Explaining interdependent action delays in multiagent plans execution

Temporal Multiagent Plan Execution: Explaining what Happened

Gianluca Torta[0000−0002−4276−7213], Roberto Micalizio[0000−0001−9336−0651], and Samuele Sormano

Dipartimento di Informatica, Università di Torino, Italy
{gianluca.torta,roberto.micalizio}@unito.it
samuele.sormano@edu.unito.it

Abstract. The paper addresses the problem of explaining failures that happened during the execution of Temporal Multiagent Plans (TMAPs), namely MAPs that contain both logic and temporal constraints about the actions conditions and effects. We focus particularly on computing explanations that help the user figure out how failures in the execution of one or more actions propagated to later actions. To this end, we define a model that enriches knowledge about the nominal execution of the actions with knowledge about (faulty) execution modes. We present an algorithm for computing diagnoses of TMAPs execution failures, where each diagnosis identifies the actions that executed in a faulty mode, and those that failed instead because of the propagation of other failures. Diagnoses are then integrated with temporal explanations, that detail what happened during the plan execution by specifying temporal relations between the relevant events.

Keywords: Temporal Multiagent Plans · Model-Based Diagnosis · SMT.

1 Introduction

Multiagent plans (MAPs) are an efficient way for accomplishing complex goals. The underlying principle is to decompose a given complex goal into subgoals, and then organize the activities of a team of agents so as that each agent achieves a subgoal autonomously while coordinating with others. Plan execution, however, is not always straightforward. The actual execution of actions, in fact, can be affected by failures. When a failure occurs, detecting and diagnosing it is of primary importance in order to resume the nominal execution.

The diagnosis of the execution of a multiagent plan (MAP) has been addressed in a number of works (see e.g., [8, 6, 7]), proposing different notions of plan diagnosis and different diagnostic methodologies. These works, however, do not model time explicitly, but only implicitly by assuming a sequence of identical time steps at which atomic actions are performed. In these approaches, thus, it is not possible to model *durative actions* [3]; however, in real world scenarios, action duration (either within a nominal range, or with unexpected delays) can strongly affect the success of the agent's plan and of its interactions with others.

Other works have addressed the diagnosis of delayed actions in MAPs [10, 9, 12]. Their objective is to provide the user with explanations of failures consisting only of actions delays; whereas, the logical effects of action failures (i.e., missing logical values that should hold as a consequence of an action) are not taken into consideration. This restriction limits the applicability of the methods by hindering their ability to handle cases of fault propagation from an action to another one due to a missing effect.

This paper contributes with a comprehensive framework addressing the diagnosis of a MAP execution by taking into account both missing effects and temporal deviations. We adopt a consistency-based notion of diagnosis: a MAP diagnosis is a subset of actions whose non-nominal behavior is consistent with the observations received so far. We then argue that, in a setting with agents interactions and durative actions, such diagnoses may not be informative enough for helping a human user figure out what happened during the plan execution. As a remedy, we enrich diagnoses with *temporal explanations* that clarify how primary action failures may have affected other actions in the MAP, even those assigned to different agents. In this paper, we do not address the role of the human in the diagnosis loop, directly. An outline of how human controllers can be involved is given [9]. We deem, however, that the synthesis of such temporal explanations is a fundamental step to increase users' awareness.

Specifically, we propose a methodology to solve a diagnostic problem by inferring the set of all the preferred diagnoses with *minimal rank* [4], i.e., with the highest (order-of-magnitude) likelihood. Our approach is based on a single, centralized diagnostic reasoner that must diagnose the behavior of a multiagent system. Since we deal with both logic and temporal constraints to model faulty action modes, the computation of all the preferred diagnoses is made by exploiting a Satisfiability Modulo Theories (SMT) solver, that is able to handle both kinds of conditions. We model propagation by considering literals that are shared among the actions (i.e., produced as an effect by an action, and consumed as a precondition by another action, even of a different agent). These shared literals can be considered as resources that are dynamically generated, and consumed, during the execution. To explain an action failure as an indirect consequence of a previous failure, thus, we focus on the events that affect the values of the literals shared by the two actions.

To the best of our knowledge, our proposal is the first one dealing with both temporal and logic aspects in the diagnosis of multiagent plans. The most similar work we are aware of is [2], where, however, the authors consider only plans with a limited number of discrete time steps amenable to a SAT encoding, and concentrate on conflicts among agents in the use of resources (e.g., road intersections).

2 Temporal Multiagent Plans

We formalize a Temporal Multiagent Plan (TMAP) P as $\langle T, A, O, CL, M\rangle$:

- T is the team of cooperating agents $ag_1, ag_2, \ldots$

- A is the set of action instances $ac_1, ac_2, \ldots$ included in the plan, each of which is assigned to a specific agent $agent(ac_i)$;
- O is a set of order constraints, that specifies a total order relation over the actions of each agent $ag \in T$; each pair $\langle ac, ac' \rangle \in O$, $ac, ac' \in A$, $agent(ac) = agent(ac')$ means that ac is the *predecessor* of ac', and ac' is the *successor* of ac;
- CL is the set of causal links between an action ac that *produces* a literal R (i.e., has $\neg R$ as pre-condition and R as effect) and another action ac' that *consumes* the literal R (i.e., has R as pre-condition and $\neg R$ as effect); in general $agent(ac)$ can be different from $agent(ac')$; in such a case we say that R is a *shared literal*;
- M is the set of all the possible behavioral modes that can be associated with the action instances in A. In particular, $M(ac)$ denotes the set of all modes associated with instance $ac \in A$. Each mode $m \in M(ac)$ is a tuple of the form $\langle label, pre, \mathit{eff}, range, rank \rangle$:
  - $label$ is the mode name;
  - pre and eff are sets of grounded literals: pre is the pre-condition for the execution of ac in mode m; whereas, eff is the set of effects obtained by performing ac in mode m[1];
  - $range$ is an interval of time corresponding to the possible durations of the action when it behaves in mode m;
  - $rank$ is a non-negative integer value representing the order-of-magnitude probability of the mode [4]: lower ranks correspond to higher probabilities.

  Set $M(ac)$ must contain at least one distinguished mode N (nominal) with rank 0. Ranks are sometimes also named *levels of surprise*, indicating how much surprising is an event for an involved operator. Therefore, they can be usualy specified by a human expert instead of learned from data that may be unavailable.

We have omitted concurrency and mutual-exclusion constraints from this definition in order to avoid excessive complexity and keep our focus on diagnosis. While the causal links in CL, as we defined it, cannot capture all the forms of mutual exclusion, we shall see that implicit mutex constraints play an important role in agents interactions through shared literals. Concurrency and other forms of mutual exclusion, e.g., the use of a resource that has a single instance, could be easily accommodated in our framework.

If we assume that all the actions will be executed in the N mode, a TMAP can be interpreted as a flexible schedule of the plan [11], that guarantees that all the causal links are respected and all the plan actions are smoothly executed. However, the TMAP also contains fundamental information associated with the possible actions failures. In particular, modes different from N are not used for the planning purpose, but for the diagnostic one; such modes allow actions to obtain different effects from the nominal, expected ones.

[1] For the sake of discussion, we assume that all modes $M(ac)$ of an action ac have the same preconditions pre.

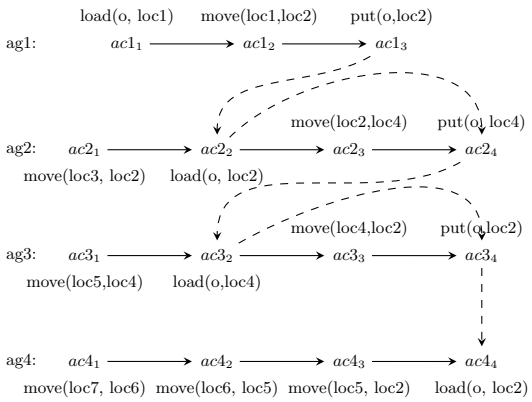

Fig. 1. An example TMAP.

Example 1. As an example TMAP P, let us consider a case with four agents: T = $\{ag1, ag2, ag3, ag4\}$ (see Figure 1). The set of actions is $A = \{ac1_1, \ldots, ac4_4\}$, with order relations O and (nominal) causal links CL as shown in the figure, respectively, by the solid and dashed arrows.

Now, we assume that the fifteen actions included in MAP P are instances of just three types of actions: *move*, *load*, and *put*. Intuitively, in the TMAP for Figure 1, the four agents have to cooperate for moving an object o from location $loc1$ to location $loc2$ and then $loc4$, and then back to $loc2$ again. For instance, ag_1 moves object o by loading it in $loc1$, and carrying it to location $loc2$. Note the nominal causal link between $ac1_3$ and $ac2_2$, meaning that in a nominal execution of the plan $ag2$ will load block o from $loc2$ after it has been moved there by $ag1$.

The table in Figure 2 shows the modes $M(ac)$ of each action type, with associated label, pre- and post-conditions, range, and rank.

For instance, in nominal mode (N), a *move(ag,p1,p2)* requires the agent to be in place $p1$, causes the agent to arrive in place $p2$, and has an execution time in the interval $[1, 3)$. The rank is 0, meaning that the N mode is preferred (i.e., the most likely). In mode $F3$, the action has an execution time in the interval $[10, 25]$, a rank 3, and leaves the agent in $p1$. Similar modes and time intervals are associated with the *load* and *put* actions. Of course, parameter o in *load* and *put* represents an object that can be manipulated by multiple agents during the TMAP execution.

It is important to note that, for all action types, the fault modes have the same pre-conditions of the N mode (but different effects, at least in terms of duration). When such common pre-conditions are not satisfied, we assume a special mode F_{PROP} (failure propagation). This mode denotes that the action was skipped (i.e., it never started). The set of the effects of F_{PROP} is therefore empty, and the action duration is 0. Also the rank is 0 because it represents a secondary failure, and hence does not contribute to the rank of the overall

act	pre	m	post	range	rank
move(ag,p1,p2)	at(ag,p1)	N	at(ag,p2)	[1,3)	0
		F1	at(ag,p2)	[3,10)	1
		F2	at(ag,p2)	[10,25]	2
		F3	$\emptyset$	[10,25]	3
load(ag,p,o)	at(ag,p), at(o,p), holds(ag,$\emptyset$)	N	$\neg$at(o,p), holds(ag,o)	[1,2)	0
		F1	$\neg$at(o,p), holds(ag,o)	[2,10)	1
		F2	$\emptyset$	[10,25]	2
put(ag,p,o)	at(ag,p), holds(ag,o)	N	at(o,p), $\neg$holds(ag,o)	[1,2)	0
		F1	at(o,p), $\neg$holds(ag,o)	[2,10)	1
		F2	$\emptyset$	[10,25]	2

Fig. 2. Example modes.

diagnosis. An action in F_{PROP} mode does not have direct effects on the world, but may have indirect effects on plan execution since some of the missing effects could be preconditions for subsequent actions.

3 Plan Execution Failure Problem

Timed Observations. We define a timed observation as a pair $\langle e, t\rangle$, where e is the observed event, and t is the time when e occurred. In our TMAP framework, an observable event can be a single ground literal, possibly with a negative polarity. For instance, $at(ag1, p1)$ and $\neg at(ag1, p1)$ are two alternative observable events. Of course, we assume that observations are reliable and consistent (i.e., the same literal does not appear with both polarities at the same time). During the execution of a plan, only a few of these events will be observed (due to *partial observability*).

Plan Execution Failure (PEF) problem. It is important to note that the agents share the same environment and resources, and cooperate with each other by exchanging services: the effects brought about by an agent may be the preconditions for the actions of another agent. In principle, therefore, the misbehavior of an agent could affect its later activities as well as other agents' activities.

We say that action ac is *ready* when its predecessor ac' s.t. $\langle ac', ac\rangle \in O$ has finished. We assume that, after an action is ready, it will execute as soon as all its pre-conditions are true. In fact, as a consequence of previous failures, the preconditions could be brought about too late, or might even not be provided at all. Let $P = \langle T, A, O, CL, M\rangle$ be a TMAP.

Definition 1. *A mapping $H : A \to M(A) \cup \{F_{PROP}\}$ is a* hypothesis *about the modes of actions in P that assigns each action $ac \in A$ with a mode $m \in M(ac)$ or special mode F_{PROP}.*

Since action modes are associated with time intervals and logic pre-/post-conditions, a hypothesis H can be used to estimate a set of possible executions of P, that may differ for the times at which actions start and end; we call these possible executions *temporal execution profiles*.

Definition 2 (Temporal Execution Profile). *Given a TMAP P, and a hypothesis H, a temporal execution profile θ is an ordered sequence of pairs $\langle s_0, t_0\rangle$, $\ldots$, $\langle s_n, t_n\rangle$, such that s_i $(i : 0..n)$ is a state of the whole system consisting of all the atoms holding at time t_i. For each $ac \in A$, the events $T_s(ac)$ (start) and $T_e(ac)$ (end) occur in exactly two states, s_i and s_k, respectively, such that t_i precedes t_k. Moreover:*

1. *each s_i is a set of atoms that are true at time t_i and that represents the state of the whole system*
2. *if ac_j starts at time t_i with mode in $M(ac_j)$, then the preconditions of ac_j for mode $H(ac_j)$ (i.e., the mode assigned by H to ac_j) hold at time t_i, and any other action that starts at time t_i, or is already in progress at that time, is not in conflict with ac_j according to the "no moving targets" rule [3], for which no two actions can simultaneously make use of a value if one of the two is accessing the value to update it;*
3. *if ac_j starts at time t_i with mode F_{PROP}, then: $t_i = t_k + \tau$ (where t_k is the end time of the predecessor ac_k of ac_j); the preconditions of ac_j do not hold at time t_i; for each $t \in [t_k, t_i]$, if the preconditions of ac_j held at time t, some other action in conflict with ac_j started or was already in progress at time t*
4. *if ac_j ends at time t_i, then the post-conditions of mode $H(ac_j)$ of action ac_j hold at time t_i*
5. *for each action $ac \in A$, the distance between the times when the action starts and terminates belong to $m.range$ where m is $H(ac)$;*
6. *s_0 is the initial given state;*
7. *s_n is the state where the effects of the last performed actions are added.*

Conditions **2** and **4** state that the pre-conditions and effects of an action ac performed with modality $m = H(ac)$ are true, respectively, when the action starts and when the action terminates. Note that condition **2** ensures that two actions that modify the same literal are executed in mutual exclusion; this is a fundamental constraint for actions that affect the value of a shared literal. Condition **3** states that an action is associated with special mode F_{PROP} only if it has not been allowed to start with true pre-conditions until a timeout τ has expired. Condition **5** imposes that in θ the duration of each action ac respects the intervals of possible durations associated with mode m assumed in H.

Of course, given a TMAP P and a hypothesis H, many temporal execution profiles can be derived: $\mathcal{T}_P(H)$ denotes the set of all possible temporal execution profiles that results from P when only the modalities in H are allowed.

More generally, since each action is associated with a number of modes, we denote with $\mathcal{T}_P$ the space of possible temporal execution profiles for the plan P obtained by considering all possible hypotheses.

Let *Obs* be a sequence of timed observations over actions in P. *Obs* can be used as a filter on $\mathcal{T}_P$ by pruning off those profiles that are not consistent with

them. More precisely, a temporal execution profile $\theta \in \mathcal{T}_P$ is consistent with Obs iff for each timed observation $\langle e, t\rangle \in Obs$, if we let t_i be the unique time instant in θ such that $t_i \leq t < t_{i+1}$, then $s_i \models e$ (where $\langle s_i, t_i\rangle \in \theta$). In other words, the timed observation $\langle e, t\rangle$ must agree with the state of the world s_i that holds at t according to τ.

It is sufficient that this does not hold for one timed observation in Obs to say that θ is not consistent with Obs. We will denote as $\mathcal{T}_P(Obs)$ the subset of the profile space consistent with Obs.

Definition 3 (PEF problem). *A Plan Execution Failure (PEF) problem is a pair $\langle P, Obs\rangle$ where P is a TMAP and Obs a set of timed observations.*

The goal of solving a PEF is to find hypotheses H that are consistent with the observations:

$$\mathcal{T}_P(H) \cap \mathcal{T}_P(Obs) \neq \emptyset. \tag{1}$$

It is well known that the number of consistency-based diagnoses can be very large, especially when the observability is low. Therefore, we are not interested in any hypothesis H that satisfies equation 1, but only in the hypotheses that also satisfy a preference criterion. More precisely, we look for solutions that minimize the rank (i.e., maximize the probability) associated with the action modes.

Definition 4. *Given a TMAP $P = \langle T, A, O, CL, M\rangle$ and a hypothesis H about actions in P, the rank of H, denoted as $rank(H)$, is*

$$rank(H) = \sum_{ac \in A} H(ac).rank.$$

In fact, since we assign rank 0 to failures that depend on previous failure, and the rank of failures that are independent can be comulated, the rank of a hypothesis is simply the sum of the ranks of the modes assumed in the hypothesis itself. Of course, there exists only one hypothesis H^0 with rank 0 in which all actions are assumed nominal.

Definition 5 (PEF solution). *Let P be a TMAP, and let $\langle P, Obs\rangle$ be a PEF problem, a solution to such a problem is an hypothesis δ such that:*

1. *δ satisfies equation (1);*
2. *$rank(\delta)$ is minimal: no other hypothesis H' that satisfies equation (1) has $rank(H') < rank(\delta)$*

As usual in a diagnostic setting, we are not interested in just one solution, but in all minimal solutions, in fact, unless other preference criteria are given, all these minimal solutions should be returned as an answer to a PEF problem.

Example 2. Let us consider the plan of example 1. Although in the original plan, action $put(o, loc2)$ of agent $ag1$ was assumed to make o available for action $load(o, loc2)$ of $ag2$, this may not be the case in a real execution scenario. Assume that the previous action of $ag2$, i.e., $move(loc3, loc2)$, had an $F1$ delay and took

8 time instants. In the meanwhile, the three *move* actions of $ag4$ have taken a total of 6 time instants, so that the object released by $ag1$ at $loc2$ at time 4 is actually loaded by $ag4$. This situation makes actions $ac2_2$, $ac2_4$, $ac3_2$, and $ac3_4$ fail with mode F_{PROP}, because they don't have the necessary preconditions to be executed. However, a diagnosis that (except for N modes) lists: $ac2_1(F1)$, $ac2_2(F_{PROP})$, $ac2_4(F_{PROP})$, $ac3_2(F_{PROP})$, $ac3_4(F_{PROP})$ is not a satisfactory *explanation* of what happened. Indeed, the fact that $ac2_1$ had a delay $F1$ does not necessarily imply all the other events and (propagation) failures: think, e.g., that the delay caused by $F1$ was just a duration of 3 time instants for $ac2_1$. In the next section we propose a notion of temporal explanation that yields more information than just the diagnosis.

4 Explaining Failure Propagations

A solution δ to a PEF problem provides a user with a labeling of (failure) modes to the plan actions that is consistent with the available observations. In particular, a special mode F_{PROP} in δ is used to denote those actions that have been affected by previously occurred action failures (i.e., it is a *secondary* failure). However, this is not in general sufficient, for the user, to understand what has actually happened. In fact, a secondary failure might be caused by the co-occurrence of two or more primary failures (e.g., when two actions delay independently and their consequences sum up affecting a third action). Such configurations are not easy to discover, and to increase the comprehension of a user, a δ diagnosis needs to be further explained to extract implicit, contingent connections between the primary failure(s) and the secondary ones.

Intuitively, failures can propagate via the shared literals, that is, via the resources produced by an action and consumed by another one. For example, an action may fail because one of the required inputs is not available at the right time, and this may happen because the producer failed in supplying it (including supplying it with too much delay), or because another action has erroneously consumed the resource in its place. Explaining δ, thus, means tracing back the temporal relations among the actions that are related to some resource of interest, and whose occurrence justifies a secondary failure.

Definition 6 (Temporal Explanation of δ w.r.t. R). *Let δ be a PEF solution to $\langle P, Obs\rangle$. A* Temporal Explanation *(*explanation *in short) $E(\delta, R)$ of δ w.r.t. a shared literal R is a set of Allen algebra relations among actions in P defined as follows. Let δ_{R+} (resp. δ_{R-}) be the subset of actions in δ that produce (resp. consume) a shared literal R. Moreover, let $\delta_R(F_{PROP})$ (resp. $\delta_R(\overline{F_{PROP}})$) be the subset of $\delta_{R+} \cup \delta_{R-}$ containing actions with mode equal to (resp. different from) F_{PROP}. Then, an explanation $E(\delta, R)$ for δ w.r.t. R is a set such that:*

- *for each $ac \in \delta_R(\overline{F_{PROP}})$, $E(\delta, R)$ specifies two Allen algebra relations ρ_{prec} and ρ_{succ} w.r.t. its predecessor and its successor in $\delta_R(\overline{F_{PROP}})$ (except for the first and last action). Relation ρ_{prec} is either* after *or* meets after*; relation ρ_s is either* before *or* meets*;*

– *for each* $ac \in \delta_R(F_{PROP})$, $E(\delta, R)$ *specifies two Allen Algebra* during *relations* ρ_R *(when ac becomes ready) and* ρ_F *(when ac timeouts and fails with* F_{PROP}*). Relations* ρ_R *and* ρ_F *relate ac either with a single action in* $\delta_R(\overline{F_{PROP}})$*, or with (the interval I in between) two actions* ac', ac'' *in set* $\delta_R(\overline{F_{PROP}})$.

Some comments are in order. First of all, note that, due to the mutual exclusion among actions that produce/consume R, the actions in $\delta_R(\overline{F_{PROP}})$ respect a total order, specified through the (*meets*) *before*/*after* relations in $E(\delta, R)$. Such an order partitions the timeline in a set Π_R of intervals of action execution and intervals between two actions.

In addition, an action $ac \in \delta_R(F_{PROP})$ that was supposed to produce/consume R, but failed because of missing pre-conditions, can actually overlap with actions in $\delta_R(\overline{F_{PROP}})$. In fact, the action has never started, and what we are interested in knowing is the interval between when ac became ready (i.e., when it became the current action for its agent), and when ac failed with mode F_{PROP}. Such events, that determine the interval W during which ac is "willing" to produce/consume R are placed in partition Π_R by *during* relation in $E(\delta, R)$. It follows that W is contextualized in $E(\delta, R)$ against all the other intervals regarding the execution of actions that have handled resource R, and hence provides the user with an explanation of why action ac could not produce/consume R during W.

A (full) explanation of a diagnosis δ is simply a set $E(\delta)$ of several sub-explanations $E(\delta, R)$, one for each shared literal R. Note that, given a diagnosis δ, it is in general possible to find several alternative explanations, corresponding to different orders of events compatible with δ. Such alternatives are equally plausible according to our model, and are therefore computed and returned to the human user. The following examples should help clarify the above definition.

Example 3. Let us refer to example 2. The *producers* of literal $R = at(o, loc2)$ are as follows: $ac1_3$, and $ac3_4$; while the *consumers* of R are: $ac2_2$, and $ac4_4$. According to example 2, the diagnosis δ (except for N modes) lists: $ac2_1(F1)$, $ac2_2(F_{PROP})$, $ac2_4(F_{PROP})$, $ac3_2(F_{PROP})$, $ac3_4(F_{PROP})$. The explanation that we have informally sketched in example 2, should now be formalized as a suitable explanation $E(\delta, R)$. Figure 3 shows $E(\delta, R)$ graphically on a diagram where time increases from left to right. Note that, besides the actions related with R and their Allen algebra relations specified by $E(\delta, R)$ (black), the schema also shows some other actions with mode assignments specified by the diagnosis δ (gray); such actions are depicted just to further increase the information conveyed by the schema to the reader.

The set of non-F_{PROP} actions that have to do with R are just $ac1_3$ and $ac4_4$, so that the timeline is partitioned in five regions (dotted vertical bars): before $ac1_3$; during $ac1_3$; between $ac1_3$ and $ac4_4$; during $ac4_4$; after $ac4_4$. The definition of explanation requires us to relate $ac1_3$ and $ac4_4$, and, in the scenario described by example 2, the relation is $ac1_3$ *before* $ac4_4$, i.e., when $ac1_3$ ends, some time passes before $ac4_4$ becomes ready and consumes $at(o, loc2)$. Note that R would be available for other consumers between the end of $ac1_3$ to the start of $ac4_4$.

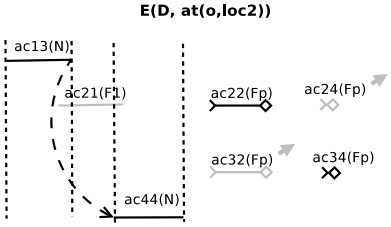

Fig. 3. An Explanation of Diagnosis $\delta = \{ac2_1(F1), ac2_2(F_P), ac2_4(F_P), ac3_2(F_P), ac3_4(F_P)\}$.

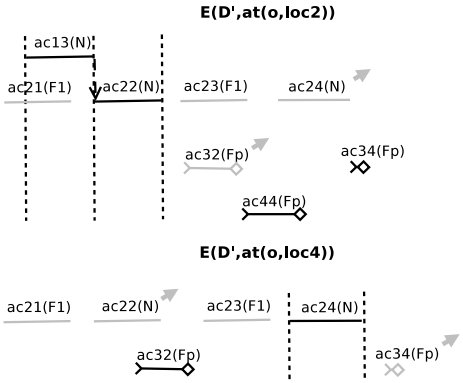

Fig. 4. An Explanation of Diagnosis $\delta' = \{ac2_1(F1), ac2_3(F1), ac3_2(F_P), ac3_4(F_P), ac4_4(F_P)\}$.

However, according to explanation $E(\delta, R)$, $ac2_2$ becomes ready and then fails with mode F_{PROP} (segment starting with $>$ and ending with $\Diamond$) only *after* $ac4_4$ ends. By looking at the figure, it is easy to see that such a delay is due to the failure with mode $F1$ of action $ac2_1$. The figure also shows actions $ac2_4$ and $ac3_2$, that are respectively a *put* and *load* related to another literal $at(o, loc4)$ (indicated by the $\nearrow$ after the actions), which fail as a consequence of the failure of $ac2_2$ (see the causal links in the plan, Figure 1). Finally, also action $ac3_4$ fails as a consequence of the failure of $ac3_2$.

Let us now consider an explanation for a different diagnosis δ', according to which actions $ac2_1$ and $ac2_3$ fail with mode $F1$ (delay), and actions $ac3_2$, $ac3_4$, and $ac4_4$ fail with mode F_{PROP} (Figure 4). The upper part of the figure shows the explanation of δ w.r.t. literal $at(o, loc2)$, while the lower part shows the explanation of δ for literal $at(o, loc4)$. In particular, $ac1_3$ produces literal $at(o, loc2)$, which is consumed by $ac2_2$ (in this case, immediately), as prescribed by the plan. However, we see that although the *put* action $ac2_4$ involving literal $at(o, loc4)$ succeeds, the associated *load* action $ac3_2$ fails with mode F_{PROP}, which in turn propagates to the failure of actions $ac3_4$ and $ac4_4$ on $at(o, loc2)$ that depend on $ac3_2$ (causal links in the plan, Figure 1).

By just looking at the upper part of the figure, then, we are left without an explanation of the failure of $ac3_2$. We have to look at the part of the figure showing the explanation for literal $at(o, loc4)$, where we realize that action $ac3_2$ became ready and then failed with mode F_{PROP} *before* action $ac2_4$ (from which $ac3_2$ depends) was executed. The cause of the delay is clearly a combination of the delays caused by the failures of actions $ac2_1$ and $ac2_3$ with mode $F1$.

5 Translation to SMT

In order to address a PEF problem by exploiting an SMT solver, we have to encode the TMAP and the observations *Obs* in the language accepted by the solver. We recall that an SMT problem is an extension of the well known propositional satisfiability (SAT) problem where formulas can contain relations and functions from various theories including real and integer linear arithmetic. Similar to SAT, when a set of formulas is satisfiable, the solver is able to return a satisfying assignment to the variables. In this work, we have adopted the Z3 solver [1]. Due to space constraints, we will focus just on the most relevant aspects of the encoding process. In order to encode action types (e.g., *move*, *load*), we need to encode the predicates that appear in their pre-conditions and effects, e.g. *at* and *holds*. We define them as *uninterpreted functions* (UF), i.e., functions for which the Z3 solver will try to find an interpretation that satisfies the set of formulas being checked. Note that most of the predicates are in fact *fluents*, i.e., they have time as one of their arguments. For example, *at(ag,p,t)* asserts that agent ag is at place p at time t. For diagnostic purposes, a fundamental predicate is *mode(ac,m)*, that defines the mode m of an action ac.

Action types, with their behaviors determined by modes, pre-conditions and effects, are expressed as *defined functions* (DF). Unlike UFs, DFs have a body that specifies how to compute the function value given the arguments. A DF receives all the parameters relevant to the action, plus two time points T_s and T_e that represent the action starting and ending times, and returns a Boolean value. For example, the signature of the *move* action is:

$$move(ag : Agent, ac : Action, from : Pos, to : Pos, T_s : Int, T_e : Int) : Bool$$

The body of the DF specifies, for each mode $m \in M(ac)$ and for the special mode F_{PROP}, the pre-conditions and the effects taken from the TMAP definition.

$$\begin{array}{ll} if\ (\textit{pre-cond}) & mode(ac, N) \Rightarrow [N\ \textit{post-cond}] \\ & \ldots \\ & mode(ac, F_k) \Rightarrow [F_k\ \textit{post-cond}] \\ else & mode(ac, F_{PROP}) \wedge [F_{PROP}\ \textit{post-cond}] \end{array}$$

The plan itself is encoded as a sequence of assertions that build the instances of action types that make up the plan. Finally, the timed observations *Obs* are easily encoded by asserting the truth of the associated fluent, e.g. the observation $\langle at(ag1, p1), t1\rangle$ will be encoded by asserting $at(ag1, p1, t1)$.

Constraints between time points are expressed as linear arithmetic relations:

$$T_e(ac) > T_s(ac);\ T_e(ac') < T_s(ac) \leq T_e(ac') + \tau$$

Note that, in the second formula, ac' is the predecessor of ac in the plan of the agent. A fundamental point that needs to be addressed by our translation is the definition of suitable *frame-axioms*, i.e., formulas prescribing that a fluent does not change if none of the actions changing it is taken. For instance, in our example logistic domain, fluent *at(ag,p,t)* is only possibly changed at the end of

a *move* action. Moreover, no other agent can change the value of *at(ag,p,t)*. So, for each action ac that is not a *move*:

$$at(ag, p, T_e(ac)) = at(ag, p, T_s(ac)) \text{ and } at(ag, p, T_s(ac)) = at(ag, p, T_e(ac'))$$

where ac' is the predecessor of ac. Things are more complicated for fluents such as *at(o,p,t)* (where o is an object) that can be changed by multiple agents. According to our assumptions, we impose that such actions must be executed in mutual exclusion. However, in general, for each action ac, we must also assert that:

$$at(o, p, T_s(ac)) = at(o, p, max(\{T_e(ac') : T_e(ac') < T_s(ac)\}))$$

where all actions ac' that can modify the fluent are considered by the $max()$ operator on the right hand side. The encoding of time and persistency relations highlights the benefits of adopting a SMT solver instead of a SAT solver for checking the consistency of hypotheses. In a SAT-encoding of a plan whose timespan is $[0, N]$, it is necessary to create a copy of each variable for each time instant in $[0, N]$. On the contrary, the SMT encoding allows us to focus just on the values of the fluents at the relevant time points, that for a TMAP are the start/end times of actions and the times of observations.

Concurrency constraints are the most difficult ones to encode, especially because we require that an action is actually executed as soon as it is possible to do so. We can't describe in detail such constraints due to lack of space. Suffice it to say that, for each shared literal R in the TMAP P, we need to introduce a predicate $wants(ac, \pm R, T)$, that denotes the fact that an action ac wants to consume $(+)$ or produce $(-)$ literal R at time T. Then, we specify a number of constraints involving the actions $P_R = P_{R+} \cup P_{R-}$ (that produce/consume R) to handle the situations that can arise during plan execution: mutual exclusion, waiting for R to be produced/consumed (possibly competing with other waiting agents), timing out and executing in mode F_{PROP}.

6 Solving PEF Problems

Given the encoding of a PEF problem in the input language of Z3, we exploit the ability of Z3 to produce an *unsat core* every time it is invoked on an unsatisfiable instance. An unsat core is a set of assertions in the input to Z3 that cannot hold simultaneously and therefore require to withdraw at least one of them in order to get satisfiability. Given the set of unsat cores that is cumulatively produced during the search for the solutions, we can avoid to explore the parts of the search space that do not *hit* (i.e., withdraw at least an assignment from) all of them. This technique is well known in diagnosis, also on approaches based on SMT [5].

Let us assume that we have a function *EncodeTMAPZ3* that, given a TMAP P, encodes it in the Z3 input language as explained in the previous section. Figure 5 shows the CBFS (Conflict-based Best First Search) diagnostic algorithm for solving a PEF specified by P and *Obs*. The algorithm is strongly based on the

CBFSDiagnosis(P = ⟨T,A,O,CL,M⟩, Obs)
1. Sys ← EncodeTMAPZ3(P)
2. Pef ← Sys ∪ EncodeObsZ3(Obs)
3. UCores ← ∅; Δ ← ∅; done? ← **false**; best ← ∞
4. **while not** done? **do**
5. σ ← NextBestPlateauResolvingConflicts(UCores)
6. **if** rank(σ) > best **then**
7. done? ← **true**
8. **else**
9. Pef_σ ← Pef ∪ EncodePlateauZ3(σ)
10. (μ, γ) ← CheckSATZ3(Pef_σ)
11. **if** $\mu \neq$ **null then**
12. best ← rank(σ)
13. **while** $\mu \neq$ **null do**
14. $\delta \leftarrow \text{project}(\mu, \{mode(ac, m) \in \mu : ac \in A\})$
15. $\Delta \leftarrow \Delta \cup \{\delta\}$
16. $\text{Pef}_\sigma \leftarrow \text{Pef}_\sigma \cup \text{EncodeAssignmentZ3}(\neg\delta)$
17. (μ, γ) ← CheckSATZ3(Pef_σ)
18. **end while**
19. **else**
20. UCores ← UCores ∪ γ
21. **end if**
22. **end if**
23. **end while**
24. **return** Δ

Fig. 5. The CBFS diagnostic algorithm.

high-level schema of Conflict-directed A^* (cd A^*) [13], with some variations explained below.

At each iteration of the top-level *while* loop, algorithm cd A^* would require to generate a full assignment of modes to actions that resolves the conflicts found so far. Instead, we generate a constraint σ on the modes of the actions with function *NextBestPlateauResolvingConflicts()* (line 5). Such a constraint: (i) contains specific assignments σ_F of faults (excluding F_{PROP}) to actions in order to hit all the unsat cores $\gamma \in$ *UCores*; (ii) constrains the remaining actions to have either mode N or F_{PROP}; (iii) has minimum rank among assignments that hit *UCores*. Therefore, σ looks as follows:

$$\begin{aligned}\sigma = & ac_1^F(\varphi_1) \wedge ... \wedge ac_m^F(\varphi_m) \wedge \\ & (ac_1^{R0}(N) \vee ac_1^{R0}(F_{PROP})) \wedge ... \wedge (ac_n^{R0}(N) \vee ac_n^{R0}(F_{PROP}))\end{aligned}$$

where actions $ac_i^F \in \sigma_F$ are assigned a specific faulty mode φ_i (excluding F_{PROP}), while actions ac_i^{R0} (where the superscript $R0$ denotes the fact that such actions contribute a rank 0 to the assignment) can take mode N or F_{PROP}. This explains the term *plateau* in the name of the function that computes constraints σ: a single constraint may indeed generate several diagnoses of equal rank (i.e.,

Explain(Pef, δ)
1. $\text{Pef}_\delta \leftarrow \text{Pef} \cup \text{EncodeAssignmentZ3}(\delta)$
2. $(\mu, \gamma) \leftarrow \text{CheckSATZ3}(\text{Pef}_\delta)$
3. **while** $\mu \neq$ **null do**
4. $\quad e_{raw} \leftarrow \text{project}(\mu, \{T_s(ac) \in \mu : ac \in A\})$
5. $\quad e_{All} \leftarrow \text{EncodeAllenAlgebra}(e_{raw})$
6. $\quad E \leftarrow E \cup e_{All}$
7. $\quad \text{Pef}_\delta \leftarrow \text{Pef}_\delta \cup \neg e_{All}$
8. $\quad (\mu, \gamma) \leftarrow \text{CheckSATZ3}(\text{Pef}_\delta)$
9. **end while**
10. **return** E

Fig. 6. The Explain algorithm.

cost) by assigning combinations of modes N or F_{PROP} to the ac_i^{R0} actions (see below).

When all the minimum rank solutions Δ to a given TMAP have already been found, a constraint σ with a higher rank than the *best* one is generated (line 6), and the algorithm returns set Δ. Otherwise, the constraint σ is added to the Z3 encoding *Pef* of the PEF problem (TMAP and observations), and the result Pef_σ is then checked by Z3 for satisfiability. If Pef_σ is unsatisfiable, Z3 returns an unsat core γ, that is added to the set *UCores*.

Otherwise, a satisfying model μ is returned by Z3. The *best* rank of solutions is updated with the rank of σ_F. Then, the algorithm enters an inner while loop where: the full assignment δ to the action modes prescribed by μ is added to the set Δ of preferred diagnoses; and then Pef_σ is checked again for satisfiability excluding δ (to avoid finding it again).

The explanations of a diagnosis δ are computed with the *Explain* algorithm shown in Figure 6. Diagnosis δ is added to the encoding *Pef* of the PEF problem solved by δ, and the result Pef_δ is checked for satisfiability with Z3. Of course, since δ is a diagnosis, the while loop is entered at least once. The times of start and end of each action are extracted from model μ, and then they are abstracted into set e_{All} of the corresponding Allen algebra relations introduced in definition 6. For example, if a *put(ag,p,o)* ends at time t, and a *load(ag',p,o)* starts at time $t + 1$, then a relation *meets* is established between *put* and *load*. After adding e_{All} to the set E of explanations of δ and negating it in Pef_δ, Z3 is called again to look for other explanations of δ.

7 Implementation and results on Test Cases

We have implemented the two SMT-based approaches to diagnosis described above as Java programs exploiting the Z3 solver. The tests have been run on a machine running Ubuntu 18.04.1 LTS, equipped with an i7 7700HQ CPU at 2.80 GHz, and 8 GB RAM. We have considered a *Logistic* domain which reflects the domain used in the examples. In such a domain, as mentioned above,

	CBFS			
	time	**#sol**	**time/sol**	**#expl**
ag 2				
ac 8 (R2)	0.48	2.0	0.24	2.0
ag 4				
ac 10 (R2)	1.32	2.5	0.53	3.0
ac 20 (R2)	6.83	4.0	1.71	6.1
ac 20 (R4)	25.53	15.6	1.64	23.2

Table 1. avg time (sec), sols, time/sol, and explanations of experiments.

agents can *move*, *load*, and *put* objects, giving rise to several kinds of inter-agent interactions. We have experimented our approach by running a number of software simulated tests under different *configurations*, defined by varying the following dimensions: $\#ag$ (2 and 4 agents); $\#ac$ (8, 10, 20 actions per agent); and $\#rnk$ (injected failures of ranks 2, 4). In order to study the effect of interactions among agents, we have introduced inter-agent links in the plans used in the configurations as follows: 2 ag x 8 act with 2 links; 4 ag x 10 act with 3 links; and 4 ag x 20 act with 7 links.

The observability rate (i.e., ratio between the number of actions with observable effects and the total number of actions) was 30%. We have chosen this level of observability because it has proved to be high enough for our algorithm to (almost) always include the diagnosis with the injected failures in the list of preferred diagnoses, and low enough to challenge our algorithm with a certain ambiguity in discriminating between the "real" diagnosis and alternative ones.

In Table 1, we show results obtained with 4 different configurations of increasing complexity. The average total time for solving the PEF problems goes from 0.48s (2 agents x 8 actions, rank 2), up to 25.53s (4 agents x 20 actions, rank 4). It should be noted that the total time includes the computation of all the preferred diagnoses, as well as their temporal explanations. If we look at the average time taken for computing each preferred diagnosis (including its explanations), the increase is more limited, going from 0.24s to 1.71s. Indeed, as the test cases become more challenging (more agents, more actions, higher rank of failures), the average number of preferred diagnoses increases (from 2.0 to 15.6), as well as the average number of associated explanations (from 2.0 to 23.2). Note that the time/sol of the 3^{rd} and the 4^{th} configurations is almost the same, despite the fact that the former has test cases with rank 2 and the latter of rank 4. This seems to indicate that the time/sol is not affected significantly by the rank of test cases.

8 Conclusions

The diagnosis of Temporal Multiagent Plans (TMAPs) has been addressed by a number of approaches in literature that focused either on diagnosing delays [10, 9, 12], or on diagnosing violation of logic conditions [8, 6, 7]. In this paper, we

have presented a novel approach that deals with both aspects. As a consequence, the propagation of failures from one action to another (and one agent to another one) is particularly complex, because it can be due to delays and/or missing logic effects. Therefore, in our framework we first single out diagnoses (possibly containing secondary failures) by means of a conflict-based search. We then explain these secondary failures by inferring the temporal profile of the production/consumption of shared resources whose misuse caused the very same failures. These temporal profiles allow a user to gain a better understanding about the causes of a secondary failure by relating it to the (primary) failure of another action that has caused an unexpected effect on some shared resource.

In this work, action failures in plan execution are considered as independent of one another, except when the actions interact through shared literals. Following [10], we may try to extend the present work by considering that action failures can be related also when they involve some common features of the agent or the environment (e.g., a motor or a traffic jam for a *move*). Since plan diagnosis is the pre-condition for plan repair, another future line of work will explore how to exploit the (on-line) computation of diagnoses to inform a re-planning process that tries to achieve (most of) the original goals.

References

1. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Int. conf. on Tools and Alg. for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
2. Elimelech, O., Stern, R., Kalech, M., Bar-Zeev, Y.: Diagnosing resource usage failures in multi-agent systems. Expert Systems with Applications **77**, 44–56 (2017)
3. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. J. Artif. Intell. Res. (JAIR) **20**, 61–124 (2003)
4. Goldszmidt, M., Pearl, J.: Rank-based systems: a simple approach to belief revision, belief update, and reasoning about evidence and actions. In: Proc. KR92. pp. 661–672 (1992)
5. Grastien, A.: Diagnosis of hybrid systems with smt: opportunities and challenges. In: Proceedings of the 21^{st} European Conf. on AI. pp. 405–410. IOS Press (2014)
6. de Jonge, F., Roos, N., Witteveen, C.: Primary and secondary diagnosis of multi-agent plan execution. Journal of Autonomous Agent and MAS **18**, 267–294 (2009)
7. Kalech, M., Kaminka, G.A.: On the design of coordination diagnosis algorithms for teams of situated agents. Artificial Intelligence **171**(8-9), 491–513 (2007)
8. Micalizio, R., Torasso, P.: Cooperative monitoring to diagnose multiagent plans. Journal of Artificial Intelligence Research **51**, 1–70 (2014)
9. Micalizio, R., Torta, G.: Diagnosing delays in multi-agent plans execution. In: Proceedings of the 20th ECAI. pp. 594–599. IOS Press (2012)
10. Micalizio, R., Torta, G.: Explaining interdependent action delays in multiagent plans execution. Autonomous Agents and Multi-Agent Syst. **30**(4), 601–639 (2016)
11. Policella, N., Smith, S.F., Cesta, A., Oddi, A.: Generating robust schedules through temporal flexibility. In: ICAPS. vol. 4, pp. 209–218 (2004)
12. Roos, N., Witteveen, C.: Diagnosis of simple temporal networks. In: Proc. of ECAI'08. pp. 593–597 (2008)
13. Williams, B.C., Ragno, R.J.: Conflict-directed a* and its role in model-based embedded systems. Discrete Applied Mathematics **155**(12), 1562–1595 (2007)