Multi-Agent Planning and Diagnosis with Commonsense Reasoning

Tran Cao Son
stran@nmsu.edu
New Mexico State University
Las Cruces, NM, USA

William Yeoh
wyeoh@wustl.edu
Washington University in St. Louis
Saint Louis, MO, USA

Roni Stern
sternron@bgu.ac.il
Ben-Gurion University of the Negev
Beer Sheva, Israel

Meir Kalech
kalech@bgu.ac.il
Ben-Gurion University of the Negev
Beer Sheva, Israel

ABSTRACT

In multi-agent systems, multi-agent planning and diagnosis are two key subfields – multi-agent planning approaches identify plans for the agents to execute in order to reach their goals, and multi-agent diagnosis approaches identify root causes for faults when they occur, typically by using information from the multi-agent planning model as well as the resulting multi-agent plan. However, when a plan fails during execution, the cause can often be related to some commonsense information that is neither explicitly encoded in the planning nor diagnosis problems. As such existing diagnosis approaches fail to accurately identify the root causes in such situations.

To remedy this limitation, we extend the Multi-Agent STRIPS problem (a common multi-agent planning framework) to a Commonsense Multi-Agent STRIPS model, which includes commonsense fluents and axioms that may affect the classical planning problem. We show that a solution to a (classical) Multi-Agent STRIPS problem is also a solution to the commonsense variant of the same problem. Then, we propose a decentralized multi-agent diagnosis algorithm, which uses the commonsense information to diagnose faults when they occur during execution. Finally, we demonstrate the feasibility and promise of this approach on several key multi-agent planning benchmarks.

CCS CONCEPTS

• **Computing methodologies** → **Multi-agent systems**; **Multi-agent planning**; *Logic programming and answer set programming*; Cooperation and coordination; • **Hardware** → **Bug detection, localization and diagnosis**; • **Theory of computation** → **Distributed algorithms**.

DAI '23, November 30-December 3, 2023, Singapore, Singapore

ACM ISBN 979-8-4007-0848-0/23/11.
https://doi.org/10.1145/3627676.3627690

KEYWORDS

Multi-Agent Systems, Multi-Agent Planning, Multi-Agent Diagnosis, Commonsense Reasoning, Decentralized Algorithms, Answer Set Programming

ACM Reference Format:
Tran Cao Son, William Yeoh, Roni Stern, and Meir Kalech. 2023. Multi-Agent Planning and Diagnosis with Commonsense Reasoning. In *The Fifth International Conference on Distributed Artificial Intelligence (DAI '23), November 30-December 3, 2023, Singapore, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3627676.3627690

1 INTRODUCTION

Multi-agent planning – the ability of a group of autonomous agents to reason about their actions and identifying sequences of actions (i.e., plans) that lead them to their goals – is a core area of AI research at the intersection of automated planning and multi-agent systems. Applications of multi-agent planning are abound, ranging from robots navigating in autonomous warehouses today [20, 21, 41] to autonomous vehicles navigating on roads in the future [7, 32]. As is the case with any application with embodied agents, failures may occur during an agent's execution. When abnormal behaviors caused by failures are observed, *automated diagnosis* techniques [10, 33, 40] are used to identify the root cause of the abnormal behavior.

However, a key assumption in both multi-agent planning methods is that the problem is fully-specified – in the sense that everything that can affect the outcomes of actions in the problem, however unlikely, is defined. For example, consider the well-known Logistics domain [22], where a set of packages need to be moved from their initial to target locations using a given fleet of vehicles such as trucks, airplanes, etc. In this domain, the `drive(truck,origin,destination,city)` action will successfully move `truck` from `origin` to `destination` if the pre-conditions that `truck` is at `origin` and both `origin` and `destination` are in `city` and are connected by a road are satisfied. This implicitly assumes some *commonsense knowledge*, typically defined as knowledge about the world that all humans are expected to know. For example, it assumed that the drive action is successful only if the `truck` is not broken, the road between `origin` and `destination` is not blocked, etc.

To ensure conciseness in the planning problem specification and tractability of solution approaches, it is reasonable that one models only knowledge (commonsense or otherwise) that is assumed to

be relevant to the problem and ignores knowledge (much of which is commonsense) that is not relevant. However, when a plan fails during execution, the cause is often related to some factor that was not taken into consideration during the planning stage. For example, perhaps the `truck` failed to reach `destination` because the road between `origin` and `destination` is not traversable due to a flood, and the problem did not consider the possibility of a flood. As such, *automated diagnosis methods need to reason about commonsense knowledge that is not explicitly encoded in the classical planning problem.*

For automated diagnosis methods to perform commonsense reasoning, it must have access to commonsense knowledge that was not used in the planning stage. However, instead of explicitly representing all commonsense knowledge and providing it as input to diagnosis methods, we take a more sensible approach by considering only a subset of commonsense knowledge that may affect the planning problem. In this paper, we propose a *Commonsense Multi-Agent STRIPS* model, which extends the traditional Multi-Agent STRIPS model [4] by including commonsense fluents and commonsense axioms that may affect the classical planning problem. Then, with access to such commonsense knowledge, we propose commonsense multi-agent diagnosis algorithms that allow agents to collaboratively identify the root cause of abnormal behaviors in a decentralized manner. Finally, we demonstrate the feasibility and promise of this class of algorithms on several key multi-agent planning benchmarks.

2 BACKGROUND

We now provide some background on *multi-agent STRIPS* (MA-STRIPS), which we will later extend to a commonsense variant, and *automated diagnosis*, which we will rely on for our multi-agent diagnosis algorithm.

2.1 Multi-Agent STRIPS (MA-STRIPS)

An MA-STRIPS problem is defined by a tuple $\langle F, I, G, \{A_i\}_i \rangle$, where F is a set of fluents, I is the initial state, G is the goal state, and A_i is the set of actions that can be performed by agent i [4]. A multi-agent plan in MA-STRIPS is often assumed to be a sequence of actions $(a_0, \ldots, a_n)$ where each action a_j is an action of some agent i (i.e., $a_j \in A_i$). A plan is a solution if it transitions the initial state to the goal state. A solution is optimal if it has the fewest actions among all solutions.

A multi-agent plan can also be a sequence of *joint actions*, if concurrent execution is allowed. We will also consider MA-STRIPS extensions in which some fluents and actions are only observable by some agents (these are called private fluents and actions). In this type of privacy-aware MA-STRIPS problems [27], a multi-agent plan comprise of a public plan that includes the actions publicly known to all agents, and set of private plans for each agent that comprise the private actions they need to perform.

Note that in MA-STRIPS, actions' effects are deterministic. Richer multi-agent planning languages that support stochastic effects and observability exists, such as Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) [3]. While the theories and algorithms developed in the proposed research may carry over to such planning languages, we limit our scope to deterministic effects, which is a reasonable abstraction for many real-world problems that have been studied extensively in the automated planning and multi-agent systems communities.

2.2 Automated Diagnosis

A diagnosis problem arises when observations indicate that the system of interest is behaving abnormally. *Model-based diagnosis* (MBD) is a popular and principled approach for solving diagnosis problems that relies on a model characterizing the behavior of the diagnosed system. This model is referred to as the *system description*. A diagnosis problem in the MBD literature is commonly defined by a tuple $\langle SD, Comps, Obs \rangle$, where SD is the system description, $Comps$ is a finite set of components, and Obs is a collection of observations about the system [11, 12, 29]. The predicate $ab(c)$ denotes the fact that the component $c \in Comps$ is "abnormal" (or defective). An MBD problem arises when the assumption that all components are healthy is inconsistent with SD and Obs (i.e., when $\bigcup_{c \in Comps} \neg ab(c) \cup SD \cup Obs$ is inconsistent). A solution to an MBD problem is a *diagnosis*, which can be defined as a set of components $\Delta \subseteq Comps$ such that the assumption that these components is faulty and all other components are healthy is consistent with SD and Obs:

$$\bigcup_{c \in Comps \setminus \Delta} \neg ab(c) \bigcup_{c \in \Delta} ab(c) \cup SD \cup Obs \text{ is consistent} \qquad (1)$$

The goal of MBD algorithms is often to identify a minimal diagnosis or preferred diagnosis in accordance to some criteria.

In the classical work introducing diagnosis from first principles, SD is a theory representing the relationship between components and state of the system and Obs consists of observations related to a single state of the system [11, 12, 29]. However, MBD has been considered for many other, significantly richer, types of system description. For example, MBD has been considered for system descriptions that include knowledge about the components' fault modes (this is known as a strong fault model), where a diagnosis specifies not only which components are faulty but also what type of fault they exhibit [6, 13, 34]. MBD has also been considered for discrete-event and dynamic systems, where SD and Obs encode a transition system and observations along a trajectory of the system, respectively [1, 2, 15, 35]. In this case, a diagnosis is a set of component-time pairs (c, t), where $c \in Comps$ and $t \geq 0$, and $ab(c, t)$ indicates that component c is defective (fails) from time step t. Prior work also applied MBD for hybrid and distributed systems, where the system description composes of a set of variables, parameters, equations, inputs, and outputs, and observations are measurements of outputs. In such a case, a diagnosis is a fault representing the deviation of parameters from their nominal values [5].

More related to the proposed research is prior work that applied MBD to diagnose a *multi-agent system* [39]. Typically, two types of failures have been investigated: Plan failures and coordination failures. In the former, faults occur due to failed execution of some actions in the plan. In the latter, faults occur due to disagreements of the agents on key components of their joint task. Appropriate algorithms have been proposed for each type of failures.

3 COMMONSENSE MULTI-AGENT STRIPS

Recall that for automated diagnosis methods to perform commonsense reasoning, it must have access to commonsense knowledge that was not used in the planning stage, but could have affected the execution of the plan. Towards that end, we propose the *Commonsense MA-STRIPS* (CMA-STRIPS) model, an extension of the classical MA-STRIPS model [4] that includes *commonsense fluents* and *commonsense axioms* that may affect the classical planning problem.

Before describing the CMA-STRIPS model, we first define a commonsense fluent as a fluent that is not affected by any action in the corresponding classical planning problem. More formally:

Definition 3.1 (Commonsense Fluent). A fluent f is a *commonsense fluent* iff $\forall a \in \{A_i\}_i : f \notin \texttt{eff}(a)$, where $\{A_i\}_i$ is the set of actions in the corresponding classical MA-STRIPS problem.

In other words, these are fluents that represent aspects of the planning problem that is presumed to remain unchanged throughout the execution of the plan. For example, the commonsense variant of the LOGISTICS domain may include a fluent ¬flood-l1 representing the fact that the location $l1$ is not flooded.

We now define the CMA-STRIPS model, which is defined by the tuple $\langle F, I, G, \{A_i\}_i, P\rangle$, where $\langle F, I, G, \{A_i\}_i\rangle$ is similar to its corresponding regular MA-STRIPS problem, except that they may include commonsense counterparts:

- $F = F^r \cup F^c$ is a set of fluents, where F^r is the set of *regular* fluents in the MA-STRIPS problem and F^c is the set of *commonsense* fluents.
- $I \subseteq F$ and $G \subseteq F$ are the initial and goal states, which may now include commonsense fluents.
- A_i is the set of actions that can be performed by agent i, whose pre-conditions $\texttt{pre}(a)$ of action $a \in A_i$ may now include commonsense fluents. However, by Definition 3.1, the effects $\texttt{eff}(a)$ of action a do not include commonsense fluents. In addition, we assume that the set A_i might also include statements declaring that certain action of agent i can interfere or is in conflict with some actions executed by others if they were to be executed in parallel. When two actions interfere, the planning algorithm should not allow them to be executed in parallel; when two actions are in conflict and one fails then the other also fails.
- $P = P^r \cup P^c$ is a set of axioms representing the relationships between the fluents F, where P^r is the set of *regular* axioms that involve regular fluents only and P^c is the set of *commonsense* axioms that involve commonsense fluents as well.

While the classical MA-STRIPS problem definition do not include regular axioms as they arguably can be compiled away, we choose to explicitly include axioms in the definition as commonsense axioms represent intuitive relationships in a concise and clear way. For example, the commonsense axiom below represents the relationship that if location $l1$ is flooded and it is connected with location $l2$ by road, then the road between the two locations is now blocked:

$$\texttt{flood-l1} \wedge \texttt{connected-l1-l2} \rightarrow \texttt{blocked-l1-l2} \tag{2}$$

With this problem definition in hand, we further restrict the problem definition to have the following property:

Property 1. *If all commonsense fluents in preconditions of actions hold in the initial state, then a solution for an MA-STRIPS problem is also a solution for the commonsense variant of the same problem.*

Recall that a solution is a plan that is feasible in the planning problem and it transitions the initial state to the goal state. It is reasonable to assume that the designers of an MA-STRIPS problem have incorporated *all* knowledge (commonsense or otherwise) that is relevant to the planning problem. Therefore, the fluents that are in preconditions of actions and whose truth value can change during the execution of a plan must be regular fluents. All other fluents in preconditions, which are the commonsense fluents, are therefore assumed to always hold. Consequently, Property 1 holds because if an action is feasible in an MA-STRIPS problem, it must also be feasible in the corresponding Commonsense MA-STRIPS problem. As a corollary to Property 1, one can then use any MA-STRIPS planner to solve CMA-STRIPS problems.

4 COMMONSENSE MULTI-AGENT DIAGNOSIS

We now describe the observation function of the agents before formulating the commonsense multi-agent diagnosis problem and describing our decentralized solution approach for diagnosing the root causes of abnormal behaviors in this problem.

4.1 Observation Function of Agents

Before describing our decentralized multi-agent diagnosis approach, we must first describe the *knowledge base* (KB) of each agent (i.e., the fluents, actions, and axioms that it knows and can observe). As a range of possibilities exist, we assume the availability of a domain-dependent mapping function:

$$M : \mathcal{A} \times F \cup P \cup \{A_i\}_i \rightarrow \{0, 1\} \tag{3}$$

that indicates whether an agent $k \in \mathcal{A}$ knows about a fluent $f \in F$, axiom $p \in P$, or action $a \in A_i$ of agent i.

There is a wide range of possible observation functions for agents, including:

- **Local Observations:** On one end of the spectrum, minimally, all agents must be able to observe their own actions and the fluents that are in the preconditions and effects of those actions. In other words, for each agent $i \in \mathcal{A}$, $M(i, a) = M(i, f) = 1$ for all actions $a \in A_i$ and fluents $f \in \texttt{pre}(a) \cup \texttt{eff}(a)$.

  Using the same LOGISTICS domain example again, if agents have *local observations* only, then truck $t1$ can observe fluents related to its own drive(t1,l1,l2,c) action for any location pairs l1 and l2 in city c; the pre-condition at-t1-l1 indicating that the truck is at l1 and the preconditions connected-l1-l2 and ¬blocked-l1-l2 indicating that the two locations are connected by a road that is not blocked; and the effects ¬at-t1-l1 and at-t1-l2 indicating that the truck is no longer at $l1$ and is at $l2$ instead. Similarly, it can also observe fluents related to its own load(o1,t1,l1) action, which loads object $o1$ onto truck $t1$ at location $l1$; pre-conditions at-o1-l1 and at-t1-l1, indicating both $o1$ and $t1$ are at $l1$; and effects ¬at-o1-l1 and in-o1-t1, indicating that $o1$ is no longer at $l1$ but is inside $t1$ instead.
- **Global Observations:** On the other end of the spectrum, all agents have global observation and are aware of all actions and fluents in the problem. In other words, for each agent $i \in \mathcal{A}$,

$M(i, a) = M(i, f) = 1$ for all actions $a \in A_i$ and fluents $f \in F$. This assumption is valid in applications, such as automated warehouses [21, 42] where there is a global observer that can communicate their observations with all agents.

In this paper, we make the most restrictive assumption and assume that all agents have local observations only. As such, our proposed diagnosis algorithm (described later) will require agents to coordinate and communicate with each other to diagnose faults that arise. Note that if agents have global observations, then diagnosing faults are trivial since all agents have complete knowledge. Finally, we also assume that an agent knows and can reason about an axiom if all the fluents in the axiom are observable to the agent. In other words, for each agent $i \in \mathcal{A}, M(i, p) = 1$ for all axioms $p \in P$ iff $M(i, f) = 1$ for all fluents $f \in F(p)$, where $F(p)$ is the set of fluents in p.

4.2 Problem Formulation

Given a plan π of an MA-STRIPS problem, a *realization* R of the plan is a trajectory $R = (s^0, a^1, s^1, \ldots, a^n, s^n)$, where $s^0 = I$ is the initial (joint) state and $a^k = \times_i a_i^k$ is the joint action of all the agents at time step k. Note that s^n may not be a (joint) goal state as the agents may fail to reach a goal state should some of their actions fail.

An *agent realization* R_i of the realization for an agent i is the projected realization with respect to that agent: $R_i = (s_i^0, a_i^1, s_i^1, \ldots, a_i^n, s_i^n)$, where $s_i^0 = I_i$ is the initial state that is observable to agent i and $a_i^k \in A_i$ is the action of agent i at time step k.

An action $a \in A_i$ of agent i is said to have *failed* in a realization of plan π if there exists a transition (s, a, s') in the agent realization such that s does not satisfy the pre-conditions $\texttt{pre}(a)$ or s' is inconsistent with the effects $\texttt{eff}(a)$. Similarly, an action $a \in A_i$ of agent i is said to be *faulty* if there exists a transition (s, a, s') such that s satisfies the pre-conditions $\texttt{pre}(a)$, but s' is still inconsistent with the effects $\texttt{eff}(a)$. Note that all failed actions are also faulty actions, but not vice versa. Finally, all actions that are not faulty are said to be *healthy*.

A *diagnosis* for a plan π and joint observation $\Omega = \times_i \Omega_i$ is a subset of agent actions A_{faulty} such that (1) there exists a realization R whose agent realizations R_i correspond to the observations Ω_i for all agents i and (2) *exactly* all actions $a \in A_{faulty}$ are faulty.

Definition 4.1 (Commonsense Multi-Agent Diagnosis). A Commonsense Multi-Agent Diagnosis problem is defined by a tuple $\langle \Pi, \pi, \Omega \rangle$, where Π is a CMA-STRIPS problem, π is a solution to Π, and Ω is the observation for some realization of π. A solution to the problem is a diagnosis for π and Ω.

4.3 MAD-DR Algorithm

We now describe our *Multi-Agent Diagnosis with Decentralized Reasoning (MAD-DR) algorithm*, a decentralized multi-agent algorithm that finds a commonsense multi-agent diagnosis during the execution of a multi-agent plan. Algorithm 1 shows its pseudocode for each "self" agent i, where it takes as inputs the CMA-STRIPS problem P, its plan π_i, and its set of neighboring agents N_i (i.e., the set of agents that it can communicate with).

Algorithm 1 MAD-DR Algorithm

Input: CMA-STRIPS problem Π, plan π_i of self agent i, and its set of neighboring agents N_i

1: $Ans \leftarrow \emptyset$ % answers maintained by the agent
2: $L \leftarrow \emptyset$ % inquiries sent to/received from neighbors
3: $LF \leftarrow \emptyset$ % failure answers received from neighbors
4: $LP \leftarrow \emptyset$ % discrepancies inquired by neighbors
5: $s_i^0 \leftarrow$ initial observable state of the agent
6: **for all** time steps $0 \leq t < length(\pi_i)$ **do**
7: **if** preconditions a_i^t are satisfied in s_i^t **then**
8: $s_i^{t+1} \leftarrow$ execute a_i^t from π_i in current state s_i^t
9: **end if**
10: **if** preconditions of a_i^t are not satisfied in state s_i^t OR effects of a_i^t are not reflected in state s_i^{t+1} **then**
11: compute $Q_i(t)$ and $Own_i(t)$
12: $Ans \leftarrow Ans \cup \{(i, f, v, v', t) \mid (f, v, v') \in Own_i(t)\}$
13: **if** $Q_i(t) \setminus Own_i(t) \neq \emptyset$ **then**
14: **for all** $(f, v, v') \in Q_i(t) \setminus Own_i(t)$ **do**
15: $i[inquiry(f, v, v', t)]k$ for $k \in N_i$
16: add (i, k, f, v, v', t) to L for $k \in N_i$
17: **end for**
18: **end if**
19: break % stop executing plan π_i
20: **end if**
21: **end for**

Algorithm 2 When Receive Message(M)

1: **if** $M = n[inquiry(f, v, v', t)]i$ and $(f, v, v', t) \notin LP$ **then**
2: % receives a new inquiry
3: $Ans, L, LF, LP =$ **query-1**$((f, v, v', t), Ans, L, LF, LP)$
4: **else if** $M = n[inquiry(f, v, v', t)]i$ and $(f, v, v', t) \in LP$ **then**
5: % receives a repeated (old) inquiry
6: $Ans, L, LF, LP =$ **query-2**$((f, v, v', t), Ans, L, LF, LP)$
7: **else if** $M = k[answer(agt, f, v, v', t')]i$ **then**
8: % receives a positive answer
9: $Ans, L, LF, LP =$ **positive**$((agt, f, v, v', t'), Ans, L, LF, LP)$
10: **else if** $M = k[answer(\perp, f, v, v', t)]i$ **then**
11: % receives a negative answer
12: $Ans, L, LF, LP =$ **negative**$((\perp, f, v, v', t), Ans, L, LF, LP)$
13: **end if**

At the start, the agent initializes its sets Ans, L, LF, and LP (Lines 1-4) that will be used for diagnosis (they will be described in detail later) as well as initial observable state s_i^0 (Line 5). Then, at each time step t, if the preconditions of its current action a_i^t are satisfied by its current state s_i^t, then it executes the action and transitions to the next state s_i^{t+1} (Lines 7-9). If the preconditions are not satisfied or (in the case where the preconditions are satisfied) the effects of the action are not reflected in the next state (i.e., there is a discrepancy between the actual effect and expected effect in the state), then it means that the action a_i^t is faulty. Consequently, the agent starts a diagnosis process (Lines 10-20). If the action is not faulty, then the agent repeats this process and attempts to execute its actions in its plan π_i until it successfully reaches its goal.

Note that we assume that an agent with a faulty action will stop executing its plan after detecting and diagnosing the fault (Line 19).

We now describe the diagnosis process, which is done using answer set programming. A diagnosis is needed when an action a_i^t is faulty, which can arise due to one of the following two conditions:

- **Condition 1:** Its preconditions are not satisfied by the current state s_i^t.
- **Condition 2:** Its preconditions are satisfied by the current state s_i^t, but its expected effects are not reflected in the next state s_i^{t+1}.

In both cases, the condition can be generalized to one where there is a discrepancy in one (or more) fluent f (that is related to the preconditions/effects) in the current/next state having a value v instead of the expected value $\tilde{v}$. Our MAD-DR algorithm makes use of this generalization, which we now describe.

4.3.1 High-Level Description. At a high level, when an agent starts a diagnosis, it first checks whether it's own action a_i^t is *abnormal* (Condition 2) or not (Condition 1).

- If it is abnormal, then it knows that its action is part of the cause of the fault. This cause is possibly incomplete because an action by a different agent may also be abnormal and is contributing to the fault. Thus, the agent needs to check if other agents also have abnormal actions that affected the next state s_i^{t+1}.
- If it is not abnormal, then the fault lies with a different agent. Thus, the agent needs to check if other agents have abnormal actions that affected the current state s_i^t.

To do so, the agent sends *inquiry* messages to its neighboring agents (e.g., agents within its communication range) asking if its neighbors know why one (or more) fluent f is having a value v instead of the expected value $\tilde{v}$.

When an agent receives such a message, if it knows the reason (for example, the agent with an abnormal action knows that its effects did not materialize), then it replies with a *positive answer* identifying the faulty agent. If it does not know the reason, then it will propagate the inquiry to its neighboring agents. If it receives a positive answer from one of its neighboring agents, it forwards that positive answer to the agent that sent the inquiry message. If it does not know the reason and it does not have any neighboring agents other than the agent that sent the inquiry message, the agent replies with a *negative answer* indicating that it does not know the faulty agent. If an agent receives negative answers from all its neighboring agents, then it replies with a negative answer to the agent that sent the inquiry as well.

Under the assumption that the communication graph of the agents is a (single) connected graph (i.e., there are no disjointed subgraphs), then the inquiring agent is guaranteed to identify the faulty agent since, in the worst case, the faulty agent will reply with a positive answer to its neighboring agent, who will propagate that answer back to the original inquiring agent. For efficiency, agents that received the positive answer along the path from the faulty agent to the inquiring agent will store the faulty information (e.g., fluent f has value v instead of v' at time step t because of agent j). That way, if another agent inquires about the same fault, the agent can reply with the information in a positive answer immediately instead of propagating to the faulty agent and back.

Algorithm 3 query-1$((f, v, v', t), Ans, L, LF, LP)$

add (f, v, v', t) to LP
add (n, i, f, v, v', t) to L
if $\exists agt, t'.(agt, f, v, v', t') \in Ans$ **then**
  $i[answer(agt, f, v, v', t')]n$
else
  $N = N_i(t) \setminus \{p, q \mid (p, q, f, v, v', t) \in L\}$
  if $N = \emptyset$ **then**
    $i[answer(\perp, f, v, v', t]n$
  else
    $i[inquiry(f, v, v', t)]k$ for $k \in N$
  end if
end if
return Ans, L, LF, LP

Algorithm 4 query-2$((f, v, v', t), Ans, L, LF, LP)$

add (n, i, f, v, v', t) to L
if $\exists agt, t'.(agt, f, v, v', t') \in Ans$ **then**
  $i[answer(agt, f, v, v', t')]n$
end if
return Ans, L, LF, LP

Algorithm 5 positive$((agt, f, v, v', t), Ans, L, LF, LP)$

add (agt, f, v, v', t') to Ans
$N = \{x \mid x \in N_i(t), (p, q, f, v, v', t) \in L, x \in \{p, q\}\}$
$i[answer(agt, f, v, v', t')]k$ for $k \in N$
return Ans, L, LF, LP

Algorithm 6 negative$((\perp, f, v, v', t), Ans, L, LF, LP)$

add (k, v, v', t) to LF
$NF = \{x \mid (x, v, v', t) \in LF\}$
if $NF = N_i(t)$ **then**
  $N = \{x \mid x \in N_i(t), (p, q, f, v, v', t) \in L, x \in \{p, q\}\}$
  $i[answer(\perp, f, v, v', t)]x$ for $x \in N$
  add (i, v, v', t) to LF
  add $(\perp, f, v, v', t)$ to Ans
end if
return Ans, L, LF, LP

Additionally, due to the decentralized nature of the algorithm, multiple agents can initiate diagnoses for different faults concurrently. Therefore, a single agent may be involved in multiple diagnoses in parallel. Further, we assume that agents will continue to execute their plans as long as their actions are not faulty. Therefore, it is possible for several faulty agents to have stopped and are either diagnosing their faults or have identified their faults, while other non-faulty agents continue to execute their plans. These two characteristics of MAD-DR differ from most existing multi-agent diagnosis algorithms, which we will further elaborate in the Related Work section (see Section 5).

4.3.2 Detailed Description. We now describe the pseudocode in more detail, especially the diagnoses component and the notations that we use.

When the self agent i diagnoses a fault at time step t, it initializes $Q_i(t)$, which is the set of discrepancies of the form (f, v, v') that indicates that f has the value v but should have the value v', and $Own_i(t)$, which is the set of discrepancies in $Q_i(t)$ that it knows is due to one of its actions. In other words, if the agent has an abnormal action a_i^t (Condition 2), then $Own_i(t) = Q_i(t)$. Otherwise (Condition 1), $Own_i(t) = \emptyset$. The agent also updates the set of answers *Ans* it maintains. Each element in *Ans* has the form (a, f, v, v', t), which means that agent a is responsible for the discrepancy (f, v, v') at time step t.

We write " $i[inquiry(f, v, v', t)]k$" to indicate that the self agent i sends to its neighboring agent k an inquiry on the discrepancy in the value of fluent f (v instead of v'). As we assume that communication is perfect, this also means that agent k receives a message $inquiry(f, v, v', t)$ from agent i. In response to an inquiry $inquiry(f, v, v', t)$ of agent i, an agent k can reply with a message of the form "$k[answer(a, f, v, v', t)]i$" indicating that it knows that agent a is responsible for the difference in value of f at time t; or "$k[answer(\perp, f, v, v', t)]i$" indicating that agent k and all its neighboring agents do not know which agent is responsible for it.

Agent i also maintains a list L of discrepancies and agents that are interested in learning about the causes of these discrepancies. Similarly, it also maintains a list LF of failure answers received from neighboring agents and a list LP of discrepancies that the agent received from neighboring agents.

When an agent receives a diagnosis message, it executes Algorithm 2. If it receives a new inquiry for the first time, then it executes the **query-1** function (Algorithm 3). If the agent is the faulty agent, then it replies with a positive answer identifying itself as the faulty agent (Lines 3-4). If it does not have any neighboring agents aside from those that sent it the inquiry, then it replies with a negative answer (Lines 7-8). Otherwise, it propagates the inquiry to its neighboring agents (Lines 9-10).

If it receives a repeated (old) inquiry, then it executes the **query-2** function (Algorithm 4). If the previous inquiry has been completed and the faulty agent has been identified, then the agent replies with a positive answer identifying the faulty agent. If the previous inquiry hasn't been completed, then the agent waits for that inquiry to be completed, after which the inquiring agent will receive an answer.

If it receives a positive answer, then it executes the **positive** function (Algorithm 5), which propagates the positive answer back to its neighboring agents that sent the inquiry for that answer.

Finally, if it receives a negative answer, then it executes the **negative** function (Algorithm 6). If the agent has received negative answers from all its neighboring agents (sans the neighboring agents that sent the inquiry), then it propagates the negative answer back to its neighboring agents that sent the inquiry.

5 RELATED WORK

Diagnosis of multi-agent systems (MAS) has been extensively studied in various settings and under different sets of assumptions. For a comprehensive discussion of prior work in this field, see a recent survey by Kalech and Natan [19].

Early work by Micalizio et al. [24] used causal models of failures and system behavior to detect failures online and invoke a diagnosis engine to isolate their root causes. Similarly, de Jonge et al. [9] proposed to first detect which actions have failed, referred to as the *primary diagnosis*, and then isolate the root cause of these failures, referred to as *secondary diagnosis* [9].

The literature on diagnosis of MAS focused on two types of faults: coordination faults [8, 17, 18, 30] and plan-related faults [14, 25, 31, 36, 37]. The root cause of coordination faults is a conflict between the beliefs of the agents, while the root cause of plan-related faults is an agent-intrinsic reason, e.g., a mechanical failure. Our work falls under the category of plan-related faults.

Most of the existing work on diagnosing MAS assume that while the agents may not have a centralized controller, the diagnosis process is centralized [9, 16, 23, 24, 28, 36–38]. Kalech et al. [18] addressed the problem of distributed diagnosis using a Distributed Constraint Satisfaction Problem (DisCSP) solver. However, they focused on coordination faults. Daigle et al. [8] proposed a distributed diagnosis algorithm but they assumed a small number of agents and also focused on coordination faults. Roos [30] applied a distributed reinforcement learning mechanism to respond to observed failures in multi-agent systems. Yet, their focus is on how to adapt the existing plan to overcome the failure, not on the diagnosis aspect. Also, they assumed the agents are self-interested, while in our case they are collaborative.

Researchers have proposed a distributed approach to diagnose multi-agent systems for cases where the agents are collaborative but still seek to preserve some of their information private [25]. They used a combination of model-based diagnosis and DisCSP, sending partial beliefs and explanations to each other until a diagnosis is reached. An alternative approach to distributed diagnosis of MAS is based on Spectrum-Based Fault Localization (SFL) [26]. While it scales well and is fairly general, unlike our proposed work, it cannot utilize knowledge about the agents plan or action models.

6 EXPERIMENTAL EVALUATION

We now describe our experimental evaluations.

6.1 Domains

We experiment with the logistics domain [22] and the multi-agent pathfinding (MAPF) domain [42]. In the logistics domain, we consider airplanes and trucks as agents. As airplanes can fly between cities, we assume that each airplane can communicate with (and knows the existence of) all other airplanes and all trucks (and vice versa). However, as each truck can only move within the city it is in, we assume that each truck can communicate with other trucks within the same city only. In other words, trucks in different cities cannot communicate with each other. In the MAPF domain, we assume that each agent can communicate with (and knows the existence of) all other agents in the problem since it is often assumed that every agent can move within a same set of locations. We experiments with the following domains:

- **Logistics** (two cities, one airplane, two trucks, six objects/packages): In this problem, the initial state is given in Figure 1 (left) with truck `t1` and objects `obj11`, `obj12`, `obj13` are in `pos1`; truck `t2` and objects `obj21`, `obj22`, `obj23` are in `pos2`; airplane `apn1` is at airport `apt2`; and static information such as position `pos1` and airport `apt1` are in city 1 and `pos2` and airport

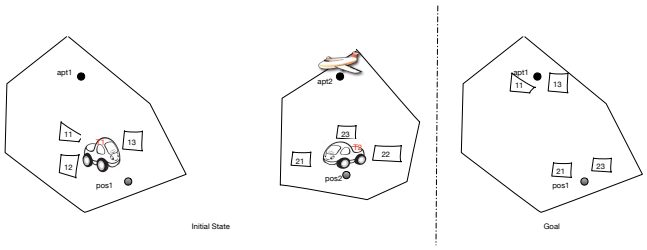

Figure 1: Logistics

apt2 are in city 2. Figure 1 (right) shows the goal, which is to have objects obj11 and obj13 at airport apt1 and objects obj21 and obj23 at position pos1.

It is easy to see that this problem needs at least 13 steps to solve: Truck t2 needs to load objects obj21 and obj23, drive to the airport apt2, unload the two objects (5 steps); Airplane apn1 then needs to load the two objects, fly to airport apt1, and unloads them (5 steps); Truck t1 then needs to load the two objects, drive to position pos1, and unload them (5 steps). In the meantime, truck t1 can deliver the two objects obj11 and obj13 to the airport apt1. Note that because we allow agents to execution actions in parallel, some actions that are required to complete the goal can overlap. Specifically, the last step of truck t2 and the first step of airplane apn1 can be done in parallel. Similarly, the last step of airplane apn1 and the first step of truck t1 can be done in parallel. Therefore, the optimal plan length for this goal is 13.

- **Multi-Agent Pathfinding (MAPF)** (3 agents in a 9×9 grid-world): To ease the creation of situations that require diagnosis, we force the paths to the goals of all agents to intersect at the center of the grid by blocking all but the middle cell of the middle column of the grid and generate the initial and goal of every robot on opposite sides of the middle column (see Figure 2).

6.2 Implementation

The system[1] is implemented using SWI-Prolog[2] and clingo.[3] Specifically, the environment simulator and the agent controller are written in SWI-Prolog and the computation of the state of the worlds after the execution of actions by agents, the diagnosis module, the computation of plans implemented by different clingo modules.

The environment simulator is responsible for computing the next state of the world by receiving the action occurrences from the agents and calling the clingo module that computes the next state of the world. The environment also randomly generates errors, preventing some actions to be successfully completed.[4] The generated errors need to be taken into consideration in the next state computation. Naturally, there are different ways to deal with unsuccessful execution of an action, such as ignoring it and continuing with the next action; or just abandoning the execution of the plan. In this paper, we adopt the view that the latter view: Once the execution of an action by an agent is unsuccessful, then all subsequent actions will be unsuccessful. This view is influenced by the MAPF domain in which an error occurrence often means that the robot is out of order. The simulator is also responsible for providing the agents with their local observations (see Section 4.1). For example, in the logistics domain, truck t1 does not know about the existence of truck t2 and, therefore, will not know about the truth value of the fluent $at(t2, pos2)$ that indicates that t2 is at pos2.

[1] Available at https://github.com/tcson62/dai.
[2] https://www.swi-prolog.org/
[3] https://potassco.org/
[4] For simplicity, we assume that this is provided as an input to the simulator.

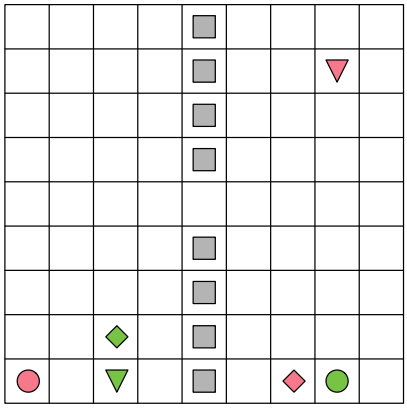

Figure 2: MAPF Configuration (Red/Green: start/end positions of agents)

The agent controller implements the main algorithm MAD-DR and all the necessary procedures described in Section 4.3. The communication between agents is facilitated by the tipc library provided by SWI-Prolog. The agent controller is responsible for executing the agent's plan. At each step, it sends to the simulator the action that needs to be executed; the agent controller receives the observations, compares them with its hypothetical state, identifies the need for diagnoses, and communicate with other agents to identify the cause if a diagnosis is needed.

The clingo modules have been developed so that it can handle commonsense fluents. For example, the default value *true* of a commonsense fluent f is encoded by an the rule $h(f,t) \leftarrow commonsense(f),\ not\ ab(f,t)$ and used in these modules. Abnormal situations (atoms of the form $ab(f,t)$) can be randomly generated and integrated with the error generation module of the simulator.

6.3 Experimental Results

To set up the experiments, we use a clingo module to generate valid concurrent plans for the agents. For example, Table 1 shows the plan for the three agents in the *Logistics* problem where '–' denotes that the agent is idle. The actions are simplified to ease the presentation. The clingo module computes the plan with the minimal horizon for all agents to achieve the goal of the problem. However, it does not optimize the individual plans, i.e., some of the agents might execute spurious actions (e.g., the airplane does not need to fly back and forth between two airports). We now describe the results of our experiments for each of our configuration domains.

Table 1: Plans for Logistics

#	Airplane 1	Truck 1	Truck 2
1	fly(apt2, apt1)	–	load(obj23, pos2)
2	fly(apt1, apt2)	load(obj13, pos1)	load(obj21, pos2)
3	fly(apt2, apt1)	load(obj11, pos1)	drive(pos2, apt2)
4	fly(apt1, apt2)	drive(pos1, apt1)	unload(obj23, apt2)
5	load(obj23, apt2)	unload(obj11, apt1)	unload(obj21, apt2)
6	load(obj21, apt2))	unload(obj13, apt1)	drive(apt2, pos2)
7	fly(apt2, apt1)	drive(apt1, pos1)	–
8	unload(obj21, apt1)	drive(pos1, apt1)	–
9	unload(obj23, apt1)	load(obj21, apn1)	–
10	load(obj13, apt1)	load(obj23, apn1)	–
11	unload(obj13, apt1)	drive(apt1, pos1)	–
12	load(obj13, apt1)	unload(obj21, pos1)	load(obj22, pos2)
13	unload(obj13, apt1)	unload(obj23, pos1)	–

6.3.1 Logistics. To see the cascading effects of action failures, we decided to test with the case that truck `t2` fails at timestep 1. Following our choice described earlier, this implies that all its actions (in subsequent timesteps) will be unsuccessful. The external input to the simulator contains one atom of the form `ab("t2", 1)` that indicates that action #1 of agent truck `t2` is abnormal (i.e., failed). We make the following observations:

- Truck `t2`:
  - It notices that its first three actions (#1, #2, #3) failed (their preconditions are satisfied but their effects did not materialize).
  - It realizes that the next two actions (#4 & #5) also failed but these failures are consequences of its own previous failed actions.
  - It also notices that its last action (#12) failed (the precondition is satisfied but its effects did not materialize).
  - It does not send any diagnosis inquiries to other agents.
- Airplane `apn1`:
  - The agent notices that it needs to start a diagnosis at step #5, when its loading action failed; it sends an inquiry about the reason for the missing object `obj23` in airport `apt2` to both trucks `t1` and `t2`.
  - The agent receives the answer from truck `t1` that it does not know the reason for the missing object.
  - The agent also receives a message from truck `t2` indicating that the reason is because some of its actions failed, namely the loading of the object at position `pos2` (#1) and the unloading of the object at airport `apt2` (#4).
  - The agent also realizes its action #6 also failed and starts a diagnosis process to identify the reason for the missing object `obj21` in airport `apt2`; similar to the other missing object, it learns that the reason is because of truck `t2`.
  - Finally, the agent realizes that its actions in steps #8 and #9 also failed, but realizes that it is a consequence of the failures of its own previous actions #5 and #6 and, thus, does not send any diagnosis inquiries to other agents.
- Truck `t1`:
  - The agent does not report any need for diagnosis until step #9.
  - The agent sends an inquiry to airplane `apn1` about the missing object `obj21` at airport `apt1`.
  - The agent receives a response from airplane `apn1` that the reason is because some of its actions failed, namely the loading of the object at airport `apt2` (#5) and the unloading of the object at airport `apt1` (#8).
  - The agent also realizes its action #10 also failed and starts a similar diagnosis to identify the reason for the missing object `obj23` in airport `apt1`; similar to the other missing object, it learns that the reason is because of airplane `apn1`.
  - Finally, the agent needs a diagnosis at steps #12 and #13, but realizes that it is a consequence of the failures of its own previous actions #9 and #10 and, thus, does not send out any diagnosis inquiries to other agents.

Overall, the number of messages sent for diagnosis is minimal (2 from truck `t1` and 4 from airplane `apn1`). We also ran the experiment with other configurations: 1 error, 2 errors, and 3 errors. We observe the similar patterns for this experiment and omit the description for brevity.

6.3.2 MAPF. In this domain, we inject an error for the first agent that *moves out* of the center cell in the grid when traversing from one side of the grid to the other. As a result, the agent will remain in that center cell and *block* other agents from reaching their goals. For this experiment, agent `a2` moves out of the center cell in step #9, and we encode this error through the atom `ab("a2", 9)`. We make the following observations:

- Agent `a2` starts a diagnosis in step #10, which is when its action to move out of the center cell failed; however, it does not send out any diagnosis inquiries to other agents as it realizes that it is the root cause of the failure.
- Agent `a3` starts a diagnosis in step #11, which is when its action to move into the center cell failed. It sends diagnosis inquries to agents `a1` and `a2` to learn why the center cell is not empty when it is supposed to be.
- Agent `a1` responds that it does not know why, but agent `a2` responds indicating that it is because its action to move out of the cell failed.
- Finally, agent `a1` starts a diagnosis in step #12, which is when its action to move into the center cell failed. Similar to the previous case, it sends diagnosis inquiries to the other two agents and agent `a2` responds indicating that it is because its action to move out of the cell failed.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we extended Multi-Agent STRIPS (MA-STRIPS) to Commonsense MA-STRIPS (CMA-STRIPS), which is a model that includes commonsense fluents and axioms. We consider the diagnosis problem in this setting and develop a decentralized multi-agent diagnosis algorithm called Multi-Agent Diagnosis with Decentralized Reasoning (MAD-DR) for CMA-STRIPS. We also present a proof-of-concept implementation of the algorithm with a simulator that allows for the agents to identify diagnoses when their actions fail. We describe our experimental evaluation of the algorithm with two popular multi-agent domains – logistics and multi-agent pathfinding. The experimental evaluation shows that the algorithm

performs reasonable well but also leaves several interesting issues for the future. First, the question of whether SWI-Prolog is the best choice for the development of a scalable and efficient decentralized diagnosis engine needs to be evaluated. Second, as we have mentioned earlier, our assumption that the failure of an action implies failure of the agent might need to be relaxed for some applications. Third, creating CMA-STRIPS benchmarks with commonsense features and axioms is also an activity that should be conducted.

ACKNOWLEDGMENTS

This research is partially supported by the National Science Foundation (NSF) of the United States under awards 1914635 and 2232055; the US-Israel Binational Science Foundation (BSF) under award 2022189; and the National Institute of Standards and Technology (NIST) of the United States via cooperative agreement 70NANB21H167. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the United States government.

REFERENCES

[1] Marcello Balduccini and Michael Gelfond. 2003. Diagnostic Reasoning with A-Prolog. *Theory and Practice of Logic Programming* 3, 4-5 (2003), 425–461.
[2] Chitta Baral, Sheila McIlraith, and Tran Cao Son. 2000. Formulating Diagnostic Problem Solving Using an Action Language with Narratives and Sensing. In *Proceedings of the International Conference on Principles of Knowledge and Representation and Reasoning (KR)*. 311–322.
[3] Daniel Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research* 27, 4 (2002), 819–840.
[4] Ronen I. Brafman and Carmel Domshlak. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 28–35.
[5] Aníbal Bregón, Matthew J. Daigle, Indranil Roychoudhury, Gautam Biswas, Xenofon D. Koutsoukos, and Belarmino Pulido. 2014. An Event-based Distributed Diagnosis Framework Using Structural Model Decomposition. *Artificial Intelligence* 210 (2014), 1–35.
[6] Vittorio Brusoni, Luca Console, Paolo Terenziani, and Daniele Theseider Dupré. 1998. A Spectrum of Definitions for Temporal Model-based Diagnosis. *Artificial Intelligence* 102, 1 (1998), 39–79.
[7] Rohan Chandra and Dinesh Manocha. 2022. GamePlan: Game-Theoretic Multi-Agent Planning with Human Drivers at Intersections, Roundabouts, and Merging. *IEEE Robotics and Automation Letters* 7, 2 (2022), 2676–2683.
[8] Matthew Daigle, Xenofon Koutsoukos, and Gautam Biswas. 2006. Distributed Diagnosis of Coupled Mobile Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 3787–3794.
[9] Femke de Jonge, Nico Roos, and Cees Witteveen. 2009. Primary and Secondary Diagnosis of Multi-Agent Plan Execution. *Autonomous Agents and Multi-Agent Systems* 18, 2 (2009), 267–294.
[10] Johan de Kleer and Kurt Konolige. 1989. Eliminating the Fixed Predicates from a Circumscription. *Artificial Intelligence* 39, 3 (1989), 391–398.
[11] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. 1992. Characterizing Diagnoses and Systems. *Artificial Intelligence* 56, 2-3 (1992), 197–222.
[12] Johan de Kleer and Brian C. Williams. 1987. Diagnosing Multiple Faults. *Artificial Intelligence* 32, 1 (1987), 97–130.
[13] Orel Elimelech, Roni Stern, and Meir Kalech. 2018. Structural Abstraction for Model-based Diagnosis with a Strong Fault Model. *Knowledge-Based Systems* 161 (2018), 357–374.
[14] Orel Elimelech, Roni Stern, Meir Kalech, and Yedidya Bar-Zev. 2017. Diagnosing Resource Usage Failures in Multi-Agent Systems. *Expert Systems with Applications* 77 (2017), 44–56.
[15] Alexander Feldman, Ingo Pill, Franz Wotawa, Ion Matei, and Johan de Kleer. 2020. Efficient Model-Based Diagnosis of Sequential Circuits. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2814–2821.
[16] Meir Kalech and Gal A. Kaminka. 2005. Towards Model-Based Diagnosis of Coordination Failures. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 102–107.
[17] Meir Kalech and Gal A Kaminka. 2007. On the Design of Coordination Diagnosis Algorithms for Teams of Situated Agents. *Artificial Intelligence* 171, 8-9 (2007), 491–513.
[18] Meir Kalech, Gal A Kaminka, Amnon Meisels, and Yehuda Elmaliach. 2006. Diagnosis of Multi-Robot Coordination Failures Using Distributed CSP Algorithms. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 970–975.
[19] Meir Kalech and Avraham Natan. 2022. Model-Based Diagnosis of Multi-Agent Systems: A Survey. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 12334–12341.
[20] Ngai Meng Kou, Cheng Peng, Hang Ma, T. K. Satish Kumar, and Sven Koenig. 2020. Idle Time Optimization for Target Assignment and Path Finding in Sortation Centers. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 9925–9932.
[21] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. 2021. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 11272–11281.
[22] Drew M McDermott. 2000. The 1998 AI Planning Systems Competition. *AI Magazine* 21, 2 (2000), 35–35.
[23] Roberto Micalizio. 2013. Action Failure Recovery via Model-Based Diagnosis and Conformant Planning. *Computational Intelligence* 29, 2 (2013), 233–280.
[24] Roberto Micalizio, Pietro Torasso, and Gianluca Torta. 2004. On-line Monitoring and Diagnosis of Multi-Agent Systems: A Model based Approach. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*. 848–852.
[25] Avraham Natan and Meir Kalech. 2022. Privacy-Aware Distributed Diagnosis of Multi-Agent Plans. *Expert Systems with Applications* 192 (2022), 116313.
[26] Avraham Natan, Meir Kalech, and Roman Barták. 2023. Diagnosis of Intermittent Faults in Multi-Agent Systems: An SFL Approach. *Artificial Intelligence* 324 (2023), 103994.
[27] Raz Nissim and Ronen I. Brafman. 2012. Multi-Agent A* for Parallel and Distributed Systems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 1265–1266.
[28] Lúcio S. Passos, Rui Abreu, and Rosaldo J. F. Rossetti. 2015. Spectrum-based Fault Localisation for Multi-Agent Systems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1134–1140.
[29] Raymond Reiter. 1987. A Theory of Diagnosis from First Principles. *Artificial Intelligence* 32, 1 (1987), 57–95.
[30] Nico Roos. 2018. Learning-Based Diagnosis and Repair. *Communications in Computer and Information Science* 823 (2018), 1–15.
[31] Nico Roos and Cees Witteveen. 2009. Models and Methods for Plan Diagnosis. *Autonomous Agents and Multi-Agent Systems* 19, 1 (2009), 30–52.
[32] Jens Schulz, Kira Hirsenkorn, Julian Löchner, Moritz Werling, and Darius Burschka. 2017. Estimation of Collective Maneuvers through Cooperative Multi-Agent Planning. In *Proceedings of IEEE Intelligent Vehicles Symposium (IV)*. 624–631.
[33] Roni Tzvi Stern, Meir Kalech, Alexander Feldman, and Gregory M. Provan. 2012. Exploring the Duality in Conflict-Directed Model-Based Diagnosis. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 828–834.
[34] Peter Struss and Oskar Dressler. 1989. "Physical Negation" Integrating Fault Models into the General Diagnostic Engine. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1318–1323.
[35] Michael Thielscher. 1997. A Theory of Dynamic Diagnosis. *Electronic Transactions on Artificial Intelligence* 1 (1997), 73–104.
[36] Gianluca Torta and Roberto Micalizio. 2018. SMT-based Diagnosis of Multi-Agent Temporal Plans. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2097–2099.
[37] Gianluca Torta, Roberto Micalizio, and Samuele Sormano. 2019. Explaining Failures Propagations in the Execution of Multi-Agent Temporal Plans. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2232–2234.
[38] Gianluca Torta, Roberto Micalizio, and Samuele Sormano. 2019. Temporal Multi-agent Plan Execution: Explaining What Happened. In *Proceedings of the International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems (EXTRAAMAS)*. 167–185.
[39] Wiebe Van der Hoek and Michael Wooldridge. 2008. Multi-Agent Systems. *Foundations of Artificial Intelligence* 3 (2008), 887–928.
[40] Brian C. Williams and P. Pandurang Nayak. 1996. A Model-based Approach to Reactive Self-Configuring Systems. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 971–978.
[41] Peter Wurman, Raffaello D'Andrea, and Mick Mountz. 2007. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 1752–1760.
[42] Peter Wurman, Raffaello D'Andrea, and Mick Mountz. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine* 29, 1 (2008), 9–20.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009