How to Repair Multi-agent Plans: Experimental Approach

Antonín Komenda[1] and **Peter Novák**[2] and **Michal Pěchouček**[1]
{komenda|pechoucek}@agents.fel.cvut.cz, P.Novak@tudelft.nl
[1]Dept. of Computer Science and Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic
[2]Dept. of Software and Computer Technology, Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology, The Netherlands

Abstract

Deterministic domain-independent multi-agent planning is an approach to coordination of cooperative agents with joint goals. Provided that the agents act in an imperfect environment, such plans can fail. The straightforward approach to recover from such situations is to compute a new plan from scratch, that is to replan. Even though, in a worst case, plan repair or plan re-use does not yield an advantage over replanning from scratch, there is a sound evidence from practical use that approaches trying to repair the failed original plan can outperform replanning in selected problems. One of the possible plan repairing techniques is based on preservation of the older plans.

This work experimentally studies three aspects affecting efficiency of plan repairing approaches based on preservation of fragments of the original plan in a multi-agent setting. We focus both on the computational, as well as the communication efficiency of plan repair in comparison to replanning from scratch. In our study, we report on the influence of the following issues on the efficiency of plan repair: 1) the number of involved agents in the plan repairing process, 2) interdependencies among the repaired actions, and finally 3) particular modes of re-use of the older plans.

Motivation

Consider a team of heterogeneous robots working together so as to execute a mission in an environment. Since the robots feature heterogeneous capabilities, it might well be that none of them is able to complete the mission on its own, however by a careful coordination and teamwork, they should be able to reach the joint objective. The team of physical robots is embodied in a dynamic environment in which various events and plan execution interruptions occur and most importantly, in which actions of the agents can fail. To execute their mission, the agents must be able to cope with such a dynamics on both, the individual, as well as the coordination level. Here we focus on the problem of multi-agent plan repair which tackles such issue.

There are several approaches capable to drive multi-agent team activities in an environment with an *a priori* unknown dynamics. Firstly, there is a body of literature dealing with and extending models of decentralized partially observable Markov decision processes (Dec-POMDPs) (Bernstein et al. 2002). A Dec-POMDP model leads to computation of a *policy* for the agents in the environment ensuring that by following it, the team reaches the joint goal. The model assumes only partial observability of the environment, a feature capable to capture various eventualities which could occur in the environment. These, however, have to be known *a priori*, so that a probabilistic model of action outcomes can be constructed before planning. Secondly, single-agent contingency (Fu et al. 2011) and conformant (Palacios and Geffner 2009) planning techniques facilitate classical-style planning for domains with non-probabilistic uncertainty in either action outcomes or state the system happens to be in. However, again, in order to plan for actions in such domains, the possible contingencies and action models in the environment must be known before the planning phase. The above discussed approaches do not scale well to larger domains, especially when the model of run-time action failures and events which could occur is *a priori* unknown.

Recently, in (Komenda, Novák, and Pěchouček 2012) the authors proposed an approach of *multi-agent (MA) plan repair* (MA-REPAIR), based on multi-agent planning (MA-STRIPS) as introduced in (Brafman and Domshlak 2008). MA-STRIPS is an approach to planning for teamwork and coordination extending the classical STRIPS-based planning techniques. According to the MA-REPAIR approach, the multi-agent team computes a team plan using a fully decentralized MA-STRIPS planning algorithm, and subsequently executes the plan, while at the same time monitoring of possible failures of plan execution. Upon an occurrence of such a failure, the team stops execution and invokes a plan repair algorithm and fixes the failed joint plan in order to reach a joint goal state from the state in which the failure occurred.

It can be argued that plan re-use in a single-agent context does not yield much advantage with respect to the computational complexity in the worst case (Nebel and Koehler 1995), since costly attempts to fix a failed plan sometimes lead to replanning from scratch anyway. In multi-agent and multi-robot settings, such as those involving teams of underwater or aerial robots, where communication is unreliable and costly, however, it is often the communication which is of higher priority than the computational complexity.

In (Komenda, Novák, and Pěchouček 2013), the authors proposed prefix and suffix-based approaches to MA plan re-

pair. Their work showed that these repairing approaches save communication in contrast to replanning from scratch in tightly coupled problems with action failures, however a research question which plan repairing techniques are more appropriate for which planning domains and problems remained unanswered. In this work, we generalize the prefix and suffix-based approaches from their work and present a study on how particular multi-agent plan repair techniques and particular parametrizations perform in different planning domains.

Multi-agent planning & repair

The problem of multi-agent plan repair as defined in (Komenda, Novák, and Pěchouček 2013) is a tuple $\Sigma = (\Pi, \mathcal{P}, s_f, k)$, where

1. $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ is a MA-STRIPS multi-agent planning problem over
   (a) a set of agents $\mathcal{A} = \alpha_1, \ldots \alpha_n$, each characterized by a set of STRIPS actions (over a propositional language $\mathcal{L}$) it can perform in an environment the agents operate in. I.e., $\alpha_i = \left\{\langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)\rangle \mid \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \subseteq \mathcal{L}\right\}$; and
   (b) an initial state $s_0 \in 2^{\mathcal{L}}$ the agents start to operate in, together with a specification of a set of final states $S_g \subseteq 2^{\mathcal{L}}$ characterizing their joint objective(s).
2. an original multi-agent plan $\mathcal{P}$ solving the problem Π, execution of which failed; and finally
3. a state $s_f \in 2^{\mathcal{L}}$ which the system happens to be in, unexpectedly after the plan execution failure occurrence of the plan execution step $k \in 1..|\mathcal{P}|$.

A solution to the MA plan repair problem $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ is a multi-agent plan $\mathcal{P}'$ solving a modified planning problem $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$. A multi-agent plan is a sequence of n-tuples, *joint actions*, with $n = |\mathcal{A}|$ corresponding to the number of agents in the team $\mathcal{A}$. Thereby a multi-agent plan is a sequence of synchronized actions of the individual agents. In order to constitute a valid multi-agent plan for a planning problem Π, firstly, each agent executing its corresponding sub-sequence of primitive actions must be capable to perform them, that is, $a_i \in \alpha_i \in \mathcal{A}$ for each individual action in every joint action $(a_1, \ldots, a_n)$ constituting the k-th step of the plan. Secondly, synchronized execution of the sequence of joint actions must lead from the initial state to some final state prescribed by the planning problem definition. That means all the inter-dependencies between the actions of the individual agents imposed by their preconditions must be satisfied along the plan execution.

In a case there are several plans repairing the original failed plan $\mathcal{P}$, the idea is to prefer those solutions, which *preserve*, that is re-use, parts of the original plan as much as possible. More formally, given two plans $\mathcal{P}_1, \mathcal{P}_2$ repairing the same multi-agent plan $\mathcal{P}$, we say that $\mathcal{P}_1$ is more preserving than $\mathcal{P}_2$ iff $\mathit{diff}(\mathcal{P}_1, \mathcal{P}) \leq \mathit{diff}(\mathcal{P}_2, \mathcal{P})$, where diff denotes the edit distance, i.e., the Levensthein distance (Levenshtein 1966) between the two plans given as arguments. Edit distance of two strings equals the minimal number of primitive edits. The adaptation to multi-agent plans corresponds to primitive edits being joint action addition, joint action deletion and a primitive action replacement operations. For further details, refer to the original papers (Brafman and Domshlak 2008; Komenda, Novák, and Pěchouček 2013).

Besides considering straightforward replanning from scratch, that is invoking the underlying multi-agent planner at the point of a failure and then executing the computed plan right away, in (Komenda, Novák, and Pěchouček 2012), the authors present two main approaches to solving multi-agent plan repair problems coined *back-on-track* and *lazy* repair respectively. Informally, the back-on-track approach tries to fix the prefix of the failed plan by computing a plan from the state in which the system happens to be right after the detection of a plan execution failure to some state along an ideal failure-free execution of the original plan. The resulting multi-agent plan re-uses some *suffix* of the original plan, if possible, and extends the plan at its beginning. The idea underlying the lazy repair is complementary. Lazy repair takes the remainder of the original plan, re-uses all its actions which still can be executed according to their preconditions regardless of the outcome and completes the plan to some goal state of the planning problem. This way, the resulting plan is composed of re-used *prefix* parts of the original plan with an appended suffix of some new repaired plan.

They also experimentally show that in some domains, these approaches lead to significant savings of communication, as well as computational resources in comparison to replanning from scratch in selected problems. Both algorithms first formulate a modified multi-agent planning problem and rely on the underlying multi-agent planner to compute a plan fragment used for re-composition into a solution plan repairing the original failed one.

A straightforward extension of the approach is to consider re-use of different parts of the original failed plan by combining parts of the prefix vs. suffix appending plan repair. Consider e.g., a repairing strategy according to which the back-on-track algorithm would not return back to an arbitrary state along the original plan's failure-free execution, but rather would consider only returning to the next few states presumed after the point of failure—this is approximately the way GPS navigation devices tend to fix a route after a missed junction point. Clearly, in some domains, this can be a beneficial approach, however not in all. An alternative idea would be to select only a minimal relevant subset of agents which should participate in the plan repair process, thereby constraining the inter-agent communication only to a subset of the agent team involved. In result, further reduction of communication needed for planning can be achieved.

Given the two above intuitions, we formulate first two hypotheses tackled in this paper.

Hypothesis 1 *Repairing algorithms minimizing the number of agents involved in the plan repairing process tend to generate lower computational and communication overheads than other strategies.*

Hypothesis 2 *Repairing algorithms reusing the original plan as a suffix generate lower computational and communication overheads than the repairing algorithms reusing the*

Algorithm 1 MA plan execution process with generalized multi-agent plan repairing algorithm.

Input: A MA planning problem $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$.
Input: Parameters f and g bounding the maximal length for reusing of the original plan as prefix and suffix.

$\mathcal{P} = \mathsf{MA\text{-}Plan}(\Pi)$
if $\mathcal{P} = \emptyset$ **then return** fail
$k = 1$
repeat
  agents perform $\mathcal{P}[k]$
  if failure detected **then**
    retrieve the current state s from the environment
    // begin: plan repairing $\Sigma = (\Pi, \mathcal{P}, s, k)$
    $f^* = f$; $g^* = g$
    repeat
      $\mathcal{P}_{\text{pre}} = \mathsf{ExecReminder}(\mathcal{P}[k..(k + f^*)], s)$
      $\mathcal{P}_{\text{suf}} = \mathcal{P}[(|\mathcal{P}| - g^*)..|\mathcal{P}|]$
      $\mathcal{P}^* = \mathsf{MA\text{-}Plan}((\mathcal{L}, \mathcal{A}, s \oplus \mathcal{P}_{\text{pre}}, S_g \ominus \mathcal{P}_{\text{suf}}))$
      if $\mathcal{P}^* \neq \emptyset$ **then**
        $\mathcal{P} = \mathcal{P}_{\text{pre}} \cdot \mathcal{P}^* \cdot \mathcal{P}_{\text{suf}}$
        break
      end if
    until tested all pairs of $f^* \in f..0$ and $g^* \in g..0$
    if $\mathcal{P} = \emptyset$ **then return** fail
    // end
    $k = 1$
  else
    $k = k + 1$
  end if
until $k > |\mathcal{P}|$

original plan as a prefix in domains featuring actions with long-term dependencies.

A long-term dependency can be visualized as a tree of consecutively dependent actions. If an action in the root of such tree has to be repaired, intuitively, it is a better idea to try to fix it as soon as possible, because not doing so can cause a snowball effect of increasing number of failing actions.

Further, we define the used algorithm, formulate last hypothesis and treat the hypotheses with three experiments.

Generalized repair algorithm

As outlined in the previous section, the algorithm used in the following experiments is a generalization of the lazy and back-on-track approaches with a number of additional modifications described in detail in the respective sections. The algorithm with the complete plan execution, monitoring and repair scheme is outlined in Algorithm 1.

The algorithm takes a multi-agent planning problem Π and two integer parameters f and g limiting the number of actions, which upon a detection of a failure can be reused during plan repair process from the currently executed *plan* $\mathcal{P}$ as a prefix or a suffix of the repairing plan. The process starts with a computation of the initial plan, which is subsequently executed. k denotes the counter of the current step in the plan execution. As the plan execution proceeds, in each step, all the agents perform their individual actions prescribed by the k-th joint action in the plan $\mathcal{P}$, denoted $\mathcal{P}[k]$.

In the case a failure is detected by the team, current state after the failure is retrieved and the plan repairing algorithm for the plan repairing problem $\Sigma = (\Pi, \mathcal{P}, s, k)$ is invoked. In each plan repairing attempt a modified multi-agent planning problem is formulated according to the current values of f^* and g^* prescribing the length of the re-used prefix and suffix of the original plan. In the case a repairing plan is found, the repairing process finishes, otherwise another attempt with a different combination of f^* and g^* is carried out. The resulting repairing plan consists of three components: the preserved prefix of the original plan $\mathcal{P}_{\text{pre}}$ reusing $\mathcal{P}[k..(k + f^*)]$, a newly computed infix $\mathcal{P}^*$ and suffix part $\mathcal{P}_{\text{suf}}$ reusing $\mathcal{P}[(|\mathcal{P}| - g^*)..|\mathcal{P}|]$, again preserving a part of the original plan $\mathcal{P}$.

The preserved prefix part of the original plan corresponds to an *executable reminder* fragment of $\mathcal{P}$, $\mathsf{ExecReminder}(\mathcal{P}, s)$, a selection of still applicable joint actions given the failure which occurred. The actions with unmet preconditions are simply omitted. Additionally, the prefix $\mathcal{P}_{\text{pre}}$ is based only on a part of the original plan effectively re-using f^* actions beginning after the k-th action of the original plan $\mathcal{P}$. The suffix part $\mathcal{P}_{\text{suf}}$ is obtained as the last g^* actions of the original plan $\mathcal{P}$.

Finally, the infix part of the plan is computed by invocation of the underlying multi-agent planner algorithm MA-Plan proposed by Nissim, et al. (Nissim, Brafman, and Domshlak 2010). The initial state of the modified planning problem is the state in which a failure-free execution of the repairing prefix $\mathcal{P}_{\text{pre}}$ would result in starting from the state s and it is denoted as $s \oplus \mathcal{P}_{\text{pre}}$. The set of goal states corresponds to a back-propagation of effects of the preserved suffix component $\mathcal{P}_{\text{suf}}$ from the set of original goals S_g. More formally, the back-propagation operator $\ominus : 2^{\mathcal{L}} \times \bigcup_{a \in \alpha \in \mathcal{A}} a \rightarrow 2^{\mathcal{L}}$ for a single action is defined as $s \ominus a = s \cup \mathsf{del}(a) \setminus \mathsf{add}(a)$. It extends to sequences of actions as follows.

Definition 1 (proposition back-propagation) *Let S' be a set of propositions back-propagated from a set of propositions S using a MA plan $\mathcal{P}$ denoted as $\ominus$ operator extension $S' = S_g \ominus \mathcal{P}$ iff $S' = (\cdots((S \ominus \mathcal{P}[m]) \ominus \mathcal{P}[m-1]) \ominus \cdots) \ominus \mathcal{P}[0]$.*

In the case a multi-agent planner finds a plan for the modified planning problem, the repairing plan takes the form $\mathcal{P}_{\text{pre}} \cdot \mathcal{P}^* \cdot \mathcal{P}_{\text{suf}}$ and gets executed from that point on. In the case no repairing plan can be found, the algorithm attempts the repair for a different combination of f^* and g^* until either a repairing plan is found, or it turns out that no repair for the failure exists.

The parametrization of the repairing process based on the f and g parameters opens an interesting research question, how do different combinations of the prefix and suffix preservation parameters influence the efficiency of the plan repairing process. Let m be the length of the re-usable part of the original plan $\mathcal{P}$, i.e., $m = |\mathcal{P}| - k$. Obviously, for $f + g < m$ there will be a gap, which has to be filled by a result of the inner planning process, in other words the original plan was *underused*. Reversely, for $f + g > m$ there

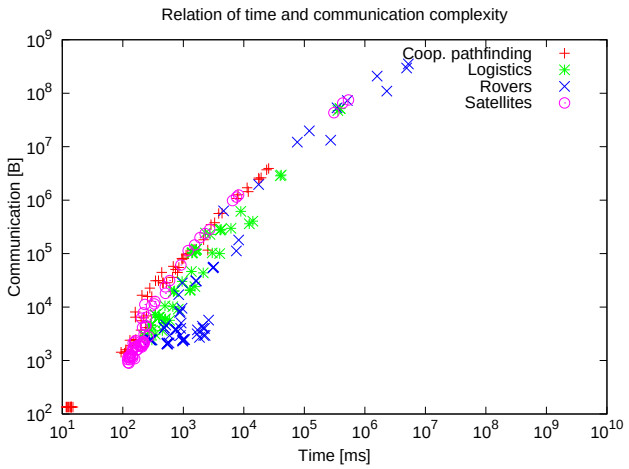

Figure 1: Relation between communicated bytes and computation time for solving the plan repairing problems.

will be an overlap, which has to be reverted, i.e., the original plan was *overused*. Intuitively, these cases are in a sense pathological. In a consequence, we propose the third, final hypothesis of this paper.

Hypothesis 3 *Repairing algorithms overusing or underusing the original plan tend to generate higher computational overheads than other algorithms.*

Experimental setup

The experiments were conducted in a synthetic setting, a simulated world with a group of agents using the plan execution, monitoring and repair loop. The world is fully observable for the agents. All failures of plan execution were generated by the simulator according to a uniform distribution over time and parametrized by a probability p of failure occurrence in each step for each experiment. A failure is generated only if there exists a plan to a goal state, which obviates problems with irreversible actions. The failures are handled by the agents immediately upon detection.

A failure is simulated by not-execution of some of the agent actions from the actual plan step. The individual action is chosen according to an uniform probability distribution over the individual actions within a joint action. As showed in (Komenda, Novák, and Pěchouček 2013), failure models with more radical impacts on the environment (e.g., state perturbations) decrease usability of the plan repairing approaches. Our motivation in this work is to study types of plan repairing, therefore we stick only to action failures.

For the implementation of the experimental setup and the repairing algorithms, we used a centralized world simulator integrating the multi-agent domain-independent planner MA-Plan (Nissim, Brafman, and Domshlak 2010). Each agent runs in its own thread and they deliberate asynchronously. The experiments were executed on 8-core processor at 3.6GHz with Java VM limited to 2.5GB of RAM.

For the experiments, we used four planning domains. Three of them originate in the standard single-agent IPC planning benchmarks. Similarly to the evaluation of the MA-Plan algorithm in (Nissim, Brafman, and Domshlak 2010; Komenda, Novák, and Pěchouček 2013), we chose domains, which are straightforwardly modifiable to a multi-agent setting: LOGISTICS, ROVERS, and SATELLITES. Additionally, we have extended the set of benchmarks by COOPERATIVE PATHFINDING coordination domain on a grid.

The experimental measurements were based on two metrics focusing on the target efficiencies: cumulative time consumed by the particular plan repairing algorithms during a single run of the simulation, i.e., the overall time spent in the algorithm (incl. the underlying planning process) excluding the initial planning phase of the scheme (Algorithm 1). The second metric was communication complexity of the process, that is the volume of communicated information in bytes among the involved agents during the plan repairing processes. Those are mainly the messages generated by the DisCSP solver in the MA-Plan planner and an additional synchronization processes minimizing the number of agents involved in the plan repairing process.

To account for differences in essential computational and communication complexity of the domains, we conducted a relationship experiment between these two measures. Figure 1 depicts the results and demonstrates that there is no essential discrepancy between the computational and communication complexity of the plan repairing solutions. That means, the following results are not biased by problems extremely hard in time and simple in communication and vice versa.

The number of repairing agents

Regardless of the theoretical results presented in (Brafman and Domshlak 2008), showing that the computational complexity of DisCSP-based multi-agent planning is not exponentially dependent on the number of the agents, in practical experiments, we faced a non-negligible dependence of the this number and required communication and computational effort. The first set of experiments analyzes this relation.

Used algorithms To validate Hypothesis 1, we have prepared an extensive set of plan repairing algorithms stemming from the Generalized repair algorithm. They can be divided into three main groups: one without agent count minimization, and two with agent minimization. First of the minimization groups reuse the original plan as a suffix and the other one as a prefix.

The difference among the algorithm instances within one of the groups lies in preference between agent minimization, size of preservation of the original plan and bound on the maximal length of the newly generated repairing plan component $\mathcal{P}^*$. This approach restrain bias prospectively caused by unbalanced influences of the agent minimization on various types of plan repair.

The approach used to minimize the number of involved agents was based on the notion of a set of *supporting agents*. The iterative process from Algorithm 1 was extended with an iteration starting only with a set of agents providing at least one action, which can contribute to the repairing plan by a required proposition(s), i.e., support part of $S_g \ominus \mathcal{P}_{\mathrm{suf}}$.

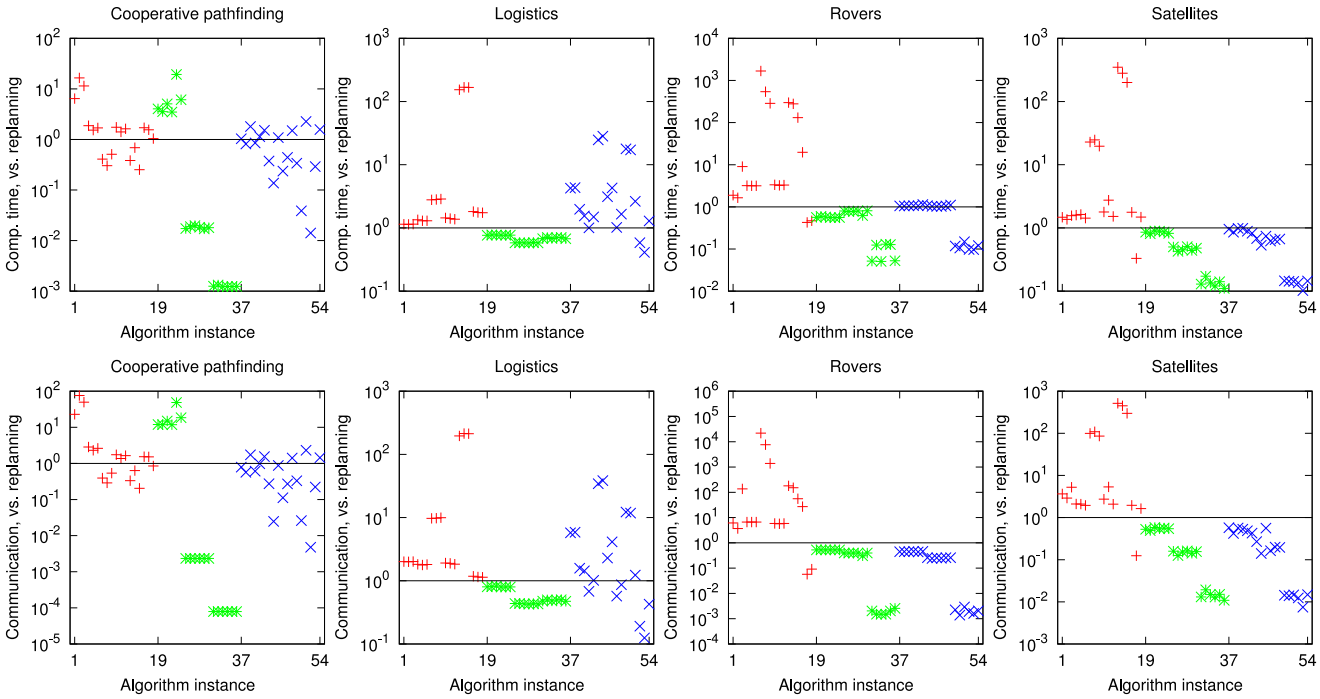

Figure 2: Comparison of various plan repairing algorithms in proportion to replanning (black line at $y = 1$) with failure probability $p = 0.3$. Each point represent a mean of several runs of one of the particular repairing algorithms. The red group contains plan repairing algorithms using only the full set of agents involved in the original planning problem, the green group contains algorithms using various techniques to minimize number of agents involved and preserving suffix of the original plan and the blue group contains algorithms also minimizing number of agents and preserving prefix of the original plan.

If such team of agents is not able to solve the plan repairing problem, the team is extended by additional agents supporting any of the current agents in the team by means of contributing to prepositions in their preconditions. If such additional agent does not exist and the team is still not containing all the agent from $\mathcal{A}$, a random agent is added into the team and the process continues.

Results and Discussion The experiments were conducted in all presented experimental domains and for all combinations of agent counts, i.e., two to four agents giving twelve domain and problem instances. Each of the group contained six variances of the algorithms giving with the problem instances 216 experiments. Each of the experiments was averaged over 5 measurements with different random seeds.

Figure 2 shows results of the first batch of experiments. The first group of repairing algorithms not minimizing number of involved agents (red color) is in most measurements in both computational and communication metrics worse than the baseline replanning algorithm. The suffix preserving algorithms minimizing numbers of agents (green color) is on the other hand nearly in all measurements better in both metrics than the baseline algorithm with an exception in the simplest COOPERATIVE PATHFINDING problems. The group of plan repairing algorithms minimizing the number of involved agents and preserving prefix part of the original plan (blue color) is on tie or better with the replanning in rather loosely coupled domains decreasing the communication and computational overheads with decreasing coupling of the domains. However in tighter coupled domains the algorithms fall behind the replanning baseline. In LOGISTICS domain, only 33% of the algorithms are better by communication overheads and only 18% by means of computational overheads. With increasing coupling the approach lose more. These results *support the first hypothesis*.

Additionally, the results revealed that the prefix-based approaches, as not the best in all agent minimizing approaches, in most of the experiments has one of the best approaches outperforming the best suffix-based approach. In LOGISTICS domain the separation between the best prefix-based and best suffix-based plan repairing algorithm is about a half an order of magnitude in favor of the one prefix preserving approach. On the other hand, in COOPERATIVE PATHFINDING, suffix-approaches gain an order and more.

Repairing of long-term dependencies

The intuition behind the second hypothesis can be rephrased as follows: If an action fails and it has potentially a lot of future dependencies, possibly of other agents or even in the goal, trying to fix it as soon as possible is rather better idea, than ignore it and try to repair it later. The experiments in this section were conducted to validate this concept.

Used algorithms The most straightforward approach here is to compare the two plan repairing algorithms re-using the whole original plan either as a prefix or as a suffix. These

$$
\begin{array}{l}
\mathsf{A}: \\
\mathsf{T}_1: \\
\mathsf{T}_2:
\end{array}
\begin{pmatrix}
\epsilon & \epsilon & \epsilon & \overline{\mathsf{l}(\mathsf{p},\mathsf{a}_1)} & \mathsf{f}(\mathsf{a}_1,\mathsf{a}_2) & \overline{\mathsf{u}(\mathsf{p},\mathsf{a}_2)} & \epsilon & \epsilon & \epsilon \\
\mathsf{l}(\mathsf{p},\mathsf{d}_1) & \mathsf{m}(\mathsf{d}_1,\mathsf{a}_1) & \overline{\mathsf{u}(\mathsf{p},\mathsf{a}_1)} & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\
\mathsf{m}(\mathsf{d}_2,\mathsf{a}_2) & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \overline{\mathsf{l}(\mathsf{p},\mathsf{a}_2)} & \mathsf{m}(\mathsf{a}_2,\mathsf{d}_2) & \mathsf{u}(\mathsf{p},\mathsf{d}_2) \\
8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 &
\end{pmatrix}
$$

Figure 3: A multi-agent plan solving the initial LOGISTICS problem used in the experiments. Empty actions are denoted as ϵ. The overlines mark public actions. The numbers in the last row represent particular counts of steps, i.e., number of actions m, to the end of the plan.

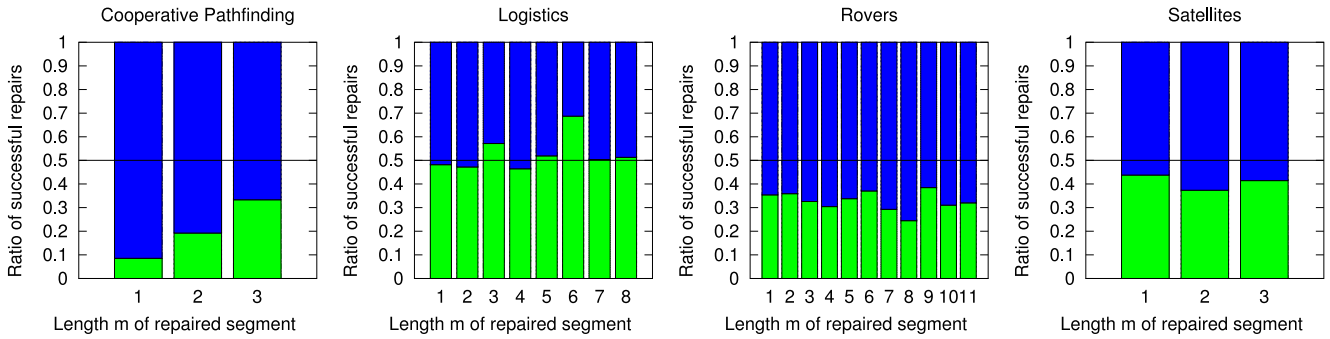

Figure 4: Comparison of success ratio against replanning between prefix-based (blue) and suffix-based (green) plan repairing.

algorithms are again modification of the plan repairing part of Algorithm 1 such that there is no iteration over various f^* and g^*, but only two fixed values. The *pure prefix algorithm* uses fixation $f^* = m, g^* = 0$ and the *pure suffix algorithm* uses only one parameter pair $f^* = 0, g^* = m$.

Furthermore, to be able to demonstrate the behavior and to explain the results, we have to present more details on LOGISTICS. The problem used in the experiments contains three agents controlling two trucks T_1 and T_2 and one airplane A. There are two cities, each with one storage depot (d_1 and d_2) and one airport (a_1 and a_2). 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 typical multi-agent plan solving this particular instance is depicted in the matrix form, see (Komenda, Novák, and Pěchouček 2013) for more detail, in Figure 3.

Results and Discussion To validate Hypothesis 2, we run the pure prefix-based and pure suffix-based repairing algorithms in all the testing domains. We have measured ratio of successful repairs of these two repairing algorithms against replanning by means of computation time. In Figure 4, we summarize the results of these experiments.

In the ROVERS and SATELLITE domains, the plans solving the problem do not contain any significant actions by means of number of future dependencies to the overall count of actions in the plan. In SATELLITES, all actions are private, therefore actions of one agent depend only on other actions of the same agent. Additionally, the individual plans of the agents are relatively short (three to four actions), therefore the private dependencies are never longer than four actions.

Multi-agent plans for the ROVER problems contain several public actions at the end of the plan, representing always only one rover communicating at one time point. Albeit the plans solving the ROVERS problems contain public actions, there are again no long dependencies among the actions. The dependencies in the private part of the plan contain three components, each containing three to four private actions. Consequently, the private dependencies are, similarly to the SATELLITE problems, maximally four actions long. The dependencies among the public actions are even shorter, as there is the same number of public actions as agents, which means maximally three-action public dependencies for three agents. The dependency link between one public action and one dependent private component increases the maximal dependent length to maximally seven actions (four private actions of the component bound to three public actions successively dependent on each other).

In such repairing problem, even if one of the leading actions in a private component fails, lazy approach solves nearly the complete problem only by re-using the original plan. More precisely, it re-uses the original solution for the rest of the private components and all the public actions except one of the failed agent. The results show, the prefix-based repair is always better then the suffix-based and the ratio between these two is stable over different points in the plan. The situation changes in the LOGISTICS domain.

In LOGISTIC with three agents and one package, there is a chain of dependent actions. Particularly, $\mathsf{u}(\mathsf{p},\mathsf{d}_2)$ depends on $\overline{\mathsf{l}(\mathsf{p},\mathsf{a}_2)}$, which depends on $\overline{\mathsf{u}(\mathsf{p},\mathsf{a}_2)}$ and so on to the first action of the plan $\mathsf{l}(\mathsf{p},\mathsf{d}_1)$. The dependency chain has six

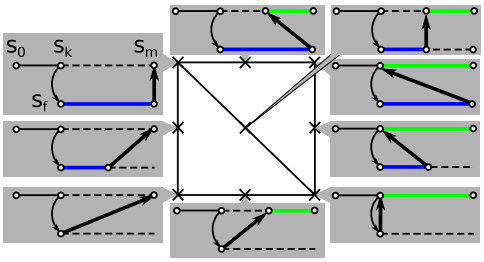

Figure 5: Scheme of a two-dimensional space representing plan repairing algorithms preserving different parts of the original plan and reusing it in different ways. The blue segments represent prefix-based re-usage and the green ones the suffix-based re-usage. The notable states are: initial state s_0, last achieved state s_k induced by the original plan, exceptional state s_f after a failure and the last anticipated state $s_m \in S_G$, provided that the original plan would be executed without a failure.

actions in the experimented plan and occupy the complete length of it. As the results show in Figure 4, there are two distinctive peaks where the suffix repair outperforms the prefix repair, additionally with an increasing trend. The first one is for repairing plans of length $m = 3$ and the other one is for $m = 6$. As we can see in Figure 3, these lengths correspond to the package hand-off points in the plan, more precisely repair of failing unloads $\overline{\mathsf{u}(\mathsf{p}, \mathsf{a}_1)}$ and $\overline{\mathsf{u}(\mathsf{p}, \mathsf{a}_2)}$. Ignoring a failure of unloading by a lazy approach causes the package is left in the last vehicle and the rest of the team finishes the executable remainder of the plan, which effectively means the vehicles are moving, but they are not transporting the package. Under such circumstances, the suffix repair only repeats the unload action and successfully continues with the rest of the original plan ending in a goal state.

One can argue that the complement load actions should be repaired more efficiently using this same argumentation as well. This is very true, however this phenomenon is not captured in the results, because of a particular implementation of the MA-Plan planner. The explanation is based on the fact the used planner efficiency is more dependent on small differences in number of involved agents, than the number of planned actions. In the case of $m = 3$, i.e., the $\overline{\mathsf{u}(\mathsf{p}, \mathsf{a}_2)}$ action, we need 2 agents to do lazy repair, because firstly we reuse executable reminder of the original plan to the last state without the package and than we have to use the planner to generate repairing plan $\mathcal{P}^*$reverting all the moves and planning again to one of the goal states. Such plan has to firstly unload the package from the airplane A and then transport it successfully by the truck T_2 to the goal destination d_2. On the other hand, the back-on-track algorithm only generates a plan repeating the unload action $\overline{\mathsf{u}(\mathsf{p}, \mathsf{a}_2)}$ and afterward continues with the original plan as a suffix. This planning problem involves only one agent, in particular, the airplane A carrying out unload of the package. The same principle can be applied to $m = 6$, but with all three agents for pure lazy repair but only 2 agents for pure back-on-track.

In the last problem of COOPERATIVE PATHFINDING, the length of a sequence of dependent actions correspond to the length of the plan, as all the actions in such plan are public and inter-dependent. Nevertheless, this is quite different "order of dependency", than in SATELLITES for example. In SATTELITES, all the actions are dependent as well, but only within one agent, whereas here, the actions are dependent across the agents. In the experimental results of the COOPERATIVE PATHFINDING a trend arises. In such dense types of inter-dependent problems, the longer are the repaired plans, the more the suffix repairing algorithm gains against the prefix one. Unfortunately with the current available implementation of the multi-agent planner, we were not able to conduct experiments with bigger grid sizes, i.e. longer repaired plans, thus the validation if the trend would continue is left for future work at this point.

The results of these experiments, namely of LOGISTICS and COOPERATIVE PATHFINDING, moderately support the second hypothesis of the paper.

Repair appropriately reusing the original plan

A fundamental principle behind a large group for repairing algorithms, firstly proposed in (Nebel and Koehler 1995), can be described as action ordering preservation or, in other words, re-usage of parts from former plans. This very principle stands also behind the two multi-agent plan repairing algorithms proposed in (Komenda, Novák, and Pěchouček 2013). It is not intuitively clear what is a good strategy for reusing the original plan, moreover related to a particular planning domain. The experiments conducted in these sections provide several insights into this issue.

Used algorithms A battery of plan repairing algorithms was prepared to validate Hypothesis 3. We modify how and how much the algorithms reuse the original plan. Such modifications lead to a two-dimensional discrete space of different plan repairing algorithms, as depicted in Figure 5, representing a structure of the repaired plan.

Each of the nine diagrams in the figure describes a variation on a resulting plan repaired by one particular modification of the algorithm in the context of execution of the original plan. The execution starts with a world in the initial state s_0 and it is anticipated to continue with help of the original plan to the last state s_m, which is one of the goal states, i.e., $s_m \in S_G$. However during execution of an action following a state s_k, execution failed and the state of the world ends up not in the state s_{k+1}, but in a state s_f, out of the anticipated sequence of states and actions. To fulfill the goal, the agents use one of the plan repairing algorithms, which under the condition of perfect execution, would transform the world from s_f to a $s_m \in S_G$.

In Figure 5, there are two dimensions depicted. One of the dimensions represent the number of actions which has to be reused from beginning of the original plan as a prefix corresponding to fixation of the iteration parameter $f^* = f$. The other dimension represents number of actions reused as suffix of the final repairing plan, i.e., fixed iteration parameter $g^* = g$. In the presented scheme, $\mathcal{P}_{\text{pre}}$ from the Algorithm 1 is denoted as a blue line, $\mathcal{P}_{\text{suf}}$ as a green line and $\mathcal{P}^*$as a black thick arrow. Since both the dimensions

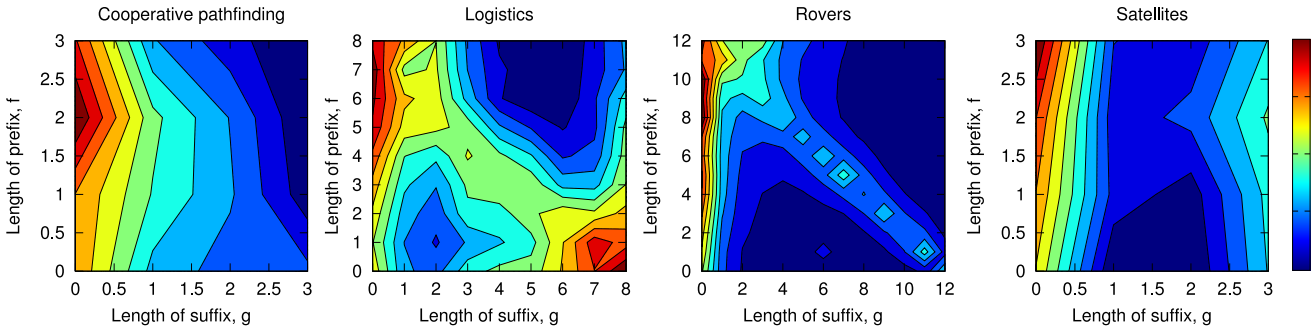

Figure 6: The maps present prefix (f on y-axis) vs. suffix (g on x-axis) preserving repairing algorithms by a success rate against replanning in the repair time for all domains with three agents and $p = 0.3$. Red color represents algorithms more often faster then replanning. The top-left to bottom-right diagonal represent algorithms neither overusing or underusing the original plan.

reuse the same original plan, the space is always a square with a side of the length m.

There are four extremes in the algorithm space. The algorithm at position $(0, 0)$ effectively degenerates from $\mathcal{P}_{\text{pre}} \cdot \mathcal{P}^* \cdot \mathcal{P}_{\text{suf}}$ to $\mathcal{P}^*$. Such process correspond to replanning from the scratch. The algorithms at positions $(m, 0)$ and $(0, m)$ represent pure repairs $\mathcal{P}_{\text{pre}} \cdot \mathcal{P}^*$ and $\mathcal{P} \cdot \mathcal{P}_{\text{suf}}$ respectively. The last extreme at (m, m) represent an algorithm, which firstly uses the executable reminder of the original plan, then using a newly generated plan $\mathcal{P}^*$ returns to the anticipated state after execution of the failed action and than reuses the original plan again to get to the goal state, i.e., the algorithm generates a full overlap of the prefix and suffix plans.

Beside the extremes, also the $(0, m), (1, m-1), ..., (m-1, 1)$, $(m, 0)$ diagonal in the space is important from perspective of the ongoing discussion. All the algorithms lying on this diagonal re-use exactly all the actions of the original plan in the original order. Meaning, the original plan is neither overused nor underused. Formally, we define:

Definition 2 *(m-*normal plan repair*) Let $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ be a multi-agent plan repairing problem, then an algorithm* R *is a* m-normal plan repair*, iff* R *solves the problem Σ by a multi-agent plan $\mathcal{P}$ with decomposition $\mathcal{P}_{\text{pre}} \cdot \mathcal{P}^* \cdot \mathcal{P}_{\text{suf}}$ and at the same time $(|\mathcal{P}_{\text{pre}}|, |\mathcal{P}_{\text{suf}}|) \in \{(0, m), (1, m-1), ..., (m-1, 1), (m, 0)\}$.*

Results and Discussion To validate the third and last hypothesis, we used a randomized sampling of the algorithm space and searched for more successful algorithms lying on the m-normal repairing diagonal by the hypothesis. The results are present in Figure 6.

The sampling experimental process measured for each encountered repairing problem the computation time of the replanning algorithm. After this base-line measurement, a tested repairing algorithm was run with a bound on the computation time based on the replanning run-time. If the algorithm performed better, a cell in the result map was incremented by one. In effect, this process rendered the presented normalized results. During the experimental execution and plan repairing, we used different lengths of the original plan, i.e., the repair was done for various m. Therefore, the resulting maps depict a continuous space, as the results with higher and lower m values were merged into the most representative m value corresponding to the initial multi-agent plan generated.

As the maps show, the hypothesis clearly holds for coupled domains with longer plans (LOGISTICS, and ROVERS). In the coupled domain of COOPERATIVE PATHFINDING, the diagonal is also present, but because of shorter repaired plans, it degenerated considerably. In the experiment with SATELLITES, the diagonal is not present.

These results support Hypothesis 3 with an auxiliary observation, that the effect is decreasing as the coupling of the domain decreases.

Final remarks

Based on the experimental results, we can come up with a summary of heuristic approaches in form of simply usable advices decreasing computation and/or communication overheads during repairing of multi-agent plans. These advices can be used for various plan repairing approaches targeting systems with planning agents. The results were verified for plan repairing techniques utilizing preservation of the original plan and using an DisCSP-based multi-agent planner to fill prospective discontinuities in the repairing plan. The advices are:

1. *Prefer smaller numbers of involved agents in the plan repairing process.*
2. *Prefer prefix repairing techniques when repairing failures with long dependencies among different agents.*
3. *Prefer m-normal plan repairing algorithms.*

This work opens several interesting questions left for the future work. Most notably, how would another implementation of the underlying multi-agent planner affect the results and would it be possible to integrate principles from single-agent search effort estimation approaches, e.g., as in (Korf, Reid, and Edelkamp 2001) to provide more precise hints how to repair during the execution and repairing process.

Acknowledgments This work was supported by the U.S. Air Force EOARD grant no. FA8655-12-1-2096.

References

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov Decision Processes. *Math. Oper. Res.* 27(4):819–840.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of ICAPS*, 28–35.

Fu, J.; Ng, V.; Bastani, F. B.; and Yen, I.-L. 2011. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *Proceedings of IJCAI*, 1949–1954.

Komenda, A.; Novák, P.; and Pěchouček, M. 2012. Decentralized multi-agent plan repair in dynamic environments (Extended Abstract). In *Proceedings of AAMAS*, 1239–1240.

Komenda, A.; Novák, P.; and Pěchouček, M. 2013. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications*. DOI: 10.1016/j.jnca.2012.12.011.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening-A*. *Artif. Intell.* 129(1-2):199–218.

Levenshtein, V. I. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady* 10:707.

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

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of AAMAS*, 1323–1330.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Int. Res.* 35(1):623–675.