Distributed Diagnosis of Multi-Agent Plans

Avraham Natan and **Meir Kalech**
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

We live in an era in which many multi-agent systems (MAS) exist, such as autonomous vehicles, automatic warehousing, trains, etc. In many of these systems, as Amazon warehouse robots, agents are assigned to operate tasks and to this end they should plan their next steps. To achieve a plan to the MAS, the agents are required to coordinate their individual plans. This is known in the literature as multi agent plan (MAP). Typically, such systems are prone to failures due to variety of reasons, such as mechanical malfunctions or software bugs. When a failure occurs, there is a need to diagnose it to identify the faulty agents. This problem has been traditionally addressed in a centralized manner, where the agents in the system share their plans and observations with a centralized diagnosing agent. However, in some systems, an agent might not be interested to share its information (e.g. its plans), for privacy reasons. To address this challenge, we propose Distributed Diagnosis of Multi-Agent Plans (DDMAP) algorithms, which identify the root cause of failures in MAP in a distributed manner, where the agents do not have to share their plans. The contributions of this paper are (1) by formalizing DDMAP as a model-based diagnosis problem, and (2) by presenting synchronous and asynchronous distributed algorithms, in which the agents coordinate to find the root cause of the MAP failure, without sharing information about their plans. Experiments on simulated scenarios show that, the synchronous algorithm performs better in terms of run-time but not in communication effort.

1 Introduction

Multi-Agent Systems (MAS) can be found in wide variety of applications, such as automatic warehousing, autonomous vehicles, airport traffic management, logistics and public transport (such as train systems) [1]. Such applications help to improve commodity and efficiency in a variety of different tasks. For example, the amazon warehouses use an array of robots for automated relocation of items in the warehouse, often freeing manpower to focus on less trivial tasks. In many of such system the agents follow a Multi Agent Plan (MAP) in order to achieve a common goal. Very often the agents follow a tightly scheduled plan with little room for errors.

In such environment agents often share reusable resources such as doorways, charge points, moving space etc. In case where an agent experiences a fault, it may result in holding a resource for longer amount of time than intended, and creating a chain reaction of agents failing to follow their original plans In some cases the result of such scenario could be halting of a production process, but in other cases it could be worse.

This raises the necessity to address any deviation of the MAP as soon as possible. To address such fault, it must be detected, isolated, and finally handled. In this work we focus on isolating the faults by diagnosing the faulty agents using Model-Based Diagnosis approach.

Previous works have already addressed the task of diagnosing multi agent systems in variety ways, such as diagnosing plan execution of agents [2, 3, 4] or diagnosing their behaviours [5, 6, 7, 8]. A recent work [1] focuses on diagnosis of temporal multi agent plan allocation (Tmara). This work assumes a centralized diagnosing agent who observes the whole system and receives the plans of the agents in the MAS in order to compute a diagnosis. Such approach might not be suitable for systems where the individual agents might want to keep some level of privacy, and thus not willing to share their plans. In such cases, submitting the plans of the agents to a centralized diagnoser violates the privacy of the agents. We propose a distributed approach where the effort of computing a diagnosis for the system becomes a joint effort of all the agents in the MAS. The agents collaborate by sharing enough information to achieve a diagnosis while keeping as much information as possible for privacy reasons.

To this end, in this work, we present a twofold distributed model-based diagnosis approach. First each agent finds a local diagnosis by modelling the system in terms of model-based diagnosis. Then, the agents collaborate by applying algorithms inspired by DisCSP algorithms [9] to find consistent global diagnosis. We present two algorithms, one synchronous and one asynchronous. Both algorithms are sound, complete, and anytime.

We evaluate our algorithms through consistent scenarios in a simulation of MAP. We found that increas-

ing the number of agents in the system, as well as the length of the plans of the agents, increases the runtime and communicated information. Surprisingly, we found that the synchronous algorithm outperforms the asynchronous algorithm in terms of run time, but not in the amount of communicated information.

2 Related Work

Diagnosis of multi agent system failures has been studied in previous works. Several approaches have been proposed for diagnosing failures in multi agent systems. In this section we survey those prior works.

We first cover works which are related to the diagnosis problem in multi-robot systems plans. Roos and Witteveen [2] introduced a model where partial observations of plan states are compared with predicted states based on normal plan execution. Extending their notion, de Jonge et al. [3] introduce the use of model-based diagnosis in primary and secondary plan diagnosis which identify the incorrect execution of actions, and the root cause of these faulty actions, respectively. Micalizio and Torasso [4] address the problem of agent diagnosis, aiming to find the faulty agents. Elimelech et al. [1] propose a model based approach to diagnose resource usage failures in multi agent systems in which the diagnosing agent doesn't have access to the multi agent plans but instead can access a temporal resource allocation plan. They model the diagnosis problem as a set of constraints over the usage of the resources.

We next cover works which are related to the diagnosis of coordination faults. Early works in this subject depicted centralized architectures. For instance, Micalizio et al. [5] have utilized causal models of failures and diagnoses to centrally detect and respond to single-robot failures and to multi-robot coordination failures. Kalech et.al. tackle the problem of diagnosing coordination faults. Continuing with their centralized approach [6], they introduced a distributed, model-based coordination-failure diagnosis approach [7]. In their work, the coordination between the robots is modeled as a constraint graph. The following year, Kalech and Kaminka [8] introduced a novel design, using the term "social diagnosis" to describe the process that diagnoses the reason why robots disagree.

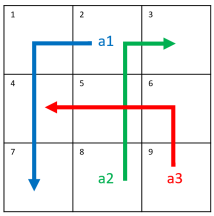
3 Problem Definition

In this section we formally define the Distributed diagnosis problem for multi-agent system (Distributed Tmara-Diag). We first present the centralized diagnosis problem for multi-agent system (Tmara-Diag) inspired by Elimelech et al. [1].

3.1 Centralized Tmara-Diag Problem

Agents in a group are assigned to operate tasks. The tasks dictate resources usage for each agent in different time steps, such that the resource usages of different agents for a given time step do not overlap. To formalize the timed resource allocation, we formally define Temporal Multi-Agent Resource Allocation (Tmara):

Definition 1 (Tmara). *Tmara is a tuple* $\langle A, R, T, plan \rangle$ *where* A, R, *and* T *are sets of agents, resources and time steps, respectively, and plan is a function* $A \times T \rightarrow R$ *that maps an agent* $a \in A$ *and time step* $t \in T$ *to the resource allocated to agent* a *at time step* t.



(a)

	t_0	t_1	t_2	t_3	Obs
a_1	2	1	4	7	4
a_2	8	5	2	3	3
a_3	9	6	5	4	5

(b)

Figure 1: A simple example to demonstrate the TMARA (left) and a TMARA-Diag problem (right)

In the example scenario in Figure 1, A is the set of agents $\{a_1, a_2, a_3\}$, R is the set of tiles $\{1, 2, ..., 9\}$, and $T = \{0, 1, 2, 3\}$ is the set of time steps. For every agent $a_i \in A$ and time step $t \in T$ the allocation function $plan(i, t)$ returns the tile that is planned for agent a_i.

A malfunctioning agent may use different resource than the one assigned to it by the Tmara. Such a deviation from the planned Tmara may result in more than one agent simultaneously trying to use the same resource. In that case, these agents may not be able to continue their execution. This may cause a chain reaction, where other agents that do follow their Tmara are also unable to continue their execution, as the resources allocated to them by Tmara are currently used by other agents who could not continue their execution. Eventually, this failed execution is revealed, at which point many agents may be using resources that were not allocated to them according to Tmara. We denote by t_{obs} this point in time in which the failure was detected, and denote by r_{obs_i} the resource used by agent a_i at time t_{obs}.

Note that if an agent a_i is using a different resource than the one allocated to it at time t_{obs} according to Tmara, it does not mean that a_i is faulty since a_i may have been forced to use this resource due to a failure of another agent. For example, consider a_3 in the scenario from figure 1. Assume agent a_1 was faulty at time step 3 and thus keeps holding its resource (tile 4) for the next time step. This causes agent a_3 to get stuck at tile 5. The system is observed at time step $t_{obs} = 3$, with $r_{obs_1} = 4$, $r_{obs_2} = 3$, $r_{obs_3} = 5$. While agent a_3 is not using its resource as planned, in this scenario a_1 is the root cause to a_3's unplanned location.

We formalize the problem of identifying which of the agents are faulty given Tmara as follows:

Definition 2 (Tmara-Diag Problem). *A* Tmara-Diag *problem is defined by the tuple* $\langle A, R, T, plan, t_{obs}, \{r_{obs_i}\}_{i=1}^{|A|} \rangle$, *where:*

- $\langle A, R, T, plan \rangle$ *forms a proper TMARA.*
- $t_{obs} \in T$ *is the time the failure is discovered.*
- $r_{obs_i} \in R$ *is the resource occupied by agent* a_i *when the system is observed at time* t_{obs}.

A Tmara-Diag *problem arises when there exists an agent* a_i *such that* $plan(i, t_{obs}) \neq r_{obs_i}$.

A solution to a TMARA-DIAG problem is a set of pairs $\langle a_i, t_i \rangle$ of faulty agents and time steps, where each pair indicates which agent was faulty and at what time. Note that we do not assume that the failure is immediately identified. Thus, t_{obs} represents the time step after the failure occurred when the system failure was detected. In addition, there may be several candidate solutions to a TMARA-DIAG problem, where each candidate is a plausible explanation for the observed behavior.

TMARA-DIAG assumes a centralized agent who makes the diagnosis. This agent holds the the information about the agents' plans and the observation. However, in this paper we discuss a distributed form of TMARA-DIAG, in which there is no centralized diagnosing agent but each agent has its own plan and observation. Such approach relies on collaborative efforts between the agents, done in a distributed manner. Each agent solves part of the problem (DISTRIBUTED TMARA-DIAG), fed by information from its fellow agents to reach a global solution to the problem.

3.2 DISTRIBUTED TMARA-DIAG problem

In DISTRIBUTED TMARA-DIAG we assume that the global plan TMARA is divided between the agents, giving every agent only its own local plan. We assume that the plans, i.e., resource usages of different agents for a given time step do not overlap.

Definition 3 (**LOCAL TMARA**). LOCAL TMARA *for agent a_i is a tuple $LT_i = \langle A, R, T, plan_i \rangle$ where A, R, T are defined as in TMARA, and $plan_i : T \rightarrow R$ is a function that maps agent a_i to the resource allocated to it at time step t.*

Returning to the example scenario in figure 1, A, R, and T are defined as in TMARA, and the function $plan_i(t)$ returns the resource that is allocated for agent a_i at time t. An agent could deviate from its LOCAL TMARA as a result of an internal fault or as a result of other agent holding a resource that was allocated for it at some time. In both cases, the agent may be holding at the time of the observation a resource that is not allocated to it. The problem of finding what caused such a deviation is the DISTRIBUTED TMARA-DIAG problem. Since we focus on a distributed diagnosis approach, every agent operates a diagnosis process based on its own plan and observation. We define first the diagnosis problem from the point of view of an individual LOCAL TMARA .

Definition 4 (**LOCAL TMARA-DIAG Problem**). *A* LOCAL TMARA-DIAG *problem for agent a_i is defined by the tuple $LTD_i = \langle A, R, T, plan_i, t_{obs}, r_{obs_i} \rangle$ where:*

- *$\langle A, R, T, plan_i \rangle$ forms a proper* LOCAL TMARA *for agent a_i.*
- *$t_{obs} \in T$ is the time the failure was discovered.*
- *$r_{obs_i} \in R$ is the resource occupied by the agent a_i when the system is observed at time t_{obs}.*

A LOCAL TMARA-DIAG *problem arises when $plan_i(t_{obs}) \neq r_{obs_i}$.*

A diagnosis to a LOCAL TMARA-DIAG problem for agent a_i is a pair $\langle a_i, t \rangle$, which indicates that agent a_i was faulty at time t. In case that the diagnosis is that agent a_i is healthy the diagnosis is an empty set. Note that there could be several alternative time steps that could explain the unexpected observation. Each one of them forms a diagnosis. We denote the set of computed local diagnoses for agent a_i as D_i. d_{i_j} denotes the j^{th} diagnosis of agent a_i. To compute a diagnosis the agent should communicate with its fellow agents. In the next section we elaborate on algorithms to compute the diagnoses. To complete the distributed diagnosis definition, we define now the global diagnosis problem, DISTRIBUTED TMARA-DIAG.

Definition 5 (**DISTRIBUTED TMARA-DIAG Problem**). *A* DISTRIBUTED TMARA-DIAG *problem is defined by a set of local diagnosis problems $\mathcal{LTD} = \{LTD_i\}_{i=1}^{|A|}$. A* DISTRIBUTED TMARA-DIAG *problem arises when $\exists a_i \in A$, s.t. $plan_i(t_{obs}) \neq r_{obs_i}$.*

A diagnosis to a DISTRIBUTED TMARA-DIAG problem is a union of the diagnoses of the LOCAL TMARA-DIAG problems. This diagnosis forms a set of pairs $\langle a_i, t_i \rangle$ of faulty agents and time steps, where each pair indicates which agent was faulty and at what time.

In the next section we show how to define LOCAL TMARA-DIAG in terms of Model-Based Diagnosis *MBD* and then we present two distributed diagnosis algorithms to solve the DISTRIBUTED TMARA-DIAG problem.

4 Method Description

In this section we provide the required background for Model-Based Diagnosis (MBD) (Section 4.1) to show how DISTRIBUTED TMARA-DIAG problem can be formalized as an MBD problem (Section 4.2). Then we present two distributed algorithms, one synchronous and one asynchronous to diagnose the faulty agents (Section 4.3).

4.1 MBD Background

MBD problems arise when the normal behavior of a system is violated due to faulty components as indicated by certain observations.

Definition 6 (***MBD* Problem**). *An MBD problem is specified by the tuple $\langle SD, COMPS, OBS \rangle$ where: SD is a system description, COMPS is a set of components, and OBS is an observation. SD takes into account that some components might be abnormal (faulty). This is specified by the unary predicate $h(\cdot)$ on components such that $h(c)$ is true when component c is healthy, while $\neg h(c)$ is true when c is faulty. A diagnosis problem (DP) arises when the assumption that all components are healthy is inconsistent with the system model and observed system behavior. This is expressed formally as follows: $SD \wedge \bigwedge_{c \in COMPS} h(c) \wedge OBS \vdash \bot$*

Diagnosis algorithms try to find diagnoses, which are possible ways to explain the above inconsistency by assuming that some components are faulty.

Definition 7 (**DIAGNOSIS**). *A set of components $\triangle$ is a diagnosis if $SD \wedge \bigwedge_{c \in \triangle} \neg h(c) \wedge \bigwedge_{c \notin \triangle} h(c) \wedge OBS \nvdash \bot$*

There may be multiple diagnoses for a given *DP*. A common way to prioritize diagnoses is to prefer minimal

diagnoses, where a diagnosis $\triangle$ is said to be minimal if no proper subset $\triangle' \subset \triangle$ is a diagnosis. A minimal cardinality diagnosis is the smallest diagnosis in terms of amount of components it contains. In this work we focus on prioritizing diagnoses by their cardinality.

4.2 Distributed Tmara-Diag as an *MBD* problem

We modeled Distributed Tmara-Diag as a set of Local Tmara-Diag problems. In this section, we model Local Tmara-Diag problem as an *MBD* problem. This means defining Local Tmara-Diag in terms of *SD*, *COMPS*, and *OBS*. The system components (*COMPS*) of agent a_i represent pairs of agent and time $\langle a_i, t\rangle$, i.e. $COMPS = \{\langle a_i, t\rangle\}_{t=0}^{|T|}$. The observation (*OBS*) represents the resource occupied by agent a_i when the system is observed, i.e. $OBS = \{R_{obs_i}\}$. *SD* represents the plans of agent a_i as well as its constraints with other agents, such as two agents could not occupy the same resource simultaneously, or that some agent must occupy some resource at some time step. Next, we define *SD*.

Defining *SD*

A healthy agent is expected to follow its plan according to Local Tmara. Otherwise, it can be either faulty, or in conflict with another faulty agent that blocks its resource. Let us define four states an agent can be in at a specific time step:

- **Healthy state.** The agent is healthy at the specific time step, and is able to advance to the next step according to its Local Tmara.
- **Faulty state.** The agent is faulty at the specific time step.
- **Conflict state.** The agent is healthy at the specific time step but it is not able to advance to the next step according to its plan.
- **Post-conflict state.** The agent was in a conflict state or post conflict state in the previous time step.

An agent could be only in one of these states. To formally define these states we first define the following predicates and functions:

- $h(a_i, t)$, which is true if a_i is healthy at time t.
- $Use(a_i, t, r)$, which is true if agent a_i at time t uses resource $r \in R$ exclusively, i.e., no other resource is used by a_i at time t.
- $UsePlanned(a_i, t)$, which is true if agent a_i uses the resource allocated to it in the LT_i for time t.
- $Available(a_i, t)$, which is true if the resource allocated to agent a_i for time t is not already used by another agent.
- $pstep(a_i, t) : A \times T \to T$, given an agent a_i and a time step t it returns the time step t' according to the Local Tmara of the resource currently held by agent a_i at time step t.
- $pres(a_i, t) : A \times T \to R$, given an agent a_i and a time step t it returns the next resource r according to the Local Tmara of the resource currently held by agent a_i at time step t

Each agent's state can be formally defined as follows.

$$healthy(a_i, t) = h(a_i, t) \wedge UsePlanned(a_i, pstep(a_i, t)) \wedge Available(a_i, pstep(a_i, t) + 1) \tag{1}$$

$$faulty(a_i, t) = \neg h(a_i, t + 1) \tag{2}$$

$$conflict(a_i, t) = h(a_i, t) \wedge UsePlanned(a_i, pstep(a_i, t)) \wedge \neg Available(a_i, pstep(a_i, t) + 1) \tag{3}$$

$$postConflict(a_i, t) = h(a_i, t) \wedge (conflict(a_i, t - 1) \vee postConflict(a_i, t - 1)) \tag{4}$$

Note that the difference between equations 1 and 3 is solely in the predicate *Available*, which means that the resource that a_i intends to use is either free, in a healthy mode (1), or taken by another agent, in a conflict mode (3). The first two parts of the equations are the same, since in both at time t the agent is healthy and the agent executed its plan up to time step t. Note that the use of the function *pstep* is intended to address the fact that it is possible that the agent has some delay due to a fault in the past, and the planned resource is delayed in some time.

We add a constraint stating that for every time step $t \in [0, t_{obs}]$, it holds that only one state is active:

$$oneState(a_i, t) = healthy(a_i, t) \otimes faulty(a_i, t) \otimes conflict(a_i, t) \otimes postConflict(a_i, t) \tag{5}$$

Next, we define the corresponding models for the different states of the agents. Depending on the state an agent is in, it can follow one of the four models outlined below:

- **Healthy model.** A healthy agent uses the next resource to the one that it is holding according to its local plan.
- **Fault model.** A faulty agent continues to hold the resource that it is currently holding.
- **Conflict model.** An agent that is in a conflict state continues to hold the resource that it is currently holding, while another agent holds its planned resource.
- **Post-conflict model.** An agent that is in a post-conflict state continues to hold the resource that it is currently holding.

We formally define the different models as such:

$$\Phi_h(a_i, t) = UsePlanned(a_i, pstep(a_i, t) + 1) \tag{6}$$

$$\Phi_f(a_i, t) = Use(a_i, t, r) \to Use(a_i, t + 1, r) \tag{7}$$

$$\Phi_c(a_i, t) = Use(a_i, t, r) \to Use(a_i, t + 1, r) \wedge \exists j \neq i \; s.t. : Use(a_j, t + 1, pres(a_i, t)) \tag{8}$$

$$\Phi_p(a_i, t) = Use(a_i, t, r) \to Use(a_i, t + 1, r) \tag{9}$$

Using the above definitions, we define the system description (*SD*) for the *MBD* problem that corresponds

to a LOCAL TMARA-DIAG problem as the conjunction of the possible states over time:

$$SD = \bigwedge_{t=0}^{|T|-1} \Big(\Big(healthy(a_i,t) \rightarrow \Phi_h(a_i,t) \Big) \wedge \Big(faulty(a_i,t) \rightarrow \Phi_f(a_i,t) \Big) \wedge \Big(conflict(a_i,t) \rightarrow \Phi_c(a_i,t) \Big) \wedge \Big(postConflict(a_i,t) \rightarrow \Phi_p(a_i,t) \Big) \Big)$$

4.3 DISTRIBUTED TMARA-DIAG Algorithms

In this section we present the distributed algorithms for solving the DISTRIBUTED TMARA-DIAG problem. Each agent is solving its own LOCAL TMARA-DIAG problem by modelling it to an *MBD* problem as defined in section 4.2. This way the agents acquire their diagnoses to the LOCAL TMARA-DIAG problems, or in short local diagnoses. Those diagnoses, however, are not necessarily equal to the global diagnoses of TMARA-DIAG problem. To compile a global diagnosis we present distributed algorithms inspired by DisCSP algorithms [9]. To this end each agent uses the diagnoses for its LOCAL TMARA-DIAG problem to generate a set of beliefs about its actual execution. It then communicates with other agents to find in a joint effort sets of beliefs that are consistent. These sets are compiled to global diagnoses. We present two distributed algorithms: one synchronous algorithm, i.e., the agents communicate in a sequence row to find a global diagnosis, and another asynchronous algorithm, where the agents communicate simultaneously. We formally define the next definitions, that will help us to present the algorithms:

Definition 8 (BELIEF). *A Belief of agent a_i is a sequence of resources $b_i = [r_{t_0}, r_{t_1}, ..., r_{t_{obs}}]$ denoting the actual execution according to the local diagnosis corresponding to this belief. We denote the set of a_i's beliefs as $B_i = \{b_{i_j}\}_{j=1}^{|B_i|}$.*

A belief of an agent is corresponding with a certain diagnosis. Assume, for instance, that one of the diagnoses of agent a_i is $\emptyset$ (i.e., healthy), then the corresponding belief (i.e., actual execution), is its original plan. However, in case that its diagnosis is that it is faulty in a certain time, then its belief about its execution could not be equal to the original plan anymore. Also, in case of a conflict, agent a_i's belief could not be equal to its original plan, since another agent a_j holds one of the resources planed to a_i. In this case agent a_i blames a_j as the cause of this conflict. This blame is defined next as the corresponding conflict cause to agent a_i's belief.

Definition 9 (CONFLICT CAUSE). *A* CONFLICT CAUSE *issued by agent a_i is a tuple $cc = \langle a_{j \neq i}, t, r \rangle \in A \times T \times R$ imposing a constraint on agent a_j s.t. a_j is expected to select a belief in which it is holding the resource r at time step t. cc_{i_j} denotes the* CONFLICT CAUSE *that corresponds to* BELIEF b_{i_j}*, and the set of a_i's conflict causes that correspond to B_i is $CC_i = \{cc_{i_j}\}_{j=1}^{|CC_i|}$.*

Each agent holds some beliefs of neighbouring agents (not necessarily of all the neighboring agents). The set in the next definition (AGENTS BELIEF VIEW) essentially contains the beliefs that neighboring agents of agent a_i shared with it.

Definition 10 (AGENTS BELIEF VIEW). *The* AGENTS BELIEF VIEW *of agent a_i is a set of beliefs $ABV_i \subseteq \{b_{k_{j_k}}\}_{k=1}^{|A|}$ s.t. $\forall b_{k_{j_k}} \in ABV_i \wedge k \neq i : \ b_{k_{j_k}} \in B_k$.*

Each belief b_{i_j} in an AGENTS BELIEF VIEW has a corresponding local diagnosis d_{i_j}. We formally define this set as an AGENTS DIAGNOSIS VIEW.

Definition 11 (AGENTS DIAGNOSIS VIEW). *The* AGENTS DIAGNOSIS VIEW *of agent a_i that corresponds to ABV_i is a set of local diagnoses $ADV_i \subseteq \{d_{k_{j_k}}\}_{k=1}^{|A|}$ s.t. $\forall d_{k_{j_k}} \in ADV_i \wedge k \neq i : \ d_{k_{j_k}} \in D_k$.*

We expect that an AGENTS BELIEF VIEW of agent a_i, together with the agent's current selected belief b_{i_j} will be consistent. A consistent AGENTS BELIEF VIEW confirms that for every two beliefs in $ABV_i \oplus b_{i_j}$, they do not occupy the same resource r at the same time step t.

Definition 12 (CONSISTENT AGENTS BELIEF VIEW). *For a given ABV_i of agent a_i and its selected belief b_{i_j}, a* CONSISTENT AGENTS BELIEF VIEW *is a set of beliefs $CABV \subseteq ABV_i \cup b_{i_j}$ that holds the condition:*

$$\forall b_{k_{j_k}}, b'_{k'_{j_{k'}}} \in CABV, \forall t \in T : \ b_{k_{j_k}}[t] \neq b'_{k'_{j_{k'}}}[t]$$

As mentioned in Definition 9, every belief $b_{k_{j_k}} \in ABV_i$, is corresponding with a conflict cause $cc_{k_{j_k}}$. AGENTS CONFLICT CAUSE VIEW is a set of conflict causes that are corresponding with the beliefs in AGENTS BELIEF VIEW. This set essentially contains the conflict causes that neighboring agents of agent a_i shared with it.

Definition 13 (AGENTS CONFLICT CAUSE VIEW). *The* AGENTS CONFLICT CAUSE VIEW *of agent a_i is a set of conflict causes $ACCV_i \subseteq \{cc_{k_{j_k}}\}_{k=1}^{|A|}$ s.t. $\forall cc_{k_{j_k}} \in ACCV_i \wedge k \neq i : cc_{k_{j_k}} \in CC_k$.*

We expect that an AGENTS CONFLICT CAUSE VIEW of agent a_i, together with the agent's current selected conflict cause cc_{i_j} will be consistent with its corresponding AGENTS BELIEF VIEW and current selected belief b_{i_j}. A consistent AGENTS CONFLICT CAUSE VIEW confirms that for every conflict cause in $ACCV_i \oplus cc_{i_j}$, if it assumes that some agent holds resource r at time step t, then if the belief of that agent is in $ABV_i \oplus b_{i_j}$, it must indeed hold resource r at time step t.

Definition 14 (CONSISTENT AGENTS CONFLICT CAUSE VIEW). *For a given $ACCV_i$ and ABV_i of agent a_i and its selected belief and conflict cause b_{i_j}, cc_{i_j}, a* CONSISTENT AGENTS CONFLICT CAUSE VIEW *is a set of conflict causes $CACCV \subseteq ACCV_i \oplus cc_{i_j}$ that holds the condition:*

$$\forall t \in T, r \in R, \forall cc_{k_{j_k}} \in CACCV : \ cc_{k_{j_k}} = \{k', t, r\} \Rightarrow \exists b'_{k'_{j_{k'}}} \in ABV_i \oplus b_{i_j} \Rightarrow b'_{k'_{j_{k'}}}[t] = r$$

Finally, we expect that the agent belief view (ABV_i) of agent a_i will be consistent with its belief set (B_i). Also we expect that the agent conflict cause view ($ACCV_i$) of agent a_i will be consistent with its conflict cause set (CC_i). In case of inconsistency, we would like to isolate the subset of (ABV_i) that cause this inconsistency. We call this subset NOGOOD.

Definition 15 (NOGOOD). *Given the* AGENTS BELIEF VIEW ABV_i, *the* AGENTS CONFLICT CAUSE VIEW $ACCV_i$ *and sets of local beliefs and conflict causes* B_i, CC_i *of agent* a_i, *a* NOGOOD *is a set* $ng \subseteq ABV_i$ *that confirms either of the next conditions:*

- $\forall b_{i_j} \in B_i,\ b_{i_j} \cup ng$ *does not form a* CONSISTENT AGENTS BELIEF VIEW
- $\forall cc_{i_j} \in CC_i,\ cc_{i_j} \cup \{cc_k\}$ *s.t.* $b_k \in ng$ *does not form a* CONSISTENT AGENTS CONFLICT CAUSE VIEW.

We denote the j^{th} *recorded nogood of agent* a_i *as* ng_{i_j}, *and the set of recorded nogoods of agent* a_i *as* NGS_i.

We next present the distributed algorithms.

Synchronous Diagnosis Algorithm (SYDIA)

The *Synchronous Diagnosis Algorithm* (*SYDIA*) assumes a hierarchical relation between the agents, and initializes the desired cardinality of the diagnosis to zero, i.e., the desired diagnosis needs to have zero faulty agents. It then starts with the first agent in the hierarchy choosing one of its local diagnoses. Based on this diagnosis it selects its corresponding belief, and passes the belief to the next agent in the hierarchy. The next agent chooses one of its local diagnoses which is consistent with the belief of the previous agent, selects its corresponding belief, and passes it to the third agent in the hierarchy, and so on. This process continues until the last agent in the hierarchy approves a consistent diagnosis with previous agents, at which point a global diagnosis has been found. The algorithm terminates once all global diagnoses for every cardinality in the range $[0, |A|]$ have been found.

Asynchronous Diagnosis Algorithm (ASYDIA)

Like the *SYDIA*, the *Asynchronous Diagnosis Algorithm* algorithm (*ASYDIA*) also assumes a hierarchical relation between the agents and also initializes the required cardinality to zero. Its way of finding sets of consistent beliefs, however, is fundamentally different from *SYDIA*. The algorithm starts with every agent selecting one of its beliefs. Then the agents jointly reason about their beliefs by exchanging of messages, while changing their selected beliefs if it is required. Each agent keeps its own AGENTS BELIEF VIEW, AGENTS CONFLICT CAUSE VIEW, and AGENTS DIAGNOSIS VIEW which includes the selected beliefs, conflict causes, and local diagnoses that other agents sent, denoted as ABV_i, $ACCV_i$, ADV_i, respectively. Note that, due to the synchronous nature of the algorithm, some agents might have those sets outdated. This may put a difficulty on agreeing on a CONSISTENT AGENTS BELIEF VIEW. To address this, each agent may mark some subsets of its ABV_i as a NOGOOD, or a set of beliefs that cannot be part of any future CONSISTENT AGENTS BELIEF VIEW. This allows the algorithm to focus the search of a CONSISTENT AGENTS BELIEF VIEW. Since this algorithm does not search for CONSISTENT AGENTS BELIEF VIEW sequentially along the agents (as *SYDIA* does), it cannot assume that it will go over all possible belief sets in some order. To address this, each time a CONSISTENT AGENTS BELIEF VIEW is found, the agents mark it as a new *nogood*, forcing the algorithm to prune this CONSISTENT AGENTS BELIEF VIEW and explore other belief sets, and thus making it possible to find every viable CONSISTENT AGENTS BELIEF VIEW. When each CONSISTENT AGENTS BELIEF VIEW of the required cardinality is found, the algorithm raises the required cardinality by one and starts over. The algorithm terminates once all global diagnoses for every required cardinality in the range $[0, |A|]$ have been found.

5 Evaluation

In this section we present experimental evaluation to examine the properties of the suggested methods, such as the number of agents and the cardinality of the faults.

5.1 Experiment Setup

Our experiments were performed on a simulation of DISTRIBUTED TMARA-DIAG problems inspired by the example presented in Figure 1. In order to simplify the experiments, we make the next assumptions: (1) the agents move on a board of 12×12 tiles, which represent the resources, (2) the agents can advance to neighbour tiles only by moving left forward or right, (3) an agent does not visit a tile it has been visited previously, (4) each agent can occupy a single tile at a time (meaning that it holds one resource at a time), and (5) a faulty agent executes a single faulty action during its execution, for a single time step.

The generation of the DISTRIBUTED TMARA-DIAG problems consists the following stages:

1. Set the number of agents and the length of the agents' plans.
2. Generate LOCAL TMARA for each agent, such that the agents' LOCAL TMARA are consistent with each other.
3. Choose a subset of agents to be faulty.
4. Simulate the agents' execution while considering the faulty agents as follows:
   - A faulty agent randomly chooses at which time step it will be faulty. Up to this time step it follows its LOCAL TMARA, then it is stuck for a single time step, and then continues to follow its LOCAL TMARA in a delay of one time step.
   - A healthy agent follows its LOCAL TMARA as long as its resource is not being used by another agent (a conflict). If it is used then it keeps holding its current resource up to the observation time.

The set of resources each agent is holding at the last time step (t_{obs}) forms the observation of the generated DISTRIBUTED TMARA-DIAG problem.

We generated DISTRIBUTED TMARA-DIAG problems for varied number of agents (2-10), time steps (2-10)

and faulty agents (1-5). We conducted 20 tests for each possible combination of values for those parameters, for a total of 2500 tests. We measured the runtime required for the different algorithms to solve the problem, the communication efforts of the different algorithms, the runtime required to reach the first diagnosis, and the amount of information units sent between the agents.

To solve the generated problem instances, we used the SAT-based algorithm to solve the LOCAL TMARA-DIAG problem of every agent, and then the *Asynchronous Diagnosis Algorithm* (*ASYDIA*) and *Synchronous Diagnosis Algorithm* (*SYDIA*) to solve the DISTRIBUTED TMARA-DIAG problem. We also test a version of the *ASYDIA* and *SYDIA* algorithms where the algorithms do not guarantee that the diagnoses will be found in a ascending order by the cardinality of the solution. We call the random order version of *ASYDIA* and *SYDIA*, *Asynchronous Diagnosis Algorithm Light* (*ASYDIAL*) and *Synchronous Diagnosis Algorithm Light* (*SYDIAL*), correspondingly. The algorithms were implemented in Java and run on a Windows machine. The constraints modelling of the SAT-based algorithm were implemented and solved using the Choco library.

5.2 Results

The plots in Figure 2 show the average runtime it took the different algorithms to solve the DISTRIBUTED TMARA-DIAG problems generated. Plot 2a shows how the runtime (y-axis) is affected by the number of of agents (x-axis) for problems with 10 time steps and 5 faulty agents, plot 2b shows the impact of the number of time steps (x-axis) on the runtime (y-axis) for problems with 10 agents and 5 faulty agents, and plot 2c shows the impact of the number of faulty agents (x-axis) on the runtime (y-axis) for problems with 10 agents and 10 time steps.

There are some trends we can see. First, the runtime grows for problems with larger number of agents, larger number of time steps and larger number of faulty agents. The runtime increases faster with the growth of the number of time steps (Figure 2b) than the number of agents (Figure 2a). This can be explained by the way the agents select a candidate belief corresponding with a diagnosis. It checks the belief for consistency only with **higher** priority agents, but for **all** of the time steps. Additionally, it can be seen that the runtime increases exponentially with the increase of faulty agents (Figure 2c). This can be explained by the fact that every faulty agent generates an amount of beliefs, which contribute to the branching factor of the diagnoses space. This means that there are more plausible diagnoses that can be found.

Second, the gap in runtime between *SYDIA* and *ASYDIA* algorithms shows clear advantage for *SYDIA*. The reason is that while in *SYDIA* an agent can assume that the beliefs of higher priority agents are all consistent with each other, an agent in *ASYDIA* cannot assume that, which is the reason for the additional computational effort. Third, figure 2 shows that finding diagnoses in increasing cardinality impacts the runtime of the algorithms. The runtime of algorithms that find diagnoses in cardinality order is higher than those that do not. This gap is especially noticeable between

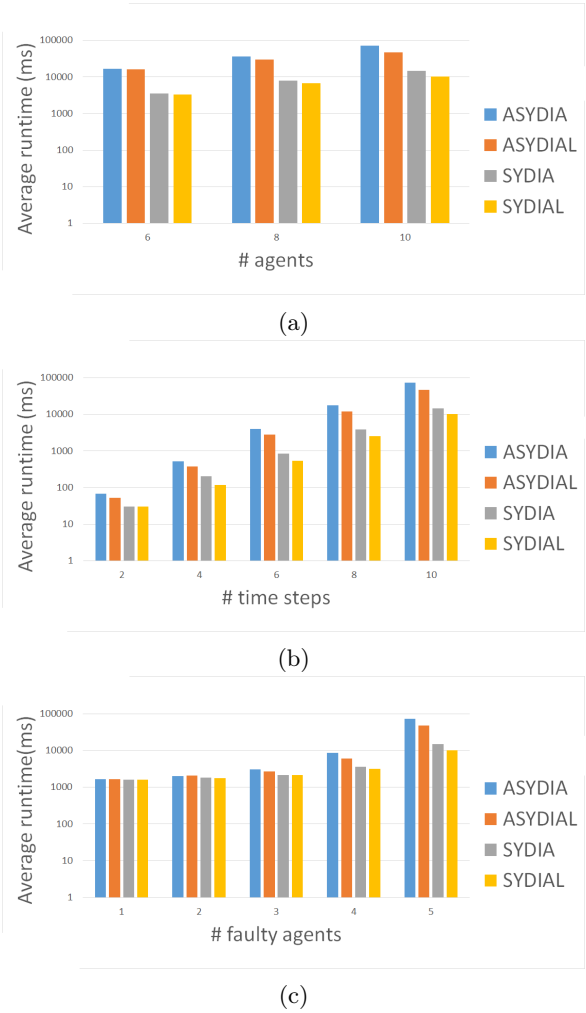
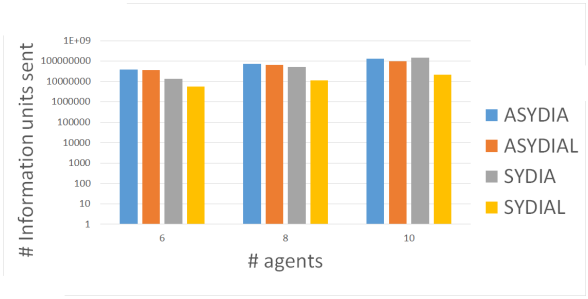


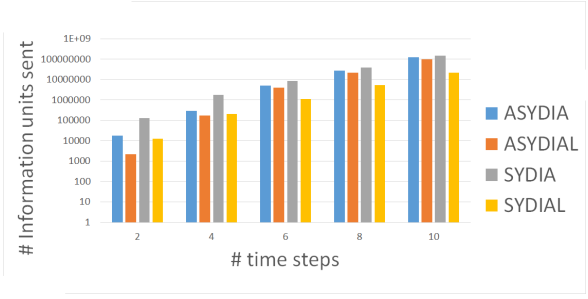
Figure 2: The runtime for solving the DISTRIBUTED TMARA-DIAG problems when varying the number of agents (a), time steps (b) and faulty agents (c).

ASYDIA and *ASYDIAL* than their counterpart synchronous algorithms. The reason for this gap is that when taking cardinality into account, algorithms sometimes dispose of valid solutions due to non matching cardinality.

Figure 3 shows the average amount of Information Units (IU's) shared between the agents for the different algorithms (y-axis). A single information unit is defined by the resource of an agent at certain time step: $\langle a_i, r, t \rangle$. Plot 3a shows how this amount is affected by the number of of agents (x-axis) for problems with 10 time steps and 5 faulty agents, plot 3b shows how this amount is affected by the number of time steps (x-axis) for problems with 10 agents and 5 faulty agents, and plot 3c shows how this amount is affected by the number of faulty agents (x-axis) for problems with 10 agents and 10 time steps. In figure 3 we can see some trends that are similar to figure 2 and some that are different. First, the amount of IUs sent increases with the number of agents, time steps, and faulty agents, since checking beliefs for consistency with less agents, time steps or faulty agents means sharing less IUs. Second, the gap in sent IUs between *ASYDIA* and *SYDIA* shows an advantage to *ASYDIA*. It can be explained



(a)



(b)

(c)

Figure 3: The amount of Information Units sent while solving the DISTRIBUTED TMARA-DIAG problems when varying the number of agents (a), time steps (b) and faulty agents (c).

by the way that the agents share information among themselves. In *SYDIA*, in order to reach a belief set of size $|A|$ that can be a candidate for a diagnosis, the agents need in turn to send information to each other until they reach such possible global diagnosis. This involves backtracking and hence increases the occasions at which information is being sent. In *ASYDIA* on the other hand, once the initial beliefs are being communicated, each agent holds a potential global diagnosis. The additional information exchanged in that case is only the information needed to change enough beliefs in order to get a CONSISTENT AGENTS BELIEF VIEW. This explains the third trend shown, where the gap in information shared between the algorithms and their cardinality agnostic versions is higher with *SYDIA* and *SYDIAL* than with *ASYDIA* and *ASYDIAL*. This can be explained by the fact that *SYDIA* and *SYDIAL* do not drop valid diagnoses due to cardinality constraints.

6 Conclusions and Future Work

In this work, we addressed the problem of diagnosing agent failures in executing their TMARA for a system where the agents are not willing to share their plans. Our approach consists of using *MBD* approach to produce local diagnoses and using distributed methods for combining these local diagnoses to global diagnoses. For the distributed methods we presented a synchronous and an asynchronous algorithms. We found that the synchronous algorithm outperforms the asynchronous one in terms of run time, and that the asynchronous algorithm outperforms the synchronous one in terms of communication overhead. We also found that both algorithms' perform better if the cardinality of the diagnosis is not taken into account.

For future work we plan to add more complex fault and conflict models. In this work the fault model dictates that an agent halts its execution for one time step. In a real environment a faulty agent could even occupy a resource that is not part of its plan. Also, we would like to explore a conflict model in which an agent waits for its resource to become available and then continue its execution.

References

[1] Orel Elimelech, Roni Stern, Meir Kalech, and Yedidya Bar-Zeev. Diagnosing resource usage failures in multi-agent systems. *Expert Systems with Applications*, 77:44–56, 2017.

[2] Nico Roos and Cees Witteveen. Models and methods for plan diagnosis. *Autonomous Agents and Multi-Agent Systems*, 19(1):30–52, 2009.

[3] Femke De Jonge, Nico Roos, and Cees Witteveen. Primary and secondary diagnosis of multi-agent plan execution. *Autonomous Agents and Multi-Agent Systems*, 18(2):267–294, 2009.

[4] Roberto Micalizio and Pietro Torasso. Plan diagnosis and agent diagnosis in multi-agent systems. In *Congress of the Italian Association for Artificial Intelligence*, pages 434–446. Springer, 2007.

[5] Roberto Micalizio, Pietro Torasso, and Gianluca Torta. On-line monitoring and diagnosis of multi-agent systems: a model based approach. In *Proceedings of the 16th European Conference on Artificial Intelligence*, pages 848–852. IOS Press, 2004.

[6] Meir Kalech and Gal A Kaminka. Towards model-based diagnosis of coordination failures. In *AAAI*, volume 5, pages 102–107, 2005.

[7] Meir Kalech, Gal A Kaminka, Amnon Meisels, and Yehuda Elmaliach. Diagnosis of multi-robot coordination failures using distributed csp algorithms. In *AAAI*, pages 970–975, 2006.

[8] 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.

[9] Makoto Yokoo, Edmund H Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on knowledge and data engineering*, 10(5):673–685, 1998.