances with *fe* fault events, where *fe* $\in \{6, ..., 13\}$. In total, we got $58,320$ instances.

For every algorithm $A \in \{$DDBA,*Random*,*Gold*$\}$ and BAMPEF instance Π, we run A until it returned a solution, denoted $A(\Pi)$. The main metrics we considered are *runtime* in seconds and *error*. The runtime includes the time required for computing useful repairs. The error was computed as the Euclidean distance between the $A(\Pi)$ and *Gold*(Π). Note that the error for *Gold* is, of course, always zero. The real Shapley values are affected by the number of participating agents and may sum up to more than one. To allow a fair comparison across different experimental configurations, we normalized every Shapley value computation such that it always sums up to one.

4.2 Results

We first examine the impact of the useful repair cardinality (k) on the error of DDBA. Table 5 shows the error of DDBA for $k = 1, \ldots, 8$, for BAMPEF instances with 12 agents, plans of length 12, 5 faulty agents, 0.9 fault probability, failure threshold 4, and 10 faulty events. The results show that the error decreases fairly fast until $k = 5$, afterwhich the decrease in error slows down significantly. Since increasing k means higher runtime, we limited k to be at most 5 in the remaining experiments, and report on averages over all evaluated problem configurations.

Table 6 shows the runtime and error for DDBA, *Random*, and *Gold*. As expected, DDBA is much faster than *Gold*. For instance, even the slowest variant of DDBA, which when considering all the useful repairs up to $k = 5$, runs in 2.077 seconds on average, while *Gold* requires 7.024 seconds on average. Compared to *Random*, we observe that DDBA is faster than *Random* for $k \leq 4$. In terms of error, the error of DDBA is higher than *Random* for $k < 4$, and lower when $k \geq 4$. Thus, our results confirm the expected trade-off provided by the useful repair cardinality parameter k between runtime and error. In our set of experiments, however, setting $k = 4$ provides an effective middle-ground between runtime and error. Hence, in the next results, we fixed k to 4. We did not report the error for *Gold* since it is always zero.

Table 7 presents the results of DDBA, *Random*, and *Gold* when varying the number of faulty events (*fe*).

The first trend we observe is that the runtime of *Gold* increases exponentially with the number of faulty events. The runtime of *Random* also increases, but at a much lower rate than *Gold*, while the runtime of DDBA is even lower. For instance, while considering 13 fault events, DDBA runs 2 times faster than *Random* on average, and 50 times faster than *Gold*. This shows the efficiency of DDBA when considering a large amount of fault events.

In terms of error, we see that the error of both *Random* and DDBA increases very slightly with the number of faulty events, which suggests that *Random* and DDBA are not influenced much by the number of fault events in the system. This is specifically of interest since this means that DDBA is scalable for larger amounts of fault events. In addition, the error of DDBA is lower than the error of *Random*. Specifically, this difference is higher when considering small numbers of fault events. This suggests that for small systems, DDBA is preferable over *Random*.

The runtime of DDBA is strongly affected by the number of useful repairs it considers. To highlight this, Table 7 also shows the number of useful repairs considered by DDBA for a different number of

Useful Repair Cardinality	1	2	3	4	5	6	7	8
Average Error DDBA	0.355	0.229	0.138	0.107	**0.089**	**0.076**	0.069	0.066

Table 5. Average error of DDBA, with the increase of the useful repair cardinality (k).

Useful Repair Cardinality		1	2	3	4	5
Average Runtime (Seconds)	Gold	7.024				
	Random	1.074				
	DDBA	**0.006**	**0.036**	**0.167**	**0.588**	2.077
Average Error	Random	0.151				
	DDBA	0.376	0.257	0.170	**0.130**	**0.107**

Table 6. Average runtime and error of *Random* and DDBA with the increase in the useful repair cardinality.

Fault Events		6	7	8	9	10	11	12	13
Average Error	Random	0.147	0.146	0.148	0.149	0.153	0.154	0.153	0.156
	DDBA	**0.112**	**0.113**	**0.120**	**0.127**	**0.133**	**0.140**	**0.146**	**0.146**
Average Runtime (Seconds)	Gold	0.052	0.152	0.334	0.737	1.745	4.449	12.505	36.215
	Random	0.062	0.129	0.256	0.467	0.718	1.239	2.174	3.546
	DDBA	**0.038**	**0.090**	**0.180**	**0.363**	**0.517**	**0.785**	**1.107**	**1.623**
DDBA Useful Repairs		34.84	58.75	92.37	138.64	198.32	272.80	367.96	476.65

Table 7. Results for a different number of faulty events.

faulty events. Indeed, the runtime of DDBA is correlated with the number of useful repairs, which increases with the number of fault events. The relative relation between the number of fault events and the number of useful repairs somewhat conforms with the theoretical analysis in Section 3.5, albiet lower than predicted. For example, for 8 faulty events, with cardinality up to $k = 4$, the upper bound of useful repairs is $\binom{8}{4} \cdot 2^4 = 1120$, while the actual number is ~ 92. This is because many subsets of the fault events are not useful repairs.

5 Discussion

In this section we highlight some of the assumptions drawn throughout the paper, which we used to simplify the problem.

Simplification assumptions: Our approach uses a simple model of the system. This is a common practice when modeling a system, that is used to address the modeling of a real-world domain in a simple way [16]. In our work, such simplifications include discrete time steps, modeling plans as discrete series of locations, and setting the same plan length for all the agents. We also assume that in each time step the agents should advance to their next position (rotation and acceleration time are not modeled), and that an agent failure manifests as a delay or acceleration. Last, we assume agents do not contest positions, and instead one of the agents moves first. Such simplifications help with creating more clear model of the problem, but also introduce limitations that should be addressed for more specific implementations.

Observation assumption: We assume complete observation. This allowed us to derive relatively accurate fault events. This assumption was inspired by a real-world example of warehouse robots, where a database keeps logs of the robot movements, so the complete execution is known. In environments that require relaxation of such assumption, Shapley values might produce less accurate blame distribution. Our approach, however, might still give the same approximation to that (less accurate) distribution.

Complete knowledge of agents' plans: An important assumption that allows us to derive the fault events is the knowledge of the agents' plans in the system. Armed with this knowledge, when observing the execution, we are able to derive delay and acceleration faults. Relaxing such an assumption would raise the need to design different value functions.

Consistent plans: We assume that the plans are consistent; If executed without any faults, the agents would reach their planned final locations. It is natural to think about cases, where faulty planners may output faulty plans. In such cases, one may need to reason about the plan itself but may also need to use the blame attribution in order to enforce some agents to deliberately create 'repairing' fault events (i.e., if the plan leads to a collision, forcing agents to slow down or speed up might help).

6 Conclusions and Future Work

We defined the problem of attributing blame to fault events, a concept orthogonal to classic diagnosis in which components of a system are labeled "faulty" and "not faulty". We built on concepts from Game Theory to devise our proposed method, which is using Shapley values to attribute blame. We showed the challenges in using Shapley values in systems with an increasing number of fault events and surveyed a current approach of Shapley approximation. Our proposed method (DDBA) improves the surveyed method. By utilizing selection of subsets of the fault events, DDBA approximates better the Shapley values while improving run-time. Theoretical evaluation explains the run-time improvement while empirical evaluation demonstrates the expected results.

For future work, we plan to focus on three main directions: (1) showing the validity of our method for different environments, (2) improving our proposed method. and (3) Demonstrating the usefulness of our method. To address (1), we plan to test different fault settings, domains, and action models of the agents. To address (2) we plan to research more sophisticated selection functions, and try to apply the same paradigm (focusing on subsets) on different blame attributing methods, such as *core* [15] and *Banzhaf index* [1, 2]. To address (3) we intend to integrate our approach with existing diagnosis and replan methods.

Acknowledgements

This research was funded by ISF grant No. 1716/17, and by the ministry of science grant No. 3-6078.

References

[1] John F Banzhaf III, 'Weighted voting doesn't work: A mathematical analysis', *Rutgers L. Rev.*, **19**, 317, (1964).
[2] John F Banzhaf iII, 'One man, 3.312 votes: a mathematical analysis of the electoral college', *Vill. L. Rev.*, **13**, 304, (1968).
[3] Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D Procaccia, *Handbook of computational social choice*, Cambridge University Press, 2016.
[4] Javier Castro, Daniel Gómez, and Juan Tejada, 'Polynomial calculation of the shapley value based on sampling', *COR*, **36**(5), 1726–1730, (2009).
[5] Hana Chockler and Joseph Y Halpern, 'Responsibility and blame: A structural-model approach', *JAIR*, **22**, 93–115, (2004).
[6] Matthew Daigle, Xenofon Koutsoukos, and Gautam Biswas, 'Distributed diagnosis of coupled mobile robots', in *Proceedings 2006 IEEE ICRA, 2006.*, pp. 3787–3794. IEEE, (2006).
[7] Femke De Jonge, Nico Roos, and Cees Witteveen, 'Primary and secondary diagnosis of multi-agent plan execution', *AAMAS*, **18**(2), 267–294, (2009).
[8] Orel Elimelech, Roni Stern, Meir Kalech, and Yedidya Bar-Zeev, 'Diagnosing resource usage failures in multi-agent systems', *ESWA*, **77**, 44–56, (2017).
[9] GN Engelbrecht and AP Vos, 'On the use of the shapley value in political conflict resolution', *Scientia Militaria: South African Journal of Military Studies*, **37**(1), (2009).
[10] Samuel Ferey and Pierre Dehez, 'Multiple causation, apportionment, and the shapley value', *JLS*, **45**(1), 143–171, (2016).
[11] Gordon Fraser, Gerald Steinbauer, and Franz Wotawa, 'Plan execution in dynamic environments', in *IEA/AIE*, pp. 208–217. Springer, (2005).
[12] Meir Friedenberg and Joseph Y Halpern, 'Blameworthiness in multi-agent settings', in *AAAI*, volume 33, pp. 525–532, (2019).
[13] Christopher Frye, Colin Rowat, and Ilya Feige, 'Asymmetric shapley values: incorporating causal knowledge into model-agnostic explainability', *Advances in NeurIPS*, **33**, 1229–1239, (2020).
[14] Amirata Ghorbani and James Zou, 'Data shapley: Equitable valuation of data for machine learning', in *ICML*, pp. 2242–2251. PMLR, (2019).
[15] Donald B Gillies, 'Solutions to general non-zero-sum games', *Contributions to the Theory of Games*, **4**, 47–85, (1959).
[16] Alicja Gosiewska, Anna Kozak, and Przemysław Biecek, 'Simpler is better: Lifting interpretability-performance trade-off via automated feature engineering', *DSS*, **150**, 113556, (2021).
[17] Joseph Halpern and Max Kleiman-Weiner, 'Towards formal definitions of blameworthiness, intention, and moral responsibility', in *Proceedings of the AAAI*, volume 32, (2018).
[18] Joseph Y Halpern, *Actual causality*, 2016.
[19] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W Durham, and Nora Ayanian, 'Persistent and robust execution of mapf schedules in warehouses', *IEEE RA-L*, **4**(2), 1125–1131, (2019).
[20] Wolfgang Hönig, TK Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig, 'Multi-agent path finding with kinematic constraints', in *ICAPS*, volume 26, pp. 477–485, (2016).
[21] Franz Hubert and Svetlana Ikonnikova, 'Strategic investment and bargaining power in supply chains: A shapley value analysis of the eurasian gas market', *Humboldt University Berlin*, (2003).
[22] Kamal Jain and Mohammad Mahdian, 'Cost sharing', *AGT*, **15**, 385–410, (2007).
[23] Meir Kalech, 'Diagnosis of coordination failures: a matrix-based approach', *AAMAS*, **24**(1), 69–103, (2012).
[24] Meir Kalech and Gal A Kaminka, 'Towards model-based diagnosis of coordination failures', in *AAAI*, volume 5, pp. 102–107, (2005).
[25] Meir Kalech and Gal A Kaminka, 'On the design of coordination diagnosis algorithms for teams of situated agents', *Artificial Intelligence*, **171**(8-9), 491–513, (2007).
[26] Meir Kalech and Gal A Kaminka, 'Coordination diagnostic algorithms for teams of situated agents: Scaling up', *Computational Intelligence*, **27**(3), 393–421, (2011).
[27] Meir Kalech, Gal A Kaminka, Amnon Meisels, and Yehuda Elmaliach, 'Diagnosis of multi-robot coordination failures using distributed csp algorithms', in *AAAI*, pp. 970–975, (2006).
[28] Meir Kalech and Avraham Natan, 'Model-based diagnosis of multi-agent systems: A survey', in *AAAI*, volume 36, pp. 12334–12341, (2022).
[29] Eliahu Khalastchi and Meir Kalech, 'Fault detection and diagnosis in multi-robot systems: a survey', *Sensors*, **19**(18), 4019, (2019).
[30] Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Daniel Harabor, Peter J Stuckey, Hang Ma, and Sven Koenig, 'Scalable rail planning and replanning: Winning the 2020 flatland challenge', in *ICAPS*, volume 31, pp. 477–485, (2021).
[31] Hang Ma, TK Satish Kumar, and Sven Koenig, 'Multi-agent path finding with delay probabilities', in *AAAI*, volume 31, (2017).
[32] Shlomi Maliah, Guy Shani, and Roni Stern, 'Collaborative privacy preserving multi-agent planning', *AAMAS*, **31**(3), 493–530, (2017).
[33] Moshe Mash, Roy Fairstein, Yoram Bachrach, Kobi Gal, and Yair Zick, 'Human-computer coalition formation in weighted voting games', *ACM TIST*, **11**(6), 1–20, (2020).
[34] Roberto Micalizio, 'A distributed control loop for autonomous recovery in a multi-agent plan', in *IJCAI*, (2009).
[35] Roberto Micalizio and Pietro Torasso, 'Cooperative monitoring to diagnose multiagent plans', *JAIR*, **51**, 1–70, (2014).
[36] Jonathan Morag, Ariel Felner, Roni Stern, Dor Atzmon, and Eli Boyarski, 'Online multi-agent path finding: New results', in *Proceedings of SoCS*, volume 15, pp. 229–233, (2022).
[37] Avraham Natan and Meir Kalech, 'Privacy-aware distributed diagnosis of multi-agent plans', *ESWA*, **192**, 116313, (2022).
[38] Martin J Osborne et al., *An introduction to game theory*, volume 3, Oxford university press New York, 2004.
[39] Guillermo Owen, *Game theory*, Emerald Group Publishing, 2013.
[40] Raymond Reiter, 'A theory of diagnosis from first principles', *Artificial intelligence*, **32**(1), 57–95, (1987).
[41] Nico Roos and Cees Witteveen, 'Models and methods for plan diagnosis', *AAMAS*, **19**(1), 30–52, (2009).
[42] Alvin E Roth, 'Introduction to the shapley value', *The Shapley value*, 1–27, (1988).
[43] L Shapley, 'Quota solutions op n-person games1', *Edited by Emil Artin and Marston Morse*, 343, (1953).
[44] Lloyd S Shapley, 'In kuhn, hw & tucker, aw', *A Value for n-person Games*, 307–317, (1953).
[45] Yoav Shoham and Kevin Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*, Cambridge University Press, 2008.
[46] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al., 'Multi-agent pathfinding: Definitions, variants, and benchmarks', in *Twelfth Annual SoCS*, (2019).
[47] Hongtao Tang, Xiaoya Cheng, Weiguang Jiang, and Shouwu Chen, 'Research on equipment configuration optimization of agv unmanned warehouse', *IEEE Access*, **9**, 47946–47959, (2021).
[48] Alejandro Torreno, Eva Onaindia, Antonín Komenda, and Michal Štolba, 'Cooperative multi-agent planning: A survey', *ACM Computing Surveys (CSUR)*, **50**(6), 1–32, (2017).
[49] Alejandro Torreno, Eva Onaindia, and Oscar Sapena, 'Fmap: Distributed cooperative multi-agent planning', *Applied Intelligence*, **41**, 606–626, (2014).
[50] Gianluca Torta, Roberto Micalizio, and Samuele Sormano, 'Explaining failures propagations in the execution of multi-agent temporal plans', in *ICAPS*, pp. 2232–2234, (2019).
[51] Stelios Triantafyllou, Adish Singla, and Goran Radanovic, 'On blame attribution for accountable multi-agent sequential decision making', *Advances in NeurIPS*, **34**, 15774–15786, (2021).
[52] Glenn Wagner and Howie Choset, 'Path planning for multiple agents under uncertainty', in *ICAPS*, volume 27, pp. 577–585, (2017).
[53] Menahem E Yaari, 'Rawls, edgeworth, shapley, nash: Theories of distributive justice re-examined', *JET*, **24**(1), 1–39, (1981).
[54] H Peyton Young, 'Monotonic solutions of cooperative games', *International Journal of Game Theory*, **14**(2), 65–72, (1985).
[55] Jiawei Zhang, Zhiheng Li, Yidong Li, and Hairong Dong, 'A bi-level cooperative operation approach for agv based automated valet parking', *TR_C*, **128**, 103140, (2021).