Decentralized Multi-agent Plan Repair in Dynamic Environments*

Antonín Komenda, Peter Novák, Michal Pěchouček

Agent Technology Center
Dept. of Computer Science and Engineering, FEE
Czech Technical University in Prague
Czech Republic

Abstract

Achieving joint objectives by teams of cooperative planning agents requires significant coordination and communication efforts. For a single-agent system facing a plan failure in a dynamic environment, arguably, attempts to repair the failed plan in general do not straightforwardly bring any benefit in terms of time complexity. However, in multi-agent settings the communication complexity might be of a much higher importance, possibly a high communication overhead might be even prohibitive in certain domains. We hypothesize that in decentralized systems, where coordination is enforced to achieve joint objectives, *attempts to repair failed multi-agent plans should lead to lower communication overhead than replanning from scratch.*

The contribution of the presented paper is threefold. Firstly, we formally introduce the multi-agent plan repair problem and formally present the core hypothesis underlying our work. Secondly, we propose three algorithms for multi-agent plan repair reducing the problem to specialized instances of the multi-agent planning problem. Finally, we present results of experimental validation confirming the core hypothesis of the paper.

1   Motivation

Classical planning and multi-agent planning based on classical planning are approaches to constructing autonomous agents and teams of agents, which attempt to achieve their objectives in an environment. The result of the planning process is traditionally a plan, a sequence of actions the agent should perform in order to achieve a given goal. When the agent is situated in a dynamic environment,

---

occurrence of various unexpected events the environment generates might lead to the plan invalidation, a failure. A straightforward solution to this problem is to invoke a planning algorithm and compute a new plan from the state the agent found itself in after the failure to a state conforming with its original objective.

Planning, as well as replanning, in the case of a failure occurrence, is a costly procedure, especially in terms of its time complexity. It is relatively straightforward to see, that in many cases, however, a relatively minor fix to the original plan would resolve the failure possibly at a lower cost. Because it is not clear what exactly are the planning domains and types of dynamic environments which would allow for such a repair approach, it can be argued that non-informed plan repair attempts can in many cases even raise the overall complexity of the approach in comparison to replanning. This would be due to futile attempts to repair the failed plan before inevitably falling back to replanning.

In general, plan repair can be seen as planning with re-use of fragments of the old plan. There is a number of works, empirically demonstrating that plan repair in various domains performs better than replanning (e.g., [8, 2, 5]). However, in [10], Nebel and Koehler theoretically analyzed plan re-use (plan repair), and conclude that in general it does not bring any benefit over replanning in terms of computational time complexity.

In situated multi-agent systems, however, the time complexity is often not of the primary importance. In such systems, often it is the communication complexity which is a higher priority concern. Consider application domains, such as e.g., undersea operations by teams of coordinated autonomous underwater vehicles. 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 signaling. In such applications, it is the communication complexity of the distributed planning algorithms which matters more than time complexity. Consequently, employment of multi-agent plain repair techniques can provide a tangible benefit over replanning for a team of robots whose multi-agent plan fails.

The motivation for our research is the intuition that multi-agent plan repair, even though not always the fastest approach, should under specific conditions generate lower communication overheads in comparison to replanning. The conditions correspond to the level of required coordination and the types of failures the environment generates. While the hypothesis is rather intuitive, our approach is significant in that we give it a rigorous treatment. Besides our preliminary approach in [7], this aspect of multi-agent planning and plan repair, while obviously important with arrival of truly distributed algorithms for multi-agent planning, such as the one by Nissim et al. [11] did not witness considerable attention of the community yet.

The contribution of the the presented paper is threefold. Firstly, after introducing the general problem of multi-agent planning stemming from the formulation due to Brafman and Domshlak [3] in Section 2, in subsequent Section 3 we formally introduce the multi-agent plan repair problem and formally state

the core hypothesis of the presented research. Secondly, still in Section 3, we propose three algorithms for multi-agent plan repair reducing the problem to specialized instances of the multi-agent planning problem. Finally, in Section 4 we present experimental validation confirming the core hypothesis of the paper. Section 5 concludes the paper by some final remarks regarding the shortcomings of our approach and future outlooks in the here described line of research.

2 Multi-agent planning

Classical, single-agent planning problem is characterized by a set of states with a unique initial state, a final state (a set of final states) and a set of actions representing the transitions between these states that the system undertakes upon performing the actions. We define multi-agent planning problem as an extension of the classical single-agent planning. We consider a number of *cooperative* and *coordinated* actors featuring possibly distinct sets of capabilities (actions), which concurrently plan and subsequently execute their local plans so that they achieve a joint goal.

An instance of a multi-agent planning problem is defined by: i) an environment characterized by a state space, ii) a finite set of agents, each characterized by a set of primitive actions (or capabilities) it can execute in the environment, iii) an initial state the agents start their activities in and iv) a characterization of the desired goal states. Before treating the problem of multi-agent planning and summarizing a state-of-the-art algorithm for solving it, we first formally introduce the underlying concepts.

2.1 Preliminaries

Consider a set of atoms $\{p_1, \ldots, p_n\}$. A *state* is a set of terms from a language $\mathcal{L} = \{p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n\}$ where $\neg p$ denotes a negation of p. We also assume the standard tautology $p \equiv \neg\neg p$ for every $p \in \mathcal{L}$. Furthermore, we require all states to be *consistent*, i.e., for a state $s \subseteq \mathcal{L}$ we have that $p \in s$ if and only if $\neg p \notin s$. Note, in general, the states do not have to be complete, i.e., it might be that there is a $p \in \mathcal{L}$, such that $\{p, \neg p\} \nsubseteq s$. $\mathcal{S}$ denotes the set of all states and we assume there is a distinguished state $\chi \in \mathcal{S}$ denoting an undefined state in which the overall system can be in. To simplify the notation, we also extend the negation to states as follows $\neg s = \{p | \neg p \in s\}$. The set of atoms corresponding to a set of terms $\phi \subseteq \mathcal{L}$ is denoted $\underline{\phi} = \{p \mid p \in \phi \text{ or } \neg p \in \phi\}$.

A *primitive action*, or simply an *action*, is a tuple $\phi a \psi$, where a is a unique action label and $\phi, \psi \subseteq \mathcal{L}$ respectively denote the sets of preconditions and effects of a. The preconditions and effects are assumed to be consistent sets of terms. Whenever the context is clear, we simply write a instead of $\phi a \psi$. *Act* denotes the set of all actions and we furthermore assume there is a distinguished empty action $\emptyset \epsilon \emptyset \in Act$ with no preconditions and no effects.

We say that an action $\phi a \psi$ is *applicable* in a state s iff $\phi \subseteq s$. An application of $\phi a \psi$ is defined by the state transformation operator $\oplus : \mathcal{S} \times Act \rightarrow \mathcal{S}$ defined

3

as follows:

$$s \oplus a = \begin{cases} (s \cup \psi) \setminus \neg\psi & \text{iff } \phi \text{ is applicable in } s, \\ \chi & \text{otherwise.} \end{cases}$$

Note that given a consistent state s and an action $\phi a \psi$, the application of a to s results in either the undefined state χ, or, in the case a was applicable in s, a consistent state s' again. The application of $\phi a \psi$ to s first enriches s with all the effects of a, however, in the case there exists some $p \in s$, s.t., $\neg p \in \psi$, the simple unification would make the resulting state inconsistent. The subsequent set subtraction of terms which were the source of such inconsistencies makes the resulting state consistent again, while at the same time preserving the effects of a. Furthermore, $\oplus$ is associative, hence we can write $s \oplus a_1 \oplus \cdots \oplus a_n$.

An *agent* $\alpha = \{\phi_{a_1} a_1 \psi_{a_1}, \ldots, \phi_{a_n} a_n \psi_{a_n}\}$ is characterized precisely by its capabilities, a finite repertoire of actions it can preform in the environment. From now on, we assume that there exists a language $\mathcal{L}$ giving rise to a state space $\mathcal{S}$.

2.2 The Problem of Multi-agent Planning

Definition 1 (multi-agent planning). A *multi-agent planning problem* is a tuple $\Pi = (\mathcal{A}, s_0, S_g)$, where $\mathcal{A}$ is a set of *agents* $\alpha_1, \ldots, \alpha_n$, featuring mutually disjoint sets of actions, an initial state $s_0 \in \mathcal{S}$ and a set of goal states $S_g \subseteq \mathcal{S}$.

Before formally defining the notion of a solution to a multi-agent planning problem, we first introduce a sequel of auxiliary notions.

Given an agent α, a *single-agent plan* P is a sequence of actions $a_1, \ldots, a_k$, s.t., $a_i \in \alpha$ for every i. $P[i]$ denotes the i-th action in P, or $P[i] = \epsilon$ in the case i is larger than the length of P, which in turn will be denoted $|P|$.

A team of agents $\mathcal{A} = \alpha_1, \ldots, \alpha_n$ can act in the environment concurrently. A joint action $\phi_{\mathbf{a}} \mathbf{a} \psi_{\mathbf{a}}$ of the team is specified by $\mathbf{a} = (\phi_{a_1} a_1 \psi_{a_1}, \ldots, \phi_{a_n} a_n \psi_{a_n})$ a tuple of actions corresponding to the individual agents, i.e., $a_i \in \alpha_i$ for each i, its preconditions $\phi_{\mathbf{a}} = \bigcup_{i=1}^{n} \phi_{a_i}$ and its effects $\psi_{\mathbf{a}} = \bigcup_{i=1}^{n} \psi_{a_i}$. $\mathbf{a}[k]$ denotes the k-th action of $\mathbf{a}$. Similarly to actions of individual agents, $\phi_{\mathbf{a}}$ and $\psi_{\mathbf{a}}$ are assumed to be consistent sets of terms. The notion of action applicability in a state s, as well as application of $\mathbf{a}$ to s straightforwardly extend from the definitions for primitive actions.

Definition 2 (multi-agent plan). Let $\Pi = (\mathcal{A}, s_0, S_g)$ be a multi-agent planning problem with $\mathcal{A} = \alpha_1, \ldots, \alpha_n$. A *synchronous multi-agent plan* $\mathcal{P} = \{P_1, \ldots, P_n\}$, consisting of single agent plans $P_1, \ldots, P_n$ respectively constructed from actions of the agents $\alpha_1, \ldots, \alpha_n$ is a solution to Π if the plan $\mathcal{P}$ satisfies the following:

1. $\mathcal{P}$ is *well-formed*, i.e., $|P_i| = |P_j|$ for all $i, j \leq n$. $|\mathcal{P}| = |P_k|$, for some $k \leq n$, denotes the length of the multi-agent plan $\mathcal{P}$;

2. $\mathcal{P}$ is *feasible*, i.e., there exists a sequel of states $s_1, \ldots, s_m$, s.t. $m = |\mathcal{P}|$ and $s_{i+1} = s_i \oplus \mathbf{a}_i$ with $\mathbf{a}_i = (P_1[i], \ldots, P_n[i])$ for all $i < m$; and finally

4

3. $\mathcal{P}$ reaches the goal S_g, i.e., there exists $s_g \in S_g$, s.t. $s_m \subseteq s_g$.

We also say that $\mathcal{P}$ solves the problem Π. Finally, *Plans*(Π) denotes the set of plans which are solutions to a given multi-agent planning problem Π.

Additionally, $\mathcal{P}[k]$ denotes the joint action of the team in the step k and $\mathcal{P}[k, i]$ denotes the primitive action of the agent i in the step k. This notation allows us to introduce the following plan-matrix notation for a multi-agent plan $\mathcal{P}$, which provides a more visual understanding of multi-agent plans:

$$\mathcal{P} = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & & a_{m2} \\ \vdots & & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}$$

where $a_{ij} = \mathcal{P}[i, j]$. Indices i ($i \leq m = |\mathcal{P}|$) and j ($j \leq n$) denote the step of the plan $\mathcal{P}$ and the agent which performs the primitive action, respectively.

We say that two multi-agent plans $\mathcal{P}_1$, $\mathcal{P}_2$ are equal ($\mathcal{P}_1 = \mathcal{P}_2$) iff they have the same length ($|\mathcal{P}_1| = |\mathcal{P}_2|$) and for all i and j we have $\mathcal{P}_1[i, j] = \mathcal{P}_2[i, j]$.

A concatenation of two multi-agent plans $\mathcal{P}_1$ and $\mathcal{P}_2$ over the same agents $\alpha_1, \ldots, \alpha_n$ is defined as a plan $\mathcal{P} = \mathcal{P}_1 \cdot \mathcal{P}_2$, where for each i and j we have $\mathcal{P}[i, j] = \mathcal{P}_1[i, j]$ if $i \leq |\mathcal{P}_1|$ and $\mathcal{P}[i, j] = \mathcal{P}_2[i - |\mathcal{P}_2|, j]$ for $i > |\mathcal{P}_1|$. In the plan-matrix notation, the concatenation would correspond to simple columns-appending operation. Note, concatenation of multi-agent plans is an associative operation.

Given a multi-agent plan $\mathcal{P}$, $\mathcal{P}[i..j]$ denotes a fragment of $\mathcal{P}$ from the step i to the step j. More precisely, $\mathcal{P}[i..j]$ is a fragment of $\mathcal{P}$ iff there exist multi-agent plans $\mathcal{P}_{prefix}$ and $\mathcal{P}_{suffix}$, such that $\mathcal{P}_{prefix} \cdot \mathcal{P}[i..j] \cdot \mathcal{P}_{suffix} = \mathcal{P}$. Finally, $\mathcal{P}[i..\infty]$ denotes the i-th suffix of the plan $\mathcal{P}$, i.e., $\mathcal{P}[i..\infty] = \mathcal{P}[i..|\mathcal{P}|]$. $\mathcal{P}_1 \cdot \mathcal{P}_2$ is said to be a *decomposition* of a multi-agent plan $\mathcal{P}$ iff $\mathcal{P} = \mathcal{P}_1 \cdot \mathcal{P}_2$.

Given two multi-agent plans $\mathcal{P}_1$ and $\mathcal{P}_2$ we can define how different they are. *diff*$(\mathcal{P}_1, \mathcal{P}_2)$ denotes the difference between $\mathcal{P}_1$ and $\mathcal{P}_2$, that is the overall number of primitive actions in $\mathcal{P}_1$, which do not correlate with the corresponding primitive actions in $\mathcal{P}_2$ and *vice versa*. Formally, *diff*$(\mathcal{P}_1, \mathcal{P}_2) = |\{(i, j) \mid \mathcal{P}_1[i, j] \neq \mathcal{P}_2[i, j]\}|$. In the case $|\mathcal{P}_1| \leq |\mathcal{P}_2|$, *diff*$(\mathcal{P}_1, \mathcal{P}_2) = $ *diff*$(\mathcal{P}_1 \cdot \mathcal{P}_\epsilon, \mathcal{P}_2)$, where $\mathcal{P}_\epsilon$ is a plan padding of $\mathcal{P}_1$ to the overall length $|\mathcal{P}_1 \cdot \mathcal{P}_\epsilon| = |\mathcal{P}_2|$ and filled with empty actions, i.e, for each i, j, we have $\mathcal{P}_\epsilon[i, j] = \epsilon$. Note that the measure *diff* is position agnostic, i.e., we define *diff*$(\mathcal{P}_1, \mathcal{P}_2) = $ *diff*$(\mathcal{P}_2, \mathcal{P}_1)$ in the case $|\mathcal{P}_1| > |\mathcal{P}_2|$.

2.3 Planning Algorithm

The above formulation of the multi-agent planning problem is well in line with the original formulation of MA-STRIPS planning due to Brafman and Domshlak [3]. The authors there additionally distinguish between the *public* and *private* actions of the individual agents. An action is public whenever its preconditions or effects involve atoms occurring in preconditions or effects of an action

belonging to another agent of the team. Formally, given a multi-agent team $\mathcal{A} = \alpha_1, \ldots, \alpha_n$, the set of public actions is defined as $Act^{pub} = \{a \mid \exists i, j : i \neq j, a \in \alpha_i, a' \in \alpha_j,$ and $(\underline{\phi_a} \cup \underline{\psi_a}) \cap (\underline{\phi_{a'}} \cup \underline{\psi_{a'}}) \neq \emptyset\}$. Recall, $\underline{\phi}$ denotes the set of non-negated atoms occurring in ϕ. $Act^{priv} = Act \setminus Act^{pub}$, where $Act = \bigcup_{i=1}^{n} \alpha_i$ is the set of all actions the team $\mathcal{A}$ can perform.

The distinction of actions to private and public turns out to be an important one. Since private actions do not depend, nor are dependencies of other actions performable by the team, planning of sequences of private actions can be implemented strictly locally by the agent the actions belongs to. In effect, the public actions become points of coordination among the multi-agent team members and a truly decentralized multi-agent planning algorithm for a planning problem Π can be implemented in two interleaving stages until a suitable multi-agent plan is found: i) a plan consisting exclusively of public actions of the agent team is calculated, and subsequently ii) the sequences of private actions between the public actions of each individual agent are computed to fill in the gaps.

The main contribution of the Brafman and Domshlak's paper lies in pointing out that the algorithms can be implemented by reduction of the first stage of the planning process to a constraint satisfaction problem (CSP) corresponding to the multi-agent planning problem with public actions only. The second stage can be subsequently solved by any classical single-agent planning algorithm. In result, solving a given multi-agent planning problem can be loosely formulated as a CSP with the following two types of constraints:

coordination constraint: a sequence of joint actions $\mathcal{P}$ (candidate multi-agent plan) satisfies the *coordination constraint* iff for every action $\phi_a a \psi_a = \mathcal{P}[k, i]$ performed by the agent α_i in the step k we have, that if a is a public action, then

- for every $p \in \phi_a$, there must exist $\phi_{a_p} a_p \psi_{a_p} = \mathcal{P}[k_p, i_p]$, such that $p \in \psi_{a_p}$ and $0 < k_p < k$ (there is some previous action which causes p to hold), and

- for no k', s.t., $k_p \leq k' < k$ there exists $\phi' a' \psi' = \mathcal{P}[k', i']$, such that $\neg p \in \psi'$ (p won't be invalidated between causing it in the step k_p and execution of a in the step k).

internal planning constraint: a sequence of joint actions $\mathcal{P}$ satisfies the *internal planning constraint* iff for every agent, the corresponding single-agent planning problem with landmarks $\{a \mid a = \mathcal{P}[k, i] \in Act^{pub}\}$ is solvable. I.e., a single-agent planning algorithm is able to fill in the gaps between the public actions in the candidate multi-agent plan.

Algorithm 1 lists the original multi-agent planning algorithm MA-Plan by Brafman and Domshlak in [3]. The algorithm iterates through CSP formulations of the planning problem according to δ, informally the number of coordination points between the agents in the multi-agent team. I.e., δ determines the number

---

Algorithm 1 MA-Plan(Π):

Input: A multi-agent planning problem $\Pi = (\mathcal{A}, s_0, S_g)$.
Output: A multi-agent plan $\mathcal{P}$ solving Π, if such exists.

  $\delta = 1$
  loop
    Construct $\mathsf{CSP}_{\Pi;\mathcal{A}}$
    if solve-csp($\mathsf{CSP}_{\Pi;\delta}$) **then**
      Reconstruct a plan $\mathcal{P}$ from a solution for $\mathsf{CSP}_{\Pi;\delta}$.
      return $\mathcal{P}$
    else
      $\delta = \delta + 1$
    end if
  end loop

---

of joint actions in a candidate multi-agent plan consisting of only public actions. Filling the gaps between the individual single-agent public actions, if possible, then gives rise to the overall multi-agent plan. In the case such a plan completion does not exist, the process continues by testing longer candidate plans.

The original multi-agent planning algorithm assumes a centralized planning architecture. I.e., it is a centralized planning algorithm computing multi-agent plans for a team of agents which are supposed to be subsequently executed in a decentralized fashion. Our motivation is however a decentralized planning/plan repair algorithm followed by a decentralized plan execution.

Nissim et al. in [11] adapted the original blueprint algorithm described above to a distributed setting. The adaptation rests on formulating the multi-agent planning problem as a distributed constraint satisfaction problem instance (DisCSP) and subsequently utilizing a a state-of-the-art DisCSP solver for solving it, plus managing the overhead involved in the resulting distributed algorithm. The resulting algorithm, however, closely follows the scheme of the original algorithm as listed in Algorithm 1. From now on, whenever we speak about the implementation of the multi-agent planning algorithm, we have in mind its decentralized version due to Nissim et al. [11].

3   Multi-agent plan repair

Consider a multi-agent planning problem $\Pi = (\mathcal{A}, s_0, S_g)$ and a plan $\mathcal{P}$ solving Π. Furthermore, consider an environment in which, apart from the actions performed by the agents of the team $\mathcal{A}$, no other exogenous events occur. We say that such an environment is *ideal*, or *non-dynamic*. The execution of $\mathcal{P}$ in such an environment is uniquely determined by the set of states $s_0, \ldots, s_m$, such that $s_{i+1} = s_i \oplus \mathcal{P}[i]$ (cf. also Definition 2).

In dynamic environments, however, it can occur that in the course of execution of $\mathcal{P}$, the environment interferes and the execution of some action $\mathcal{P}[i]$

from the plan $\mathcal{P}$ does not result in precisely the state s_{i+1} as defined above. We could say that at step i an unexpected event occurred in the environment. For simplicity, we consider only unexpected events happening exclusively in the course of execution of some action (as if it took a non-zero time), not such which could occur while the agent is deliberating the execution (i.e., as if the deliberation was instantaneous).

Note that not all unexpected events in a dynamic environment necessarily lead to problems with execution of the plan $\mathcal{P}$. However, there are at least two cases of such events, which can be considered a *plan execution failure.*

A *weak failure* of execution of the plan $\mathcal{P}$ at step i w.r.t. the multi-agent planning problem Π is such, when the state s_f resulting from an attempt to perform an action $\phi_{\mathbf{a}}\mathbf{a}\psi_{\mathbf{a}} = \mathcal{P}[i]$ for some i does not satisfy some of the postconditions of $\mathbf{a}$, i.e., $\psi_{\mathbf{a}} \not\subseteq s_f$.

A *strong failure* of execution of the plan $\mathcal{P}$ at step i w.r.t. the planning problem Π occurs whenever the i-th action of $\mathcal{P}$ cannot be executed due to its inapplicability. I.e., the execution of the plan up to the step i resulted in states $s_0, s'_1 \ldots, s'_i$, possibly with some weak failures occurring in the course of execution of the plan fragment and $\mathcal{P}[i]$ is not applicable in s'_i.

The weak and the strong plan execution failures are, however, just two examples of a plan failure. There certainly are application domains in which weak failures can be tolerated as far as the goal state is reached after execution of the multi-agent plan. Alternatively, there might be domains in which other types of plan execution failures can occur, e.g., any change of the state not caused by the involved agents can be considered a failure as well. To account for the range of various types failures, from now on, we only require that a plan execution monitoring process determines some plan execution failure at a step i which results in some failed state s_f.

Definition 3 (multi-agent plan repair)**.** Let $\Pi = (\mathcal{A}, s_0, S_g)$ be a multi-agent planning problem. A *multi-agent plan repair problem* is a tuple $\Sigma = (\Pi, \mathcal{P}, s_f, k)$, where $\mathcal{P}$ is a multi-agent plan solving the planning problem Π, k is the step of $\mathcal{P}$ in which its execution failed and $s_f \in \mathcal{S}$ is the corresponding failed state.

A solution to the plan repair problem Σ is a multi-agent plan $\mathcal{P}'$, such that $\mathcal{P}'$ is a solution to the planning problem $\Pi' = (\mathcal{A}, s_f, S_g)$. We say that $\mathcal{P}'$ *repairs* $\mathcal{P}$ in s_f. In the case $Plans(\Pi') = \emptyset$, we say that the plan is *irreparable* given the failure occurring at the state s_f.

Given two multi-agent plans $\mathcal{P}_1$ and $\mathcal{P}_2$ both repairing a multi-agent plan $\mathcal{P}$ for a problem Π in a state s_f, we say that $\mathcal{P}_1$ is *preserving $\mathcal{P}$ more* than $\mathcal{P}_2$ iff $diff(\mathcal{P}_1, \mathcal{P}) \leq diff(\mathcal{P}_2, \mathcal{P})$ and denote the relation by $\mathcal{P}_1 \preceq \mathcal{P}_2$. The *minimal repair of the multi-agent plan* $\mathcal{P}$ is such a plan $\mathcal{P}_{\min} \in Plans(\Pi')$, which is minimal w.r.t. the mutual differences between the plans solving Π'. I.e., $\mathcal{P}_{\min} \in \underset{\mathcal{P}' \in Plans(\Pi')}{\arg\min} \; diff(\mathcal{P}, \mathcal{P}')$.

Note, there might be several distinct minimal repairs of a given multi-agent plan.

In general, the multi-agent plan repair problem can be reduced to solving a modified multi-agent planning problem and thus gives rise to a straightforward plan repair algorithm based on *replanning* in two steps: 1) construct the multi-agent replanning problem Π' as prescribed in Definition 3, and subsequently 2) utilize the MA-Plan algorithm (Algorithm 1) to solve the problem Π'.

The original motivation underlying this paper was the hypothesis that attempts to repair failed multi-agent plans lead to lower communication overhead than replanning. Clearly, not all planning problems could benefit from such a mechanism. Since we focus on multi-agent planning problems, which in a sense *enforce coordination* among the members of a multi-agent team, we firstly introduce the concept of k-coordinated multi-agent planning problems.

Definition 4 (k-coordination)**.** We say that a multi-agent plan $\mathcal{P}$ is k-*coordinated* iff each fragment $\mathcal{P}$ of length k contains at least one joint action containing a public action. Formally, for every $\mathcal{P}'$, s.t. $\mathcal{P} = \mathcal{P}_{prefix} \cdot \mathcal{P}' \cdot \mathcal{P}_{suffix}$ with $|\mathcal{P}'| = k$, there exist i and j so that $\mathcal{P}'[i,j] \in Act^{pub}$.

We say that a multi-agent problem Π is k-*coordinated* iff all the plans $\mathcal{P} \in Plans(\Pi)$ solving Π, which cannot be compressed are k-coordinated. A plan $\mathcal{P} \in Plans(\Pi)$ can be *compressed* iff it contains a fragment $\mathcal{P}'$, s.t. $\mathcal{P} = \mathcal{P}_{prefix} \cdot \mathcal{P}' \cdot \mathcal{P}_{suffix}$ and $\mathcal{P}_{prefix} \cdot \mathcal{P}_{suffix} \in Plans(\Pi)$.

We informally say that multi-agent planning problems leading to plans containing coordination points (public actions) placed relatively frequently throughout the plans are tightly coordinated. More formally, a multi-agent planning problem Π is *tightly coordinated* if it is k-coordinated and k is relatively low in comparison to the lengths of plans from $Plans(\Pi)$. In the case k is relatively high w.r.t. the plan lengths, we say that the problem is *loosely coordinated* and finally, if the plans do not involve public actions, i.e., coordination is not needed at all, we say the problem is *uncoordinated*.

The core hypothesis of the paper can be then formulated as follows:

Hypothesis 1. Multi-agent plan repair approaches producing more preserving repairs than replanning tend to generate *lower communication overhead* for tightly coordinated multi-agent problems.

A crisper, though perhaps a more challenging version of the hypothesis would express the communication overhead in terms of the average communication complexity:

Hypothesis 2. When applied to tightly coordinated planning problems, multi-agent plan repair algorithms producing more preserving repairs than replanning feature a *lower average communication complexity* than replanning.

In the remainder of this paper, we approach resolution of Hypothesis 1. Treatment of Hypothesis 2 is beyond the scope of this paper and is left for future work.

Algorithm 2 Back-on-Track-Repair(Σ)

Input: A multi-agent plan repair problem $\Sigma = (\Pi, \mathcal{P}, s_f, k)$, with $\Pi = (\mathcal{A}, s_0, S_g)$ and a sequence of states $s_0, \ldots, s_m$ execution of $\mathcal{P}$ generates in the ideal environment.
Output: A multi-agent plan $\mathcal{P}'$ solving Σ if a solution exists.

    Construct $\Pi^{back} = (\mathcal{A}, s_f, \{s_0, \ldots, s_m\})$
    if MA-Plan(Π^{back}) returns a solution $\mathcal{P}^{back}$ **then**
        Retrieve the state s_j of $\mathcal{P}$ to which $\mathcal{P}^{back}$ returns
        return $\mathcal{P}' = \mathcal{P}^{back} \cdot \mathcal{P}[j \ldots \infty]$
    end if

3.1 Back-on-track Repair

Unexpected event occurring in an environment can cause a failure in execution of a plan performed by some multi-agent team in that environment. The result is that the overall state of the system is not the one expected by the undisturbed plan execution at the particular time step. A straightforward idea to fix the problem is to utilize a multi-agent planner to produce a plan from the failed state to the originally expected state and subsequently follow the rest of the original multi-agent plan from the step in which the failure occurred. The following multi-agent plan repair approach, coined *back-on-track* (BoT) repair, is inspired by this idea, in fact a slight generalization of it.

Definition 5 (back-on-track repair)**.** Let $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ be a multi-agent plan repair problem and $\Pi' = (\Delta, s_f, S_g)$ being the corresponding modified multi-agent replanning problem.

    We say that a plan $\mathcal{P}' \in Plans(\Pi')$ is a *back-on-track repair* of $\mathcal{P}$ iff there is a decomposition of $\mathcal{P}'$, such that $\mathcal{P}' = \mathcal{P}^{back} \cdot \mathcal{P}[i..\infty]$ for some $i \leq |\mathcal{P}|$.

    $\mathcal{P}' = \mathcal{P}^{back} \cdot \mathcal{P}[i..\infty]$ is said to be a *proper back-on-track repair* iff $|\mathcal{P}[i..\infty]| > 0$. I.e., $\mathcal{P}'$ preserves some non-empty suffix of $\mathcal{P}$.

    Informally, the back-on-track approach tries to preserve a suffix of the original plan and prefix it with a newly computed plan $\mathcal{P}^{back}$ starting in s_f and leading to some state along the execution of $\mathcal{P}$ in the ideal environment. Note, that all plans from $Plans(\Pi')$ are back-on-track repairs of the original plan. The length of the preserved suffix of the original plan provides a handle on the repair quality ordering of the plans. The longer the preserved suffix, the more preserving the plan is. On the other hand, even when the plan repair problem Σ is indeed solvable, there might not be any valid proper back-on-track repair of the original planning problem.

    Algorithm 2 realizes a multi-agent plan repair procedure according to the back-on-track plan repair principle. Since the MA-Plan algorithm searches for the shortest plan from the initial state to a goal state, the Back-on-Track-Repair computes plans which return back to the original one in the shortest possible way.

Algorithm 3 Lazy-Repair(Σ)

Input: A multi-agent plan repair problem $\Sigma = (\Pi, \mathcal{P}, s_f, k)$, with $\Pi = (\mathcal{A}, s_0, S_g)$ and $|\mathcal{A}| = n$.

Output: A multi-agent plan $\mathcal{P}'$ solving the problem Σ according to the lazy approach, if a solution exists.

Construct $\mathcal{P}_{[k..\infty]}$, the executable remainder of $\mathcal{P}$ from the step k and state s_f

Construct $\Pi^{lazy} = (\mathcal{A}, s_{|\mathcal{P}|}, S_g)$

let $\mathcal{P}^{lazy}$ be a solution to MA-Plan(Π^{lazy}) if such exists.

return $\mathcal{P}_{[k..\infty]} \cdot \mathcal{P}^{lazy}$

3.2 Simple Lazy Repair

The back-on-track multi-agent plan repair approach seeks to compute a new prefix to some suffix of the original plan and repair the failure by their concatenation. An alternative approach, coined *lazy*, attempts to preserve the remainder of the original multi-agent plan and close the gap between the state resulting from the failed plan execution and a goal state of the original planning problem.

Let s_f be the state resulting from a failure in execution of a multi-agent plan $\mathcal{P}$ in a step k. We say that a sequence of joint actions $\mathcal{P}'$ is an *executable remainder* of $\mathcal{P}$ from the step k and the state s_f iff there exists a sequence of states $s_k, \ldots, s_{|\mathcal{P}|}$, such that $s_k = s_f$, $s_{i+1} = s_i \oplus \mathcal{P}'[i - k + 1]$ and for every step i and every agent j, we have that $\mathcal{P}'[i - k + 1, j] = \mathcal{P}[i, j]$ in the case $\mathcal{P}[i, j]$ is applicable in the state s_i and $\mathcal{P}'[i - k + 1, j] = \epsilon$ otherwise.

The following definition provides a formal definition of the lazy approach.

Definition 6 (simple lazy repair)**.** Let $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ be a multi-agent plan repair problem and $\Pi' = (\mathcal{A}, s_f, S_g)$ being the corresponding modified multi-agent replanning problem.

We say that a plan $\mathcal{P}' \in Plans(\Pi')$ is a *lazy repair* of $\mathcal{P}$ iff there is a decomposition of $\mathcal{P}'$, such that $\mathcal{P}' = \mathcal{P}_{[k..\infty]} \cdot \mathcal{P}^{lazy}$, where $\mathcal{P}_{[k..\infty]}$ is the executable remainder of $\mathcal{P}$ from the step k, execution of which results in the state s_{lazy} when starting in s_f, and $\mathcal{P}^{lazy}$ is a solution to the multi-agent planning problem $\Pi^{lazy} = (\mathcal{A}, s_{lazy}, S_g)$.

Algorithm 3 realizes multi-agent plan repair based on the lazy repair approach described above.

The back-on-track approach always succeeds to compute some multi-agent plan repairing the original plan from the failed state in the case the replanning form scratch would compute such a plan from that state as well. The lazy approach is in this sense incomplete, as it might happen that the execution of the executable remainder of the original plan diverges to a state from which no plan to some goal state exists. Given that dynamic environment in general could generate irreparable failures, this incompleteness cannot be considered a

shortcoming of the lazy approach in general. Of course in domains in which no irreparable unexpected event might occur, while at the same time the agents are allowed to perform actions potentially having irreversible and potentially harmful effects, the lazy approach has to be employed with caution.

3.3   Repeated Lazy Repair

In dynamic environment plan failures occur repeatedly, i.e., even after a repair of a failed plan, it is possible for the repaired plan to fail again. In this situation both the back-on-track, as well as the lazy multi-agent plan repair algorithms lead to prolonging the really executed plan. In the case of the back-on-track approach, this is inevitable, since upon the repair, the subsequent plan execution process immediately processes the newly added plan fragment. In the case of the lazy repair, however, upon occurrence of another failure during execution of the repaired plan, it is not always necessary to prolong the overall multi-agent plan.

The intuition behind the *repeated lazy plan repair* approach is that a failure during execution of an already repaired plan makes the previous repair attempt irrelevant and its result can be discarded, unless the failure occurred already in the plan fragment appended by the previous repair. The following definition formally introduces the extension of the lazy multi-agent plan repair approach. For clarity, from now on, we refer to the lazy multi-agent plan repair approach introduced in Definition 6 as *simple lazy repair*.

Definition 7 (Repeated lazy repair)**.** Let $\Pi = (\mathcal{A}, s_0, S_g)$ be a multi-agent planning problem with a solution $\mathcal{P}$ and $\Sigma_1 = (\Pi, \mathcal{P}, s_{f_1}, k_1)$ be a multi-agent plan repair problem with a lazy repair solution $\mathcal{P}_{\Sigma_1} = \mathcal{P}_{[k_1..\infty]} \cdot \mathcal{P}_{suffix}$ and $\Sigma_2 = (\Pi, \mathcal{P}_{\Sigma_1}, s_{f_2}, k_2)$ be a multi-agent plan repair problem the system is currently facing.

We say that $\mathcal{P}'$ is a *repeated lazy repair* of $\mathcal{P}_1$ iff

1. $\mathcal{P}'$ is a simple lazy repair solution to $\Sigma' = (\Pi, \mathcal{P}, s_{f_2}, k_2)$ in the case $k_2 \leq |\mathcal{P}_{[k_1..\infty]}|$; and

2. $\mathcal{P}'$ is a simple lazy repair solution to Σ_2 otherwise.

The repeated lazy repair leads to a straightforward extension of the lazy plan repair algorithm listed in Algorithm 3. Note, that the repeated lazy repair algorithms enables a plan execution model which preserves significantly longer fragments of the original plan. That is, upon a failure, instead of trying to repair the failed plan right away, as both the back-on-track and simple lazy plan repair algorithms do, the system can simply proceed with execution of the remainder of the original plan and only after its complete execution the lazy plan repair is triggered. The approach simply ignores the plan failures during execution and postpones the repair the very end of the process, hence the "*lazy*" label for the two algorithms.

Algorithm 4 Repeated-Lazy-Repair($\Sigma_1, \mathcal{P}_{\Sigma_1}, \Sigma_2$)

Input: A multi-agent planning problem $\Pi = (\mathcal{A}, s_0, S_g)$, a multi-agent plan $\mathcal{P} = \mathcal{P}_{[k..\infty]} \cdot \mathcal{P}_{suffix}$ solving Π, and two multi-agent plan repair problems $\Sigma_1 = (\Pi, \mathcal{P}, s_{f_1}, k_1)$ with a solution $\mathcal{P}_{\Sigma_1}$ and $\Sigma_2 = (\Pi, \mathcal{P}_{\Sigma_1}, s_{f_2}, k_2)$.
Output: A multi-agent plan solving Σ_2 if a solution exists.

   if $k_2 \leq |\mathcal{P}_{[k..\infty]}|$ **then**
      Construct $\Sigma' = (\Pi, \mathcal{P}, s_{f_2}, k_2)$
   else
      $\Sigma' = \Sigma_2$
   end if
   return a solution to Lazy-Repair(Σ') if such exists
---

4 Experimental validation

To verify the Hypothesis 1, we conducted a series of experiments with implementations of the multi-agent plan repair algorithms described in Section 3. Below, we firstly describe the experimental setup used for the experiments and subsequently interpret the data collected and revisit Hypothesis 1.

4.1 Experimental Setup

The experiments were based on a two-stage algorithm. In the first phase, for a given domain a multi-agent plan was computed using the MA-Plan algorithm based on a distributed constraint satisfaction solver for computing the candidate coordination plans and implementation of a best-first-search action planning algorithm as part of FF [6] for computing the local, single-agent plans. We used the implementation of the distributed multi-agent planner authored by Nissim et al. also used for the experiments conducted in their paper [11]. In the second phase, we executed the multi-agent plan. In the course of the plan execution, we simulated the environment dynamics by producing various plan failures according to a variable failure probability. The plan execution was monitored and upon a failure detection a plan repair algorithm was invoked. Algorithm 5 lists the pseudo-code of the process.

Before execution of each plan step, the joint action is checked for applicability in the current state. In the case it is not applicable, a plan repair algorithm is invoked and the execution continues on the repaired plan. Otherwise, the state is updated with the joint action.

The execute-fail function in the algorithm either updates the current state by the joint action provided as a parameter as if in the ideal environment, or to generate an unexpected event occurring in the simulated dynamic environment.

We distinguish two types of plan failures: *action failures* and *state perturbations*. Both failure types are parametrized by a uniformly distributed probability P, which determines whether a simulation step fails, or not. Both failure types are weak failures. That is, they are not handled immediately, but can preclude

---

Algorithm 5 Plan execution and monitoring algorithm.

Input: An initial multi-agent planning problem $\Pi = (\mathcal{A}, s_0, S_g)$.

> $\mathcal{P} = \mathsf{MA\text{-}Plan}(\Pi)$
> $s = s_0;\ step = 1$
> **repeat**
>   **if** $\phi_{\mathcal{P}[step]} \not\subseteq s$ **then**
> $\mathcal{P} = \mathsf{Repair}(\Pi, \mathcal{P}, s, step)$
> $step = 1$
>   **end if**
> $s = \mathsf{execute\text{-}fail}(s, \mathcal{P}[step])$
> $step = step + 1$
> **until** $step > |\mathcal{P}|$

---

the plan execution and later result in a strong failure. Upon detection, a strong failure is handled by one of the plan repairing algorithms.

An *action failure* is simulated by not-executing some of the individual agent actions from the actual plan step. The individual action is chosen according to a uniformly distributed probability. The individual action is removed from the joint action and the current state is updated by the modified joint action.

The other simulated failure type, *state perturbation,* is parametrized by a positive non-zero integer *c*, which determines the number of state terms, which are removed from the current state, as well as the number of terms which are added to it. The terms to be added or removed are selected also randomly from the domain language according to a uniform distribution.

The experimental setup was implemented as a centralized simulator of the environment integrating a decentralized multi-agent domain-independent planner $\mathsf{MA\text{-}Plan}$. The individual agents are initialized by a planning domain, together with a particular planning problem instance. Each agent runs in its own thread and they deliberate asynchronously. The agents send peer-to-peer messages among themselves. Message passing is mediated by the centralized simulator as well. The messages are sent in the DisCSP phase by the integrated solver, which is a part of the $\mathsf{MA\text{-}Plan}$ planner.

The experiments were performed on *Phenom Quad Core 9950* processor at 2.6GHz with *Java Virtual Machine* limited to 2.5GB of RAM. The individual measurements were parametrized by the plan failure probability P and each problem instance was executed 6–10 with various value samples. The resulting data are, in the figures, presented with natural distribution. The candlestick charts depict the difference between the minimal and the maximal measurements, together with the standard deviation.

4.2   Test Problems, Algorithms and Metrics

The experiments were conducted on three planning domains. The domains originate in the standard benchmark single-agent ICP planning domains pub-

lished at [1]. Similarly to [11], we chose domains, which are straightforwardly modifiable to multi-agent planning problems: LOGISTICS (3 agents), ROVERS (3 agents), and SATELLITES (2–5 agents).

The LOGISTICS domain is a tightly coordinated in that it requires relatively frequent coordination among the involved agents: airplanes and trucks need to wait for each other to load or unload the transported packages. The ROVERS domain is loosely coordinated in that it requires coordination only at the end of plans: there is a single shared communication channel between one of the rovers and the receiving station. Finally, the SATELLITES domain is uncoordinated in that it does not need any coordination between the satellites acquiring images individually.

To evaluate validity of Hypothesis 1, the multi-agent planning problems were tested on the experimental setup against a plan repair algorithm implementing replanning from scratch and two of the repair algorithms Back-on-Track-Repair (Algorithm 2) and Repeated-Lazy-Repair (Algorithm 4) introduced in Section 3.

Efficiency problems of the MA-Plan implementation limited the experiments to plans with maximally two landmarks (coordination points). The measurements of Back-on-Track-Repair algorithm runs were negatively influenced by sensitivity of the planner implementation to the number of terms in the goal state. Additionally, the Back-on-Track-Repair algorithm could not leverage disjunctive goal form (cf. Definition 5) and this was emulated by iterative process testing all term conjunctions in a sequence and thus resulting in multiple runs of the DisCSP solver instead of a single run with disjunctive goal.

We used three metrics to evaluate the measurements:

execution length is the overall number of joint actions the experimental setup executed.

planning time is the measured cumulative time consumed by the underlying MA-Plan planner used for generating initial and repairing plans; and finally

communication is measuring the number of messages passed between the agents during the planning or plan repair process. That is messages generated by the DisCSP solver in the MA-Plan planner.

4.3   Results and Discussion

The first batch of experiments directly targets validation of Hypothesis 1: *multi-agent plan repair is expected to generate lower communication overhead in tightly coordinated domains.* We used LOGISTICS as a tightly coordinated domain and dynamics of the simulated environment modeled as action failures. Figure 1 depicts the results of the experiment. The communication overhead generated by the Back-on-Track-Repair algorithm is on average only 59% (36% at best) of that generated by the replanning approach. Furthermore, the Repeated-Lazy-Repair algorithm performed even better and on average produced only 43% (11% at best) of the communication overhead generated by the replanning algorithm. In result, the experiments *strongly support* our hypothesis.
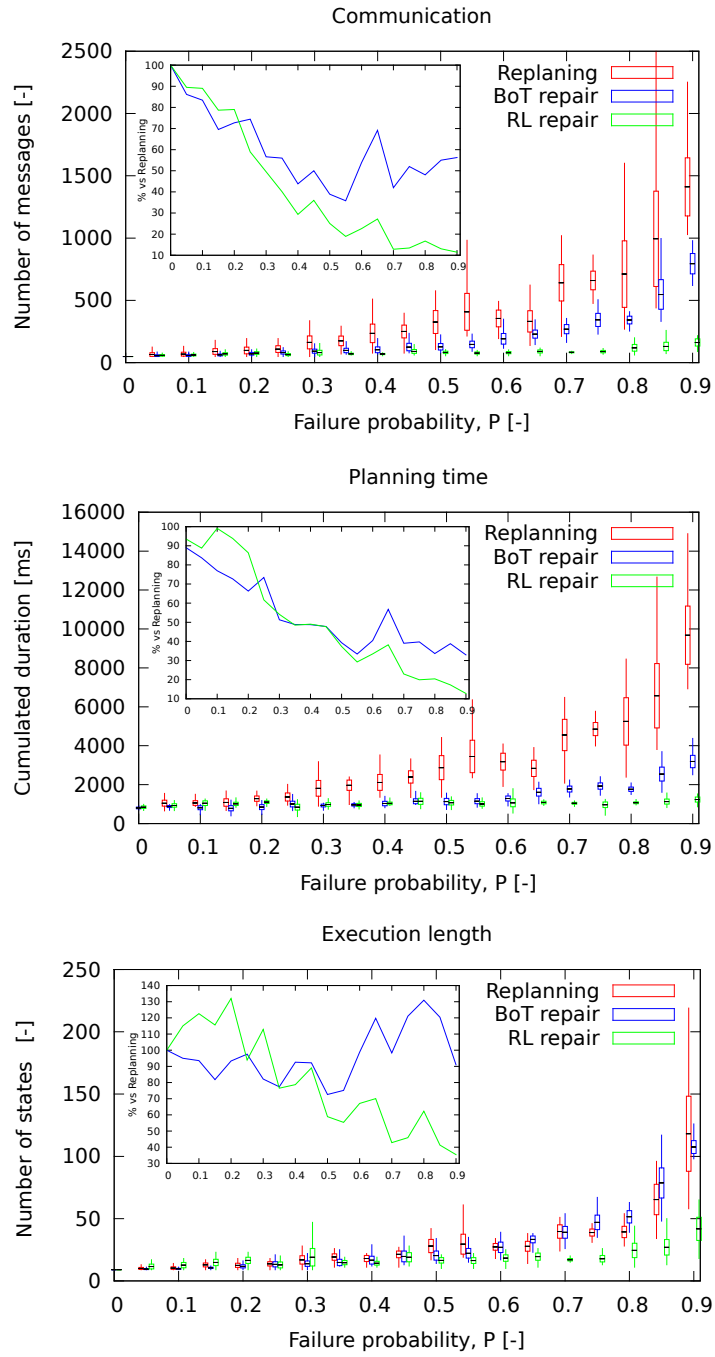
Figure 1: Experimental results for LOGISTICS domain with 3 agents and action failures.
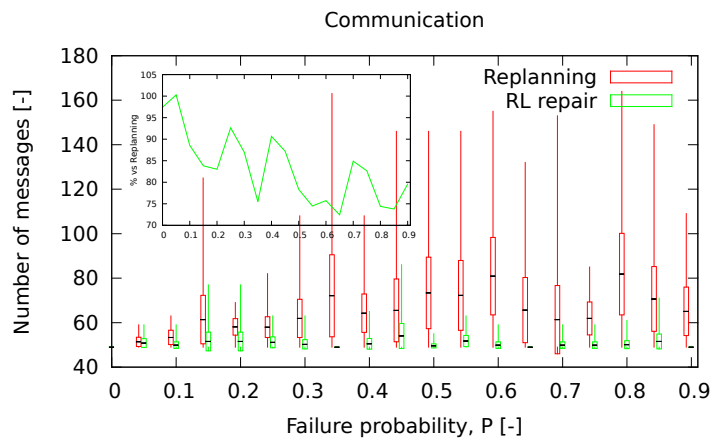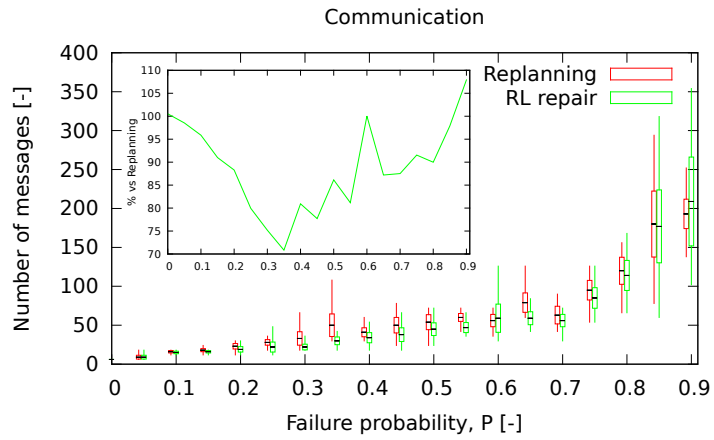
Figure 2: Experimental results for ROVERS domain with 3 agents and action failures (top). Experimental results for LOGISTICS domain with 3 agents and state perturbations with $c = 1$ (bottom).
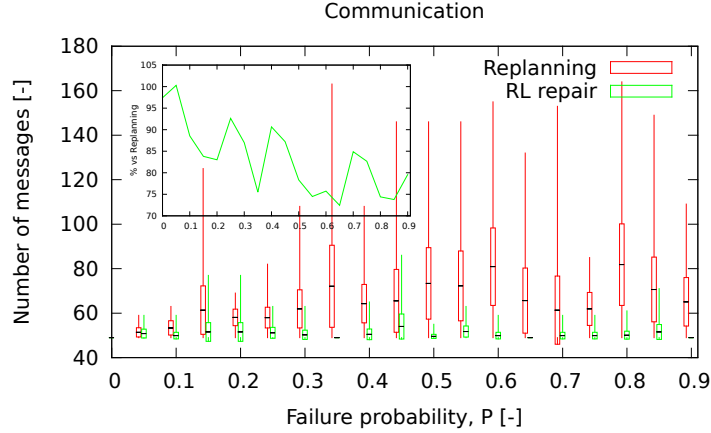
Figure 3: Experimental results for LOGISTICS domain with 3 agents and state perturbations with $c = 1$.

Additionally, the overall time spent in the planning phase (used by the MA-Plan algorithm) by the plan repair algorithms was 54% (34% at best) and 51% (12% at best) for Back-on-Track-Repair and Repeated-Lazy-Repair respectively. The execution length was lower in comparison the replanning approach as well being in average 96% (72% at best, 130% at worst) by Back-on-Track-Repair and lower being 81% (34% at best, 132% at worst) for Repeated-Lazy-Repair.

The second batch of experiments focused on boundaries of validity of the positive result presented above. In particular, we validated the condition on the coordination tightness and feasibility of failures. The auxiliary hypothesis we validated states: *with decreasing coordination tightness of the domain, the communication efficiency gains of repairing techniques should decrease. For loosely coordinated domains the communication efficiency of plan repair should be on-par with that of the replanning approach.*

To validate the auxiliary hypothesis we ran experiments with the ROVERS as a loosely coordinated domain. Figure 3 (top) presents the results supporting the hypothesis.

The third batch of experiments targeted the perturbation magnitude of the plan failures. The second auxiliary hypothesis we validated states: *communication efficiency gain of plan repairing in contrast to replanning should decrease as the difference between a nominal and related failed state increases.* The underlying intuition is that, in the case the dynamic environment generates only relatively small state perturbations and the failed states are "not far" from the actual state, the plan repair should perform relatively well. On the other hand, if the state essentially "teleports" the agents to completely different states, replanning tends to generate more efficient solutions than plan repair.

To answer this hypothesis, we have prepared another LOGISTICS experiment

employing state perturbations as the model of the environment dynamics. Figure 3 (bottom) depicts results of the experiment for $c = 1$. The perturbed state for $c = 1$ is produced by removing one term from the actual state and adding another one. As the chart shows, under random perturbations the plan repairing technique lost its improvement against replanning. For stronger perturbations with $c = 2, 3, 4$, the ratio between plan repairing and replanning remained on average the same. The trend of the absolute numbers of messages, planning time and execution length was slightly decreasing, as the probability of opportunistic effects increased.

Finally, we conducted a series of experiments with a non-coordinated SAT-TELITES domain. The results depicted in Figure 3 show the anticipated lower plan repair communication efficiency in contrast to replanning.

5 Final remarks

In the presented paper, we formally introduced the problem of multi-agent plan repair, proposed three algorithms for solving it and experimentally validated the hypothesis stating that under certain conditions, multi-agent plan repair approach tends to be more efficient in terms of the communication overhead it generates in comparison to the replanning approach. Our results well support the core hypothesis of the paper and we additionally performed a series of experiments validating its boundary conditions.

The line of research underlying this paper well correlates with recent works on classical single-agent planning sub-domains, such as partial ordered plan monitoring and repairing [9], conformant and contingency planning, plan re-use and plan adaptation. Environment dynamics is also handled by approaches based on Markov decision processes. The main difference to our approach is that the state perturbations utilized in our experiments have *a priori* unknown probabilities. Our own recent approach to the problem of multi-agent plan repair in [7] can be seen only as a precursor to the formal and rigorous treatment of the problem in this paper. Therein, we described the first steps towards formal treatment of the problem, as well as proposed two specific incomplete algorithms for solving the problem, very distinct from the ones presented here.

There are several open challenges resulting from the presented work. Firstly, the multi-agent planning framework (MA-STRIPS) is not expressive enough to describe certain aspects of concurrent actions and should be extended to this end, what, we suspect, will also influence the multi-agent planning complexity analysis. In particular, there is no way to account for joint actions which have effects strictly different than the unity of the individual actions involved. Another issue is that there is no way to enforce or forbid concurrent execution of certain individual actions. Secondly, the framework is not able to describe concurrent resource consumption, which is not an issue in single-agent STRIPS [4] planning, but in the multi-agent extension two individual concurrently executed actions might "consume" the same precondition, even though it is undesirable in the domain. Thirdly, there is a need for more efficient implementations of

multi-agent planners with more features as the gap between the state-of-the-art classical planners and multi-agent planners is enormous. Fourthly, there is a lack of standardized planning benchmarks for multi-agent planning, especially considering tightly coordinated planning problems. Such are needed to further evaluate the hypotheses presented in this paper. Finally, we leave out the work towards resolving the validity of Hypothesis 2 aiming at investigation of complexity issues of multi-agent plan repair to future work.

References

[1] The international planning competition, ICAPS, http://ipc.informatik.uni-freiburg.de/.

[2] T. C. Au and H. Munoz-Avila. On the complexity of plan adaptation by derivational analogy in a universal classical planning framework. *Advances in Case-Based Reasoning*, pages 13–27, 2002.

[3] Ronen I. Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proc. of ICAPS*, pages 28–35, 2008.

[4] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proc. of the 2nd International Joint Conference on Artificial Intelligence*, pages 608–620, 1971.

[5] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proc. of ICAPS*, 2006.

[6] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[7] Antonín Komenda and Peter Novák. Multi-agent plan repairing. In Christopher Amato, Emma Brunskill, Guy Shani, and Matthijs Spaan, editors, *Proceedings of Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities, DMPOUW 2011, IJCAI'11 collocated workshop*, 2011.

[8] 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.

[9] Christian Muise, Sheila A McIlraith, and J Christopher Beck. Monitoring the execution of partial-order plans via regression. In *Proc. of 22nd International Joint Conference on Artificial Intelligence*, pages 1975–1982, 2011.

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

[11] Raz Nissim, Ronen I. Brafman, and Carmel Domshlak. A general, fully distributed multi-agent planning algorithm. In *Proc. of AAMAS*, pages 1323–1330, 2010.