Domain-independent Multi-agent Plan Repair

Antonín Komenda

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Czech Republic

Peter Novák

Department of Software and Computer Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
The Netherlands

Michal Pěchouček

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Czech Republic

Abstract

Achieving joint objectives in distributed domain-independent planning problems by teams of cooperative agents requires significant coordination and communication efforts. For systems facing a plan failure in a dynamic environment, arguably, attempts to repair the failed plan in general, and especially in the worst-case scenarios, 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

Email addresses: antonin.komenda@agents.fel.cvut.cz (Antonín Komenda), P.Novak@tudelft.nl (Peter Novák), michal.pechoucek@agents.fel.cvut.cz (Michal Pěchouček)

Preprint submitted to Journal of Network and Computer Applications September 29, 2012

communication overhead might be even prohibitive in certain domains. We hypothesize that in decentralized systems, where frequent coordination is required to achieve joint objectives, attempts to repair failed multi-agent plans should lead to lower communication overhead than replanning from scratch.

Here, we formally introduce the *multi-agent plan repair problem.* Building upon the formal treatment, we present the core hypothesis underlying our work and subsequently describe three algorithms for multi-agent plan repair reducing the problem to specialized instances of the multi-agent planning problem. Finally, we present an experimental validation, results of which confirm the core hypothesis of the paper. Our rigorous treatment of the problem and experimental results pave the way for both further analytical, as well algorithmic investigations of the problem.

Keywords: Multi-agent plan repair, Decentralized multi-agent planning, Communication complexity, Experimental evaluation

---

Domain-independent Multi-agent Plan Repair

Antonín Komenda

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Czech Republic

Peter Novák

Department of Software and Computer Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
The Netherlands

Michal Pěchouček

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Czech Republic

---

1. Motivation

Multi-agent planning based on classical planning is an approach to constructing control mechanisms for a team of possibly heterogeneous autonomous agents which compute and subsequently execute plans for the individual agents so as to achieve some joint team objective in the environment. When an agent is situated in a dynamic environment, occurrence of various unexpected events in the environment might lead to invalidation of the plan, a

Email addresses: antonin.komenda@agents.fel.cvut.cz (Antonín Komenda), P.Novak@tudelft.nl (Peter Novák), michal.pechoucek@agents.fel.cvut.cz (Michal Pěchouček)

Preprint submitted to Journal of Network and Computer Applications September 29, 2012

failure. A straightforward solution to this problem is to invoke a planning algorithm and compute a new plan from the state the failure occurred in to a state conforming with its original objective.

In general, replanning in the case of a failure occurrence is a costly procedure, especially in terms of its time complexity. 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 benefit from 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.

Plan repair can be seen as planning with re-use of fragments of the old plan. Even though there is a number of works, empirically demonstrating that plan repair in some domains performs better than replanning (e.g., [1, 2, 3]), Nebel and Koehler in [4] theoretically analyzed plan re-use (plan repair), and concluded that in general it does not bring any benefit over replanning in terms of computational time complexity. Taking into an account the performance of modern classical planners on modern hardware, the benefit of repairing failed plans would often be relatively low from the time-complexity perspective.

In situated multi-agent systems, however, the time complexity is often not of the primary importance. 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 the 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.

Study of multi-agent planning system in previous literature [5, 6, 7] reveal a fact that local planning for individual agents has usually lower computational complexity than solving the global coordination problem. Therefore, at least intuitively, plan re-use techniques, which effectively simplify the coordination part of the problem, should improve the time complexity and, as the communication complexity is often tightly related to the time complexity, consequently lower the communication complexity as well.

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 overhead in comparison to replanning. The conditions correspond to the amount and frequency of minimal required coordination and the types of failures the environment generates. While the hypothesis is rather intuitive, investigation of the particular types of domains and the corresponding suitable repair algorithms deserves a deeper attention.

The contribution of the presented paper is threefold. Firstly, after introducing the general problem of multi-agent planning stemming from the formulation in [6], we give a rigorous treatment to the problem of multi-agent plan repair and formulate a notion of relative coordination frequency

of a multi-agent planning problem. In turn, this formal approach allows us to state the core hypothesis of the presented research in a more formal and precise manner. Secondly, we propose three decentralized algorithms for multi-agent plan repair reducing the problem to specialized instances of the multi-agent planning problem including proofs of their correctness. Finally, we present experimental validation confirming the core hypothesis of the paper. The paper concludes with final remarks regarding the shortcomings of our approach and future outlooks in the here described line of research.

2. Multi-agent Planning

We treat the problem of multi-agent planning as an extension of the classical single-agent planning in the manner adapted from MA-STRIPS planning in [6]. We consider a number of *cooperative* and *coordinated* actors featuring distinct sets of capabilities (actions), which concurrently plan and subsequently execute their local plans so as to 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. The following formal restatement of the MA-STRIPS problem and our adaptations thereof constitute the preliminaries enabling us to state the core hypotheses of the paper in a formal manner, as well as provide the necessary background for the algorithms and their proofs introduced later in Section 3. The formal preliminaries build upon the original algorithm for the MA-STRIPS problem proposed by Brafman and Domshlak

in [6].

A *state* $s \subseteq \mathcal{L}$ is a set of atoms from a finite set of propositions $\mathcal{L} = \{p_1, \ldots, p_m\}$. Given $p \in s$, we say that p *holds* in s, otherwise p *does not hold* in s. In that sense, states are complete, i.e., it cannot happen that there is a $p \in \mathcal{L}$, such that p's validity in s is unknown. $\mathcal{S} = 2^{\mathcal{L}} \cup \{\chi\}$ denotes the set of all states together with a distinguished state $\chi \in \mathcal{S}$ denoting an undefined state.

A *primitive action* (*action*) an agent can perform in an environment is a tuple $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$, where a is a unique action label and $\mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)$ respectively denote the sets of preconditions, add effects and delete effects of a taken from some $\mathcal{L} = \{p_1, \ldots, p_m\}$. *Act* denotes the set of all actions and we furthermore assume there is a distinguished empty action $\epsilon = \langle \emptyset, \emptyset, \emptyset \rangle \in Act$ with no preconditions and no effects. Whenever $\mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \subseteq \mathcal{L}$, we say that a is defined over $\mathcal{L}$.

We say that an action a is *applicable* in a state s iff $\mathsf{pre}(a) \subseteq s$. An application of a is defined by the state transformation operator $\oplus : \mathcal{S} \times Act \to \mathcal{S}$ so that $s \oplus a = (s \cup \mathsf{add}(a)) \setminus \mathsf{del}(a)$ iff a is applicable in s. In the case a is not applicable in s, $s \oplus a$ results in a distinguished undefined state χ. Note, we do not require that $\mathsf{add}(a) \cap \mathsf{del}(a) = \emptyset$, rather we simply assume that the effects negate each other strictly according to the definition of $\oplus$. Furthermore, $\oplus$ is left-associative, hence we can write $s \oplus a_1 \oplus \cdots \oplus a_n$.

An *agent* $\alpha = \{a_1, \ldots, a_n\}$ is characterized precisely by its capabilities, a finite repertoire of actions $a_i \in Act$ it can preform in the environment.

Definition 1 (MA-STRIPS)**.** A *multi-agent planning problem* is a tuple $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$, where

1. $\mathcal{L}$ is a finite set of atoms;
2. $\mathcal{A}$ is a set of *agents* $\alpha_1, \ldots, \alpha_n$ with actions defined over $\mathcal{L}$, featuring, besides the empty action ϵ, otherwise mutually disjoint sets of actions. I.e., $\alpha_i \cap \alpha_j = \{\epsilon\}$, whenever $i \neq j$;
3. $s_0 \in \mathcal{S}$ is an initial state; and finally
4. $S_g \subseteq \mathcal{S}$ is a set of goal states.

From now on, given a set of agents $\mathcal{A}$ as defined above, $Act = \bigcup_{i=1}^{n} \alpha_i$ denotes the set of all actions which can be performed among the agents of the team $\mathcal{A}$, the team capabilities.

The definition of a MA-STRIPS multi-agent planning problem is adapted from the original formulation in [6]. 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 $\mathbf{a} = \langle \mathsf{pre}(\mathbf{a}), \mathsf{add}(\mathbf{a}), \mathsf{del}(\mathbf{a}) \rangle$ of the team is specified by $\mathbf{a} = (a_1, \ldots, a_n)$ a tuple of actions corresponding to the individual agents, i.e., $a_i \in \alpha_i$ for each i, its preconditions $\mathsf{pre}(\mathbf{a}) = \bigcup_{i=1}^{n} \mathsf{pre}(a_i)$ and its effects $\mathsf{add}(\mathbf{a}) = \bigcup_{i=1}^{n} \mathsf{add}(a_i)$ and $\mathsf{del}(\mathbf{a}) = \bigcup_{i=1}^{n} \mathsf{del}(a_i)$. $\mathbf{a}[k]$ denotes the k-th action of $\mathbf{a}$. The notions of action applicability in a state s, as well as application of $\mathbf{a}$ to s straightforwardly extend from the definitions for primitive actions, hence we can write $s \oplus \mathbf{a}$. Note, as we are building on top of the MA-STRIPS formalism, at this point we do not rule out, nor specifically handle

joint actions in which the effects of individual agents' actions cancel out each other. In general, however, such considerations need to be tackled. Later on in this section, we comment on such joint actions in more detail.

Definition 2 (multi-agent plan)**.** Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ 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 $i, j \leq n$. Additionally, $|\mathcal{P}| = |P_k|$ for every $k \leq n$, 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
3. $\mathcal{P}$ reaches the goal S_g, i.e., $s_m \in S_g$.

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 $\mathcal{P}$, with $a_{ij} = \mathcal{P}[i, j]$, providing a more visual understanding of multi-agent plans

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

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 $\mathcal{P}[i,j] = \mathcal{P}_2[i - |\mathcal{P}_1|, j]$ for $i > |\mathcal{P}_1|$. Note, concatenation of multi-agent plans is left-, as well as right- associative operation, so we can write $\mathcal{P} = \mathcal{P}_1 \cdot \mathcal{P}_2 \cdot \cdots \cdot \mathcal{P}_n$.

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*. *diff* $(\mathcal{P}_1, \mathcal{P}_2)$ corresponds to *Levensthein distance* [8], in literature also referred to as *edit-distance*, between two strings corresponding to the sequences of actions of the individual plans. Adaptation of the notion of Levensthein distance between two multi-agent plans corresponds to the number of atomic edits, that is insertion of an empty joint action, empty joint action deletion and individual action replacement, needed to transform one plan into the other. The cost of the atomic edits is assumed to be equal. This model of plan difference is also closely related to the MODDELINS modification problem for single-agent plans described in [4].

To introduce the MA-Plan algorithm for solving MA-STRIPS problems as formulated in [6], we finally need to distinguish between the public and private actions of 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. The private actions are those, which are not affected by actions of the other agents.

Let $\mathsf{atoms}(a) = \mathsf{pre}(a) \cup \mathsf{add}(a) \cup \mathsf{del}(a)$ and similarly $\mathsf{atoms}(\alpha)$ be the sets of atoms required or affected by the action a and the agent α respectively. Given a multi-agent team $\mathcal{A} = \alpha_1, \ldots, \alpha_n$ with actions defined over the set of atoms $\mathcal{L}$, the set of *public* actions is defined as $Act^{pub} = \{a \mid a \in \alpha_\mathrm{i} \text{ and } \mathsf{atoms}(a) \subseteq \mathcal{L} \setminus \mathsf{atoms}(\alpha_i)\}$. Consequently, the set of private actions is defined as $Act^{priv} = Act \setminus Act^{pub}$.

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. The algorithm MA-Plan for solving a planning problem Π can be thought of in two interleaving stages until a suitable multi-agent plan is found: i) computation of a plan consisting exclusively of suitable coordination points of the agent team, and subsequently ii) computation of sequences of private actions filling the gaps between the public actions of each individual agent. While the second stage can be computed in a local manner by each individual agent without interactions with its peers, a truly decentralized multi-agent algorithm for the first stage requires a non-trivial

amount of interaction between the agents.

One of the main contributions of the Brafman and Domshlak's paper [6] lies in the observation that the MA-Plan algorithm can be implemented by reduction of the first stage to a constraint satisfaction problem (CSP) (cf. e.g., [9]). In the CSP, each agent is represented by a single variable ranging over possible plans of the individual agent and two types of constraints:

coordination constraint: a sequence of joint actions $\mathcal{P}$ (candidate multi-agent plan) corresponding to a multi-agent planning problem $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ satisfies the *coordination constraint* iff for every action $a = \mathcal{P}[k, i]$ performed by the agent α_i in the step k we have, that if a is a public action, then

- for every $p \in \mathsf{pre}(a)$, there must exist $a_p = \mathcal{P}[k_p, i_p]$, such that $p \in \mathsf{add}(a_p)$ and $0 < k_p < k$ (there is some previous action which causes p to hold), or $p \in s_0$ in which case we set $k_p = 1$; and
- for no k', s.t., $k_p \leq k' \leq k$ there exists $a' = \mathcal{P}[k', i']$, such that $p \in \mathsf{del}(a')$ (p won't be invalidated between causing it in the step k_p and execution of a in the step k).

The constraint ensures that the dependencies of all the public actions occurring in the overall multi-agent plan are satisfied, possibly by actions performed in advance by other team members.

internal planning constraint: a sequence of joint actions $\mathcal{P}$ corresponding to a multi-agent planning problem $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ 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

Algorithm 1 MA-Plan(Π):

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

Output: A multi-agent plan $\mathcal{P}$ solving Π, if such exists.

```
δ = 1
loop
  construct CSP_{Π;A}
  if solve-csp(CSP_{Π;δ}) then
    reconstruct a plan P from a solution for CSP_{Π;δ}
    return P
  else
    δ = δ + 1
  end if
end loop
```

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. The constraints ensure that each individual plan is locally executable by the particular agent.

Note, the formulation of the coordination constraint renders joint actions with $\mathsf{add}(\mathbf{a}) \cap \mathsf{del}(\mathbf{a}) \neq \emptyset$ invalid. It is the non-strict inequalities in the condition 2 of the coordination constraint, together with the definition of public actions, which ensure the local consistency of joint actions.

Algorithm 1 lists the original multi-agent planning algorithm MA-Plan by Brafman and Domshlak in [6]. 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 of joint actions in a candidate multi-agent plan containing 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, possibly not terminating in the case where no solution to the given multi-agent planning problem exists.

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.

In [10], Nissim and Brafman adapted the original blueprint algorithm from [6] 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 state-of-the-art DisCSP solver for solving it, plus managing the overhead involved in the resulting distributed algorithm. From now on, whenever we speak about the implementation of the multi-agent planning algorithm, we refer to its decentralized version as described in [10].

As mentioned in previous sections, the multi-agent planner from Nissim and Brafman [10] is built on a DisCSP solver and a centralized single-agent heuristic search planner. The algorithm used as the DisCSP solver is a customized Asynchronous Backtracking (ABT) solver [11] with Asynchronous Forward Checking [12] heuristics. In the planner, the best-first-search algo-

rithm is employed with helpful action and landmark heuristics. The planner is a part of the FastForward planning suite [13]. The planning process passes four separate phases: i) centralized preparation of the DisCSP instance, ii) initialization of the solving process for the DisCSP in a special agent representing the goal requirements, iii) decentralized solving of the DisCSP problem and iv) decentralized finalization of the DisCSP solving process.

In the final decentralized phase the agents already know their local plans, given a multi-agent solution exists, and execute them in a distributed manner.

3. Multi-agent Plan Repair

Consider a multi-agent planning problem $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ 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 failure-free and 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 dynamic environments 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 the action $\mathbf{a} = \mathcal{P}[i]$ does not satisfy some of the postconditions of $\mathbf{a}$, i.e., $\mathsf{add}(\mathbf{a}) \not\subseteq s_f$.

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. In practice, it makes the most sense to monitor for strong failures in system's evolution. Most weak failures either lead to a strong failure later on in the plan execution, or were irrelevant. Of course, except for the case when a weak failure leads to a future failure to reach a goal state, i.e., when some atom supposed to be included in a goal state fails to be effected by an action in the plan. There also 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. Thus, monitoring for weak, strong, or even other types of plan execution failures

can strongly depend on the target application. 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 $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ 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 $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$. We say that $\mathcal{P}'$ *repairs* $\mathcal{P}$ in s_f. In the case $\mathit{Plans}(\Pi') = \emptyset$, 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 $\mathit{diff}(\mathcal{P}_1, \mathcal{P}) \leq \mathit{diff}(\mathcal{P}_2, \mathcal{P})$ 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 $\mathcal{P}_{\min} \in \mathit{Plans}(\Pi')$, which is minimal w.r.t. the mutual differences between the plans solving Π'. That is,

$$\mathcal{P}_{\min} \in \underset{\mathcal{P}' \in \mathit{Plans}(\Pi')}{\arg\min} \mathit{diff}(\mathcal{P}, \mathcal{P}')$$

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: i) construct

the multi-agent replanning problem Π' as prescribed in Definition 3, and subsequently ii) utilize the MA-Plan algorithm to solve the problem Π'.

While the notion of minimal repair of multi-agent plans is based on the number of changes the repaired plan contains w.r.t. the original plan, also other metrics selecting distinguished plan repairs could be considered. We will discuss examples of such later in the paper.

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 need to provide an indication of which planning problems tend to benefit from the plan repairing approach. The following notion of *coordination frequency* formalizes the idea.

Definition 4 (coordination frequency)**.** Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be a multi-agent planning problem with a solution $\mathcal{P}$. We say that $\mathcal{P}$ is δ-*coordinated* iff it contains at δ coordination points, i.e., joint actions including at least one public actions of some individual agents. In the case $\delta = 0$, that is $\mathcal{P}$ does not contain any public action, we say that $\mathcal{P}$ is *uncoordinated.*

Relative coordination frequency $cf(\mathcal{P})$ of a δ-coordinated plan $\mathcal{P}$ denotes the frequency of coordination point occurrence per single step in the plan and is defined as

$$cf(\mathcal{P}) = \frac{\delta}{|\mathcal{P}|}$$

Relative coordination frequency $cf(\Pi)$ of a multi-agent planning problem Π denotes the minimal coordination frequency required to solve Π and is defined as

$$cf(\Pi) = \min_{\mathcal{P} \in Plans(\Pi)} cf(\mathcal{P})$$

The notion of relative coordination frequency of plans relates to the fractional amount of coordination corresponding to a single step in a plan execution. It straightforwardly extends to planning problems viewed as sets of plans solving them. We simply look for solutions requiring minimal relative amount of coordination required to solve the problem. The notion of relative coordination frequency allows for comparison and ordering of multi-agent planning problems according to the amount of coordination they minimally require for solving them. Informally, we will call problems with relatively low $cf(\Pi)$ *loosely coordinated* and those with $cf(\Pi)$ closer to 1 *tightly coordinated*. Note that a problem with $cf(\Pi) = 0.5$ is still tightly coordinated, as for each coordination step, there is only one uncoordinated step. Multi-agent planning problems with $cf(\Pi) = 0$ will be called *uncoordinated*.

Note, it still might be the case that even though a multi-agent planning problem can be solved without any coordination, i.e., $cf(\Pi) = 0$, there still can exist coordinated plans in $Plans(\Pi)$, which are more efficient, e.g., shorter than the uncoordinated ones. For instance, consider a domain where the objective is that an agent A reaches a destination d. The agent A could move from its starting position to d on its own, albeit slowly and resulting in a relatively long plan. Alternatively, A could be transported quickly to d by another agent B. The latter plan would be shorter in terms of overall number of steps, but would require coordination. In result, repair of such

a plan would be costlier in terms of communication overhead it incurs than the uncoordinated one, our main concern here.

The core hypothesis of the paper can be now stated more formally.

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 should feature a* lower average communication complexity *than replanning.*

As shown in [6], δ turns out to play an important role in time-complexity analysis of the MA-STRIPS problem (cf. also Algorithm 1). Above, we hypothesize that it is the relative frequency of coordination points along the plans, which turns out to play a role in the communication complexity of plan repair. Plan repair for problems which require some coordination quite often along the plans should lead to re-use of fragments including relatively large number of coordination points, which do not have to be planned for again and thus leads to reduction of required communication in the repair process.

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.

In the following subsections, we describe three plan repairing algorithms. Sketches of ideas behind these algorithms were initially proposed in [14], in the following we provide novel descriptions and formalizations of the algorithms in full details.

3.1. Back-on-track Repair

Unexpected event occurring in an environment can cause a failure in execution of a plan performed by a multi-agent team in that environment. The result would be that the overall state of the system won't be the one expected by an 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 $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ being the corresponding modified multi-agent replanning problem.

We say that a plan $\mathcal{P}' \in \mathit{Plans}(\Pi')$ 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

Algorithm 2 Back-on-Track-Repair(Σ)

Input: A multi-agent plan repair problem $\Sigma = (\Pi, \mathcal{P}, s_f, k)$, with $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ and a sequence of states $s_0, \ldots, s_m$, a failure-free execution of $\mathcal{P}$ would generate.

Output: A multi-agent plan $\mathcal{P}'$ solving Σ, if a solution exists.

construct $\Pi_{back} = (\mathcal{L}, \mathcal{A}, s_f, \{s_0, \ldots, s_m\} \cup S_g)$
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 $\mathcal{P}' = \mathcal{P}_{back} \cdot \mathcal{P}[j \ldots \infty]$
else
  return $\mathcal{P}' = \chi$
end if

original plan, 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, 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 ordering of the plans according to the quality of repair. 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. The length of the overall repaired plan, however, depends also on the selection of a particular goal state $s_g \in \{s_0, \ldots, s_m\} \cup S_g$ of the planning problem Π_{back}. If the planning algorithm selects s_g according to an ordering from s_m to s_0 and later on the remaining states from S_g for the same lengths of possible $\mathcal{P}_{back}$ plans, the overall repaired resulting plan would also be the shortest, under a condition the result is a proper back-on-track repair.

The algorithm rests on invocation of the underlying multi-agent planner, hence its correctness relies on the correctness of the underlying planner. The following lemma states the soundness of Algorithm 2.

Lemma 6 (Back-on-Track-Repair soundness)**.** *Let* $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ *be a multi-agent planning problem with agents situated in a dynamic environment in which the environment can interfere with the plan execution and let* $\mathcal{P}$ *be a solution to* Π*. Let also* s_f *be a state resulting from an interference of the environment, a plan failure, at a step* k *of execution of the plan* $\mathcal{P}$*.* $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ *denotes the corresponding multi-agent plan repair problem.*

Unless the execution of Back-on-Track-Repair(Σ) *finishes with the undefined plan* χ*, a failure-free execution of the resulting plan* $\mathcal{P}'$ *leads to some goal state of the original multi-agent planning problem* Π*.*

Proof. Follows straightforwardly from the construction of Π_{back} and that $\mathcal{P}$ is a solution to Π. Either $\mathcal{P}_{back}$ leads to some state along the ideal execution trace of the original plan $\mathcal{P}$ and then the remainder of $\mathcal{P}$ leading to the final state $s_m \in S_g$ is reused, or a failure-free execution of $\mathcal{P}_{back}$ would lead directly to some final state $s_{end} \in S_g$ without reusing a part of $\mathcal{P}$. □

Furthermore, upon a failure of a plan execution, if there exists a plan from

the failed state to a final state of the original multi-agent planning problem, the back-on-track algorithm is able to find a solution to the corresponding multi-agent plan repair problem.

Lemma 7 (Back-on-Track-Repair completeness)**.** *Let* $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$*,* $\mathcal{P}$*,* s_f*,* k *and consequently* Σ *be as assumed in Lemma 6.*

If there exists a solution to the modified multi-agent planning problem $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$*, then the execution of* Back-on-Track-Repair(Σ) *algorithm finishes and finds* $\mathcal{P}' \neq \chi$*, a solution repair of* $\mathcal{P}$*.*

Proof. Again, follows straightforwardly from construction of Π_{back} in the algorithm. Observe that if there is a solution plan to the problem $\Pi = (\mathcal{L}, \mathcal{A}, s_f, S_g)$, then there also must exist at least the same solution to the modified planning problem $\Pi_{back} = (\mathcal{L}, \mathcal{A}, s_f, \{s_0, \ldots, s_m\} \cup S_g)$. That is, in the worst case, the back-on-track approach resorts to re-planning from scratch. □

The lemmas 6 and 7 establish how the back-on-track plan repair approach inherits its correctness from the underlying multi-agent planner. Note however, the algorithm is only *partially complete*, because in cases when there is no solution to a given multi-agent planning problem, it is not ensured that the algorithm MA-Plan terminates. Provided a totally complete multi-agent planning algorithm, directly replacing MA-Plan, total completeness of the Back-on-Track-Repair algorithm could be straightforwardly established by the lemmas above.

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

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

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

Algorithm 3 Lazy-Repair(Σ)

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

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

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

simulate execution of $\mathcal{P}_{[k..\infty]}$ from s_f on, resulting in a final state s_{lazy}

construct $\Pi_{lazy} = (\mathcal{L}, \mathcal{A}, s_{lazy}, S_g)$

$\mathcal{P}_{lazy} =$ MA-Plan(Π_{lazy})

return $\mathcal{P}_{[k..\infty]} \cdot \mathcal{P}_{lazy}$, **unless** $\mathcal{P}_{lazy} = \chi$ in which case **return** χ

Similarly to the back-on-track algorithm, Algorithm 3 inherits its correctness from the underlying multi-agent planner invoked internally.

Lemma 9 (Lazy-Repair soundness)**.** *Let* $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$*,* $\mathcal{P}$*,* s_f*,* k *and* Σ *be as assumed in the Lemma 6.*

Unless the execution of Lazy-Repair(Σ) *finishes with the undefined plan* χ*, a failure-free execution of the resulting plan* $\mathcal{P}'$ *leads to some goal state of the original multi-agent planning problem* Π*.*

Proof. In whichever state s_{lazy} a failure-free execution of the executable remainder of $\mathcal{P}$ ends up, if existing, the solution plan to the problem Π_{lazy} will take the system from there to some final state corresponding to the original multi-agent planning problem Π. The executable remainder of $\mathcal{P}$ from the state in which the failure occurred will get reused in the resulting plan. □

Unlike the back-on-track algorithm, the lazy approach is in general 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 a goal state exists. The notion of the algorithm completeness has to be weakened to domains in which the agent team is at least capable to revert its own actions.

Definition 10 (connected multi-agent planning domain)**.** Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be a multi-agent planning problem. Let also $Act = \alpha_1 \times \cdots \times \alpha_n$, with $\alpha_1, \ldots, \alpha_n \in \mathcal{A}$, and $\mathcal{S} = 2^{\mathcal{L}}$. We say that the planning problem induces a *connected planning domain* iff for every state $s \in \mathcal{S}$ and a joint action $\mathbf{a} \in Act$, there exists a solution to the multi-agent planning problem $\Pi' = (\mathcal{L}, \mathcal{A}, s \oplus \mathbf{a}, s)$, i.e, a plan $\mathcal{P} = \mathbf{a}_1, \ldots, \mathbf{a}_k$, such that $s = s \oplus \mathbf{a} \oplus \mathbf{a}_1 \oplus \cdots \oplus \mathbf{a}_k$.

In essence, the definition of connected multi-agent planning domain states that it is in the scope of capabilities of the multi-agent team $\mathcal{A}$ to "undo", or "revert", effects of any of its own actions. Note, a single-agent version of the definition (with an omnipotent agent $\overline{\alpha} = \bigcup_{\alpha_i \in \mathcal{A}} \alpha_i$) would also suffice, since we require that $\epsilon \in \alpha$ for every $\alpha \in \mathcal{A}$ and in a consequence any joint action of the team can be transformed into a corresponding multi-agent plan of length n with only a single agent acting in any given step of the plan.

The following lemma states that the Lazy-Repair algorithm is complete in connected planning domains.

Lemma 11 (Lazy-Repair completeness)**.** *Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ inducing a connected multi-agent planning domain and we assume $\mathcal{P}$, s_f, k, as well as Σ are as in the Lemma 6. Let also s_{lazy} correspond to the state to which a failure-free execution of an executable remainder $\mathcal{P}_{[k..\infty]}$ of $\mathcal{P}[k..\infty]$ would lead.*

If there exists a solution plan $\mathcal{P}'$ to the multi-agent planning problem

$\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$, *then the execution of* Lazy-Repair(Σ) *algorithm finishes and finds a plan* $\mathcal{P}^* \neq \chi$ *, a solution repair of* $\mathcal{P}$*.*

Proof. Let $\mathcal{P}_{[k..\infty]} = \mathbf{a}_{k+1}, \ldots, \mathbf{a}_m$ be the executable remainder of $\mathcal{P}[k..\infty]$ and let $s_{k+1}, \ldots, s_m$ be the states resulting from a failure-free execution of $\mathcal{P}_{[k..\infty]}$, i.e., $s_{j+1} = s_j \oplus \mathbf{a}_j$ for $k+1 < j < m$. Since the agent team acts in a connected planning domain, any of its actions is reversible, that is, its effects can be undone. Therefore for execution of each action $\mathbf{a}_j$ above, there must exist a sequence of plans $\mathcal{P}^{\leftarrow}_{\mathbf{a}_j}$, each being a solution to the planning problem $\Pi^{\leftarrow}_{\mathbf{a}_j} = (\mathcal{L}, \mathcal{A}, s_{j+1}, s_j)$. Since we assume that there exists a plan $\mathcal{P}'$ solution to the problem $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$, the plan $\mathcal{P}^* = \mathcal{P}_{[k..\infty]} \cdot \mathcal{P}^{\leftarrow}_{\mathbf{a}_{m-1}} \cdot \cdots \cdot \mathcal{P}^{\leftarrow}_{\mathbf{a}_{k+1}} \cdot \mathcal{P}'$ is a solution for the plan repairing problem $\Sigma = (\Pi, \mathcal{P}, s_f, k)$. That is, the solution plan first executes $\mathcal{P}_{[k..\infty]}$, the executable remainder of the original plan $\mathcal{P}$ from the point of failure (as defined by the algorithm), then "undoes/reverts" effects of all the performed actions in $\mathcal{P}_{[k..\infty]}$ and thus returns to the state s_f, and finally executes the plan $\mathcal{P}'$, existence of which we assume. □

The corollary of the line of reasoning leading to the proof of completeness of the lazy repair approach is that despite non-existence of irreversible environment interferences in some domains, it is the agent team whose actions can break the system evolution beyond repair. For illustration, even though it is not in the ability of the physical environment to push a robot over a cliff, it is indeed in its own powers to jump from it during execution of an executable remainder of some, otherwise harmless plan, which failed shortly before. In such domains, the lazy approach has to be employed with caution.

To conclude, similarly to the back-on-track approach, Lemma 11 states

only partial completeness of the Lazy-Repair algorithm the underlying multi-agent planner does not ensure termination.

3.3. Repeated Lazy Repair

In a 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 an already repaired plan, it is not always necessary to prolong the overall multi-agent plan. In the case a second failure occurs while still executing the plan fragment from the original plan preserved by the first repair, the suffix appended by the first repair can be discarded and replaced by a new plan suffix repairing the second failure, should it be necessary.

The following definition formally introduces *repeated lazy* (RLazy) *plan repair*, an extension of the lazy multi-agent plan repair approach introduced in Definition 8. For clarity, from now on, we refer to the lazy multi-agent plan repair introduced in the previous subsection as *simple lazy repair*.

Definition 12 (repeated lazy repair)**.** Let $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ be a multi-agent plan repairing problem. Let also $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be the corresponding multi-agent planning problem with a solution of the form $\mathcal{P} = \mathcal{P}' \cdot \mathcal{P}_{fix}$. In the case this is the first failure encountered during execution of $\mathcal{P}$, we have

$|\mathcal{P}_{fix}| = 0$ and thus $\mathcal{P} = \mathcal{P}'$. Otherwise, $\mathcal{P}$ is a simple lazy repair solution of some (previously solved) plain repair problem $\Sigma_p = (\Pi, \mathcal{P}_p, s_{f_p}, k_p)$ composed of an executable remainder of $\mathcal{P}_p$ (represented as $\mathcal{P}'$) and a repair suffix $\mathcal{P}_{fix}$.

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

1. $\mathcal{P}''$ is a simple lazy repair solution to $\Sigma' = (\Pi, \mathcal{P}', s_{f_p}, k)$ in the case $k \leq |\mathcal{P}'[k_p..\infty]|$ (the failure occurred still within the executable remainder of $\mathcal{P}_p[k_p..\infty]$); or otherwise
2. $\mathcal{P}''$ is a simple lazy repair solution to $\Sigma' = (\Pi, \mathcal{P}, s_{f_p}, k)$.

The repeated lazy repair leads to a straightforward extension of the simple lazy plan repair algorithm listed in Algorithm 3. The intuitive benefit of the straightforward application of the repeated lazy repair approach is that it should lead to shorter executed plans than would result from usage of the simple lazy repair. Consider a plan execution failure at step k_1 of a plan $\mathcal{P}$. Simple lazy repair approach would fix it by appending a suffix $\mathcal{P}_1$ resulting in the plan $\mathcal{P}_{[k_1..\infty]} \cdot \mathcal{P}_1$. Simple lazy repair of a second failure at a step k_2 occurring still somewhere in the fragment $\mathcal{P}_{[k_1..\infty]}$ would result in a solution $\mathcal{P}_{[k_2..\infty]} \cdot \mathcal{P}_1 \cdot \mathcal{P}_2$ with a suffix $\mathcal{P}_2$, the solution to the second plan repair problem. Unlike that, upon occurrence of the second failure the repeated lazy repair discards the previously computed suffix $\mathcal{P}_1$ and replaces it with a new suffix $\mathcal{P}'_2$, resulting in a repair solution $\mathcal{P}_{[k_2..\infty]} \cdot \mathcal{P}'_2$. The idea is that in many domains $\mathcal{P}'_2$ should be shorter than the length of the combined suffix $\mathcal{P}_1 \cdot \mathcal{P}_2$. This could be especially beneficial in domains in which subsequent failures can even revert, or otherwise fix the ones occurring previously.

As with the previous two plan repairing approaches, we conclude the discourse with proofs of repeated lazy repair approach correctness.

Algorithm 4 Repeated-Lazy-Repair(Σ)

Input: A multi-agent plan repairing problem $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ with $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ and its solution $\mathcal{P}$. In the case $\mathcal{P}$ is a lazy repair solution of a (previously solved) plain repair problem $\Sigma_p = (\Pi, \mathcal{P}_p, s_{f_p}, k_p)$, it takes the form $\mathcal{P} = \mathcal{P}' \cdot \mathcal{P}_{\mathit{fix}}$. Otherwise, in the case this is the first failure encountered, $|\mathcal{P}_{\mathit{fix}}| = 0$.

Output: A multi-agent plan solving $\Sigma = (\Pi, \mathcal{P}, s_f, k)$.

if $k \leq |\mathcal{P}'_{[k_p..\infty]}|$ **then**
  return Lazy-Repair($(\Pi, \mathcal{P}', s_f, k)$)
else
  return Lazy-Repair($(\Pi, \mathcal{P}, s_f, k)$)
end if

Lemma 13 (Repeated-Lazy-Repair soundness)**.** *Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$, $\mathcal{P}$, s_f, k and Σ be as assumed in the Lemma 6.*

Unless the execution of Repeated-Lazy-Repair(Σ) *finishes with the undefined plan χ, a failure-free execution of the resulting plan $\mathcal{P}'$ leads to some goal state of the original multi-agent planning problem Π.*

Proof. Follows immediately from the soundness of the simple lazy repair approach in Lemma 9. □

Lemma 14 (Repeated-Lazy-Repair completeness)**.** *Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ inducing a connected multi-agent planning domain and we assume $\mathcal{P}$, s_f, k, as well as Σ are as in the Lemma 6.*

If there exists a solution plan to the multi-agent planning problem $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$, then the execution of Repeated-Lazy-Repair(Σ) *algorithm fin-*

ishes and finds a plan $\mathcal{P}' \neq \chi$, a solution repair of $\mathcal{P}$.

Proof. Again, follows straightforwardly from the proof of completeness of the simple lazy repair algorithm. Note, the proof of Lemma 11 is independent of how exactly does the final state to which the executable remainder of the original plan leads to looks like, it can be arbitrary. Therefore, when we arbitrarily modify the executable remainder of the original plan, as in Algorithm 4, the proof still holds. That is, if there exists a plan $\mathcal{P}''$ from s_f to some state in S_g, then in connected domains, there must exist at least the plan firstly executing the executable remainder of the original plan, subsequently a plan reverting its effects back to s_f and than finally performing the steps of $\mathcal{P}''$. □

4. Multi-agent Plan Repairing Process

The multi-agent planning, executing, monitoring and repairing process has two phases. In the first phase, for a given domain a multi-agent plan is constructed using the MA-Plan algorithm. In the second phase, the plan is executed by the agents acting in a shared environment. In the course of the execution, the dynamics of the environment can interfere, possibly resulting in a failure of the executed plan. Since the plan execution is monitored by the multi-agent team, or a centralized observer, upon a failure detection a plan repair algorithm is invoked. In turn, to find particular repairing plans, the MA-Plan algorithm is invoked as specified by the plan repairing algorithms introduced in the previous section.

Scheme listed in Algorithm 5 shows the pseudo-code of the process. Since we assume complete information there is no difference between a decentral-

ized and a centralized monitoring, hence for clarity, the algorithm instantiates the centralized version. As a consequence of the information completeness assumption, also the execution of the centralized initialization of the MA-Plan algorithm does not negatively affect the amount of communication in the system.

Before execution of each plan step, the algorithm checks whether a failure occurred and if so, invokes a plan repair algorithm. We do not explicitly articulate what a failure amounts to, since this can be application specific. Plausible options include checking for weak, or strong failures, i.e, validity of effects of the previously executed action, or validity of preconditions of the action to be executed next. Alternatively, in some applications it might be useful to check for any exogenous change of the current state not caused by the involved agents.

Finally, the algorithm accounts for the possibility that the plan repairing process can result in finding no solution to the failure. If that is the case, the algorithm finishes with the final plan equal to the undefined plan χ. Note however, that Algorithm 5 does not necessarily terminate. Termination of of the scheme relies on two factors. Firstly, it is the termination property of the underlying multi-agent planner invoked by the plan repair algorithms discussed in Section 3. Secondly, unless no repair to the occurred failure can be found, the algorithm terminates when it is capable to fully execute the computed plan. In environments where failures can occur relatively frequently, it can however happen that the plan execution, monitoring and repair process would continually repair recurring failures sooner than the previous repair was fully executed. In a consequence, this would lead to a gradual prolonga-

Algorithm 5 Plan execution and monitoring scheme.

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

$\mathcal{P} = \mathsf{MA\text{-}Plan}(\Pi)$
if $\mathcal{P} = \chi$ **then return** fail
$k = 1$

repeat
  agents perform $\mathcal{P}[k]$
  if failure detected **then**
    retrieve the current state s from the environment
    $\mathcal{P} = \mathsf{Repair}((\Pi, \mathcal{P}, s, k))$
    $k = 1$
  else
    $k = k + 1$
  end if
until $\mathcal{P} = \chi$ or $k > |\mathcal{P}|$

tion of the executed plan so that it will never reach the end of its execution. Informally, for such domains, we could state that the Algorithm 5 terminates when the plan repair process generates sharply shorter repaired plans than is the time horizon in which the failures in the environment tend to occur. Results in [15] discuss steps towards a formal analysis of such a planning horizon and classification of various planning domains with respect to the frequency of failures occurring in an environment and the likelihood that an agent completes its plans without an interruption.

Instantiation of the execution, monitoring and repair scheme with the repeated lazy repair algorithm allows for an alternative plan execution model. The planning process invocation in the repair algorithm could be delayed until the execution of the preserved fragment of the original plan finishes. Such an approach could preserve significantly longer fragments of the original plan than instantiation of the original scheme in Algorithm 5 with the Repeated-Lazy-Repair algorithm. 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 invoked from the listed plan execution scheme would do, the system can simply proceed with execution of the remainder of the original plan and only after it finishes, 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. In some domains, such an approach could significantly decrease the number of multi-agent planner invocations and in a consequence save a large amount of communication overhead.

5. Experimental Validation

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

5.1. Experimental Setup

The experiments were based on a presented two-stage plan repairing algorithm, where we distinguish two types of plan failures: *action failures* and *state perturbations*. Both failure types are parametrized by an uniformly distributed probability P, which determines whether a simulation step fails, or not (a failure is generated only if there exists a plan to a goal, which obviate problems with irreversible actions). Both failure types are weak failures. That is, they are not handled immediately, but can preclude 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-execution of some of the individual agent actions from the actual plan step. The individual action is chosen according to a uniform probability distribution over the positions within a joint action. The individual failed action is then 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.

We implemented the experimental setup as a centralized simulator of the environment integrating the multi-agent domain-independent planner MA-Plan. The individual agents are initialized by the planner initialization process, together with a given planning problem instance. Each agent runs in its own thread and they deliberate asynchronously. The agents send peer-to-peer

messages between themselves via a centralized simulator as well. The messages are sent by the integrated MA-Plan planner exclusively in the DisCSP phase.

The experiments were performed on *FX-8150 8-core* processor at 3.6GHz 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 10 times with various value samples. The resulting data are, in the figures, presented with the natural distribution. The candlestick charts depict the differences between the minimal and the maximal measurements, together with the standard deviation. The accompanying charts represent the percentage ratio between the measured variable for the particular repairing method and replanning from scratch (normalized at the 100% level). Since we are using the planner MA-Plan as a black-box algorithm, the relative proportion to the replanning approach bear a higher significance than the particular absolute numbers. The values presented in the result table are average values from the measurements of the same parametrization.

5.2. Test Problems, Algorithms and Metrics

The experiments were conducted on four planning domains. Three of the domains originate in the standard single-agent IPC planning benchmarks [16]. Similarly to the evaluation of the MA-Plan implementation in [10], we chose domains, which are straightforwardly modifiable to the multi-agent setting: LOGISTICS (2–6 agents), ROVERS (2–4 agents), and SATELLITES (2–6 agents). Additionally, we have extended the set of IPC-based domains by a well known coordination domain COOPERATIVE PATHFINDING (2–4 agents).

Instances of LOGISTICS problems are about transporting packages between locations by a fleet of heterogeneous transport vehicles. A representative example of a LOGISTICS problem Π^{log3}—used as one of the experiments—contains three agents controlling two trucks $\mathsf{T1}$ and $\mathsf{T2}$ and one airplane A. There are two cities, each with one storage depot ($\mathsf{d1}$ and $\mathsf{d2}$) and one airport ($\mathsf{a1}$ and $\mathsf{a2}$). The trucks can move $\mathsf{m}(\mathsf{from}, \mathsf{to})$ only within their cities, i.e., between one depot and one airport. The airplane can fly $\mathsf{f}(\mathsf{from}, \mathsf{to})$ among all airports in the environment, but cannot land at the depots. All vehicles can load $\mathsf{l}(\mathsf{package}, \mathsf{location})$ and unload $\mathsf{u}(\mathsf{package}, \mathsf{location})$ a package at a location. Initially, there is one package p at one of the depots and the goal is to transport it to the other depot in the other city. The trucks start at the depots and the airplane starts at one of the airports. A multi-agent plan solving this particular instance is $\mathcal{P}^{log3} =$

$$\begin{array}{c} \mathsf{A}: \\ \mathsf{T1}: \\ \mathsf{T2}: \end{array} \begin{pmatrix} \epsilon & \epsilon & \epsilon & \overline{\mathsf{l}(\mathsf{p},\mathsf{a1})} & \mathsf{f}(\mathsf{a1},\mathsf{a2}) & \overline{\mathsf{u}(\mathsf{p},\mathsf{a2})} & \epsilon & \epsilon & \epsilon \\ \mathsf{l}(\mathsf{p},\mathsf{d1}) & \mathsf{m}(\mathsf{d1},\mathsf{a1}) & \overline{\mathsf{u}(\mathsf{p},\mathsf{a1})} & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \mathsf{m}(\mathsf{d2},\mathsf{a2}) & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \overline{\mathsf{l}(\mathsf{p},\mathsf{a2})} & \mathsf{m}(\mathsf{a2},\mathsf{d2}) & \mathsf{u}(\mathsf{p},\mathsf{d2}) \end{pmatrix},$$

the coordination frequency for such problem is $cf(\Pi^{log3}) = \frac{4}{9} = 0.\bar{4}$, because $\delta = 4$ and length of the plan $|\mathcal{P}^{log3}| = 9$ and the presented plan $\mathcal{P}^{log3}$ is minimal from the perspective of the coordination points. For the context of the experiments, the LOGISTICS domain is 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 parallel version of the LOGISTICS domain (PAR) for 5 and 6 agents involves two parallel logistics sub-problems.

Problems of the ROVERS domain describe space exploration missions carried out by autonomous rovers equipped for three types of tasks: soil analysis

s, rock analysis r and imaging i. The resulting data from the tasks has to be communicated $\mathsf{c}(\mathsf{s},\mathsf{r},\mathsf{i})$ back to the Earth in one data package over a communication channel available only for one of the rovers at a time. The data can be communicated only if they are prepared $\mathsf{p}(\mathsf{s}/\mathsf{r}/\mathsf{i})$. The rock and soil analysis can be executed provided that the rover is at suitable position and has empty analytical store. The store can be emptied, if required. The rovers can move among predefined waypoints with a known information about the samples. Images can be taken only from suitable positions and with a camera calibrated and in a correct mode. An example problem Π^{rov3} used as one of the experiments has three fully equipped rovers R1, R2 and R3. A solution $\mathcal{P}^{rov3} =$

$$\begin{array}{l} \mathsf{R1}: \\ \mathsf{R2}: \\ \mathsf{R3}: \end{array} \left(\begin{array}{ccccccccc} \underbrace{\begin{array}{ccccccc} \cdots & \mathsf{p(r1)} & \cdots & \mathsf{p(i1)} & \cdots & \mathsf{p(s1)} \\ \cdots & \mathsf{p(r2)} & \cdots & \mathsf{p(i2)} & \cdots & \mathsf{p(s2)} \\ \cdots & \mathsf{p(r3)} & \cdots & \mathsf{p(i3)} & \cdots & \mathsf{p(s3)} \end{array}}_{\text{10 private actions}} & & \epsilon & & \overline{\mathsf{c(s1,r1,i1)}} & & \epsilon \\ & & \epsilon & & \epsilon & & \overline{\mathsf{c(s2,r2,i2)}} \\ & & \overline{\mathsf{c(s3,r3,i3)}} & & \epsilon & & \epsilon \end{array} \right)$$

has coordination frequency $cf(\Pi^{rov3}) = \frac{3}{13} \doteq 0.23$ following the same procedure as presented in previous paragraph with LOGISTICS. Therefore, for the context of the experiments, the ROVERS domain is loosely coordinated in that it requires coordination only at the end of plans.

The SATELLITES domains describe planning for a set of independent satellites providing various types of deep space imagery i from the orbit. Each imaging instrument on board of a satellite has to be firstly turned to point at one of predefined target directions. Secondly, each imaging instrument has to be powered, switched on and calibrated before it can take an image $\mathsf{t}(\mathsf{i})$ in one of predefined modes. A solution of one of the experimental instance Π^{sat3} using three satellites S1, S2 and S3 is $\mathcal{P}^{sat3} =$

$$\begin{array}{l} \mathsf{S1}: \\ \mathsf{S2}: \\ \mathsf{S3}: \end{array} \underbrace{\begin{pmatrix} \cdots & \mathsf{i(i1)} \\ \cdots & \mathsf{i(i2)} \\ \cdots & \mathsf{i(i3)} \end{pmatrix}}_{\text{3 private actions}}.$$

In this case the coordination frequency $cf(\Pi^{sat3}) = \frac{0}{3} = 0$, as there is no public action in an optimal plan. Therefore the domain is uncoordinated in that it does not need any coordination between the satellites acquiring images individually.

Finally, in the COOPERATIVE PATHFINDING domain, a team of robots move on a 3×3 grid (positions x1y1 to x3y3), where only a single robot can occupy one cell. The goal for the robots is to move $\mathsf{m}(\mathsf{from}, \mathsf{to})$ to initial positions occupied by the other robots in the initial state. A representative problem Π^{cp3} contains three robots R1, R2 and R3 and a solution plan $\mathcal{P}^{cp3}$ is

$$\begin{array}{l} \mathsf{R1}: \\ \mathsf{R2}: \\ \mathsf{R3}: \end{array} \begin{pmatrix} \overline{\mathsf{m(x1y2, x1y1)}} & \overline{\mathsf{m(x1y1, x2y1)}} \\ \overline{\mathsf{m(x2y1, x3y1)}} & \overline{\mathsf{m(x3y1, x3y2)}} \\ \overline{\mathsf{m(x3y2, x2y2)}} & \overline{\mathsf{m(x2y2, x1y2)}} \end{pmatrix},$$

consequently the coordination frequency $cf(\Pi^{cp3}) = \frac{2}{2} = 1$ as each action in an optimal plan is public and therefore the domain represent a fully coordinated problems.

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 (BoT repair, Algorithm 2) and Repeated-Lazy-Repair (RLazy

repair, Algorithm 4) introduced in the previous section.

Efficiency problems of the original MA-Plan implementation (in [10]) limited the experiments to plans with maximally six landmarks, coordination points, per agent. Additionally, the Back-on-Track-Repair algorithm could not leverage disjunctive goal form (cf. construction of Π_{back} in Algorithm 2) and this was emulated by an 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 four metrics to evaluate the measurements:

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

planning time was the measured cumulative time consumed by the underlying MA-Plan planner used for generating initial and repairing plans,

repairing time is the overall time spent in MA-Plan invocations minus the first planning process of the initial plan; and finally,

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

5.3. 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.*

Domain	Agents	Repairing time [ms]			No. of messages [-]			Communication [kB]			Exec. length [-]		
		BoT	RLazy	Replan	BoT	RLazy	Replan	BoT	RLazy	Replan	BoT	RLazy	Replan
LOGISTICS	2	115.3	116.1	145.6	13.9	8.2	10.9	2.0	1.7	2.3	8.8	14.3	10.7
	3	178.0	149.6	257.2	33.7	18.0	59.5	5.0	3.6	6.2	13.2	18.7	15.9
	4	266.2	162.1	479.3	89.5	29.1	114.7	13.3	**6.6**	**26.5**	15.5	21.5	18.1
LOGISTICS (PAR)	5	73.7	74.0	81.3	23.2	21.8	22.5	4.4	4.4	5.0	13.5	14.6	14.9
	6	126.0	84.1	110.7	49.6	32.5	60.9	9.3	6.8	9.6	11.4	12.4	12.0
COOP. PATHFINDING	2	23.9	115.6	93.2	2.4	2.2	2.2	0.6	0.6	0.6	2.8	3.2	2.6
	3	28.1	261.5	374.6	12.9	20.0	12.7	0.7	4.5	3.4	2.4	3.1	3.4
	4	29.4	19568.6	6529.8	20.0	14k	19.1	**0.9**	3002.0	**5.1**	2.3	4.0	3.3
ROVERS	2	381.3	179.8	249.8	13.0	7.4	10.0	2.7	1.9	2.7	14.7	20.3	14.5
	3	374.5	300.6	489.9	15	13.6	22.4	3.3	3.6	6.3	12.5	19.5	14.5
	4	798.3	634.4	650.5	42.5	30.0	29.1	7.6	8.3	8.9	15.3	22.1	13.6
SATELLITES	2	80.3	67.5	67.5	6.2	4.9	3.8	1.3	1.1	0.9	5.8	7.5	5.1
	3	126.9	81.9	139.9	15.8	7.6	15.3	2.9	1.8	3.7	6.9	7.0	6.5
	4	139.2	144.4	176.5	21.8	14.5	18.2	3.8	3.6	4.7	6.7	6.9	5.6
	5	154.5	232.9	222.3	30.0	17.7	25.9	5.6	4.8	7.3	5.7	6.7	5.5
	6	1452.8	48093.9	23027.7	57.3	32.5	38.2	11.6	9.4	11.7	6.9	7.1	5.7

Table 1: Results of experiments for all domains with probability $P = 0.3$ and action failures. The highlighted cells are the best results for a particular domain and a particular metrics. The bolded results distinctively support the core hypothesis of the paper.

LOGISTICS and COOPERATIVE PATHFINDING, as tightly and fully coordinated domains with dynamics of the simulated environment modeled as action failures, are suitable experiments to provide required insight. Table 1 shows results for a fixed failure probability $P = 0.3$ and Figures 1, 2 and 3 depict the results of the experiment for 3 agents LOGISTICS with variable probability P.

The highlighted results for 4-agent LOGISTICS in the table shows that the communication overhead generated by the Repeated-Lazy-Repair (RLazy) algorithm is at 25% of that generated by the replanning (Replan) approach. For 4-agent COOPERATIVE PATHFINDING, the communication overhead generated by the Back-on-Track-Repair (BoT) is at 18% of that generated by the replanning. Additionally, the communication overhead decreases with the increasing number of agents in the problems. I.e., the plan repairing algorithms scale better than replanning from scratch. The trends in Figures 1, 2 and 3 for LOGISTICS domain show, that the results are also valid for higher values of P. Furthermore the overhead decreases with increasing failure probabilities. The communication overhead generated in the experiment for various probabilities P by the Back-on-Track-Repair algorithm is, over all the measured probabilities, on an average at 59% (36% at best) of that generated by the replanning approach. 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 a consequence, the experiments *strongly support* our hypothesis.

The overall time spent in the planning phase (used by the MA-Plan algorithm) by the plan repair algorithms echoes the results for the communication

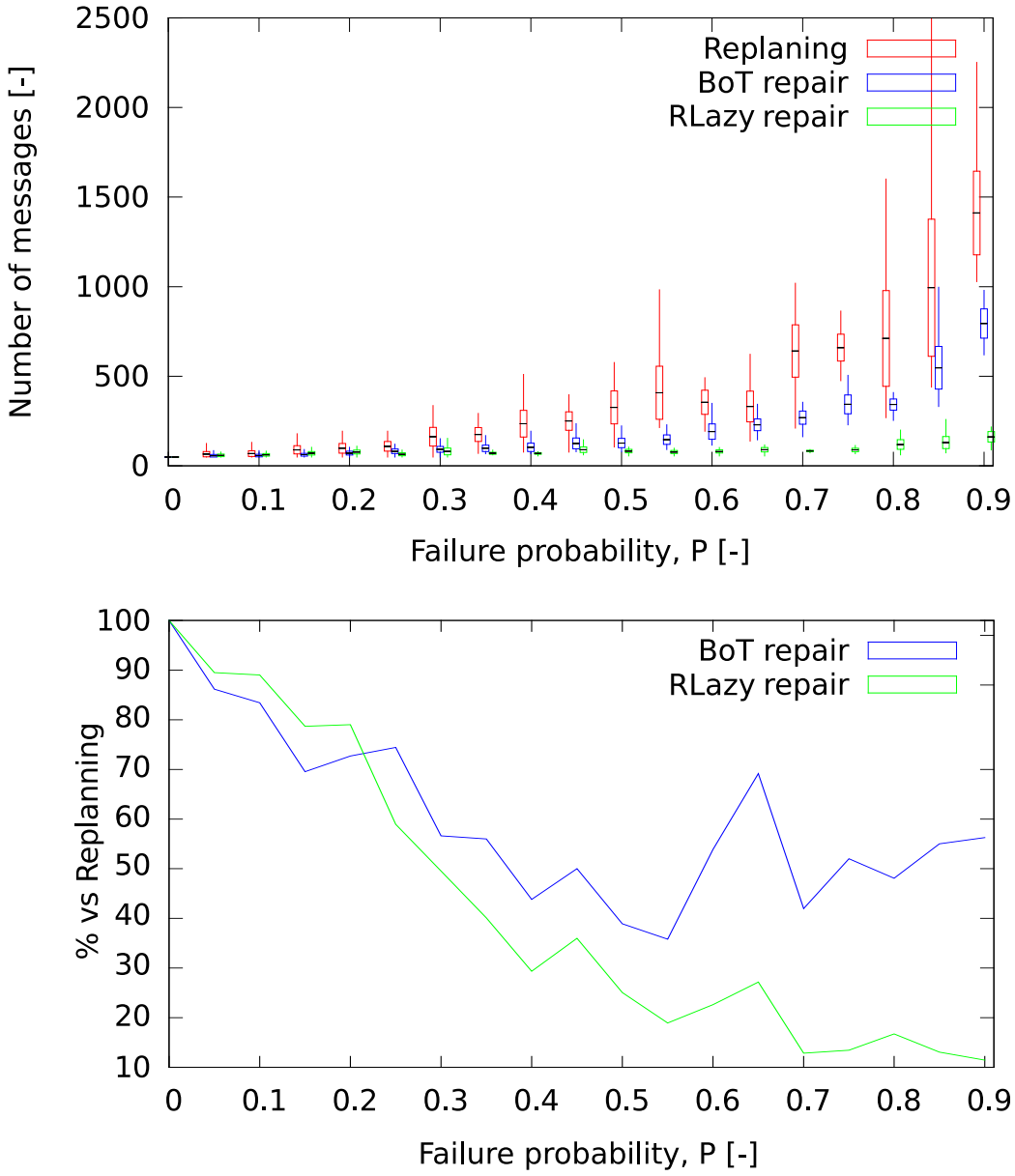

Figure 1: Experimental results of the communication metrics for LOGISTICS domain with 3 agents and action failures.

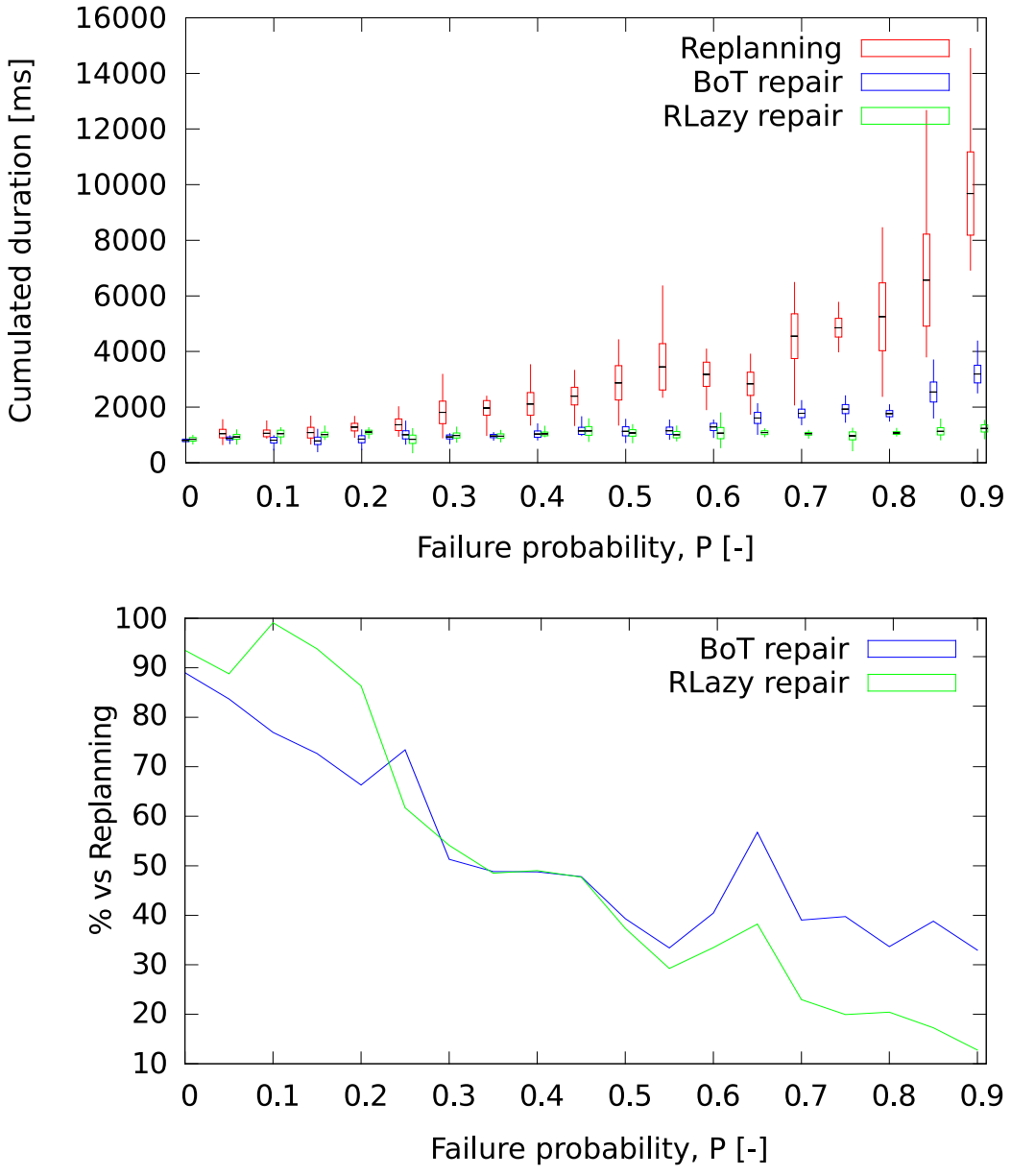

Figure 2: Experimental results of the planning time metrics for LOGISTICS domain with 3 agents and action failures.

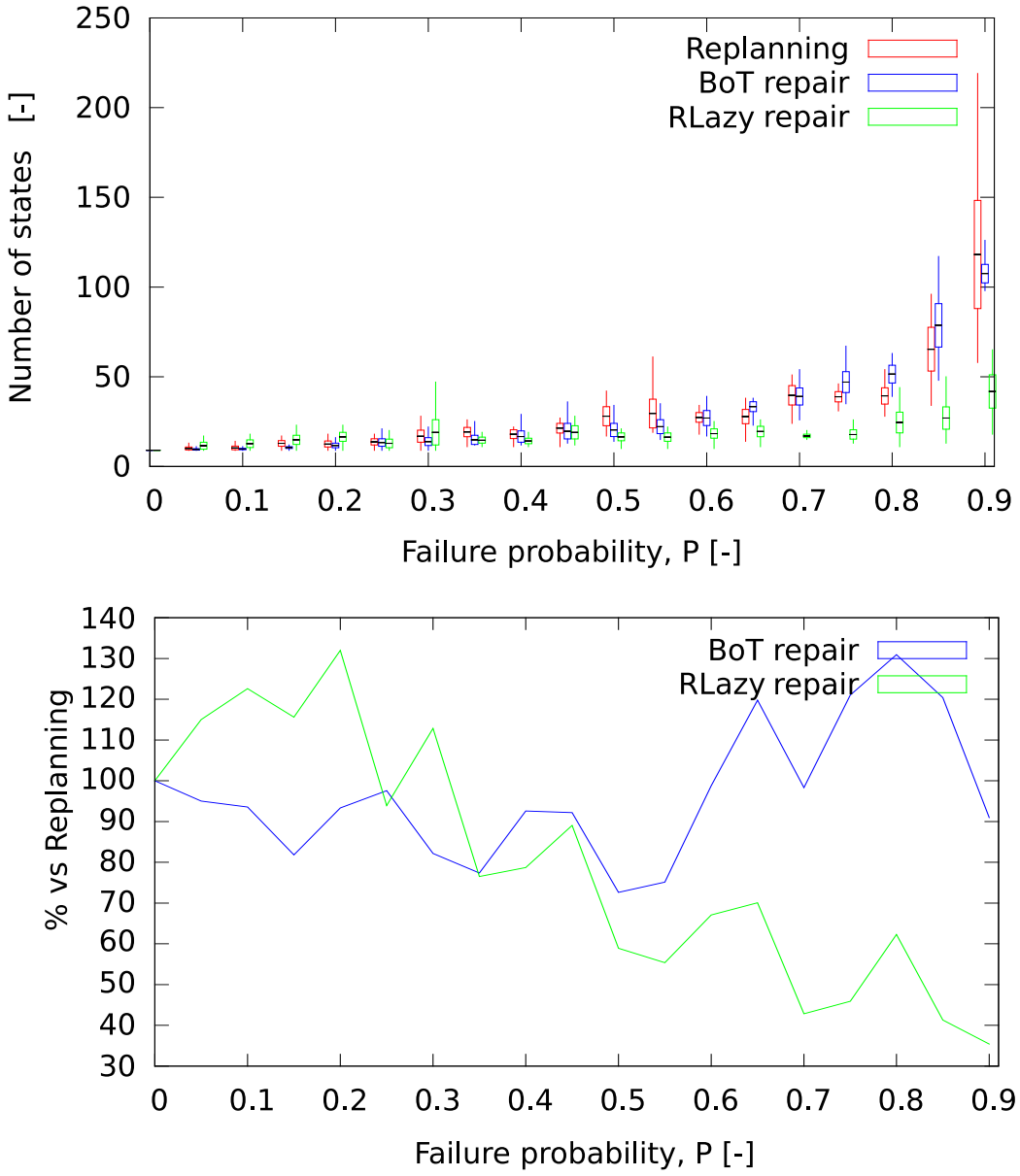

Figure 3: Experimental results of the execution length metrics for LOGISTICS domain with 3 agents and action failures.

overhead. Plan repairing scales better with higher numbers of agents in both LOGISTICS and COOPERATIVE PATHFINDING. On average, over all the measured probabilities P in 3-agent LOGISTICS, the computational efficiency was at 54% (34% at best) and at 51% (12% at best) for Back-on-Track-Repair and Repeated-Lazy-Repair respectively in comparison to replanning. Figure 1 depicts these results.

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 frequency of the planning 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 ROVERS as a loosely coordinated and SATELLITES as an uncoordinated planning problem. The results in Table 1 shows that the plan repairing algorithms are only slightly better (maximally 10%) in terms of the generated communication overhead than replanning, regardless of the number of agents. The trend in Figure 4 shows similar results for various failure probabilities P. The presented results support the auxiliary hypothesis.

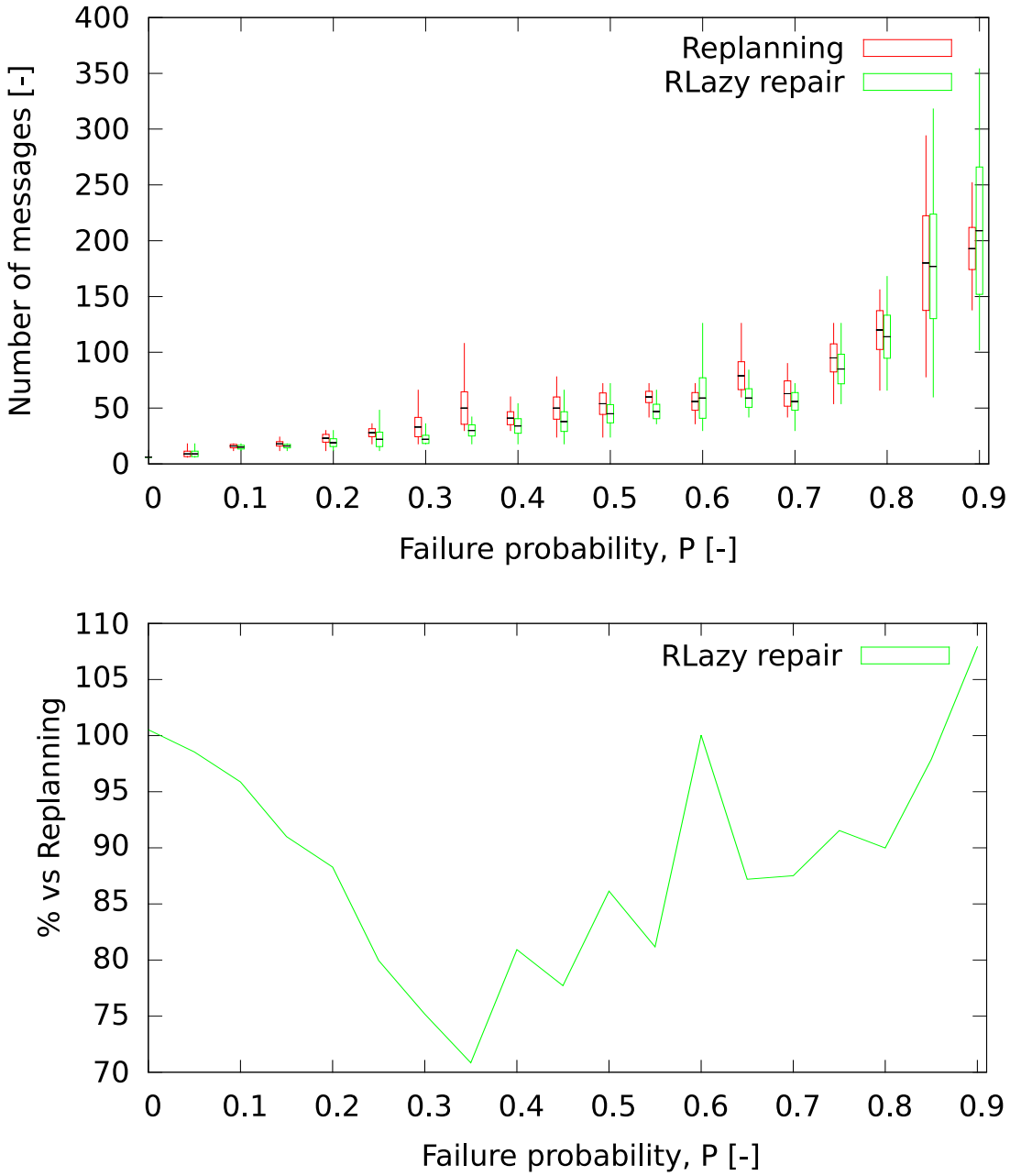

Figure 4: Experimental results for ROVERS domain with 3 agents and action failures.

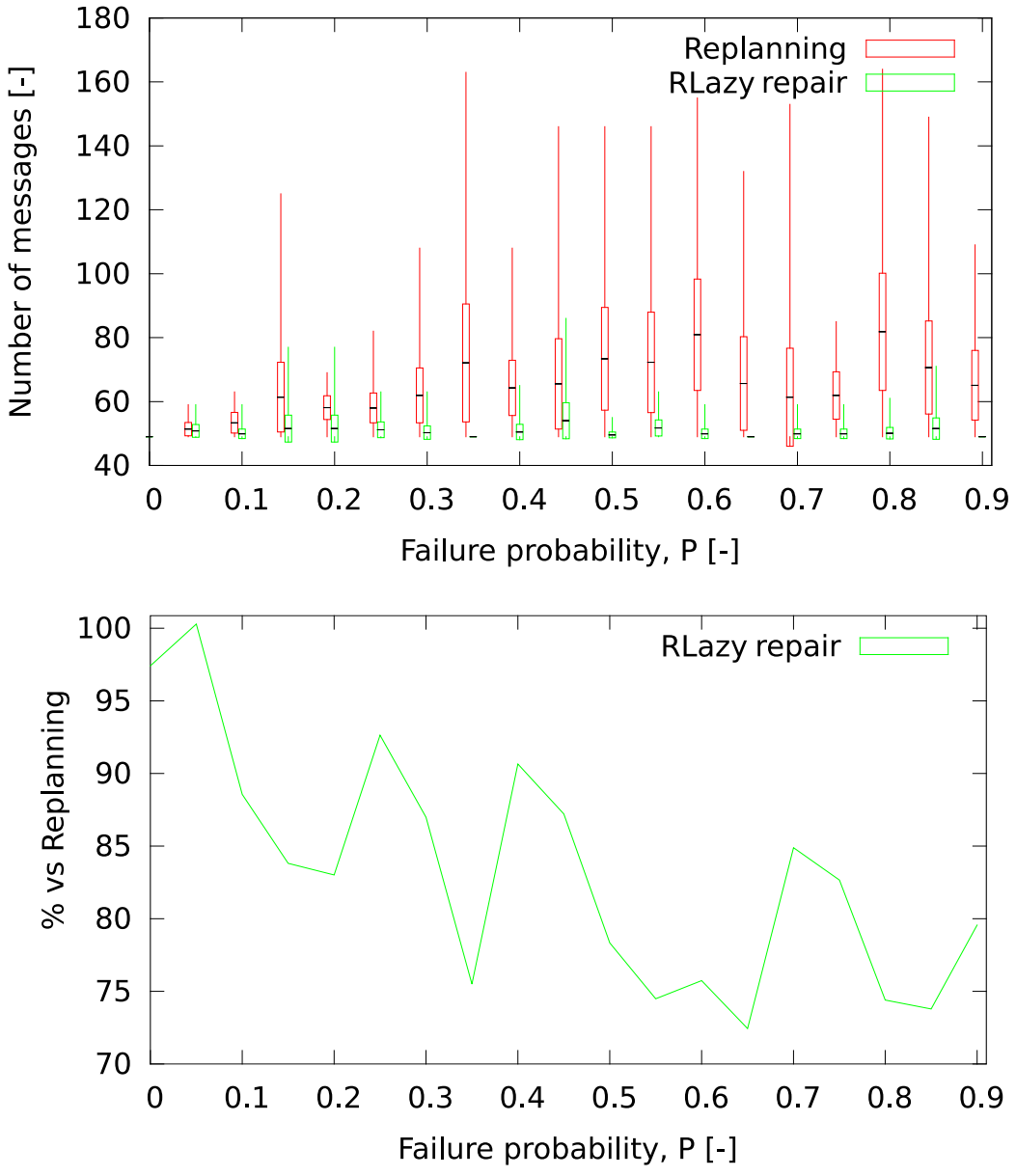

Figure 5: Experimental results for LOGISTICS domain with 3 agents and state perturbations with $c = 1$.

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 the nominal and the corresponding failed states 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 tackle this hypothesis, we modified the LOGISTICS experiment to simulate state perturbations as the model of the environment dynamics. Figure 5 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 $c = 2, 3, 4$ (not showed in the figure), 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.

Beside supporting the presented hypotheses the results also show the differences between the two plan repairing algorithms. Table 1 highlights the best results for communication volume and planning time. In most cases the Repeated-Lazy-Repair algorithm is more efficient in communication than

the Back-on-Track-Repair algorithm. The exceptions are the COOPERATIVE PATHFINDING and ROVERS domains with higher numbers of agents. These problems share high combinatorial complexity (COOPERATIVE PATHFINDING in coordination and ROVERS in local planning) and therefore more plan preserving techniques, as Back-on-Track-Repair, benefit.

6. Final Remarks

In the presented paper, we i) formally introduced the problem of multi-agent plan repair, ii) formulated a notion of relative coordination frequency of a planning problem based on Brafman and Domshlak's number of coordination points, iii) proposed three algorithms for solving the repair problem and proved their correctness, and finally iv) formulated, as well as experimentally validated the hypothesis stating that under certain conditions, *multi-agent plan repair approaches tend to be more efficient in terms of the communication overhead they generate in comparison to replanning from scratch.* Our results well support the core hypothesis of the paper and we additionally performed a series of experiments validating its boundary conditions articulated by the series of auxilliary hypotheses in Section 5.

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 [17], conformant [18] and contingency planning [19], plan re-use [3] and plan adaptation [20]. 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 [21] 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 a formal treatment of the problem, as well as proposed two specific incomplete algorithms for solving the problem, very distinct from the ones presented here. A sketch of the ideas behind the three plan repairing algorithms were additionally published in our recent work [14], however without a formal description in full details and exhaustive evaluation. Those are presented herein.

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. This, 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 powerful enough to capture subtleties in situations, such as concurrent resource consumption. This is not an issue in single-agent STRIPS [22] 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. Consider e.g., two trucks loading the same box at the same time, or two robots entering a narrow door simultaneously. Thirdly, there is a need for more efficient and feature-full implementations of multi-agent planners, as the gap between the state-of-the-art classical planners and multi-agent planners is still enormous. Fourthly, there is a lack of standard-

ized planning benchmarks for multi-agent planning, especially considering tightly coordinated planning problems. Exploration of such is needed to further evaluate the hypotheses presented in this paper. Finally, we left out the work towards resolving the validity of Hypothesis 2 aiming at analytical investigation of complexity properties of multi-agent plan repair to future work.

[1] R. v. d. Krogt, M. d. Weerdt, Self-interested planning agents using plan repair, in: Proceedings of the ICAPS 2005 Workshop on Multiagent Planning and Scheduling, 2005, pp. 36–44.

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

[3] M. Fox, A. Gerevini, D. Long, I. Serina, Plan stability: Replanning versus plan repair, in: Proceedings of ICAPS, 2006, pp. 212–221.

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

[5] J. S. Cox, E. H. Durfee, An efficient algorithm for multiagent plan coordination, in: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '05, ACM, New York, NY, USA, 2005, pp. 828–835.

[6] R. I. Brafman, C. Domshlak, From one to many: Planning for loosely coupled multi-agent systems, in: Proceedings of ICAPS, 2008, pp. 28–35.

[7] A. Jonsson, M. Rovatsos, Scaling up multiagent planning: A best-response approach, in: F. Bacchus, C. Domshlak, S. Edelkamp, M. Helmert (Eds.), Proceedings of ICAPS, AAAI, 2011.

[8] V. I. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions and Reversals, Soviet Physics Doklady 10 (1966) 707.

[9] R. Dechter, Constraint processing, Elsevier Morgan Kaufmann, 2003.

[10] R. Nissim, R. I. Brafman, C. Domshlak, A general, fully distributed multi-agent planning algorithm, in: Proceedings of AAMAS, 2010, pp. 1323–1330.

[11] P. Prosser, Hybrid algorithms for the constraint satisfaction problem, Computational Intelligence 12 (3) (1993) 268–299.

[12] R. Zivan, A. Meisels, Asynchronous forward-checking for DisCSPs, Constraints 12 (2007) 131–150.

[13] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, Journal of Artificial Intelligence Research 14 (2001) 253–302.

[14] A. Komenda, P. Novák, M. Pěchouček, Decentralized multi-agent plan repair in dynamic environments (Extended Abstract), in: Proceedings of AAMAS, 2012, pp. 1239–1240.

[15] P. Novák, W. Jamroga, Agents, Actions and Goals in Dynamic Environments, in: T. Walsh (Ed.), IJCAI 2011, Proceedings of the 22nd Interna-

tional Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, IJCAI/AAAI, 2011, pp. 313–318.

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

[17] C. Muise, S. A. McIlraith, J. C. Beck, Monitoring the execution of partial-order plans via regression, in: Proceedings of 22nd International Joint Conference on Artificial Intelligence, 2011, pp. 1975–1982.

[18] A. Albore, H. Palacios, H. Geffner, Compiling uncertainty away in non-deterministic conformant planning, in: H. Coelho, R. Studer, M. Wooldridge (Eds.), ECAI, Vol. 215 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2010, pp. 465–470.

[19] A. Albore, H. Palacios, H. Geffner, A translation-based approach to contingent planning, in: C. Boutilier (Ed.), IJCAI, 2009, pp. 1623–1628.

[20] H. Muñoz-Avila, M. T. Cox, Case-based plan adaptation: An analysis and review, IEEE Intelligent Systems 23 (4) (2008) 75–81.

[21] A. Komenda, P. Novák, Multi-agent plan repairing, in: Proceedings of Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities IJCAI-DMPOUW Workshop, pp. 1–6.

[22] R. Fikes, N. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, in: Proceedings of the 2nd International Joint Conference on Artificial Intelligence, 1971, pp. 608–620.