Multi-agent Plan Repairing

Antonín Komenda and Peter Novák
Agent Technology Center
Dept. of Cybernetics, Faculty of Electrical Engineering
Czech Technical University in Prague
Czech Republic

Abstract

Coordinated multi-agent planning and acting in dynamic and uncertain environments poses a number of challenges. We present a conceptual framework formalising the problem of multi-agent planning and subsequent plan repair. As a first step towards tackling the problem of multi-agent plan repair, we introduce a sequel of three algorithms. As a preliminary experiment, we evaluate and compare one of them with re-planning from scratch in a synthetic domain of multi-robot cranes and show computational and communication gains of the plan repairing technique.

1 Introduction

Dealing with dynamic and uncertain environments in which unexpected events may occur or actions of agents may fail is one one of the core research topics in Artificial Intelligence. This problem is a particularly pressing issue in the domain of embodied robotics, where the agents interact with the real world in its unconstrained complexity. Recognising the difficulties of classical planning in such domains led to emergence of several approaches tackling this problem. In the field of automated planning, among the most important ones belong *contingency planning*, *conformant planning*, or *distributed continual planning* [2].

One of the main problems in planning for a single agent situated in a dynamic environment is the efficiency of an agent's planning versus the rate of plan execution failure. In cases when the agent's planning algorithms are significantly faster than the dynamics of the environment, the pragmatic approach to simply re-plan from scratch is currently the prevalent solution to the problem. By performing a formal time complexity analysis of the problem Nebel and Koehler in [3] show that this approach is in fact a very reasonable solution. In particular, as a result of their work they show that when repeated (re-)planning is needed, it is not possible to achieve a provable efficiency gain of plan reuse over plan generation. Informally, there is not much to be gained by naive attempts to repair failed plans in order for the autonomous agent to continue its activities.

With maturing of the field of multi-agent systems and transfer of the relevant results to multi-robotic application domains, the problem of *plan repair vis-à-vis* dynamic and uncertain environments gains, however, a new importance. Consider application domains, such as e.g., underwater coordinated operations by teams of autonomous underwater vehicles (AUVs). While the state of the art technology allows to employ relatively powerful computers on board of such robots, the communication links are extremely constrained and expensive; wireless networks cannot be deployed and communication is performed mostly using acoustic signalling. In such applications, it is the communication complexity of the distributed multi-agent planning algorithms which matters more than time, or space complexity. Consequently, employment of multi-agent plain repair techniques can provide a tangible benefit over re-planning from scratch for the team of robots whose multi-agent plan fails.

In this paper we present first steps towards a formal approach tackling the problem of multi-agent plan repair. The contribution of this work is twofold. Firstly, in Section 2, we introduce a formal conceptual framework articulating the problem of multi-agent planning and the corresponding problem of multi-agent plan repair. Secondly, in Section 3, we present a sequel of three multi-agent plan repair algorithms and discuss their respective properties. We conclude the paper by presenting an experimental evaluation of one of the algorithms in Section 4 and discussion of the related work in Section 5.

2 Preliminaries

Even though various aspects of multi-agent planning were studied in the course of the last decade (cf. e.g., [6]), surprisingly a formal treatment and a complexity analysis of the general problem of multi-agent planning appeared only recently in the work of Brafman and Domshlak [1]. In the following we recapitulate and extend their formalism of multi-agent planning (Subsection 2.1), introduce an abstract plan-execute-monitor architecture capable to detect plan failures (Subsection 2.2) and finally introduce the multi-agent plan repair problem (Subsection 2.3).

2.1 Multi-agent planning

Let from now on $\mathcal{L}$ be a propositional language. The language $\mathcal{L}$ comes with the entailment relation $\models\colon 2^{\mathcal{L}} \times \mathcal{L} \rightarrow \{\top, \bot\}$ mapping a set of propositions, a theory in $\mathcal{L}$, and a formula to Boolean truth values according to the set inclusion relation.

$\mathcal{P}$=(

A_1:	P_1=	a_1^1	a_2^1	a_3^1 ⋯	a_m^1
A_2:	P_2=	a_1^2	a_2^2	a_3^2 ⋯	a_m^2
⋮					
A_n:	P_n=	a_1^n	a_2^n	a_3^n ⋯	a_m^n)

Figure 1: Depiction of a multi-agent plan structure.

I.e., having a theory $S \in 2^{\mathcal{L}}$ and a proposition $\varphi \in \mathcal{L}$, $S \models \varphi$ if and only if $\varphi \in S$. W.l.o.g., we assume $\top, \bot \in \mathcal{L}$.

A *multi-agent planning problem* is defined as a tuple $\Pi = \langle \varphi, S, s_{\text{init}}, S_{\text{goal}} \rangle$ where φ is a set of *agents* $A_1, \ldots, A_n$, S denotes the set of all possible states and $s_{\text{init}} \in S$ and $S_{\text{goal}} \subseteq S$ respectively denote the initial state of the multi-agent system and the set of desirable goal states. Each agent is characterised by sets of their respective individual capabilities. The capabilities are described using STRIPS as a set of quadruples $\phi_{pre} a \phi^+_{post} \phi^-_{post}$. $\phi_{pre} \in 2^{\mathcal{L}}$ denotes the set of preconditions of the action a, where a is the action's label. $\phi^+_{post}, \phi^-_{post} \in 2^{\mathcal{L}}$ respectively denote the sets of add and delete effects of execution of the action a. To say that an agent A is capable to execute the action a, from now on we simply write $a \in A$. Furthermore, we assume that each agent is capable to execute the empty action $\{\top\}\epsilon\{\top\}\{\top\}$, i.e., $\epsilon \in A_i$ for every $0 \leq i \leq n$.

Note that our version of STRIPS action specification language does not involve variables. Assuming only finite domains of variables, w.l.o.g., we assume already grounded action specifications, i.e., such where a first-order action specification with variables is translated into a set of primitive ground actions without variables by considering all instantiations of the action over the domains of the involved variables.

Having a multi-agent planning problem (MA-plan) $\Pi = \langle \varphi, S, s_{\text{init}}, S_{\text{goal}} \rangle$, we are looking for a set of individual plans $P_1, \ldots, P_n$ for the agents $A_1, \ldots, A_n$ in the form of sequences of actions, such that (i) each agent is capable to carry out its individual plan on its own, and (ii) after the synchronised execution of the individual plans, the system is in one of the goal states S_{goal}. Formally, considering agents $A_1, \ldots, A_n$, we are seeking a n-tuple $\mathcal{P} = (P_1, \ldots, P_n)$, such that each $P_i = a_1^i, \ldots, a_m^i$, where $a_j^i \in A_i$ and $|P_i| = |P_j| = m$, for every $1 \leq i, j \leq n$. We also denote $|\mathcal{P}| = m$. In the following we write $P[k]$ to select a_k, the k-th action of the individual plan $P = a_1, \ldots, a_k, \ldots, a_m$. Consequently, we write $\phi[k]_{pre}$, $\phi[k]^+_{post}$ and $\phi[k]^-_{post}$ do denote the pre- and post-conditions of the action $P[k]$. The structure of a MA-plan is depicted in Figure 1.

Since every agent is capable to perform the empty action ϵ, the constraint for the same length of plans can be trivially fulfilled for any multi-agent plan by adding ϵ-padding to the end of shorter individual plans.

Synchronised execution of a multi-agent plan $(P_1, \ldots, P_n)$ in a state s_0 is characterised by a sequel of states $s_0, s_1, \ldots, s_m$ where each state s_j is obtained from s_{j-1} by executing the joint action of the agents $\mathcal{A}[j] = \langle P_1[j], \ldots, P_n[j] \rangle$ for all $0 < j \leq m = |P_1|$. We write

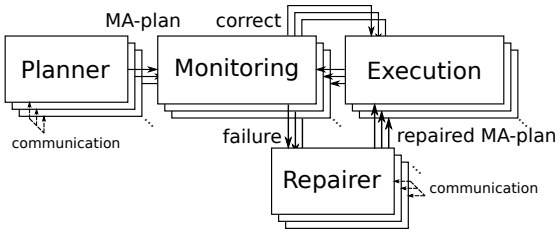


Figure 2: The plan execution and monitoring architecture – each layer for each agent.

Algorithm 1 Plan execution and monitoring algorithm.

```
1:  P = plan(Π)
2:  k := 0
3:  while k ≠ |P| do
4:      s_curr := exec(s_curr, ⋃_{i=1}^n φ_i[k]^-_post, ⋃_{i=1}^n φ_i[k]^+_post)
5:      if cannot_proceed(P, s_curr, k) then
6:          P = repair(Π, P, s_curr, k)
7:          if P = Fail then
8:              return Fail
9:          end if
10:     end if
11:     k := k + 1
12: end while
13: return True
```

$s_j = s_{j-1} \oplus \mathcal{A}[j]$, formally

$$s_j = s_{j-1} \setminus \{\bigcup_{i=1}^{n} \phi_i[j]^-_{post}\} \cup \{\bigcup_{i=1}^{n} \phi_i[j]^+_{post}\}$$

The joint action $\mathcal{A}[j]$ is *executable* in a state $s_{j-1} \in S$ if and only if preconditions of each individual action are satisfied in s_{j-1}, i.e., $s_{j-1} \models \phi_{pre}$ for every $\phi_{pre} \in \bigcup_{i=1}^{n} \phi[j]_{pre}$.

We say that the multi-agent plan $(P_1, \ldots, P_n)$ is *sound* w.r.t the MA-plan problem $\Pi = \langle \varphi, S, s_{\text{init}}, S_{\text{goal}} \rangle$ iff the synchronised execution of $(P_1, \ldots, P_n)$ is characterised by the sequel of states $s_0, s_1, \ldots, s_m$, such that $s_0 = s_{\text{init}}$, $s_m \in S_{\text{goal}}$ and each joint action $\mathcal{A}[j]$ is executable in the state s_{j-1}.

2.2 Plan execution and monitoring

In dynamic and uncertain environments, an agent's actions and plans may not always lead to the desired consequences, or can turn out to be not executable. To account for such cases, a cautious agent must be able not only to execute its actions, but also monitor its own progress and detect failures. Generally speaking, besides a planning component, implementation of an agent should also include monitoring and a plan repairing component. Figure 2 depicts a generic multi-agent plan-execute-monitor architecture. More concretely, considering a multi-agent plan produced by a suitable MA-plan planner, the abstract execution-monitoring algorithm checks in every state the soundness of the next step before advancing. If necessary, it invokes a plan repairing procedure. The Algorithm 1 lists a sketch of such an algorithm.

The multi-agent planning function $plan(...)$ takes an instance of the MA-plan problem as an input and returns

a multi-agent plan $\mathcal{P}$. Function $exec(\ldots)$ executes a joint action defined by sets ϕ^{+}_{post} and ϕ^{-}_{post} and returns changed state. Plan failure is detected by the condition $cannot\_proceed(\mathcal{P}, s, k)$ which is true iff there exists an individual plan P_i of the agent A_i, such that the precondition $\phi[k+1]_{pre}$ of the action $P_i[k]$ is not satisfied in s, i.e., $s \not\models \phi[k+1]_{pre}$. Hence, the next joint action of the plan $\mathcal{P}$ cannot be executed as planned, because one of the preconditions necessary for the step is not satisfied. We consider full information, i.e., all agents can observe failures of other agents. Upon plan execution failure, the plan repairing function $repair(\Pi, \mathcal{P}, s, k)$ takes as an input the considered instance of the multi-agent plan problem, the executed multi-agent plan $\mathcal{P}$, a state s in which the execution of $\mathcal{P}$ failed (could not proceed any more), and k the step in which the execution of the plan $\mathcal{P}$ failed in.

2.3 Multi-agent plan repair

As already sketched in the previous subsection, upon detection of plan execution failure, a plan repair should be attempted. In general, a plan repair problem $\mathcal{R}$ can be defined as a tuple $\mathcal{R} = (\Pi, \mathcal{P}, s_{\mathrm{F}}, k)$ comprising the original MA-plan problem Π, the currently executed multi-agent plan $\mathcal{P}$, the state s_{F} in which the execution of $\mathcal{P}$ was interrupted (couldn't proceed anymore) and $0 < k < |\mathcal{P}|$ the number of the step in which the execution of $\mathcal{P}$ was interrupted. Furthermore, we assume that there was a reason for the joint plan so that it couldn't proceed further than state s_{F}. Provided the execution of $\mathcal{P}$ is characterised by the sequence of states $s_0, \ldots, s_{|\mathcal{P}|}$, there must be an individual plan P_i in $\mathcal{P}$, such that the precondition $\phi[k+1]_{pre}$ of the next action to be executed $P_i[k+1]$ is not satisfied in s_F, i.e., $s_F \not\models \phi[k+1]_{pre}$. A solution of the plan repair problem (MA-repair) is of the same type as that of the plain multi-agent planning problem Π, i.e, a joint plan $\mathcal{P}'$, which is sound w.r.t. the MA-plan problem $\Pi = \langle \varphi, S, s_{\mathrm{F}}, S_{\mathrm{goal}} \rangle$. Optionally, we can require the resulting multi-agent plan $\mathcal{P}'$ to be constructed from $\mathcal{P}$ with use of minimal number of changes. Since the optimal plan repair problem is not the focus of this paper, we leave out its precise formal definition.

In the above paragraph, we focus only on a single type of plan failure, the critical one, i.e., when the cooperative plan simply cannot proceed any more. The reasons for such a situation might be twofold. Either (i) some previous individual action of the multi-agent plan failed, i.e., some post-condition of the action did not become true after the action's execution, or (ii) the environment interfered and invalidated a previously established condition so that the forthcoming action couldn't be safely executed. In both cases, the failure is detected at the latest possible point, i.e., when the condition becomes vital for the subsequent step in the joint plan, that is its precondition.

In general, we can identify several types of plan failures. Alternatively, post-conditions of actions could be checked after their execution in order to detect action failures. This approach is however vulnerable to checking irrelevant conditions, i.e., such which are never used as a precondition of some future action in $\mathcal{P}$.

Algorithm 2 Naive plan repair algorithm

1: $s_{\mathrm{curr}} := simulate(\mathcal{P}, s_{\mathrm{init}}, k)$
2: **while** $S_{\mathrm{goal}} \not\subseteq s_{\mathrm{curr}}$ **do**
3: $\quad \mathcal{P}' := plan(\varphi, S, s_{\mathrm{F}}, s_{\mathrm{curr}})$
4: $\quad$ **if** $\mathcal{P}' \neq \emptyset$ **then**
5: $\quad\quad$ **return** $\mathcal{P}' \cdot tailplan(\mathcal{P}, k)$
6: $\quad$ **else**
7: $\quad\quad k := k + 1$
8: $\quad\quad s_{\mathrm{curr}} := simulate(\mathcal{P}, s_{\mathrm{init}}, k)$
9: $\quad$ **end if**
10: **end while**
11: **return** $Fail$

Algorithm 3 Blind Repairing Algorithm

1: $\mathcal{P}' := (P'_1 = \epsilon, ..., P'_n = \epsilon)$
2: $s_{\mathrm{curr}} := simulate(\mathcal{P}, s_{\mathrm{init}}, k)$
3: $\varphi^* := (A | A \in \varphi \wedge \exists a \in A : \phi^{a+}_{post} \subseteq s_{\mathrm{curr}} \setminus s_{\mathrm{F}} \vee \phi^{a-}_{post} \subseteq s_{\mathrm{F}} \setminus s_{\mathrm{curr}})$
4: **for all** $A_i \in \varphi^*$ **do**
5: $\quad$ **if** $\exists a_r : a_r \in A_i, \phi^{a_r}_{pre} \subseteq s_{\mathrm{F}} \wedge \phi^{a_r+}_{post} \subseteq s_{\mathrm{curr}} \wedge s_{\mathrm{curr}} \not\subseteq \phi^{a_r-}_{post}$ **then**
6: $\quad\quad P'_i[1] = a_r$
7: $\quad$ **else**
8: $\quad\quad$ **if** $\exists A_d \exists a_d : A_d \in \varphi, a_d \in A_d : \phi^{a_d}_{pre} \subseteq s_{\mathrm{F}} \wedge \phi^{a_d+}_{post} \subseteq s_{\mathrm{curr}} \wedge s_{\mathrm{curr}} \not\subseteq \phi^{a_d-}_{post}$ **then**
9: $\quad\quad\quad P'_d[1] = a_d$
10: $\quad\quad$ **end if**
11: $\quad$ **end if**
12: **end for**
13: **if** $\exists i : P'_i \neq \epsilon$ **then**
14: $\quad$ **return** $\mathcal{P}' \cdot tailplan(\mathcal{P}, k)$
15: **else**
16: $\quad$ **return** $tailplan(\mathcal{P}, k)$
17: **end if**

3 Plan repairing algorithms

3.1 Naive Repairing Algorithm

Algorithm 2 lists the simplest algorithm designed. It uses iteratively prolonged repairing plan from the point of the failure to a particular point in the old plan (the k-th step). The function $simulate(\ldots)$ denotes the ideal plan execution function up to the k-th step if everything works out right according to the joint action specification (w.r.t. the effects of the involved actions). Formally,

$$simulate(\mathcal{P}, s_j, k) = ((s_j \oplus \mathcal{A}[j+1]) \oplus \mathcal{A}[j+2]) \oplus \cdots \oplus \mathcal{A}[k]$$

simulates iteratively the ideal execution of the joint actions $\mathcal{A}[j+1], ..., \mathcal{A}[k]$ on the state s_j. The function $tailplan(\mathcal{P}, k)$ cuts off beginning of the MA-plan $\mathcal{P}$ up to the k-th action and returns the rest.

The complexity of the Naive Repairing Algorithm (NRA) respects the inner planner complexity. The worst case complexity is *PSPACE-complete* (since a general planning fits the *PSPACE-complete* complexity class).

Algorithm 4 Iterative Blind Repair Algorithm

$k^{\text{orig}} := k$
$s_{\text{F}}^{\text{orig}} := s_{\text{F}}$
$\mathcal{P}' := \mathcal{P}$
while $S_{\text{goal}} \not\subseteq s_{\text{curr}}$ **do**
  $\mathcal{P}' := blindRepair(\Pi, \mathcal{P}', s_{\text{F}}, k)$
  if $\forall i \in (0, ..., m) : \phi_i'[k]^+_{post} \subseteq s_{\text{F}} \wedge s_{\text{F}} \not\subseteq \phi_i'[k]^-_{post}$ **then**
    return $\mathcal{P}'$
  else
    $k := k + 1$
    $s_{\text{F}} := simulate(\mathcal{P}', s_{\text{init}}, k)$
  end if
end while
return $naiveRepair(\Pi, \mathcal{P}, s_{\text{F}}^{\text{orig}}, k^{\text{orig}})$

3.2 Blind Repairing Algorithm

The Blind Repairing Algorithm (BRA) tries to solve a failure by adopting an alternative action(s) solving the failed effects. Firstly it finds agent(s) which caused the problem (their effects do not meets the simulated state). After the agents are found, for each agent an alternative action is tried to be found. If the personal alternative cannot be found a wide alternative is tried (another agent can adopt an action which solves the failure). Finally, even if the wide alternative cannot be found, the problem is ignored to be solved in the next steps prospectively (the agens which cannot proceed are executing ϵ). If a failure is ignored the repairing algorithm is called again in the next step by the main monitor-execution loop, however the context is already changed thus a new solution can be found. Algorithm 3 lists the pseudo-code of the blind plan repairing algorithm.

The algorithm is not sound in the pure form as it can miss a solution of the repairing problem. Its modification towards the soundness can be managed simply by adding a fail-safe call of NRA if the algorithm iteratively fails at the goal state. The iterative form of the algorithm is listed in Algorithm 4.

The non-iterative form has linear complexity (iteration over the number of the agents). The iterative extension has minimal quadratic complexity (number of the agents times the length of the plan). Maximal complexity reflects the complexity of the NRA as it can be used as a fail-safe process.

3.3 Locality Repairing Algorithm

The last repairing algorithm (see Algorithm 5) is based on an assumption, the communication is not costless, i.e. we count amount of information needed to be transmitted for each repairing of an action (for the Naive Repairing Algorithm, it is the complete global state plus the final state to one repairer agent, and then the result back to the rest of the agents $c = 2 * 2 * n$, n is number of the agents, i.e. number of the state to transfer and c is communication volume for each (re-)planning process; for the Blind Repairing Algorithm the mean value is $c = n/2$, i.e. a median of the probability of number of agents involved in reparation of one action).

Before presenting the final plan repair algorithm, the locality repairing, let us introduce the notion of a *causality graph*

Algorithm 5 Locality Repairing Algorithm

$\overline{v} := (1, 1, 0)$
loop
  $\varphi^* := combinations(\varphi, A, \lceil v_1 \rceil)$
  $\pi = \{\rho | \forall A \in \varphi^* : \rho = A : alterns(\lceil v_2 \rceil, \lceil v_3 \rceil)\}$
  $\mathcal{P} = compatible(\pi)$
  if $\mathcal{P} \neq \emptyset$ **then**
    return $\mathcal{P}$
  end if
  $\overline{v} := \overline{v} + \overline{V}$
end loop

exploited by the algorithm.

Each MA-plan $\mathcal{P}$ can be represented as a graph G, where vertices V represent the actions and edges E represent causal links among the actions:

$$\begin{aligned} G &= (V, E) \\ V &= \{v_i | \forall i : a_i \in \mathcal{P}\} \\ E &= \{e_{ij} | e_{ij} = (v_i, v_j) \textbf{ iff } \phi^{a_i}_{post} \cap \phi^{a_j}_{pre} \neq \emptyset\}. \end{aligned}$$

Using a causality graph G, a causality relation $\leftarrow$ can be defined as $a_i \leftarrow a_j$ **iff** $e_{ij} \in E$. A transitive closure of $\leftarrow$ (oriented path in the causal graph) will be denoted as $\overset{*}{\leftarrow}$.

The proposed plan repairing algorithm is based on the planning technique presented in [4]. From the communication perspective, it tries to solve the problem as locally as possible, iteratively including more and more agents which tries to solve the problem also locally. Therefore, the agents communicate as few as possible information (failed actions). Algorithm 5 lists the pseudo-code of the Locality Repair Algorithm (LRA).

The vector $\overline{v}$ represents three counters: v_1 number of agents involved in the repairing process, v_2 represents length of the repairing plans, and v_3 represents number of actions removed from the old plan. The counters are increased by a constant vector $\overline{V}$ if a valid plan is not found. Various parametrisation of the algorithm can be done using the $\overline{V}$ vector (e.g., $\overline{V} = (0.1, 1, 0)$ represents increase of the number of the involved agents for each 10 actions of the repairing plan length).

The $combination(\varphi, A, n')$ function generates a set of possible combinations of n' agents prioritising combinations with the agent A. The function $A : alterns(q, r)$ generates alternative repairing plans from the perspective of the agent A of length q with r removed actions from the current plan. The process of action removing follows a path in the causality graph G, formally the process successively removes actions on path $a_k \overset{*}{\leftarrow} a_{k+r}$. The generated plan alternatives have to satisfy the initial conditions defined by the state s_{F} and the final conditions defined by state s_{k+r}. The function $compatible(\pi)$ returns a plan based on the alternatives in the set π, where only mutually compatible alternatives are preserved. Two alternatives are compatible if delete effects of one do not preclude usage of the other. The first compatible alternative tuple found is used.

Repairing Dimensions The Locality Repairing Algorithm is based on the principle of the Blind Repairing Algorithm. The main difference is that the algorithm is sound inherently in the pure form.

The main problem of the BRA is that the repairs can be done only by one changed action for an agent A in the step k. Additionally, the changes cannot be backtracked as the process works only in the successive manner. On one hand, this fact simplifies the process and implies the process has only linear computational complexity. On the other hand, the process can miss quite simple solutions using only two successive actions provided by the failure causing agent. The Locality Repairing Algorithm takes into the account all these possibilities and thus searches through the complete space of possible repairs.

More precisely, there are three dimensions in the space of the possible repairs. The first dimension v_1 is common with the BRA and it describes which agents undertake the repairing actions. The second dimension v_2 describes how many and which actions one agent uses for its local repair. The last one v_3 describes how deep the repair goes through the current plan. The last dimension is similar to the successive direction of the NRA. The differences are (i) the repairing process can backtrack and (ii) it primarily follows the causality graph in an opposite direction against the causality links.

In the exhaustive case, all the combinations of the future actions have to be treated as possibilities for repairing. The only aspect decreasing the full range of $2^{|A|\lceil v_2 \rceil}$ combinations for one agent A is that only compact successive sequences of the actions based on the causal links need to be considered. The number of the combinations thus only $(\lceil v_2 \rceil + 1)^{|A|}$.

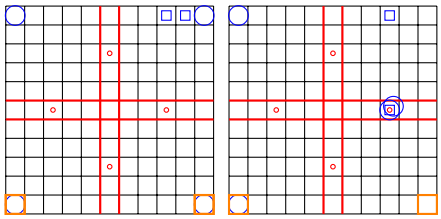
4 Experimental evaluation

For an evaluation and a preliminary comparison, we have designed and implemented a synthetic experimental environment, a multi-agent planner based on [4], and the Naive Repairing Algorithm (see Section 3.1). The Naive Repairing Algorithm is compared with re-planning from scratch.

4.1 Simulated Environment

As an environment we use a restricted instance of the Crane-Robot domain problem which we call Box-Movers. In Box-Movers, there are k mover agents (representing a robot and crane as one entity), which can each move in a $w \times h$ grid and b distinguishable boxes each of which has one target position. The boxes cannot be laid upon each other. Each grid point can be accessed by only one mover with the exception of s points, which can be accessed by two neighbouring agents (hand-over points). Each mover can carry only one box at a time.

The initial state defines points where the movers start and where the boxes lie. The movers have three possible actions (i) move, (ii) load, and (iii) unload. A mover can *move* in its accessible grid by one cell horizontally, vertically, and diagonally (including the hand-over points) whether or not it is carrying a box. A mover can *load* a box, if it is at the same point as the box. A mover can *unload* a box if it is carrying it, and there is not another box in the unload position.



(a) (b)

Figure 3: The example environment, where blue circles represent movers and blue squares represent the boxes. The orange cells are the respective target points and the red points are the hand-off points. The red lines represent areas reachable by the respective agents. (a) The initial configuration for 4 movers and 2 boxes. (b) Box1 (10, 0) is being handed from Mover3 (top-right) to Mover4 (bottom-right).

The model of the failures is based on a probability P with an uniform distribution. The probability describes if a *move* action carrying a box drops the box. The detection of the failure is reliable which means the failure is always detected. $P = 1$ represents a failure of each executed move action (this case is not included in the evaluation as the plan would have infinite length). If $P = 0$, the execution is deterministic and without failures.

A solution of the problem is a plan of actions for each mover which relocates the boxes from their initial points to their target points.

An example instance of the domain is depicted in Figure 3, where $k = 4, w = 6, h = 6, b = 2$, and $s = 4$.

4.2 Experimentation Methodology

The experiments are comparing the Naive Repairing Algorithm and re-planning from scratch for varying probability P. There are three comparison metrics used: (i) computational complexity, (ii) communication complexity, and (iii) length of the resulting plans.

The computational complexity measure is based on a number of action sequences generated by the planning, re-planning, and plan repairing processes. All such sequences have to be checked for soundness and mutual compatibility, which makes them the most complex part of the algorithm.

The communication complexity counts a number of messages, which are necessary to coordinate the planning processes. The messages contain only comparable sizes of data (the largest structure transferred by one message is a single-agent action sequence). More complex data packages are split into more messages.

The length of the final plan includes also all repairing plans.

4.3 Experiment Results and Discussion

The results are depicted in three figures according to the mentioned metrics. As we can see in Figure 4, the complexity of the plan repairing algorithm (NRA) is considerably lower (12%) than the complexity of the re-planning. The plan repairing technique uses the planner during the execution only for short and simple planning problems. On the other hand,

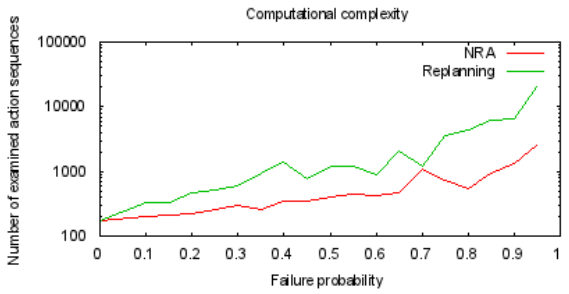


Figure 4: Comparison of NRA and re-planning from scratch – computational complexity metrics.

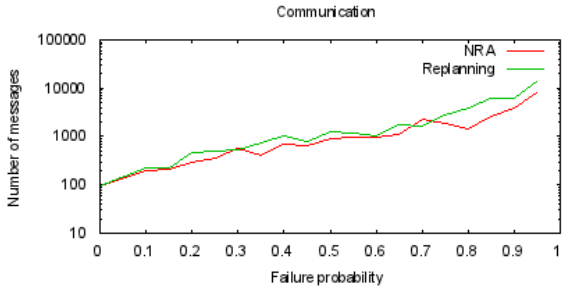


Figure 5: Comparison of NRA and re-planning from scratch – communication complexity metrics.

the re-planning approach in the earlier phases of the execution has to plan complete plans towards the goal state.

The communication complexity (see Figure 5) of the plan repairing algorithm is also lower than the re-planning from scratch (57%). As the planning problems creating repairing plans are simpler than the complete re-planning problems, the planning process need less messages to find the compatible personal action sequences of the agents. The more advanced plan repairing techniques as Locality Repairing Algorithm are theorised to even more decrease the number of the messages required for the repairing process according to the v_2 dimension (other agents) of the vector $\overline{V}$.

The plan lengths, depicted in Figure 6, shows the increase of the plan length for the plan repairing technique (59%). It is caused by repeated extension of the initial plan by the repairing plans. Study and reduction of this effect can be one of the possible future works.

5 Discussion and conclusions

In the presented work, we are making first steps towards a thorough investigation of the multi-agent plan repair problem, where we specifically focus on study of communication complexity of the re-planning and plan repair process. As already discussed in Section 1, plan repair in general is not a deeply studied topic. One of the main reasons is that it can be shown that plan repair, in its most general form in single-agent context, is not more efficient than planning from scratch, i.e., re-planning (cf. [3]). In multi-agent context, where communication among agents can be severely constrained, however,

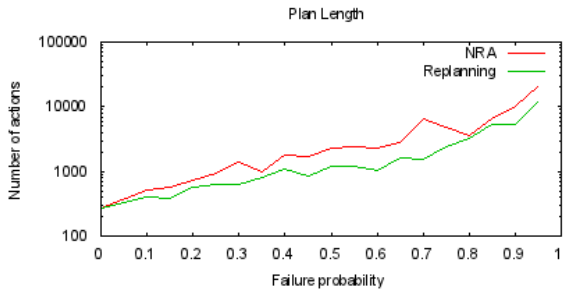


Figure 6: Comparison of NRA and re-planning from scratch – plan length metrics.

distributed repair algorithms aiming at fixing the failed plan in a local context only concerning a relevant subset of agents of the team can be a more viable option than straightforward re-planning.

Recently, in a sequel of works culminating in the dissertation [5], van der Krogt and de Weerdt study plain repair and its uses for multi-agent planning [6]. Unlike our case, they study plan repair in single-agent context. When transposed to multi-agent planning domain, they consider the problem of multi-agent planning for self-interested agents which are capable to independently achieve their respective goals. Our primary motivation, however, is to study multi-agent planning in teams of cooperative agents, which are forced to work together in order achieve their individual goals and none of the team members is able to bring about the goal on its own (cf. Subsection 4.1).

References

[1] Ronen I. Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric A. Hansen, editors, *ICAPS*, pages 28–35. AAAI, 2008.

[2] Edmund H Durfee, Charles L Ortiz, and Michael J Wolverton. A Survey of Research in Distributed, Continual Planning. *AI Magazine*, 20(4):13–22, 1999.

[3] Jana Koehler Nebel, B. Plan reuse versus plan generation: a theoretical and empirical analysis. *Artificial Intelligence*, 76(1-2):427–454, July 1995.

[4] Raz Nissim, Ronen I. Brafman, and Carmel Domshlak. A general, fully distributed multi-agent planning algorithm. In Wiebe van der Hoek, Gal A. Kaminka, Yves Lespérance, Michael Luck, and Sandip Sen, editors, *AAMAS*, pages 1323–1330. IFAAMAS, 2010.

[5] Roman van der Krogt. *Plan repair in Single-Agent and Multi-Agent Systems*. PhD thesis, Delft University of Technology, Delft, The Netherlands, 2005.

[6] Roman van der Krogt and Mathijs de Weerdt. Self-interested planning agents using plan repair. In *Proceedings of the ICAPS 2005 Workshop on Multiagent Planning and Scheduling*, pages 36–44, 2005.