HIPR: An Architecture for Iterative Plan Repair in Hierarchical Multi-agent Systems

Swarup Kumar Mohalik[1*], Mahesh Babu Jayaraman[1], Ramamurthy Badrinath[1], Aneta Vulgarakis Feljan[2]

[1] Ericsson Research, Bangalore-560037, India.
[2] Ericsson Research, Kista-16480, Sweden.

* Corresponding author. Tel.: 91-9945698671; email: swarup.kumar.mohalik@ericsson.com
Manuscript submitted March 7, 2017; accepted July 30, 2017.
doi: 10.17706/jcp.13.3.351-359

Abstract: In large scale multi-agent systems, both planning for system goals and replanning during plan execution to handle failures are compute-intensive. Since replanning requires faster response time because it happens during plan execution, a lot of focus in the AI planning literature has been on incremental methods, such as plan repair, plan modification etc., which avoid re-synthesizing a complete plan. In this paper, we propose HIPR - an architecture and supporting algorithms for fast replanning in multi-agent systems with two fairly general characteristics: (1) where the agents are organized hierarchically based on attributes like location/administration and (2) where most failures are localized i.e. only a few agents are affected by the failure while most of the agents at large remain unaffected. HIPR exploits these characteristics to identify the smallest group of agents that are affected by the failure and to synthesize new plan fragments for only these agents. The localization to smaller number of agents generates smaller replanning problems and hence more efficient solutions. We illustrate application of HIPR on a small, yet realistic route planning use case.

Key words: Artificial intelligence, hierarchical domains, planning, replanning.

1. Introduction

In a multi-agent system (MAS), system-wide goals are achieved through plans for individual agents and tight coordination among them (Refer to Chapter 11.4 in Russell and Norvig [1]). The system adapts to failures in plan execution caused by unexpected changes in the environment state through dynamic replanning for the agents. Since naive implementation of replanning where plans are generated for the entire system is costly, plan repair techniques are usually more attractive. However, in a MAS, plan repair must take into account re-coordination of the plans of the agents which results in significant conceptual and algorithmic complexity. In this paper, we propose a simple and intuitive method for plan repair in a MAS where (1) agents are organized in a hierarchical fashion, and (2) where most failures are localized so that the system-wide impact is infrequent.

The traditional MAS comprises a set of agents that are peers. However, natural groupings are introduced in real-life deployments because of the spatial and organizational nature of the agents. For example, the agents deployed for a set of devices in a building floor are grouped through a gateway agent, a set of fleet vehicles are grouped through an aggregator agent for the purposes of scheduling and driver assignment, agents belonging to an enterprise belong to a group for management, maintenance and accounting

purposes etc. Such grouping leads to a natural hierarchical organization of the agents, where the agents at the leaf level provide the real operational capabilities and the ones at higher levels are responsible for deciding the actions for the operational agents and for the coordination among them. Such an organization of agents can be exploited for efficient plan repair.

The other characteristic we exploit is the localized nature of failures due to the loosely-coupled, asynchronous nature of agents in most real-life systems. Localization implies that only a small number of agents are affected by any failure and that the necessary repairs can also be restricted to a few agents. In cases where the coordination requirements force us to repair the plans of other indirectly affected agents as well, we identify a minimum number of agents from the agent hierarchy that need replanning and compute the corresponding plan fragments to be repaired. Since often the required degree of coordination reduces at higher levels of the hierarchy (e.g. agents are farther from each other or are in different organizations), the level at which replanning needs to be done is expected to be small. This can result in smaller (sub)plan synthesis problems leading to efficient plan repair.

Our work is based on the action-based planning formalism where a sequence of actions is synthesized to achieve a final goal state from a given initial state using the knowledge about the precondition and effect of the actions. The main contributions of this work is an architecture and supporting methods for repairing a failed plan in an iterative manner over increasing levels of the hierarchy of agents. We call this Hierarchical Iterative Plan Repair (HIPR) architecture. The salient points of HIPR are: (1) decomposition of global plans for individual agents, (2) identification of the fragments of plans for a group of agents that do not need coordination with agents outside the group and (3) replacement of these fragments with new plans ensuring the progress towards the original goal state. These techniques are illustrated on a small, yet realistic route planning use case.

The paper is organized as follows. In Section 2, we present preliminary notions for the planning framework of hierarchical multi-agent systems as an extension of classical planning. In Section 3, we present the route-planning use case. Sections 4 through 6 present our contributions: the notion of local plans and their execution is given in Section 4, the core technique of identifying a suitable plan fragment and replacing it by another is presented in Section 5, and the complete HIPR architecture is described in Section 6. In Section 7, we recall and compare related work in the literature and conclude with direction for future work in Section 8.

2. Planning Framework

In this paper we consider hierarchical multi-agent systems (or HMAS for short). Let AG be a pool of agents in the HMAS. At any point of time, the agents are arranged in a tree hierarchy. The agents at the leaf (lowest level) are the *operating agents* and the internal agents (higher levels) are the *managing agents*. The root of the tree is called the *Root agent*. The set of operating agents that are descendants of agent A is noted by $desc(A)$. For an operating agent A, $desc(A) = \{A\}$.

Let $D = \langle Obj, Pred, Act \rangle$ be the domain of the root (or, the system domain), where Obj is a set of objects, $Pred$ a set of predicates over the objects and Act, a set of actions specified through precondition/effect pairs: for an action $a \in Act$, $precond(a)$, $effect(a) \subseteq Pred$ denote the precondition and effect of action a resp.

Denote the set of objects and predicates occurring in an action (precondition or effect) by $objSet(a)$ and $predSet(a)$ resp. This is extended to sets of actions. Each action is mapped to a unique operating agent by a one-to-one function $\mu(.)$. The domain for an agent is derived from this mapping: $D_A = \langle Obj_A, Pred_A, Act_A \rangle$, where $Act_A = \{a | a \in Act \text{ and } \mu(a) \in desc(A)\}$, $Obj_A = \text{objSet}(Act_A)$ and $Pred_A = predSet(Act_A)$.

A State of an agent A is defined as a consistent subset of predicates from $Pred_A$. An action a is enabled in a given state S if its precondition is satisfied. If the same action is executed in the state, it causes the state S to change to a new state S' according to its effects. We denote this state transition by $\delta(S, a) = S'$.

A Plan for an agent A is a partial order $(P, \leq)$ of actions from Act_A, where $\leq$ is the reflexive, transitive closure of the immediate dependence relation $<_m$ which is derived from the precondition and effect predicates of the actions. For any action a in the plan, let $In(a) = \{a_j \mid a_j <_m a\}$ and $Out(a) = \{a_j \mid a <_m a_j\}$. $In(a)$ denotes the actions on which a depends upon, $Out(a)$ denotes the actions depending upon a.

A plan describes some possible sequences in which actions can be executed. Given a plan P, let $execSeq(P)$ be all the linearized sequences of the partial order plan. For a state s_0 and a sequence $x = a_1 \ldots a_n \in execSeq(P)$, an execution results in a sequence of transitions $s_0\{a_1\}s_1\{a_2\} \ldots \{a_n\}s_n$, where for all i=1…n, $\delta(s_{i-1}, a_i) = s_i$. Abusing notation, we denote this fact as $\delta(s_0, x) = s_n$. It can be proved that from a given initial state, all the sequences from $execSeq(P)$ lead to the same final state. So, we may assert $\delta(s_0, P) = s_n$ without any ambiguity.

For an agent A, a problem Pr_A is defined as a pair (I_A, G_A) where I_A is the initial state and G_A, the goal state. The plan synthesis question for agent A is to synthesize a plan P_A using its domain D_A such that $\delta(I_A, P_A) = G_A$.

Note: From the perspective of an operating or managing agent A, all the definitions above are identical to those in the classical planning problem for single agent within the domain DA. Hence, existing single agent planners like OPTIC [2], FF [3] can be used to synthesize plans for the planning problems specified for the agent.

3. Motivating Example

The HMAS and the replanning techniques proposed in this paper are illustrated through the following example where there are three operating agents: the *plane agent* controls a plane, the *train agent* controls a train and a car agent controls three cars (namely, car_1, car_2 and car_3). The Land Transport Agent (LTA) is a managing agent for the train and the car agents. At the root is the Route Planning Agent (RPA) which provides route planning service to users. The organization hierarchy of the agents and the routes are shown in Fig. 1. The circles denote airports, the squares denote train stations and the triangles denote road milestones. These constitute the location points.

The predicates $at_x(\ell), x \in \{person, t(train), p(plane), car_1, car_2, car_3\}$ denote that a person or vehicle x is at location point ℓ. Predicates $segment_x(a, b)$ denote the type x of segment (i.e. air, train route, road) between points a and b. For each vehicle type x, we have the action $move_x(a, b)$ which is specified by the precondition $\langle at_x(a), segment_x(a, b), at_{person}(a) \rangle$ and the effect $\langle at_x(b), not(at_x(a)), at_{person}(b), not(at_{person}(a)) \rangle$. Note that the action allows movement of a vehicle only when there is a passenger. This simplification in the model is only to ease the explanation. Initially, the plane is at point a (asserted by the predicate $at_p(a)$), the train is at point 3, car_1, car_2 *and* car_3 are at points 6, 8 and c resp.

Suppose the goal of a person is to travel from Point a to Point 10 specified as a problem $(\{at_{person}(a)\}, \{at_{person}(10)\})$. One possible global plan (action sequence) to achieve this goal is $\langle move_p(a, b), move_p(b, 3), move_t(3,4), move_t(4,5), move_t(5,6), move_{car_1}(6,9), move_{car_1}(9,10) \rangle$. This can be decomposed into three local plans for the three agents (plane, train and car): $\langle move_p(a, b), move_p(b, 3) \rangle$, $\langle move_t(3,4), move_t(4,5), move_t(5,6) \rangle$ and $\langle move_{car_1}(6,9), move_{car_1}(9,10) \rangle$, where there are following cross-agent dependencies between the actions: $move_p(b, 3) \rightarrow move_t(3,4)$ and $move_t(5,6) \rightarrow move_{car_1}(6,9)$. The dependencies follow from an obvious constraint that a person must reach a point first before leaving it by possibly other modes.

Assume the train is at Point 4 and it learns that the track 4-5 is under repair. Then the train agent can reason locally and find that it can follow the alternate route 4-7-8-6. The corresponding change in the local plan for the train agent is $\langle move_t(4, 7), move_t(7, 8), move_t(8, 6) \rangle$, which leads to the original local goal of

the person (to reach point 6). This modification in the local plan for the train agent does not affect the existing plan for car_1 at all.

Now consider the case when the train is at the station 4 and learns that the station at Point 6 is not available. In this case, since there are no train routes to point 6, train agent cannot repair its plan within its domain and hence escalates this to the LTA. The LTA finds a route 4-7-8-11-9-10 and decomposes the corresponding plan into the local plans $\langle(move_t(4,7),\ move_t(7,8)\rangle$ (train) and $\langle move_{car_2}(8,11), move_{car_2}(11,9), move_{car_2}(9,10)\rangle$(car) with the cross-agent dependency $move_t(7,8) \rightarrow move_{car_2}(8,11)$.

Further, consider the case when the Person is at Point 3 and the train is cancelled due to engine malfunction. The train agent cannot obviously find any alternate plan. When it escalates the issue to the LTA, the LTA is also unable to find any alternatives to reach Point 10 from point 3 using the train and car agents. Hence, it cascades the issue to the RPA, which synthesizes a new plan $\langle move_p(3,c), move_{car_3}(c,1), move_{car_3}(1,2), move_{car_3}(2,10)\rangle$. When decomposed to the operating agents, we get local plans $\langle move_p(3,c)\rangle$ (plane) and $\langle move_{car_3}(c,1),\ move_{car_3}(1,2), move_{car_3}(2,10)\rangle$(car) with cross-agent dependency $move_p(3,c) \rightarrow move_{car_3}(c,1)$.

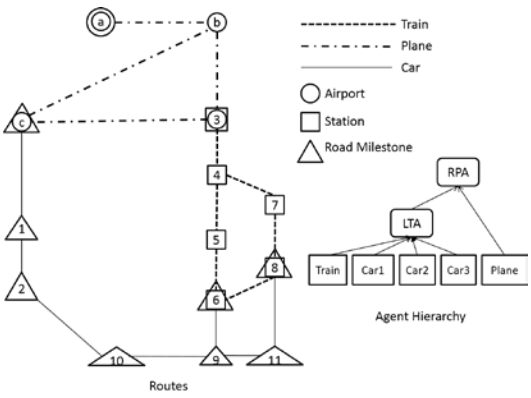


Fig. 1. Example of an HMAS and replanning.

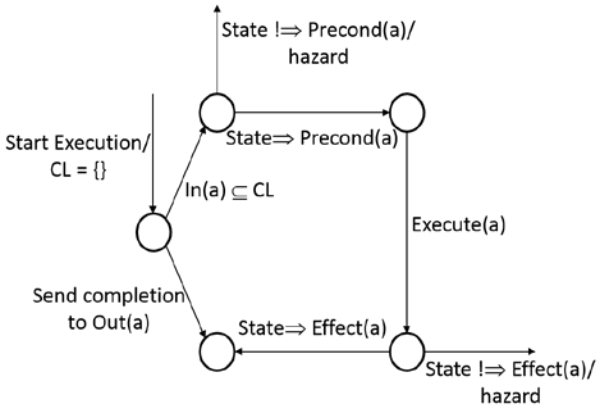


Fig. 2. Control for execution of a local plan.

4. Local Plans and Execution

Given a managing agent A, a plan P_A and an operating agent $i \in desc(A)$, the local plan of i in P_A is defined as

$P_A \uparrow i$, which is obtained by projecting P_A to Act_i and replacing each action a in the projection by $\langle In(a), precond(a), a, effect(a), Out(a)\rangle$. The execution platform for the operating agents has a signaling facility using which the agents inform the other agents about the completion of execution of local actions. Using this, each operating agent executes a local plan following the finite control state machine shown in Fig. 2. The explanation of the control is given below.

Definition [Local Execution]. The execution control is triggered through a signal *Start execution* after a new local plan has been updated (explained in Section 6). At the start, it initializes a list CL (Completion List) to $\emptyset$. Then, it waits for the enabling condition of the next action a in the plan. During the wait, it updates the CL at the receipt of completion notifications from other operating agents. When $In(a) \subseteq CL$, it implies that the dependencies for the action a have completed execution. Then, the control calls a monitor to check whether *precond (a)* holds in the current state of the world. If it holds, the action a is executed (e.g. through library routines or web services). At the completion of execution, completion notification is sent to agents whose actions are in $Out\ (a)$. The control calls the monitor to check *effect (a)* to ensure that the action completed successfully. In the cases where precondition or effect do not hold,

the monitor sends a hazard signal to the agent, which then proceeds to the replanning phase (again, explained in Section 6).

It is not difficult to see that the procedure above essentially enforces the topologically sorted order on the global plan. This ensures the correctness of local executions even though they are carried out in a distributed fashion.

5. Core Algorithm for Plan Repair

In this section, we describe the core algorithm for plan repair which depends upon (1) identifying a suitable fragment of the plan for replacement and (2) synthesizing a new plan to replace the old plan fragment.

5.1. Identification of A Suitable Plan Fragment

Given an agent A, the plans have a natural topological prefix order $\preceq$ ($\prec$ is the strict order) among them. Given a plan P_A, $a \preceq$-prefix of P_A is called a *plan fragment*. When the agent is in a configuration $\langle S_A, P_A \rangle$, and a hazard is detected, we need to identify a plan fragment that can be replaced without any conflict with the execution of other agents. Note that we have taken plan fragment as a prefix of the current plan for ease of presentation. The result is applicable for repairing future subplans as well if one can predict the hazards early.

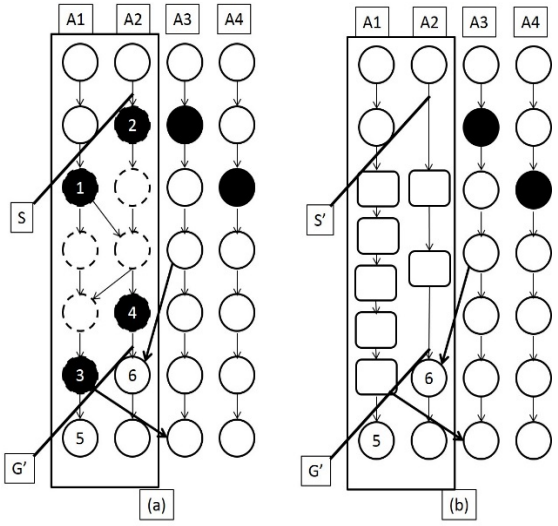


Fig. 3. Replanning surgery.

```
1:  procedure PLANREPAIR(AGENT A)
2:          Pf  = get maximal contingency-free plan fragment
3:          Sf  = get local destination state
4:          Problem =  (Sa,Sf)
5:          Domain = DA(predSet(Pf))
6:          Pr = call Planner(Domain, Problem)
7:          if Planner succeeds then
8:                  for agent a ∈ desc(A) do
9:                  Replace (Pf ↑ a) by (Pr ↑ a)
10:                 Return True
11:         Return False
```

Algorithm 1. Plan repair.

Definition [Contingency-Free Plan Fragment]. Given a global plan P, a plan fragment P' for an agent A is called contingency-free if no action in P' has any IN-dependency outside of $desc(A)$ (i.e. $In(a) \setminus desc(A) = \emptyset$) and the only actions with OUT-dependency, if any, are the maximal actions of P' (i.e. $Out(a) \setminus desc(A) = \emptyset$). In addition, if for all plan fragments $P'' \succ P'$, P'' is not contingency-free, then P' is called *maximal contingency-free*.

See Fig. 3(a). Assume that the agents *A1* and *A2* are the descendants of a managing agent *A*. Let the dark circles 1 and 2 mark the beginning of the current plan of *A*. The maximal contingency free plan fragment is upto the actions 3 and 4, since inclusion of action 6 violates the IN-dependency condition and inclusion of action 5 violates the OUT-dependency condition.

Intuitively, the actions in the contingency-free fragment are independent of the other current actions in the global plan and hence they can be replaced without affecting the existing coordinations (which are outside of the contingency-free plan fragment). By definition, for the Root agent, the entire global plan P_R is

contingency-free. Hence, at the Root agent, plan replacement essentially means synthesizing a fresh plan for the entire system.

5.2. Synthesis of a Replacement Plan

When we identify a contingency-free plan fragment (P_f) and can compute the final state (S_f) that the fragment results in, we can replace the fragment by another which also results in the same state S_f. We call this state the *local destination state*.

Definition [Local Destination State]. Given an agent A, the current expected state $S_c \subseteq Pred$, current plan P and a contingency-free plan fragment P_f, let $S = \delta(S_c, P_f)$. The local destination state S_f is the largest subset of S such that (*) each predicate in S_f is either in the goal state or used by an action in the rest of the plan ($P \setminus P_f$) in its precondition.

The condition (*) is because in the original execution, the actions in P_f might have introduced predicates that are not necessary to achieve the final goal. Hence, we retain only the essential subset in the local destination state.

Given the above, for an agent A at the point of hazard, the plan repair algorithm is designed as shown in Algorithm 1. Note the restriction of the domain (objects, predicates and actions) to only the predicates occurring in P_f, which ensures that no new dependencies are created when new plan fragments are introduced. Also observe that when A is an operating agent, the decomposition procedure is trivial. The correctness of the algorithm follows from the facts that (1) the final goal G depends only on the local destination state of agent A, (2) the replacement procedure ensures that the local destination state is achieved in spite of the hazard and (3) the local nature of replacement leaves the rest of the global plan unaffected. Fig. 3(b) illustrates the replacement of the maximal contingency-free plan fragment.

6. HIPR Architecture

In this section, we describe the complete HIPR architecture for HMAS based on the plan repair algorithm from the previous section.

The topology of the HMAS and interface signals are given in Fig. 4. The internal structure of each agent (both operating and managing) is shown in Fig. 5. Each agent has two components: Execution Master and Replanning Module. There is also a monitoring module whose service is available to all the agents.

Operating Agent: An operating agent receives a plan either from an ancestor, or from its own replanning module. It switches from the current plan to this new plan and starts executing the local plan (as per Section 4).

When the monitor detects a hazard, it stalls the execution module and triggers the replanning module. The replanning module invokes the plan repair algorithm. If there is a plan, then it is given to the execution module to replace the selected local plan fragment. This triggers the stalled execution module to start executing the new plan. If no feasible plan is found, then the replanning agent sends a signal to the parent and continues stalling.

Managing Agent: A managing agent has a (dummy) executing agent which actually does not execute any action, but maintains the partial order plans of all its descendants and all the completion notifications. When an action is completed, it updates the plans to keep track of the status at any time.

The replanning module is triggered by a replan signal from one of its children agents. Like the operating agents, it invokes the plan repair Algorithm. If a plan is found, it decomposes the plans into local plans and sends them to the descendant operating agents. If no feasible plan is found, it stalls and sends the replan signal to its parent. If the managing agent is the root agent, it sends the failure signal to the user and awaits further instructions.

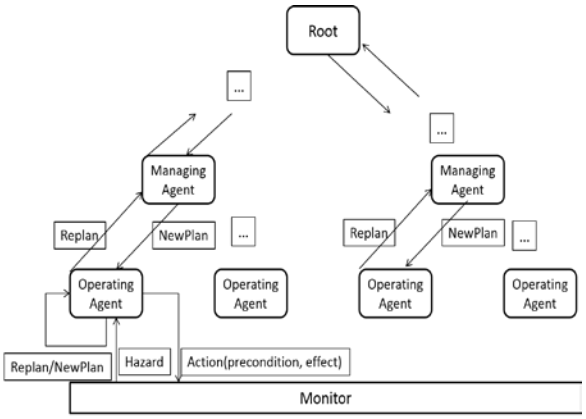


Fig. 4. HIPR architecture.

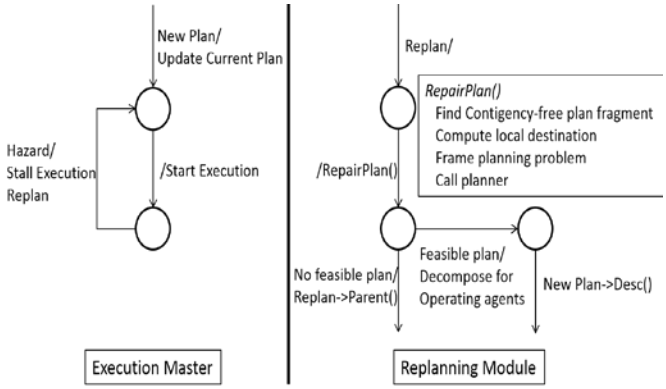


Fig. 5. Modules in agents of HMAS.

At a hazard, if there is an alternate global plan for the system, HIPR can also find it at the root agent in the worst case when the iteration reaches the root level of agent hierarchy (assuming, of course, that we use a planner which is complete). This ensures the correctness of HIPR. But, due to the bottom-up iteration, HIPR may discover alternate local plans at lower levels than the root, leading to better performance.

6.1. Revisiting the Example

In the route planning example (Fig. 1), when the segment 4-5 is not available, there is a hazard since the action $move_t(4,5)$ is not possible because its precondition $\{at_t(4), segment_t(4,5)\}$ is not satisfied. At this point, the maximal contingency-free plan fragment for the train agent is $\langle move_t(4,5), move_t(5,6)\rangle$. The local destination state S_f is $\{at_{person}(6)\}$, since the predicate $at_t(6)$ is not part of goal state and is also not essential in the rest of the plan. The actual current state S_a is $\{at_{person}(4), at_t(4)\}$ along with the predicates for all the segments except $segment_t(4,5)$. The replacement plan for the planning problem $\langle S_a, S_f\rangle$ can be synthesized using D_{train} alone, which is $\langle move_t(4,7), move_t(7,8), move_t(8,6)\rangle$.

When station 6 is not available, we model the situation as the state S_a with all segments except $segment_t(5,6)$ and $segment_t(8,6)$. One can see that the problem $\langle S_a, S_f\rangle$ cannot be solved using D_{train} alone. Now. considering the managing agent LTA at this hazard, the maximal contingency-free plan fragment is $\langle move_t(4,5), move_t(5,6), move_{car_1}(6,9), move_{car_1}(9,10)\rangle$. The local destination state S_f is $\{at_{person}(10)\}$, since the predicate $at_{car_1}(10)$ is not part of the final goal and no action is left after the contingency-free fragment. Now, the planning problem $\langle S_a, S_f\rangle$ can be solved in D_{LTA} which contains the moves of both train and car. The resulting plan is $\langle move_t(4,7), move_t(7,8), move_{car_2}(8,11), move_{car_2}(11,9), move_{car_2}(9,10)\rangle$ is decomposed for the train and car agents accordingly.

When the person is at point 3 and the train is not available, $\{at_t(3)\}$ is false. Hence the original plan faces a hazard since the action $\{move_t(3,4)\}$ cannot be carried out. It is easy to see that the train agent alone and even the LTA agent cannot find plans in the restricted domains. Hence, the root agent RPA considers the case. For RPA, the maximal contingency-free plan fragment P_f is $\langle move_t(3,4), move_t(4,5), move_t(5,6), move_{car_1}(6,9), move_{car_1}(9,10)\rangle$. The local destination state S_f is $\{at_{person}(10)\}$. Now, the planning problem $\langle S_a, S_f\rangle$ can be solved in D_{RPA} which contains the actions of plane and car agents. The resulting plan $\langle move_p(3,c), move_{car_3}(c,1), move_{car_3}(1,2), move_{car_3}(2,10)\rangle$ is decomposed for the plane and car agents.

7. Related Work

There is an extensive body of AI literature on replanning techniques. The reader can find a good survey of

the approaches in Talamadupula *et al.* [4] and Endo *et al.* [5]. The Partial Satisfaction Planning (PSP) technique described in [4] for plan repair in multi-agent systems derives completely new plans for each agent at the occurrence of a hazard taking into account the coordinations (called *commitments*) in the earlier plan. In our approach, only a small fragment of the existing plan for a set of agents is replaced by a new plan without ever touching the coordinations with the agents outside the set. So we do not have to code the commitments as new soft goals for the new problem as is done in [4]. Moreover, in our approach, any necessary change in coordination among a group of agents under a managing agent is implemented by global planning at the level of the managing agent.

In [6], the authors suggest re-coordinating among agents to repair an invalid local plan. This can lead to cascading the replanning activity to many agents and may need time to stabilize. In the worst case, this can lead to complete replanning for all the agents. In comparison, we take advantage of the locality and hierarchical agents so that in case of a hazard, instead of iteratively replanning agents for re-coordination, we replan for all the peer agents as a whole. This is conceptually simpler, though the larger scale may strain the planners. Therefore, characterization of domains for localized failures is critical for our approach.

The graph-based multi-agent replanning algorithm in [7] iteratively expands the zone around a hazard and uses distributed CSP (constraint satisfaction problem) techniques to find alternate plans. In a broad sense, the approach in Bonnet-Torres *et al.* [8] is similar to that in [7] though the methods to select the larger zone and find alternate plans are different. Our overall approach is similar to [7] and [8]. However, we have a 0-1 approach regarding coordination between agents. belonging to different groups. At any hazard, we try to repair the plan without disturbing the existing coordination among agents. If it is not possible, then we take advantage of the hierarchical structure to let a higher level agent replan for the entire group. The work on multi-agent path planning in [9] realizes efficient replanning through Q-learning techniques and key waypoints. We believe that our work is complementary and can increase the applicability of the work in [9].

The approach in [10] is similar in spirit to our work in that it selects new plans to minimize the recoordination requirements. However, it deals with plans derived from Hierarchical Task Networks (HTN). HIPR is based on action-based planning which has significant difference in the algorithms developed for plan repair. An important observation is that the complexity of replanning is dependent on the domain under analysis. This is noted in the work of [11] and should be examined in the HIPR framework.

8. Conclusion and Future Work

Conceptually, HIPR seems to offer efficient replanning in hierarchical multi-agent systems with localized failures, as exemplified through the use case. Large scale studies are in progress with use cases from IoT-based systems that are mostly hierarchical and exhibit localization of failure. In addition, we are investigating the following features to boost the performance of HIPR.

Since the efficiency of HIPR depends upon identifying the right level of the managing agent for the plan repair, it would be advantageous to converge to this level as soon as possible. One could use analytics on the history of hazards and plan repairs to estimate the probable levels.

In case of multiple hazards, replanning for one hazard done at a level may get overridden by replanning for another hazard. Hence, we need to find a suitable scheduling to minimize the wasteful plan repair steps.

In the classical plan synthesis problem, optimized plans are generated taking into account one or more metrics. But the local plan repair in HIPR can impact the metrics originally guaranteed by a global plan. Therefore, it would be desirable to control the plan repair algorithm so that the deviation is minimal.

References

[1] Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd ed.). Pearson Education.

[2] Benton, J., Coles, A. J., & Coles, A. (2012). Temporal planning with preferences and time-dependent continuous costs. *Proceedings of ICAPS: Vol. 77.*

[3] Hoffmann, J. FF Planner Tool. Retrieved from https://fai.cs.uni-saarland.de/hoffmann/metric-ff.html

[4] Talamadupula, K., Smith, D., Cushing, W., & Kambhampati, S. (2013). A theory of intra-agent replanning. *DTIC Document*.

[5] Shoma, E., Masataro, A., & Fukunaga, A. (2016). Evaluation of a simple, window-based, replanning approach to plan optimization. *Proceedings of the 8th Workshop on Heuristics and Search for Domain Independent Planning (HSDIP), ICAPS* (pp. 5-11).

[6] Van, R., Krogt, Der., & Weerdt, M. (2005). Plan repair as an extension of planning. *Proceedings of ICAPS: Vol. 5*. (pp.161–170).

[7] Zhang, J. F., Nguyen, X. T., & Kowalczyk, R. (2007). Graph-based multiagent replanning algorithm. *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*.

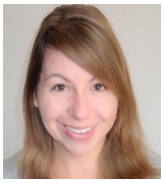
[8] Bonnet-Torrès, O., & Tessier, C. (2006). Cooperative team plan: Planning, execution and replanning. *AAAI Spring Symposium on Distributed Plan and Schedule Management,* 25-32.

[9] Su, X., Zhao, M., Zhao, L., Zhang, Y. (2016). A novel multi stage cooperative path re-planning method for multi UAV. *PRICAI,* 482-495.

[10] Bartold, T., & Durfee, E. (2003). Limiting disruption in multiagent replanning. *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*. *ACM*, 49–56.

[11] Guido, B., & Rossana, D. (2008). A replanning algorithm for decision theoretic hierarchical planning: Principles and empirical evaluation. *Applied Artificial Intelligence, 22(10),* 937-963.

Swarup Kumar Mohalik is a principal engineer in Ericsson Research, Bangalore. His expertise is in the theory of automata, logic and formal methods. He has been working in the areas of formal specification and verification of real-time embedded software, model based testing and AI planning techniques.



Aneta Vulgarakis Feljan is a senior researcher in the field of management and operation of complex systems at Ericsson Research. She was a scientist in software architecture and usability at ABB Corporate Research. Her research interests include model-based development, knowledge engineering, formal analysis and AI planning.

Mahesh Babu Jayaraman is a senior researcher in Ericsson Research in the field of management and operation of complex systems applied to intelligent management of smart cities, intelligent transport systems and intelligent automation in Telco and cloud space. His research interests are AI planning, knowledge management and analytics. His specialization areas are network and element management systems, operations support solutions(OSS), network planning and policy management.

Ramamurthy Badrinath is working as a principal engineer in Ericsson Research where he is a team member who deals with the management and operations of cyber-physical systems. He worked with HP in various capacities as an architect in the earlier time. He served as a faculty at IIT Kharagpur for about nine years. His areas of interest include cloud systems, high performance computing, semantic web and infrastructure management.