Czech Technical University in Prague
Faculty of Electrical Engineering

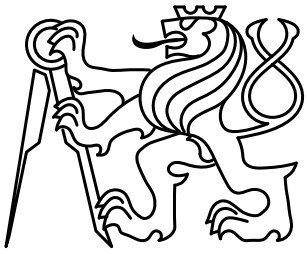
Doctoral Thesis

June 2013

Antonín Komenda

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science and Engineering



Domain-independent Multiagent Plan Repair

Doctoral Thesis

Antonín Komenda

Prague, June 2013

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Artificial Intelligence and Biocybernetics

Supervisor: Prof. Dr. Michal Pěchouček, MSc.

Dedicated to my sister Alena Alisa Komendová.

Acknowledgments

This work would not be possible without the support of my supervisor Prof. Michal Pěchouček and daily supervisor Dr. Peter Novák. They both guided me consistently and openly during the research and coauthored most of the works which this thesis stands on. My gratitude also goes to Dr. Jiří Vokřínek who helped me from the first moments we met at the university and skillfully directed my early steps and to Michal Štolba who is currently my closest coworker and without whom the latest research results could not be of the quality they are now.

Especially the final miles of the work would be impossible without a thorough and methodical motivation from my dear wife Martina.

Abstract

Achieving joint objectives in distributed domain-independent planning problems by teams of cooperative agents requires significant coordination and communication efforts. Provided that the agents act in an imperfect environment, their plans can fail. The straightforward approach to recover from such situations is to compute a new plan from scratch, that is to replan. *An alternative approach is to reuse the original plan and repair it. Even though, in a worst case, plan repair or plan reuse 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.*

This thesis formally introduces the multiagent plan repair problem. *Building upon the formal treatment, algorithms for multiagent plan repair are described using transformation of the problem to specialized instances of a multiagent planning problem. The algorithms are theoretically analyzed from the perspective of soundness and completeness properties as well as time and communication complexity.*

To demonstrate practical impacts of the plan repair techniques, after description of the implementation of the algorithms, an extensive experimental evaluation of the algorithms is presented with discussion of the results. The experiments focus on four hypotheses: (i) multiagent plan repair approaches producing more preserving repairs than replanning tend to generate lower communication overhead for planning problems requiring strong coordination, (ii) repair approaches minimizing the number of agents involved in the plan repair process tend to generate lower computational and communication overheads than other strategies, (iii) repair approaches solving a problem in the execution as soon as possible generate lower computational and communication overheads than the other repair algorithms in domains featuring actions with long-term dependencies and (iv) repair approaches overusing or underusing the original plan tend to generate higher computational overheads than other algorithms.

Finally, the plan repair approaches are validated by adaptation and deployment to a high-fidelity simulation by a software engineering methodology designed for guidance of such process and by replacement of the used inner multiagent planning approach with a newly designed and implemented experimental prototype of a multiagent forward-search planner.

Contents

Chapter 1

Introduction

Research in artificial intelligence endeavors to describe and utilize rational behavior for machines. A crucial part of such behavior is the ability to deliberate about sequences of actions or shortly *plans*. Building of plans is indisputably the most important precursor for any other form of such deliberation. In the context of intelligent construction of plans by machines, we talk about *automated planning*. The story, however, often does not end with a prepared plan. Under various circumstances, the plans have to be updated to take into account unanticipated phenomena. In such situations we talk about replanning, plan adaptation or *plan repair*, as far as the process is fixing an original plan. The deliberation can be understood as a centralized computational process or decentralized interweaving of more processes. The latter case can be additionally restricted by a need of the computational processes not to reveal all the information they are working with. In such cases, we are dealing with *multiagent* systems or particularly in the context of the aforementioned fixing of plans with *multiagent plan repair*.

Automated planning, especially the centralized form, has already succeeded in various practical problems [46, 47]. The first presented is planning for a mission on Mars carried out by the Mars Rovers [8, 18] and generally planning and scheduling for various space missions at NASA [9]. Another example represents software systems for planning of sheet-metal bending operations and assembly tasks [22]. Automated planning was also successfully used in various logistic domains such as organization of ship containers at docks[1] or cargo transportation[2].

The motivation for the research described in this thesis stems from practical needs for efficient and robust approach to coordination of teams of *intelligent entities* solving problems across various domains. To introduce practical intentions here, the domain to mention first is planning of cooperation during a disaster relief operations where various parties with heterogeneous abilities have to cooperate to fulfill a common goal. The other example is a multi-robotic team of autonomous

[1] http://www.navis.com/solutions/container
[2] http://www.soloplan.com/

vehicles providing support in a military mission. Such vehicles can considerably vary in their abilities and limitations, *e.g.*, the bandwidth among a couple of robotic submarines will be much lower than the bandwidth among a group of large-scale Unmanned Aerial Vehicles (UAVs).

In the mentioned practical problems, there are several distinctive properties which more precisely frame the context. Firstly, it is plausible to presume that the members of the team always want to *cooperate*, as there are only common goals and there are no rational reasons to act competitively, or even adversarially. Secondly, the plan repair techniques have to be flexible enough to be usable in various domains. Such *domain-independent* approaches are less prone to a bias to particular domains and overspecialization of the planners. Furthermore, they provide universal solutions presumably usable even in newly discovered problems and reusable in wider number of related tasks. Domain-independent solutions are more effectively developed as they are more likely to solve problems for a larger community. Lastly, the problems caused by a nondeterministic environment, which are the very causes for plan repair in the first place, are realistic, *e.g.*, wheels of a mobile robot can slip, causing an executed `drive` action to fail to move the robot at all, however executing the same `drive` action displacing the robot from one city to another across a country is not realistic in real-world related problems. Additionally, as in principle, the disturbances cannot be precisely mapped on the real environment, the *failure model* has to be treated as *unknown*. These distinctive properties define the problem of this thesis as:

> *Cooperative domain-independent multiagent repair of plans with an unknown failure model.*

The repair process is carried out by a set of intelligent entities. Such entities can be any computational systems able to interact with the target environment. The computational system comprising the intelligent entities is presumed to be divisible into a couple of processes allowed to interoperate in form of message passing. The processes will be denoted as *agents* and message passing among them will be denoted as inter-agent *communication*. The focus of this thesis is on the deliberative agents working with plans, therefore the interface between an agent and the environment requires *execution of actions* and *perceiving of facts* valid in the environment. In effect, this means the interface can have a human behind who executes the actions by driving a real truck in the real world or a robotic subsystem carrying out the execution of the actions by motors moving robotic arms or a simulator of action execution in a virtual environment.

With this focus set on the computational agents and the communication among them, the motivation can be wrapped up with the *metrics* of the interest both for the practical and theoretical research of the plan repair algorithms. In classical planning, the *computational complexity* is the key metrics targeted by the research, however in decentralized systems and therefore also in the multiagent systems, the *communication complexity* gains in importance. This is especially true for a possible deployment of the algorithms on systems with low communication bandwidth among the agents like the aforementioned underwater robotic teams.

1.1 Problem Statement

The previous section defined the problem of this thesis in a declarative way, however the problem can be formulated more concretely as a question:

How to efficiently repair a multiagent plan after a divergence from ideal execution?

The question contains three distinctive parts which will be described in more details. From the right, the *divergence from ideal execution* means there is a process executing a multiagent plan which can fail and therefore the execution can diverge from a presumed ideal course. The *multiagent plan* describes actions the agents has to undertake to transform the environment they are acting in from an initial state to a common goal. The actions, initial and goal states describe the particular problem the agents solve. And finally, the core of the question is the notion of an *efficient repair* of the multiagent plan which is a process effectively modifying the plan in such way that the execution can continue with the repaired plan even after the divergence. The question does not specify what algorithmic approach the repair should follow, however the fundamental idea of the thesis is to utilize parts of the repaired plan, thus save required communication and computation effort which was already performed in the previous process of construction of the original multiagent plan.

To answer the stated question, a systematic approach was adopted. Firstly, the problem was split into several detailed and focused subproblems:

1. To formally define multiagent plan repair.
2. To design and to formally verify multiagent plan repair algorithms.
3. To experimentally verify required properties of the designed algorithms.
4. To validate the designed algorithms.

The first subproblem is to propose an appropriate formalization of the multiagent plan repair problems. The formalization should be based on the state of the art in multiagent planning to allow usage of previous techniques and implementations. Based on the formalization, the main problem is how to theoretically design approaches in form of formal definitions and practical algorithms to solve the problem of multiagent plan repair. Since the focus here is on the principle of plan reuse, the approaches should transform and supplement parts of the original plan resulting in a new fixed plans. These approaches has to be both theoretically and experimentally verified to comply the requirements of soundness, completeness and efficiency. Since the verification focuses on correctness of the solutions, the final question is if the proposed approaches are valid with respect to the motivation of the research, therefore the proposed algorithms have been tested in a particular application domain and with two different multiagent planning approaches.

1.2 Contributions and Accomplishments

This thesis compiles a series of work done on multiagent plan repair which were in the initial phases influenced by work of Jiří Vokřínek and Michal Pěchouček on task-oriented multiagent problem solving and social commitments [65]. The first results in this direction were published in [38, 37, 35] and in coauthored publications [63, 64, 69]. The work was from the beginning related to the particular domain of tactical and disaster relief missions of simulated autonomous assets. The contribution of these works was mostly in the sense of applied research and design of specialized solutions for particular domains with various forms of dynamism (usually with *a priori* unknown models). Generalization of this work led to an abstract representation of *multiagent plans by means of social commitments* in [36].

The work in [30, 31, 39, 34, 32], done in cooperation with Peter Novák, was firstly focused on definition of novel theoretical foundations for domain-independent multiagent plan repair principles. Based on work of Brafman and Domshlak [7] a *formal definition of multiagent plan repair* was presented in [30] with preliminary algorithmic approaches to multiagent plan repair. The formal framework and the algorithms were precised in [31, 33]. The final iteration presented *design and formal validation of multiagent plan repair algorithms* in [34] with proofs of soundness and completeness of the algorithms. The article also extended the *experimental validation of the algorithms* from the previous publication with an extensive results from various domains with an extended and optimized version of the planner. The experimental evaluation was precised in [32] with an additional plan repair algorithm connecting and generalizing the previous ones.

The applied part of the work focused on adaptation of plan repair for highly-detailed simulations of the real world. In [54], the simulation was described with a particular domain of a tactical mission. Furthermore, the publication proposed first steps towards a development process for seamless deployment of theoretically backed algorithms into simulated environments similar to the real world. Detailed description of the *deployment process* together with summary of requirements on a suitable *high-fidelity simulator* were presented in [39] together with evaluation of the process.

Most recently, the work focused on design and analysis of different techniques for multiagent planning, as the experiments revealed scalability issues with the used multiagent planning technique. Preliminary results of this work were published in cooperation with Michal Štolba in [62].

The main contribution of this thesis is a compact and detailed presentation of the results from the mentioned publications. The experimental parts were reformulated into an coherent form and presented with several unpublished details. Additionally the thesis provides an interconnection of the applied and theoretical results.

The most notable unpublished parts of this thesis are theoretical complexity analysis of the plan repair algorithms in Section 4.5, detailed description of the modifications of the used multiagent planner in Section 4.6, more detailed description of evaluation of the plan repair deployment in Sections 6.3, 6.4 and 6.5 and details on the newly designed planner in Chapter 7.

1.3 Organization

The thesis is organized into three main parts. The first one in Chapter 2 focuses on a summary of works in areas of multiagent planning and centralized plan repair, defining the state of the art for the context of the research in this dissertation. The part is concluded with definition of the context of the multiagent plan repair problem and summarizes the presumptions and limitations for the context of this thesis.

The second part establishes the theoretical foundations required for the latter formal definition of the plan repair algorithms and their theoretical analysis and experimental evaluation. In Chapter 3, the problem of multiagent plan repair is defined formally. Chapter 4 begins with statement of the main hypotheses of the thesis and continues with definition of the plan repair algorithms with soundness and completeness proofs. A unifying approach is presented in Section 4.4. The complexity analysis and implementation details are presented in Sections 4.5 and 4.6 respectively. The last chapter of the second part, Chapter 5, provides an experimental evaluation of the stated hypotheses.

The last part in Chapter 6 and Chapter 7 presents two different validation approaches, one by application of plan repair using a designed software engineering approach into high-fidelity simulation of tactical missions and the other by usage of a different planner than the one used in the core experiments of the work.

The thesis is concluded with a summary of achievements and directions for future research stemming mostly from the theoretical analysis of the algorithms and the synthetic evaluation.

Chapter 2

Related Work

Multiagent plan repair as defined in this thesis is a problem of synthesis of multiagent plans ideally reusing parts of an older plan generated by a multiagent planning process. Before a precise definition of what plan repair is, a strong preliminaries and groundwork are required. This chapter provides a summary of the state of the art building the required background for the next chapters defining multiagent plan repair formally and proposing the algorithmic solutions.

Multiagent plan repair builds upon two main areas of research: *multiagent planning [7]* and *classical plan repair* [48]. Both of these areas build on *classical planning* with its long history of thorough studies of plan generation for deterministic domains both in theory and practice [46]. Generalization of the planning research areas is *distributed planning* and generalization of classical plan repair is coined here as *distributed plan repair*. Relations among the areas are outlined in Figure 2.0.1.

The research in domain-independent *multiagent planning* [7] assumes a team of agents working together to come up with a set of cooperative plans executable by the respective agents which planned them. Additionally, the motivation of the multiagent planning is in the possibility to

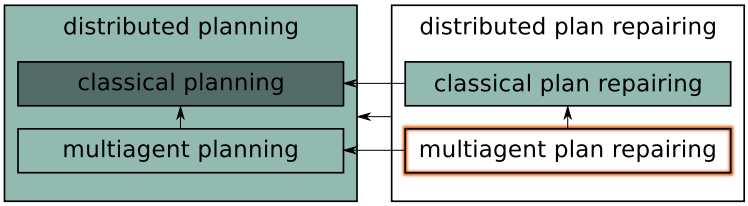


Figure 2.0.1: Related work as the context of the related research for this dissertation. The focus of this work is in the orange field. The darker the areas are the more they were already studied. The arrows represent dependence and influence among the areas.

factorize (partition) one large planning problem to more planning agents which can work only with a subset of the information and therefore preserve some private knowledge about the planning problem. In Section 2.1, the state of the art in multiagent planning is summarized.

Classical plan repair approaches arose together with the research on plan generation in classical planning. These approaches had weakened one restriction of the classical planning—the environment was not fully deterministic. Initially, plan execution and execution monitoring were studied in such environments. If the monitoring component detected that the current plan cannot be executed any more, some kind of correction mechanism had to be run to fix the problem and allow the system to continue its execution. In Section 2.3, the classical plan correction mechanisms will be described in more detail.

Historically, classical plan repair inherited a centralized architecture and complete information about the planning problem from classical planning. Distributed planning and *distributed plan repair* can be understood as abstractions of their respective centralized and multiagent counterparts. Since the distributed forms do not pose any restrictions on the number of parallel computational processes solving the planning problem, they generalize centralized single-process classical planning. In multiagent planning, there is a restriction on the way, how the planning process is decentralized among the particular agents. These restrictions are part of the planning problem definition, as each action is mapped to one agent only. Each agent can use only its own actions for planning and later for execution, therefore it is a restricted form of the distributed planning as well.

Beside the mentioned work closely related to multiagent planning and classical plan repair, there are also several loosely related approaches solving similar problems.

Firstly, there is a wide area of research done in *domain-specific* planning techniques. In the case-based planning [61] plans or repairs are generated using a library of predefined plan elements. General partial global planning [11] provides a coordination framework for agents employing domain-dependent planners. TALPlanner [15] works with predefined domain-dependent temporal information which helps with complex coordination of agents with various temporal constraints. Distributed Hierarchical Task Network (HTN) [13, 17] planning uses domain-dependent receipts how to refine particular planning operators.

The second group contains approaches based on decentralized *Partially Observable Markov Decision Processes* (Dec-POMDPs) [5]. Such approaches are based on probabilistically described uncertainty in the environment. Solutions for such problems are in a form of policies prescribing which actions should the agents execute in particular states of the environment and the goals are defined in form of rewards. An input to a solver of a Dec-POMDP problem has to contain all probabilities of the possible transitions among the state describing the environment. This coerce a model of an environment for a Dec-POMDP problem to be known *a priori*.

The last group of related approaches is based on *game theoretical* research. The main differences are the same as for Dec-POMDPs, since the games in extensive form are generalization of Dec-

POMDPs [55]. Additionally, in game theoretic approaches, the agents can be considered not only cooperative, but also competitive or even adversarial. As the focus of the planning in this thesis is on strictly cooperative agents, the game theoretic approaches are overgeneralized for the scope of this work.

2.1 Multiagent Planning

Multiagent planning is a specific form of distributed planning. *Distributed problem solving and planning* is an inherent part of the research field of distributed artificial intelligence. Durfee introduced his distributed problem solving and planning [16] by a statement that it is a subfield of distributed artificial intelligence emphasizing *"the collective effort of the agents to solve the common problems"*.

One of the related strategies for distributed and therefore multiagent planning from the field of distributed problem solving is the *task sharing* approach and inspired coordination techniques as presented in [67]. The principle is based on passing of tasks from a busy agent to vacant agents. The task sharing approaches in general require a hierarchy of agents. Such hierarchy can be predefined, as presented for instance in [16], or dynamically built during the process of problem solving as showed in [67]. The hierarchical approaches are not effective in strongly heterogeneous multiagent systems. In such systems, each agent has different abilities and thus a managed organizational structure is more effective [56]. Additionally, the task-based approaches need an auxiliary synthesis phase which merges plans of the particular agents into one global plan. Technically, several agents are responsible for integration of the partial results of other members of the team. In the hierarchical approaches the agents higher in the hierarchy merges the results of their subordinates. The multiagent planning used for the proposed plan repair algorithms will be strictly based on the dynamic organizational structure which is dynamically established by solving the coordination subproblem during the planning.

In situations where deliberative agents or generally computational processes require planning, it is quite obvious the planning problem has to be solved in a distributed manner. In [16], there is an overview of techniques solving such *distributed planning* problems.

The basic elements of (distributed) planning are *states* and *actions*. States describe the planning environment in particular configurations and actions prescribe possible ways, how the states can be transformed into each other. States and actions together describe a state transition system of a planning problem. One of predefined *goal states* is required to be the final state after execution of a sequence of actions beginning with a predefined *initial state*. Such sequence of action is denoted as a *plan* and it is a solution to a (distributed) *planning problem.*

Each distributed planning problem has to be appropriately represented. In [46], one of the popular *planning representations* presented is the classical representation. It uses a concise representation of the actions in the form of *operators*, such planning problems are also denoted as in the

lifted form. In contrast to actions, operators can have non-zero arity and therefore can represent exponential number of actions. The classical representation is based on predicate logic. Elements of the environment are substituted by constant symbols, relations among the elements are described by predicates, and a particular state is a set of grounded atoms. Actions are instances of operators and an operator in the classical representation is a triplet

$$o = (\mathsf{name}(o), \mathsf{pre}(o), \mathsf{eff}(o)),$$
$$\mathsf{name}(o) = n(x_1, \ldots, x_k),$$

where the name consists of an operator symbol n and its parameters in the form of variable symbols x_i. The preconditions $\mathsf{pre}(o)$ are set of literals which must hold in a state in order to allow usage of the operator and the effects $\mathsf{eff}(o)$ are literals that become true in the transformed state after usage of the operator.

The proposed formalization in this thesis will be based on the STRIPS representation [19]. The importance of STRIPS does not lie only in the particular algorithm presented in [19]. From today's perspective, the representation used for description of the STRIPS planning problem is probably more important, therefore the term STRIPS is commonly used for planning problems described by a set of *facts* describing a state, actions with preconditions described by those facts, sets of facts to add and delete after action execution and a set of goal states described as a set of facts required to hold in the final state of the plan. In the original Fikes and Nilsson's work, STRIPS was described in the lifted form, that means by operators with parameters. For this thesis, the term STRIPS is important in the sense of referring to the representation and formal language of the input to a STRIPS-compatible planner, because the formalization of the plan repair problems is built upon it. The formalization is based on classical representation, each action is consisting of a label and three groups of expressions which are called: *preconditions* $\mathsf{pre}(o)$, *deletions* $\mathsf{del}(o)$ (referring to negative effects of $\mathsf{eff}(o)$), and *additions* $\mathsf{add}(o)$ (referring to positive effects of $\mathsf{eff}(o)$) and a transition function which defines how an action applied in a state changes the particular facts, formally $s_{i+1} = (s_i \setminus \mathsf{del}(o)) \cup \mathsf{add}(o)$, where s_i represents a state at step i.

Regardless the particular representation, the distributed planning problems can be according to [16] described in one of the following forms: centralized planning for distributed plans, distributed planning for a centralized plan and distributed planning for distributed plans.

In the first case of *centralized planning for distributed plans*, the problem is to generate a set of plans for particular agents in the environment. Since the agents can work simultaneously, the plans need to be executed in parallel. This requirement is fulfilled in *partially ordered plans* as the ordered sets of the actions act as a plan for the particular agents. A domain-dependent HTN planner where decomposition can involve other agents [68] is an example of this form of distributed planning.

The *distributed planning for a centralized plan* can be described as a cooperative planning of

various special planners. In strongly heterogeneous environments, it is a well-founded assumption that one planning principle is not enough. An effective heuristic can be known for one problem and an expressive representation for another. In such case, the problem can use different planners for different subproblems. An example can be a STRIPS planner planning a high-level overall plan and a special path planner which refines movement actions.

Distributed planning for distributed plans is a synergy of the two previous principles and it is most suitable for the multiagent planning as it combines planning by more distinctive processes represented by the agents, for particular elements in the environment embodied by the same agents. The first work formally defining *deterministic domain-independent multiagent planning* was written by Brafman and Domshlak [7]. According to the Durfee's forms of distributed planning, it fits to the last and most challenging group of distributed planning for distributed plans, however, in the contrast to distributed planning as defined by Durfee, the Brafman and Domshlak's approach added several specific presumptions.

The key distinction of multiagent planning as defined by Brafman and Domshlak is precise separation of private and public knowledge of the agents during planning. That means, actions and facts describing the environment in the planning process can be known only by one particular agent which makes them effectively private or known by more than one agent making them public. The requirement for *preservation of private knowledge* cannot be solved otherwise than by distribution of the planning process in such way the particular private knowledge stays with the agent which it is private for. There cannot exist a trustworthy central authority as, by definition, private knowledge is defined only by one agent knowing it, therefore any other authority had to be understand as another agent and the information cannot be private if it is shared even with a trustworthy authority.

The private and public distinction as defined in [7] has another additional benefit in possible simplification of the planning problem for loosely coupled domains. Such plan factorization approaches are not new in classical planning as a technique for simplification of several planning problems [6]. The specificity of the multiagent planning is in the way how the factorization has to be done. In the factorization approaches of classical planning the key variable is how the problem is factorized, however in the multiagent problems, the factorization is naturally defined by the abilities of the agents which dictates precisely how the factorization has to be done. The price for this simplification is a decreased generality of the factorization approach.

The only implementation of a deterministic domain-independent multiagent planner available in the time of realizing the research described in the following chapters was from Nissim, et al. [51]. The planner uses a *Distributed Constraint Satisfaction Problem (DisCSP)* solver for coordination of the agents. From the perspective of Durfee's distributed problem solving, the DisCSP algorithm would be responsible for the task sharing and the synergy phases. The construction of the local plans is done by a forward-chaining heuristic search based on the STRIPS formalization. Current state-of-the-art forward-chaining heuristic planners from the perspective of efficiency in computation time

and plan quality are based on various types of complex automatically extracted heuristics together with relatively basic search techniques using variations on the textbook version of A^* and *Best-First Search* (BFS) algorithms. Relationships among such heuristics and their mutual differences and similarities (incl. dominance of some heuristics against others) are presented in [23]. Comprised heuristics in the used multiagent planner by Nissim, et al. are *Fast-Forward*, *helpful actions* and *landmarks* heuristics.

As mentioned in the previous paragraph, multiagent planner used in this work utilizes DisCSP for solving the coordination subproblem. This principle goes back to a neoclassical planning approach based on a centralized form of *Constraint Satisfaction Problem (CSP)* solving as a combinatorial search. An overview of approaches for planning-as-CSP can be found in [3]. State-of-the-art models for classical planning-as-CSP are presented in [4]. Notable models are (i) GP-CSP [14], and (ii) CSP-PLAN[44]. In the next section, a basic CSP-based planning approach is described in more details.

2.2 CSP-based Planning

One of the neoclassical planning techniques is based on utilization of compilation of a planning problem into a *Constrains Satisfaction Problem* (CSP) [44]. Benefits of such approach are both on theoretical and on practical basis. The theoretical results can help with complexity studies borrowed from the research of CSP solving complexity and the practical results provide ready-to-use implementations and tested algorithms for CSP solving usable for planning problems after the compilation.

CSPs use variable-based representation similar to the state-variable representation in classical planning [46]. There is a set of variables, each accompanied by a domain of possible values. The definition of the variables is supplemented by a set of n-ary constraints restricting possible combinations of the values of the variables. A solution of a CSP is an complete assignment of the values to their respective variables not violating any of the constraints.

The compilation from planning to CSP solving is an iterative process with successively incrementing length of the prospective solution. The process begins with a CSP for finding a plan of one action and if such solution cannot be found, it continues with two actions, three actions and so on.

The resulting CSP variables form layers of two types: (i) state layers and (ii) action layers. Firstly, there is only one CSP variable in the action layer with a domain of all actions in the planning problem and a set of CSP variables representing facts which holds after application of that action. In the next increment, there are two action layers of the CSP variables representing a sequence of two actions in a particular order in the resulting plan. The two action layers are interleaved with two variable layers representing the facts holding after application of the first and

the second action respectively. There are four types of constrains representing the relations among the layers, particularly among the actions and the facts from the neighboring layers:

- constraints describing the initial state and the related preconditions of the first used action;
- constraints describing the effects of the last action and the related required goal facts;
- constraints on the effects and preconditions of neighboring actions;
- constraints assuring the frame axiom[1].

If the compiled CSP has a solution, the particular values of the variables in the action layers straightforwardly determine a plan solving the original planning problem. If a solution cannot be found the length of the plan is incremented by one and the search continues with a newly created CSP for a longer plan.

The same principle described in this section for CSP-based planning can be used for DisCSP-based planning as well. Additionally, the principle can be restricted only to a subset of actions and facts. If such subset represents only the public information in a multiagent planning problem, the DisCSP describes the coordination planning subproblem as described in [51].

2.3 Classical Plan Repair

One of the first papers proposing a unified and well defined framework for planning, plan execution, monitoring and repair is from 1988 written by Ambros-Ingerson and Steel [1]. The framework was called IPEM (Integrated Planning, Execution and Monitoring) and it was based on IPE framework integrating only planning and execution. The framework was based on the TWEAK partial plan representation (similar to nowaday HTNs [17] or Universal Classical Planning (UCP) [27]).

Phases of the planning process included *synthesis*, *inspection*, *modification*, and *execution*. The first phase described building of the plan. The inspection phase provided validation and verification of the plan considering the requested requirements. In cases, the plan become inconsistent with the environment, it had to be fixed w.r.t the unanticipated state of the environment by plan adaptation. Finally, the plan directed behavior of the system in the environment during the execution phase.

An important highlight for the context of this work is that the framework inherently included an early form of a plan repair algorithm within the adaptation phase. The algorithm was used to adapt to both dynamism of the environment and to execution failures. The plan repair process was based on five operations: (i) reduction prior, (ii) reduction parallel, (iii) reduction new, (iv) linearize and v) expand. These operations were used according to the type of the unexpected event during the execution. The reductions represented three ways of adding new actions into the

[1] A successor fact layer preserves all values of the variables not affected by any executed action(s) in the previous layer.

plan if an action has unsupported preconditions, linearization represented reordering of actions and expansion replaced actions with another action segments.

An intensively cited paper [48] by Nebel and Koehler from 1995 proposed a first theoretical analysis formally investigating the complexity of planners reusing parts of the old plans. The result of the paper stated:

> *"It is not possible to achieve a provable efficiency gain of [plan] reuse over [plan] generation."*

Additionally, the paper provided proofs that plan reuse can be strictly more complex than plan generation from scratch, assuming conservative plan modification. These results do not imply there cannot exist a plan reusing or repair algorithm with an increased efficiency, it rather shows there are caveats of using such techniques in general.

As a formal instrument to build the proofs, the paper used a principle similar to regular expressions describing possible modifications of the original plan. The model is called MODDELINS after the three possible changes in the plan: (i) action sequence modification, (ii) action sequence deletion and (iii) insertion of a new action sequence. The proposed plan repair algorithms in this dissertation builds upon these principles.

The paper from Nebel and Koehler somehow decreased interest of the planning community in study of the plan repair techniques, since it showed that a general plan repair algorithm cannot be more efficient than plan generation. However, the need for planning techniques usable in real-world conditions stayed and stay to the present day. The main difference between the classical planning problems and the real-world planning problems can be summarized as (i) uncertain action effects, (ii) dynamic environment and (iii) incomplete information. A paper [12] summarizing these needs was written by des Jardins et al. *Distributed, Continual Planning* (DCP) defined in this paper describes a set of techniques tackling such problem. Since the conditions defined for DCP differ from the classical planning, the paper [12] stated that:

> *"Plan-repair methods in current systems are typically at either the reactive level or the generative level; methods for smoothly integrating plan-repair techniques at multiple levels of abstraction and varying time scales are needed."*

Without a bigger reception by the planning community, Au et al. published a paper [2] in 2002 opposing, in a sense, the paper from Nebel and Koehler [48]. The principle of plan adaptation (usable also as a plan repair technique) described in the paper used a principle called derivational analogy. The *derivational analogy* takes into account not only the goal of the process, but also the history of the problem solution. In the plan adaptation context, it means that the construction process of a plan is used later in the process of plan adaptation. An important statement of the paper was that plan adaptation (repair) by derivational analogy is more efficient then planning from scratch (in special cases, plan repair by analogy has logarithmic complexity and planning from

planning fundamentals	classical planning states, actions, operators, init and goals
planning representation	classical with multiagent extension of STRIPS
action ordering in plans	partial ordered plans with restrains on simultaneous actions
distribution form	distributed planning and plan repair for distributed plans
problem factorization	multiagent based on separation of private and public knowledge

Table 2.1: Summary of planning-related approaches backing multiagent plan repair as defined in this thesis.

scratch is PSPACE-complete). This statement does not disprove the conclusions of the paper [48], because those were restricted only on conservative plan changes. The plan adaptation by the analogy is not based on the conservative plan changes, although for a long time, the conservative changes were considered as the most efficient.

A long track of publications [40, 42, 41] pushing forward the plan repair research in the context of the DCP was produced by van der Krogt and his colleagues in the years from 2004 to 2006. The papers particularly address the statement from the paper by des Jardins et al. calling for smooth integration of plan repair techniques, planning, plan execution and monitoring.

Most of the mentioned works in the previous paragraphs were based on the UCP techniques, particularly HTN planning. Serina with colleagues published in [21] an extension of their graph planner moving the research towards domain-independent plan repair. The extension of the *Local search for Planning Graphs* planner (LPG) by plan adaptation (LPG-adapt) directly uses repairs of the planning graph, in contrast to the other approaches which repairs the resulting plans.

2.4 Context for Multiagent Plan Repair

The previous sections stake a boundary of the research this thesis builds upon. Proposed multiagent plan repair utilizes research done in this areas in various forms (see a summary in Table 2.1).

Classical planning is domain-independent but centralized, however even so, it plays a crucial role in the planning research as a whole and provides the fundamental basics. Furthermore, the multiagent plan repair uses extension of the classical planning representations in form of the STRIPS formalization and problem definitions. Used plans prescribing behavior of the agents are in a restrained form of plans with a simultaneous actions in time steps of the execution and of the same lengths for all participating agents. The plans are fully instantiated, therefore both the used planner and repair algorithms have to produce sequences of grounded operators, *i.e.*, the (primitive) actions. All this is accordingly to the definition of multiagent planning by Brafman and Domshlak. Motivation is to preserve distinction between the private and public actions, since the uniqueness of the private knowledge argument holds for multiagent plan repair as well. According to Durfee's typology of distributed planning, motivation behind the proposed approaches is distributed plan

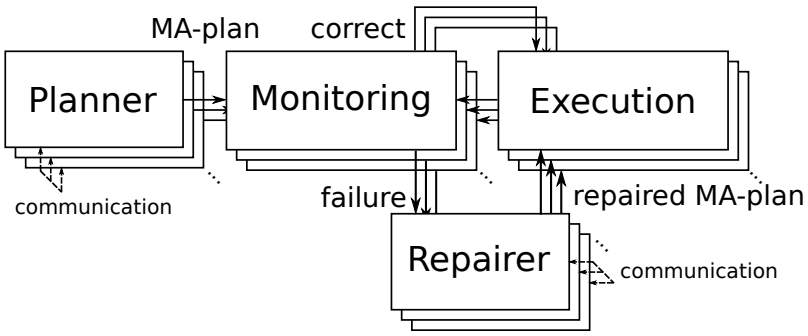


Figure 2.4.1: The plan execution and monitoring architecture—each layer is for each agent. After the initial planning, the multiagent plan (MA-plan) is passed on to the monitoring components which check if the current state concur with the plan and if so the plan is passed on to execution of the first actions in the plan. After execution, the plan is passed on again to the monitoring component and the process is repeated. If an failure is detected, the plan is repaired and after the repair the execution continues.

repair of distributed plans. Such approach preserves private knowledge during plan repair and allows agents in the team execute their specific actions leading them towards the common goals.

As the agents are in a dynamic and uncertain environment, actions and plans may not always lead to desired consequences or can turn out to be not executable. To account for such cases, a cautious agent must be able not only to execute its actions, but also monitor its own progress and detect failures during execution of its actions. Generally speaking, besides a planning component, implementation of an agent should also include monitoring and a plan repair component. Figure 2.4.1 depicts a generic multiagent plan-execute-monitor architecture. More concretely, considering a multiagent plan produced by a suitable multiagent planner, the abstract execution-monitoring algorithm checks in every state the soundness of the next step before advancing. If necessary, it invokes a plan repair procedure.

Creating an effective plan repair procedure has its limitations and caveats as Nebel and Koehler showed. To minimize such risks in this thesis, the plan repair algorithms were created independently on a particular multiagent planner and possibly utilizing its benefits for multiagent repair as well as for multiagent planning. The planner used in the rest of the thesis (with the exception of Chapter 7) is based on a compilation of the coordination part of a multiagent planning problem to a DisCSP problem and a forward-chaining planning for the individual plans of the agents. Primal authors of the planner are Nissim and his colleagues [51], however the experimental work described in Chaper 5 required nonnegligible amount of additional work, namely bugfixing and efficiency optimizations as presented in Section 4.6. This planner was used both as the initial planner and as the planning component in the repair algorithms. The proposed plan repair algorithms are based on the principles of the MODDELINS modifications [48] and bring them to the multiagent setting.

In the context of the classical plan repair research, this dissertation tries to answer:

> *What are the particular caveats of the plan reuse techniques based on* MODDELINS *from* [48] *extended into multiagent setting?*

In other words, when it is beneficial to use plan repair techniques based on conservative plan changes. The derivational analogy approach can be understood as a similar approach eligible for domain-dependent planning techniques as HTN planning.

Finally, to frame the context of the multiagent plan repair problem precisely, a summary of presumptions and limitations considered in this dissertation follows:

- The planning *domain* and the planning *problem are static* for one execution run. That means that no new actions and agents can appear or disappear during the execution.
- The environment is *fully observable*, *i.e.*, monitoring of changes in the environment is for all agents granted and instantaneous. The planning processes of the particular agents are not considered as part of the environment.
- *Communication is immeasurably faster than execution* of any action. In effect, there cannot be any interferences between duration of action execution and message passing duration.
- *Communication is perfect.* All sent messages are delivered and without any change of their contents.
- All *plan repair algorithms use a multiagent planner* as an inseparable component.

Chapter 3

Formal Foundations for Multiagent Plan Repair

One of the first requirements preceding the research presented in this work was a formalism allowing to describe all the theorized plan repair algorithms and helping to precisely formulate several properties of the solutions in general. This chapter describes its last form with all the necessaries for the latter description of the particular algorithms.

3.1 Multiagent Planning

A first step towards repair of plans for agent teams leads to description of such plans and techniques for their building. For the context of this work, the problem of multiagent planning is treated as an extension of the classical singleagent planning in the manner adapted from MA-Strips planning in [7] with a planner utilizing a DisCSP solver and singleagent forward-search planner.

The definition begins with a team of *cooperative* and *coordinated* agents 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 multiagent 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 adaptations thereof constitute the preliminaries enabling to state the core hypotheses, as well as provide the necessary background for the algorithms and their proofs introduced later in Chapter 4.

A *state* $s \subseteq \mathcal{L}$ is a set of atoms from a finite set of propositions $\mathcal{L} = \{p_1, \ldots, p_m\}$. p *holds* in s, given $p \in s$, otherwise p *does not hold* in s. In that sense, states are complete. That means, 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* (or simply an *action*) an agent can perform in an environment is a triplet $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$, where a is a unique action label and $\mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)$ denote the respective 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 furthermore, it is assumed there is a distinguished empty action $\epsilon = \langle \emptyset, \emptyset, \emptyset \rangle \in Act$ with no preconditions and no effects. Whenever $\mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \subseteq \mathcal{L}$, a is defined over $\mathcal{L}$.

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 that $\mathsf{add}(a) \cap \mathsf{del}(a) = \emptyset$ is not required, as it is assumed that the effects negate each other strictly according to the definition of $\oplus$. Furthermore, $\oplus$ is left-associative, hence $s \oplus a_1 \oplus \cdots \oplus a_k$ can be written.

Similarly to the transformation operator $\oplus$, it is defined a reverse-transformation operator $\ominus : \mathcal{S} \times Act \to \mathcal{S}$ for a single action as $s \ominus a = (s \cup \mathsf{del}(a)) \setminus \mathsf{add}(a)$. The operator is left-associative as well, therefore $s \ominus a_1 \ominus \cdots \ominus a_k$ can be written.

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 *multiagent planning problem* is a quadruplet $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$, 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. That is, $\alpha_i \cap \alpha_j = \{\epsilon\}$, 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.

Before formally defining the notion of a solution to a multiagent planning problem, a sequel of auxiliary notions has to be introduced.

Given an agent α, a *singleagent 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 $a_i \in \alpha_i$ 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 $s \oplus \mathbf{a}$ can be used. At this point, the definitions do not 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, such joint actions will be commented on in more detail.

Definition 2 (multiagent plan)**.** Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be a multiagent planning problem with $\mathcal{A} = \alpha_1, \ldots, \alpha_n$. A *synchronous multiagent 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.*, $|P_k| = |P_l|$ for all $1 \leq k, l \leq n$. Additionally, $|\mathcal{P}| = |P_i|$ for every $1 \leq i \leq n$, denotes the length of the multiagent 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 $s_{k+1} = s_k \oplus \mathbf{a}_k$ with $\mathbf{a}_k = (P_1[k], \ldots, P_n[k])$ for all $1 \leq k < m$ and finally
3. $\mathcal{P}$ reaches the goal S_g, *i.e.*, $s_m \in S_g$.

The statement will be also used in an alternative form: $\mathcal{P}$ solves the problem Π. Finally, $\mathit{Plans}(\Pi)$ denotes the set of plans which are solutions to a given multiagent 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 to introduce the following plan-matrix notation for a multiagent plan $\mathcal{P}$, with $a_{ij} = \mathcal{P}[i, j]$, providing a more visual understanding of the defined multiagent plans used in the rest of the dissertation:

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

Using the definition of a multiagent plan, operations on such plans can be defined. Two multiagent 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 holds $\mathcal{P}_1[i, j] = \mathcal{P}_2[i, j]$.

A concatenation of two multiagent 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 holds $\mathcal{P}[i, j] = \mathcal{P}_1[i, j]$ if $i \leq |\mathcal{P}_1|$ and $\mathcal{P}[i, j] = \mathcal{P}_2[i - |\mathcal{P}_1|, j]$ for $i > |\mathcal{P}_1|$. Concatenation of multiagent plans is left-, as well as right- associative operation, so $\mathcal{P} = \mathcal{P}_1 \cdot \mathcal{P}_2 \cdot \cdots \cdot \mathcal{P}_n$ can be written.

Given a multiagent 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 multiagent 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 $\mathcal{P}$, that is, $\mathcal{P}[i..\infty] = \mathcal{P}[i..|\mathcal{P}|]$. $\mathcal{P}_1 \cdot \mathcal{P}_2$ is said to be a *decomposition* of a multiagent plan $\mathcal{P}$ iff $\mathcal{P} = \mathcal{P}_1 \cdot \mathcal{P}_2$.

Using the notion of a multiagent plan, two directions of proposition propagation can be defined. Since the process extends the transformation operators, the same symbols will be used in extended forms $\oplus : \mathcal{S} \times (Act \times Act \times \cdots) \to \mathcal{S}$ and $\ominus : \mathcal{S} \times (Act \times Act \times \cdots) \to \mathcal{S}$ as follows:

Definition 3. (proposition propagation) Let S' be a set of propositions propagated from a set of propositions S using a multiagent plan $\mathcal{P}$ denoted as $S' = S \oplus \mathcal{P}$ iff $S' = S \oplus \mathcal{P}[1] \oplus \mathcal{P}[2] \oplus \cdots \oplus \mathcal{P}[m]$, where $m = |\mathcal{P}|$.

Definition 4. (proposition back-propagation) Let S' be a set of propositions back-propagated from a set of propositions S using a multiagent plan $\mathcal{P}$ denoted as $S' = S \ominus \mathcal{P}$ iff $S' = S \ominus \mathcal{P}[m] \ominus \mathcal{P}[m-1] \ominus \cdots \ominus \mathcal{P}[1]$, where $m = |\mathcal{P}|$.

Given two multiagent plans $\mathcal{P}_1$ and $\mathcal{P}_2$, $\mathit{diff}(\mathcal{P}_1, \mathcal{P}_2)$ will denote 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*. $\mathit{diff}(\mathcal{P}_1, \mathcal{P}_2)$ corresponds to *Levensthein distance* [43], in literature also referred to as the *edit-distance*, between two strings corresponding to the sequences of actions of the individual plans. Adaptation of the notion of Levensthein distance between two multiagent 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 singleagent plans described in [48].

To introduce the MA-Plan algorithm for solving MA-STRIPS problems as formulated in [7], finally a distinction between the public and private actions of individual agents has to be defined. 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 an action a or an agent α respectively. Given a multiagent 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 $Act^{pub}_{\alpha} = \{a \mid a \in \alpha \text{ and } \mathsf{atoms}(a) \subseteq \mathcal{L} \setminus \mathsf{atoms}(\alpha)\}$ and denoted also as Act^{pub} if the agent context is clear. 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 effectively defining the multiagent factorization of the problem. 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 multiagent team members. The algorithm MA-Plan for solving a planning problem Π can be thought of in two interleaving stages until a suitable multiagent 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 multiagent 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 [7] lies in the observation that the MA-Plan algorithm can be implemented by reduction of the first stage to a constraint satisfaction problem (CSP). 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 multiagent plan) corresponding to a multiagent 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 holds 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 $1 \leq k_p < k$ (there is some previous action which causes p to hold), or $p \in s_0$ in which case $k_p = 1$ is set; 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 multiagent 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 multiagent planning problem $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ satisfies the *internal planning constraint* iff for every agent, the corresponding singleagent planning problem with landmarks $\{a \mid a = \mathcal{P}[k, i] \in Act^{pub}\}$ is solvable, meaning a singleagent planning algorithm is able to fill in the gaps between the public actions in the candidate multiagent 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 $k_p \leq k' \leq k$ in the second condition of the coordination constraint, together with the definition of public actions, which ensure the local consistency of joint actions.

Algorithm 3.1 MA-Plan(Π):

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

```
1: δ = 1
2: loop
3:    construct CSP_{Π;A}
4:    if solve-csp(CSP_{Π;δ}) then
5:       reconstruct a plan P from a solution for CSP_{Π;δ}
6:       return P
7:    else
8:       δ = δ + 1
9:    end if
10: end loop
```

Algorithm 3.1 lists the original multiagent planning algorithm MA-Plan by Brafman and Domshlak in [7]. The algorithm iterates through CSP formulations of the planning problem according to δ, informally the number of coordination points between the agents in the multiagent team. That means, δ determines the number of joint actions in a candidate multiagent plan containing public actions. Filling the gaps between the individual singleagent public actions, if possible, then gives rise to the overall multiagent plan. In the case such a plan completion does not exist, the process continues by testing longer candidate plans (as restated in Section 2.2), possibly not terminating in the case where no solution to the given multiagent planning problem exists.

The original multiagent planning algorithm assumes a centralized planning architecture. It is a centralized planning algorithm computing multiagent plans for a team of agents which are supposed to be subsequently executed in a decentralized fashion. Motivation of this work is however a decentralized planning followed by a decentralized plan execution and prospective decentralized plan repair.

In [51], Nissim et al. adapted the original blueprint algorithm from [7] to a distributed setting. The adaptation rests on formulating the multiagent 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, the implementation of the multiagent planning algorithm MA-Plan will refer to its decentralized version as described in [51].

3.2 Multiagent Plan Repair

Based on the required formalization of multiagent planning, the formal definitions of multiagent plan repair can be proposed.

Consider a multiagent planning problem $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ and a plan $\mathcal{P}$ solving Π. Further-

more, consider an environment in which, apart from the actions performed by the agents of the team $\mathcal{A}$, no other exogenous events occur. 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 $s_{k+1} = s_k \oplus \mathcal{P}[k]$ (see 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 $\mathcal{P}[k]$ from the plan $\mathcal{P}$ does not result in precisely the state s_{k+1} as defined above. That is at step k an unexpected event occurred in the environment. For simplicity, only unexpected events happening exclusively in the course of execution of some action are considered (as if it took a non-zero time), not such which could occur while the agent is deliberating the execution (as if the deliberation was instantaneous). These presumptions were specified in Section 2.4.

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 multiagent planning problem Π is such, when the state s_f resulting from an attempt to perform the action $\mathbf{a} = \mathcal{P}[k]$ does not satisfy some of the positive effects of $\mathbf{a}$, that is, $\mathsf{add}(\mathbf{a}) \not\subseteq s_f$.

A *strong failure* of execution of the plan $\mathcal{P}$ at step k w.r.t. the planning problem Π occurs whenever the k-th action of $\mathcal{P}$ cannot be executed due to its inapplicability. It means, the execution of the plan up to the step k resulted in states $s_0, s_1 \ldots, s_k$, possibly with some weak failures occurring in the course of execution of the plan fragment and $\mathcal{P}[k]$ is not applicable in s_k.

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 multiagent 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, which happens, 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, a plan execution monitoring process has to determine some plan execution failure at a step k which results in some failed state s_f.

Definition 5 (multiagent plan repair)**.** Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be a multiagent planning problem. A *multiagent plan repair problem* is a quadruple $\Sigma = (\Pi, \mathcal{P}, s_f, k)$, where $\mathcal{P}$ is a multiagent 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.

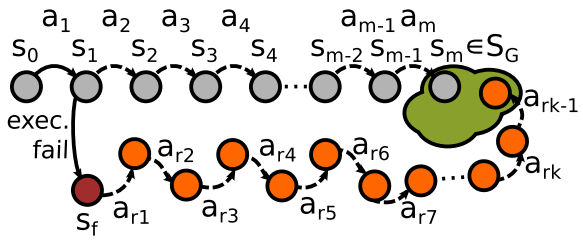


Figure 3.2.1: A visual representation of the replanning principle. The gray nodes represent planned states of evolution of the environment, s_0 is the initial state and S_G is a set of possible goal states. The dashed arcs represent planned but not yet executed actions, the solid ones were already executed. Only the joint action $\mathbf{a}_1$ was executed without a failure. The execution of $\mathbf{a}_2$ failed and the environment ended in a distinctive state s_f. The orange nodes represent planned states of the replanned plan $(\mathbf{a}_{r1}, \ldots, \mathbf{a}_{rk})$.

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

Given two multiagent plans $\mathcal{P}_1$ and $\mathcal{P}_2$ both repair a multiagent plan $\mathcal{P}$ for a problem Π in a state s_f, $\mathcal{P}_1$ 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 multiagent 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,

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

Note that there might be several distinct minimal repairs of a given multiagent plan.

In general, the multiagent plan repair problem can be reduced to solving a modified multiagent planning problem and thus gives rise to a straightforward plan repair algorithm based on *replanning* (see Figure 3.2.1) in two steps: (i) construct the multiagent replanning problem Π' as prescribed in Definition 5, and subsequently (ii) utilize the MA-Plan algorithm to solve the problem Π'.

While the notion of minimal repair of multiagent 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. The additional metrics will be discussed in Sections 4.5 and 5.2.

Since the dissertation focus on multiagent plan repairing problems, which in a sense enforce coordination among the members of a multiagent team, a definition of an appropriate property is needed to indicate which planning problems tend to benefit from the plan repair approach. The following notion of *coordination frequency* formalizes the idea.

Definition 6 (coordination frequency)**.** Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be a multiagent planning problem with a solution $\mathcal{P}$. $\mathcal{P}$ is *δ-coordinated* iff it contains at least δ coordination points, that are, 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, then$\mathcal{P}$ 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 multiagent 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. It is simply a solution requiring minimal relative amount of coordination required to solve the problem. The notion of relative coordination frequency allows for comparison and ordering of multiagent planning problems according to the amount of coordination they minimally require for solving them. Informally, problems with relatively low $cf(\Pi)$ will be called *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. Multiagent planning problems with $cf(\Pi) = 0$ will be called *uncoordinated*.

Note, it still might be the case that even though a multiagent planning problem can be solved without any coordination $cf(\Pi) = 0$, 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.

As shown in [7], δ turns out to play an important role in time complexity analysis of the MA-STRIPS problem. Hypothetically based on the previous paragraphs the relative frequency of coordination points along the plans, seems to play a role in the communication complexity of plan repair as well. Plan repair for problems which require some coordination quite often along the plans should lead to reuse of fragments including relatively large number of coordination points, which do not have to be planned for again and thus should lead to reduction of required communication in the repair process.

Chapter 4

Plan Repair Algorithms

Before describing the plan repair algorithms the hypotheses will be stated more formally, based on the formalization from the previous chapter.

The idea behind the first hypothesis is based on an intuition that plan repair algorithms preserving larger parts of the original plan has to solve simpler problem and therefore will need less communication.

Hypothesis 1. *Multiagent plan repair approaches producing more preserving repairs than replanning tend to generate* lower communication overhead *for tightly coordinated multiagent problems.*

The second 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 the planning could be achieved.

Hypothesis 2. *Repair approaches minimizing the number of agents involved in the plan repair process tend to generate lower computational and communication overheads than other strategies.*

The third hypothesis is related to chains of interdependent actions. 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 rapidly increasing numbers of further failing actions.

Hypothesis 3. *Repair approaches reusing a suffix of the original plan generate lower computational and communication overheads than the repair algorithms reusing prefix of the original plan in domains featuring actions with long-term dependencies.*

The last research question treated in this dissertation is how do different combinations of the prefix and suffix preservation parameters influence the efficiency of the plan repair process. On

one hand, there can be a gap between the prefix and suffix reused parts of the original plan, which has to be filled by a result of the inner planning process, in other words the original plan was *underused*. Reversely, there can as well be an overlap, which has to be reverted, *i.e.,* the original plan was *overused*. These cases are in a sense pathological. In a consequence, the final hypothesis is proposed as:

Hypothesis 4. *Repair approaches overusing or underusing the original plan tend to generate higher computational overheads than other algorithms.*

In the following sections, four proposed plan repair algorithms based on the presented formalization will be presented.

4.1 Back-on-Track Repair

Unexpected event occurring in an environment can cause a failure in execution of a plan performed by a multiagent team in that environment. The result would be that the overall state of the system will not 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 multiagent planner to produce a plan from the failed state to the originally expected state and subsequently follow the rest of the original multiagent plan from the step in which the failure occurred. The following multiagent plan repair approach, coined *Back-on-Track* (BoT) repair (see Figure 4.1.1), is inspired by this idea, in fact a slight generalization of it.

Definition 7 (Back-on-Track repair)**.** Let $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ be a multiagent plan repair problem and $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ being the corresponding modified multiagent replanning problem.

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 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 that all plans from $\mathit{Plans}(\Pi')$ are Back-on-Track repairs of the original plan. The length of the preserved suffix of the original plan provides indication for 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 4.1 realizes a multiagent plan repair procedure according to the Back-on-Track plan repair principle. Since the MA-Plan algorithm searches for the simplest plan from the initial state

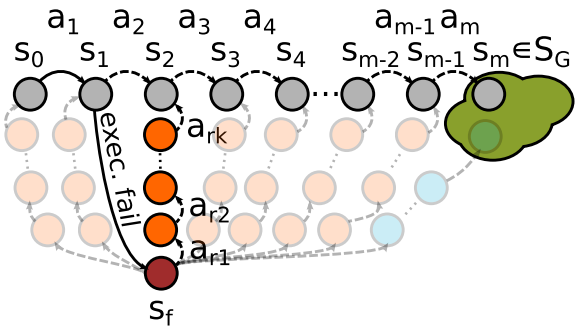


Figure 4.1.1: A visual interpretation of the Back-on-Track (BoT) plan repair approach. The nodes and arcs has the same meaning as in Figure 3.2.1. The translucent orange states with their respective actions represent various possibilities of the BoT repair plans. The cyan nodes represent a solution by replanning.

Algorithm 4.1 Back-on-Track-Repair(Σ)

Input: A multiagent 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 multiagent plan $\mathcal{P}'$ solving Σ, if a solution exists.

1: construct $\Pi_{back} = (\mathcal{L}, \mathcal{A}, s_f, \{s_0, \ldots, s_m\} \cup S_g)$
2: **if** MA-Plan(Π_{back}) returns a solution $\mathcal{P}_{back}$ **then**
3: retrieve the state s_j of $\mathcal{P}$ to which $\mathcal{P}_{back}$ returns
4: **return** $\mathcal{P}' = \mathcal{P}_{back} \cdot \mathcal{P}[j..\infty]$
5: **else**
6: **return** $\mathcal{P}' = \chi$
7: **end if**

to a goal state, the Back-on-Track-Repair computes plans which return back to the original one in the simplest 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 depends on invocation of the underlying multiagent planner, hence its correctness relies on the correctness of the underlying planner. The following lemma states the soundness of Algorithm 4.1.

Lemma 8 (Back-on-Track-Repair soundness)**.** *Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be a multiagent 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 multiagent 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 multiagent 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 multiagent planning problem, the Back-on-Track-Repair is able to find a solution to the corresponding multiagent plan repair problem.

Lemma 9 (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 8.*

If there exists a solution to the modified multiagent 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 replanning from scratch. □

The lemmas 8 and 9 establish how the Back-on-Track plan repair approach inherit its correctness from the underlying multiagent planner. Note however, the algorithm is only *partially complete*, because in cases when there is no solution to a given multiagent planning problem, it is not ensured that the algorithm MA-Plan terminates. Provided a totally complete multiagent planning algorithm, directly replacing MA-Plan, total completeness of the Back-on-Track-Repair algorithm could be straightforwardly established by the lemmas above.

4.2 Simple-Lazy Repair

The Back-on-Track multiagent plan repair approach tries to compute a new prefix to some suffix of the original plan and repair the failure by their concatenation. An alternative approach, coined *Lazy* repair, attempts to preserve the remainder of the original multiagent plan and close the gap between the state resulting from the failed plan execution and a goal state of the original planning problem (see Figure 4.2.1).

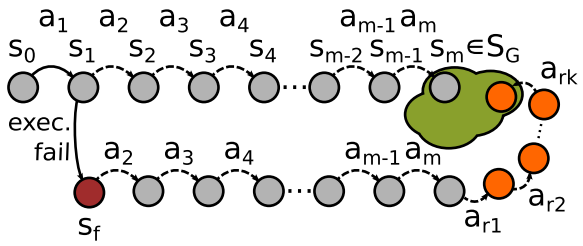


Figure 4.2.1: The Lazy multiagent plan repair approach. The nodes and arcs has the same meaning as in Figure 3.2.1. Note that both the original plan and the plan used form the state s_f use the same actions $\mathbf{a}_2, \ldots, \mathbf{a}_m$.

Algorithm 4.2 Lazy-Repair(Σ)

Input: A multiagent plan repair problem $\Sigma = (\Pi, \mathcal{P}, s_f, k)$, with $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$.
Output: A multiagent plan $\mathcal{P}'$ solving the problem Σ, if a solution exists.

1: construct $\mathcal{P}_{[k..\infty]}$, the executable remainder of $\mathcal{P}[k..\infty]$ from the state s_f
2: simulate execution of $\mathcal{P}_{[k..\infty]}$ from s_f on, resulting in a final state s_{lazy}
3: construct $\Pi_{lazy} = (\mathcal{L}, \mathcal{A}, s_{lazy}, S_g)$
4: $\mathcal{P}_{lazy} =$ MA-Plan(Π_{lazy})
5: **return** $\mathcal{P}_{[k..\infty]} \cdot \mathcal{P}_{lazy}$, **unless** $\mathcal{P}_{lazy} = \chi$ in which case **return** χ

Let s_f be the state resulting from a failure in execution of a multiagent plan $\mathcal{P}$ in a step k. Then 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, $\mathcal{P}'[i-k+1, j] = \mathcal{P}[i, j]$ holds 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 10 (Simple-Lazy repair)**.** Let $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ be a multiagent plan repair problem and $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ be the corresponding modified multiagent replanning problem.

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 $\mathcal{P}' = \mathcal{P}_{[k..\infty]} \cdot \mathcal{P}_{lazy}$, 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 $\mathcal{P}_{lazy}$ is a solution to the multiagent planning problem $\Pi_{lazy} = (\mathcal{L}, \mathcal{A}, s_{lazy}, S_g)$.

Algorithm 4.2 realizes multiagent plan repair based on the Lazy approach described above.

Similarly to the Back-on-Track algorithm, Algorithm 4.2 inherits its correctness from the underlying multiagent planner invoked internally.

Lemma 11 (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 8.*

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 multiagent 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 multiagent 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 12 (connected multiagent planning domain)**.** Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be a multiagent planning problem. Let also $Act = \alpha_1 \times \cdots \times \alpha_n$, with $\alpha_1, \ldots, \alpha_n \in \mathcal{A}$, and $\mathcal{S} = 2^{\mathcal{L}}$. 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 multiagent 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 multiagent planning domain states that it is in the scope of capabilities of the multiagent team $\mathcal{A}$ to *revert* effects of any of its own actions. Note, a singleagent version of the definition (with an omnipotent agent $\overline{\alpha} = \bigcup_{\alpha_i \in \mathcal{A}} \alpha_i$) would also suffice, since $\epsilon \in \alpha$ was required for every $\alpha \in \mathcal{A}$ and in a consequence any joint action of the team can be transformed into a corresponding multiagent 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 13 (Lazy-Repair completeness)**.** *Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be inducing a connected multiagent planning domain and let $\mathcal{P}$, s_f, k, as well as Σ are as in the Lemma 8. 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 multiagent 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 $k+1 \leq j < m$. 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

it is assumed 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 repair 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 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 is assumed. □

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 13 states only partial completeness of the Lazy-Repair algorithm the underlying multiagent planner does not ensure termination.

4.3 Repeated-Lazy Repair

In a dynamic environment, plan failures occur repeatedly. 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 multiagent plan repair algorithms lead to prolonging of the executed plan. For 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 multiagent 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 repair the second failure, should it be necessary.

The following definition formally introduces *Repeated-Lazy* (RLazy) plan repair (see Figure 4.3.1), an extension of the Lazy multiagent plan repair approach introduced in Definition 10. For clarity, from now on, the Lazy multiagent plan repair introduced in the previous subsection will be referred as *Simple-Lazy* repair.

Definition 14 (repeated lazy repair)**.** Let $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ be a multiagent plan repair problem. Let also $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be the corresponding multiagent 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 $\mathcal{P}$, holds $|\mathcal{P}_{fix}| = 0$ and thus $\mathcal{P} = \mathcal{P}'$. Otherwise, $\mathcal{P}$ is a Simple-Lazy repair solution of some (previously

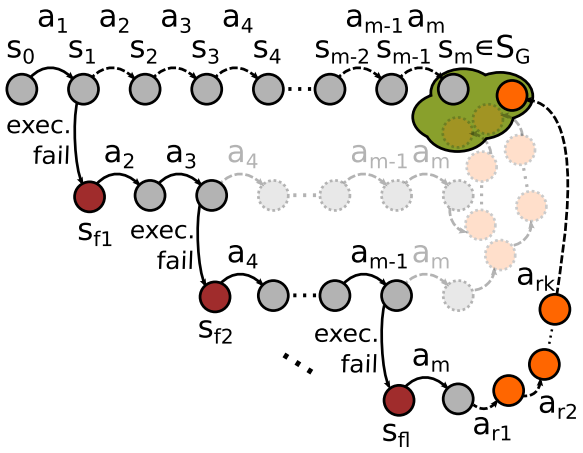


Figure 4.3.1: The Repeated-Lazy plan repair approach. The nodes and arcs has the same meaning as in Figure 3.2.1. The translucent parts are the saved planning problems by the repeating principle.

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 $\mathcal{P}_{fix}$.

$\mathcal{P}''$ 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 4.2. 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 repair approaches, proofs of correctness of the Repeated-Lazy repair algorithm concludes this section.

Algorithm 4.3 Repeated-Lazy-Repair(Σ)

Input: A multiagent plan repair 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 $\mathcal{P} = \mathcal{P}' \cdot \mathcal{P}_{fix}$. Otherwise, in the case this is the first failure encountered, $|\mathcal{P}_{fix}| = 0$.

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

```
1: if k ≤ |P'_[k_p..∞]| then
2:    return Lazy-Repair((Π, P', s_f, k))
3: else
4:    return Lazy-Repair((Π, P, s_f, k))
5: end if
```

Lemma 15 (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 8.*

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 multiagent planning problem Π.*

Proof. Follows immediately from the soundness of the Simple-Lazy repair in Lemma 11. □

Lemma 16 (Repeated-Lazy-Repair completeness)**.** *Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ inducing a connected multiagent planning domain and $\mathcal{P}$, s_f, k, as well as Σ are as in the Lemma 8.*

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

Proof. Follows straightforwardly from the proof of completeness of the Simple-Lazy repair. Note, the proof of Lemma 13 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 the executable remainder of the original plan, as in Algorithm 4.3, is arbitrarily modified, 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 $\mathcal{P}''$. □

4.4 Generalized Repair

The first two presented algorithms, namely the Simple-Lazy and Back-on-Track, are orthogonal to each other in the way how they reuse the original plans. The Simple-Lazy approach reuses prefix of the original plan as an executable remainder and the Back-on-Track approach reuses suffix as a plan fragment. These two approaches can be combined into one algorithm using the original plan

both as a prefix and a suffix together. Such approach *generalizes* the first two approaches and combines the original plan by both fashions as shown in Figure 4.4.1.

Definition 17 (generalized repair)**.** Let $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ be a multiagent plan repair problem and let $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ be the corresponding modified multiagent replanning problem.

A plan $\mathcal{P}' \in \mathit{Plans}(\Pi')$ is a *Generalized repair* of $\mathcal{P}$ parametrized by index vectors F and G iff there is a decomposition of $\mathcal{P}'$, such that $\mathcal{P}' = \mathcal{P}_{[k..(k+f)]} \cdot \mathcal{P}_{\mathit{fix}} \cdot \mathcal{P}[(|\mathcal{P}| - g)..\infty]$, where $\mathcal{P}_{[k..(k+f)]}$ is the executable remainder of $\mathcal{P}$ from k to $k + f$ for some $f \in F$ and $\mathcal{P}[(|\mathcal{P}| - g)..\infty]$ is fragment of the original plan from $|\mathcal{P}| - g$ to $|\mathcal{P}|$. The elements of F and G has to be from interval $\langle 0, |\mathcal{P}| - k\rangle$ to meet the requirements of the decomposition parts $\mathcal{P}_{[k..k+f]}$ and $\mathcal{P}[|\mathcal{P}| - g..\infty]$. It holds that $|F| = |G|$ and for $\forall i$ s.t. $1 \leq i \leq |\mathcal{P}|$, there is no $1 \leq j \leq |\mathcal{P}|, j \neq i$ such that $(f_i, g_i) = (f_j, g_j)$.

To illustrate the generalized notion of this repair, based on the definition, it can be shown that for $F = (|\mathcal{P}| - k), G = (0)$ the approach will end as the Simple-Lazy approach and for $F = (0), G = (|\mathcal{P}| - k, \ldots, 0)$ as the Back-on-Track approach. In the first case, the original plan is reused in the form of the executable remainder of length $|\mathcal{P}| - k$ equally to the definition in Section 4.2. In the second case, the definition of the index vector G implies trying to reuse as long as possible part starting with length $|\mathcal{P}| - k$ and ending with length 0, equally to the Back-on-Track approach in Section 7. Finally, $F = (0), G = (0)$ describes replanning.

It could be argued that generalization reusing the original plan only as prefix and suffix parts is in fact not general, *e.g.*, by means of a MODDELINS scheme presented in [48]. According to MODDELINS, the reuse scheme describing the presented generalized repair approach would be

$$(a_1, a_2, \ldots, a_f, *, a_{|\mathcal{P}|-g}, a_{|\mathcal{P}|-g+1}, \ldots, a_{|\mathcal{P}|}),$$

however a scheme

$$(a_1, \ldots, a_f, *, a_{f+1}, \ldots, a_{f+g}, *, a_{|\mathcal{P}|-h}, \ldots, a_{|\mathcal{P}|})$$

would be also possible, but Generalized repair cannot directly represent it.

The motivation of the approach in this section is not to be general in the sense of reuse pattern of the original plan, but be general from perspective of reuse of the original plan as executable remainder together with a plan fragment. For such case, the generalized repair scheme is the only one making sense, since breaking the $\mathcal{P}_{[k..(k+f)]}$ or the $\mathcal{P}[(|\mathcal{P}| - g)..\infty]$ parts into smaller chunks could only diverge from a efficient repair plan. Breaking the fixing part $\mathcal{P}_{\mathit{fix}}$ into smaller parts could be straightforwardly replaced by extension of the $\mathcal{P}_{[k..(k+f)]}$ and $\mathcal{P}[(|\mathcal{P}| - g)..\infty]$ parts and therefore it is not needed to explicitly consider it. Finally, the generalization does not consider the repeating notion of the Repeated-Lazy repair as it is about the way of running the repair process than about the plan preserving approach *per se*.

The algorithm for the Generalized plan repair approach is outlined in Algorithm 4.4. In the

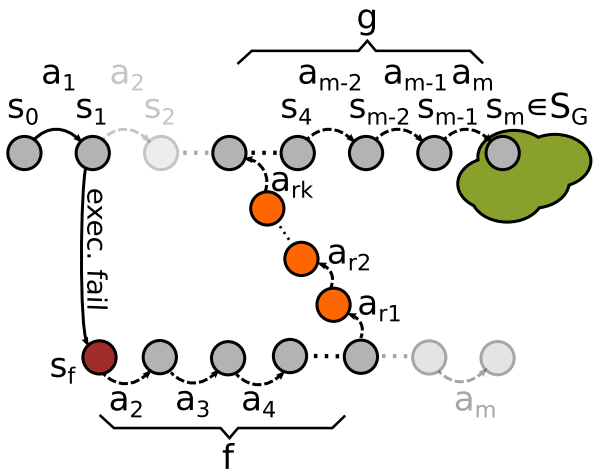


Figure 4.4.1: The Generalized plan repair approach. The nodes and arcs has the same meaning as in Figure 3.2.1. The repair plan $(a_{r1}, ..., a_{rk})$ is used to connect the prefix and suffix of the original plan. The parameters f and g prescribed how many action are used in the prefix way or the suffix way.

case, a failure is detected by the agent team, the current state after the failure is retrieved and the plan repair algorithm for the plan repair problem $\Sigma = (\Pi, \mathcal{P}, s, k)$ is invoked. In each plan repair attempt a modified multiagent planning problem is formulated according to the current values of f and g prescribing the length of the reused prefix and suffix of the original plan. These parameters are took as elements of two vectors of indices F and G additionally parametrizing the approach in contrast to the previous algorithms, which were fixed from perspective of the lengths of the original plan reuse. In effects, the current parametrization prescribes the successive repair attempts.

If a repair plan is found, the repair process finishes, otherwise another attempt with a different combination of f and g is made (selection of F has priority over G in the combination of indices). The resulting repairing plan consists of three components: the preserved prefix of the original plan $\mathcal{P}_{\text{pre}}$, a newly computed infix $\mathcal{P}^*$ and suffix part $\mathcal{P}_{\text{suf}}$, again preserving a part of the original plan $\mathcal{P}$.

The preserved prefix part of the original plan corresponds to an *executable remainder* (see Section 4.2) fragment of $\mathcal{P}$, $\mathsf{ExecRemainder}(\mathcal{P}, s_f)$. The actions with unmet preconditions are in the remainder simply omitted. Additionally, the prefix $\mathcal{P}_{\text{pre}}$ is based only on a part of the original plan effectively reusing 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 multiagent planner algorithm MA-Plan. The initial state of the modified planning problem is the state in which a failure-free execution of the repair prefix $\mathcal{P}_{\text{pre}}$ would result in starting from the state s_f, that is propagation $s_f \oplus \mathcal{P}_{\text{pre}}$. The set of goal states $S_g \ominus \mathcal{P}_{\text{suf}}$ corresponds to a back-propagation

Algorithm 4.4 Generalized-Repair(Σ, F, G)

Input: A multiagent plan repair problem $\Sigma = (\Pi, \mathcal{P}, s_f, k)$.
Input: Parameters F and G prescribing the lengths for reusing of the original plan as prefix and suffix respectively.

1: f, g =initial pair of $f \in F$ and $g \in G$
2: **repeat**
3: $\mathcal{P}_{\text{pre}} = \mathsf{ExecRemainder}(\mathcal{P}[k..(k+f)], s_f)$
4: $\mathcal{P}_{\text{suf}} = \mathcal{P}[(|\mathcal{P}| - g)..\infty]$
5: $\mathcal{P}^* = \mathsf{MA\text{-}Plan}((\mathcal{L}, \mathcal{A}, s_f \oplus \mathcal{P}_{\text{pre}}, S_g \ominus \mathcal{P}_{\text{suf}}))$
6: **if** $\mathcal{P}^* \neq \emptyset$ **then**
7: $\mathcal{P} = \mathcal{P}_{\text{pre}} \cdot \mathcal{P}^* \cdot \mathcal{P}_{\text{suf}}$
8: **break**
9: **end if**
10: **until** tested all pairs of $f \in F$ and $g \in G$
11: **if** $\mathcal{P} = \emptyset$ **then return** fail

of effects of the preserved suffix component $\mathcal{P}_{\text{suf}}$ from the set of original goals S_g.

If the multiagent planner finds a plan for the modified planning problem, the repair 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 repair plan can be found, the algorithm attempts the repair for a different combination of f and g until either a repair plan is found, or it turns out that no repair for the failure exists.

As in the previous three plan repair approaches, the approach description will be concluded with proofs of soundness and completeness of Algorithm 4.4. The algorithm rests on invocation of the underlying multiagent planner, hence its correctness relies on the correctness of the underlying planner.

Lemma 18 (Generalized-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 8.*

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

Proof. Regardless what particular state $s_f \oplus \mathcal{P}_{\text{pre}}$ the failure-free execution of the executable remainder of $\mathcal{P}$ ends up in, the solution plan, if exists, for the problem $\Pi^* = (\mathcal{L}, \mathcal{A}, s_f \oplus \mathcal{P}_{\text{pre}}, S_g \ominus \mathcal{P}_{\text{suf}})$ will take the system from $s_f \oplus \mathcal{P}_{\text{pre}}$ to a state $S_g \ominus \mathcal{P}_{\text{suf}}$ corresponding to the original multiagent planning problem Π. Either $\mathcal{P}_{\text{suf}}$ 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 $\mathcal{P}^*$ would lead directly to some final state $s_{end} \in S_g$ without reusing a part of $\mathcal{P}$ as suffix. □

The first half of the proof resembles the soundness proof of Lazy repair described in Algo-

rithm 4.2. The other part equals to the Back-on-Track Algorithm 4.1 soundness proof. As mentioned before, in Generalized repair these two approaches merge, therefore the proofs are based on the same argumentation.

The completeness of the generalized repair will be proven for a case, where the planning problem has no dead-ends, $0 \in F$ and $0 \in G$.

Lemma 19 (Generalized-Repair completeness)**.** *Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be a multiagent planning problem and let $\mathcal{P}$, s_f, k, as well as Σ are as in the Lemma 8. Let F and G be integer vectors with an interval domain for the elements $\langle 0, |\mathcal{P}| - k\rangle$ (see Definition 17). And finally, let $0 \in F$ and $0 \in G$.*

If there exists a solution plan to the multiagent planning problem $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$, then the execution of Generalized-Repair(Σ,F,G) *algorithm finishes and finds a plan $\mathcal{P}' \neq \chi$, a solution repair of $\mathcal{P}$.*

Proof. The algorithm tests all combinations of F and G values. It eventually tests the required combination $f = 0$ and $g = 0$. Based on the definition of the algorithm, in such case, $\mathcal{P} = \mathcal{P}_{\text{pre}} \cdot \mathcal{P}^* \cdot \mathcal{P}_{\text{suf}}$ degenerates to $\mathcal{P} = \mathcal{P}^*$ since ExecRemainder$(\mathcal{P}[k..(k+f)], s_f)$ = ExecRemainder$(\mathcal{P}[k..k], s_f) = \emptyset$ and $\mathcal{P}[(|\mathcal{P}| - g)..\infty] = \mathcal{P}[|\mathcal{P}|..|\mathcal{P}|] = \emptyset$. The $\mathcal{P}^*$ is result from MA-Plan, therefore the completeness depends on the completeness of the planner. □

4.5 Complexity Analysis

In this section, the presented plan repair algorithms will be theoretically studied from perspective of a classical complexity metrics and one additional metrics suitable for distributed algorithms. The classically studied metrics is time complexity. Additionally, in multiagent systems, one can use a metrics based on an asymptotic ratio of communication volume required for an algorithm to finish to size of the input, as in the case of the time complexity.

4.5.1 Time Complexity of MA-Plan

All plan repair algorithms presented in the previous sections use the multiagent planner as a component, therefore its complexity is an key part of the further analysis. The time complexity of the multiagent planning based on solving of coordination CSP and internal heuristic search (and therefore the MA-Plan implementation of the planner) was studied in [7], therefore the analysis of the time complexity from [7] will be recalled in the following paragraphs.

Informally, it is *"the number of times [needed] to verify that a certain choice of coordination-sequence length forms a basis for a solution × the complexity of the verification process"*. To formally describe the time complexity of the MA-Plan approach, firstly the authors of [7] define size of the CSP domains for each agents' CSP variable (see Section 3.1). Each value of each domain represents

one possible coordination sequence for one agent. Such sequences consist of at most δ coordination points defined as pairs (a, t) with a public action a and $1 \leq t \leq n\delta$ for n agents.

The idea of the $n\delta$ limit for the virtual time points t can be demonstrated on an example with $n = 2$ agents and $\delta = 3$ with precisely three used coordination points for both agents:

$$\begin{array}{c} \alpha: \\ \beta: \end{array} \begin{pmatrix} a_1^\alpha & * & a_2^\alpha & * & a_3^\alpha & * \\ * & a_1^\beta & * & a_2^\beta & * & a_3^\beta \end{pmatrix}.$$

The example shows the longest possible coordination pattern for that particular instance as both the agents use all coordination points possible and the actions depends on each other such that no prolonging of the pattern is possible. In that case, the first coordination point is $(a_1^\alpha, 1)$ and the last one $(a_3^\beta, n\delta)$, where $,n\delta = 6$.

The size of the CSP domain as defined in [7] is for an agent α

$$|D_\alpha| = \sum_{d=1}^{\delta} \begin{pmatrix} n\delta \\ d \end{pmatrix} \cdot |Act_\alpha^{pub}|^d = O((n\delta|Act_\alpha^{pub}|)^{\delta+1}).$$

The term $\begin{pmatrix} n\delta \\ d \end{pmatrix}$ represents all possible combinations of d virtual time points for the public actions (*e.g.*, for $d = 2, n\delta = 6$ there are 15 of them $\{(1,2),(1,3),\ldots,(1,6),(2,3),(2,4),\ldots,(5,6)\}$) and the term $|Act_\alpha^{pub}|^d$ represents all possible public action sequences of length d (*e.g.*, for $d = 2$ and $|Act_\alpha^{pub}| = 2$ the sequences are $\{a_1a_1, a_1a_2, a_2a_1, a_2a_2\}$), therefore for each d, the complete term in the sum counts the number of possible coordination sequences for d coordination points. Finally, the summed up result represent the number of all possible coordination sequences for one agent.

The domain size is then used in the final time complexity formula for the *internal planning constraints* (ipc) in the CSP (see Section 3.1) in the following form

$$O(f(\mathcal{I}) \cdot n \cdot \max_{\alpha \in \mathcal{A}} |D_\alpha|) = O(f(\mathcal{I}) \cdot n(n\delta|Act^{pub}|)^{\delta+1}) = O_{ipc},$$

where the term $f(\mathcal{I})$ represents maximal complexity of individual planning $\mathcal{I}$ with a function f describing the cost of switching from regular planning.

The complexity induced by the *coordination constrains* (cc) is in [7] derived from time complexity of Adaptive-Tree-Consistency algorithm (ATC) for solving CSP problems. The complexity is based on a tree-width ω of the CSP constraint graph [10] which is

$$O(n \cdot \max_{\alpha \in \mathcal{A}} |D_\alpha|^{\omega+1}) = O(n(n\delta|Act^{pub}|)^{\delta\omega+\epsilon}) = O_{cc},$$

where $\epsilon = \delta + \omega + 1$ is dominated by $\delta\omega$. According to [7], the constraint graph is isomorphic to the moral graph of agent interaction graph, therefore ω can be treated as a tree-width of the agent

interaction graph. An *agent interaction graph* describes dependencies of the agents on each other defined by public actions.

The final complexity is sum for the complexities for the particular constraints

$$O_{ipc} + O_{cc} = O(f(\mathcal{I}) \cdot n(n\delta|Act^{pub}|)^{\delta+1} + n(n\delta|Act^{pub}|)^{\delta\omega+\epsilon}) = O_{MAP}.$$

Note that the complexity has no direct exponential dependence on the number of agents n, has no direct exponential dependence on the length of the individual plans of the agents and has no direct exponential dependence on the size of the original planning problem $|\Pi|$. However, the complexity of the individual planning $f(\mathcal{I})$ is in general still exponential in the size of the individual planning problems.

4.5.2 Time Complexity of the Plan Repair Algorithms

Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be the original multiagent planning problem and $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$ be a related multiagent replanning problem. The time complexity of the planning problem Π is complexity of the MA-Plan algorithm as showed in the previous section

$$O(f(\mathcal{I}) \cdot n(n\delta q)^{\delta+1} + n(n\delta q)^{\delta\omega+\epsilon}),$$

where for brevity $q = |Act^{pub}|$. Straightforwardly, the replanning complexity is in general the same as of planning, since the only difference is another initial state s_f. That means n, q and ω are the same[1]. δ depends on the particular initial state, however there is no guarantee that δ for s_f will be generally higher or lower then δ for s_0.

From [48], it is known that plan reuse cannot be generally less complex than replanning from scratch. For the Back-on-Track-Repair (see Section 4.1), the inner planning is defined as $\Pi_{back} = (\mathcal{L}, \mathcal{A}, s_f, \{s_0, \ldots, s_m\} \cup S_g)$, therefore the time complexity is

$$O(|\mathcal{P}|) \cdot O_{MAP} + O_{back} = O(|\mathcal{P}| \cdot f(\mathcal{I}) \cdot n(n\delta q)^{\delta+1} + |\mathcal{P}| \cdot n(n\delta q)^{\delta\omega+\epsilon} + 2|\mathcal{P}|) = O_{BoT}.$$

The complexity of retrieving the returning state is $|\mathcal{P}|$ (at worst a linear walk over the original plan) and the concatenation with the suffix of the original plan is represented by another linear operation of maximally $|\mathcal{P}|$ operations, therefore there is an additive term $O_c = 2|\mathcal{P}|$. Planning for a set of disjunctive goals, which are $\{s_0, \ldots, s_m\}$ can be described as m plannings for the single goals, therefore the complexity of MA-Plan O_{MAP} is multiplied by length of $\mathcal{P}$ which equals to m.

In the Simple-Lazy-Repair (see Section 4.2), firstly, the executable remainder is constructed by a simulation of time complexity O_{rem} which in the worst case require $|\mathcal{P}|$ testings of all ac-

[1]The difference in sizes of different states cannot be larger then $|\mathcal{L}|$, which bounds it by a constant.

tions' $|Act|$ possible preconditions $|\mathcal{L}|$. After construction of the inner planning problem $\Pi_{lazy} = (\mathcal{L}, \mathcal{A}, s_{lazy}, S_g)$, the result is concatenated with the original plan, formally

$$O_{MAP} + O_{rem} + O_{lazy} = O(f(\mathcal{I}) \cdot n(n\delta q)^{\delta+1} + n(n\delta q)^{\delta\omega+\epsilon} + |\mathcal{P}| \cdot |Act| \cdot |\mathcal{L}| + |\mathcal{P}_{lazy}|) = O_{SLazy}.$$

Since the length of the resulting plan part $|\mathcal{P}_{lazy}|$ is polynomial w.r.t. δ, no additional exponential dependence is added here.

The Repeated-Lazy-Repair (see Section 4.3) uses the Simple-Lazy-Repair as an inner repair technique and in worst case it degenerated to it, therefore the asymptotic time complexity is the same

$$O_{SLazy} = O_{RLazy}.$$

Note that in cases better then the worst case, the Simple-Lazy-Repair algorithm decreases the number of usages of Simple-Lazy, therefore its practical complexity (in contrast to the worst case) would be rather near to $O_{SLazy} \cdot \rho^{-1}$, where ρ represent the frequency of the failures.

The last proposed plan repair algorithm—the Generalized-Repair (see Section 4.4)—is parametrized by two index vectors F and G which are used in a repeated search for a solution of the multiagent plan repair problem. The time complexity is informally *how many times the algorithm needs generate and check a repair strategy* $\times$ *what is the complexity of the generate and check procedure.* The generate and check procedure consists of two proposition propagation procedures (by Definitions 3 and 4) each in worst case using the same principle as the executable remainder[2] and one inner planning, hence

$$O(f(\mathcal{I}) \cdot n(n\delta q)^{\delta+1} + n(n\delta q)^{\delta\omega+\epsilon} + 2|\mathcal{P}| \cdot |Act| \cdot |\mathcal{L}|) = O_{G1}.$$

The procedure with complexity O_{G1} is in the Generalized-Repair used maximally $|F| \cdot |G|$ times. If a solution is found, two additional concatenations are needed to finally build the solution, therefore

$$\begin{aligned} O(|F| \cdot |G|) \cdot O_{G1} + O(|\mathcal{P}_{\text{pre}}|) + O(|\mathcal{P}_{\text{suf}}|) &= \\ O(|\mathcal{P}|^2 \cdot f(\mathcal{I}) \cdot n(n\delta q)^{\delta+1} + |\mathcal{P}|^2 \cdot n(n\delta q)^{\delta\omega+\epsilon} + 2|\mathcal{P}|^3 \cdot |Act| \cdot |\mathcal{L}| + |\mathcal{P}_{\text{pre}}| + |\mathcal{P}_{\text{suf}}|) &= O_{GEN}, \end{aligned}$$

where $|F| \cdot |G| = O(|\mathcal{P}|^2)$, as the index vectors can parametrize at most all combinations of the indices to the original plan $\mathcal{P}$ by Definition 17. Similarly to the Simple-Lazy-Repair, the lengths of the resulting plan parts $|\mathcal{P}_{\text{pre}}|$ and $|\mathcal{P}_{\text{suf}}|$ are polynomial w.r.t. δ, thus no additional exponential dependence is added here.

All the resulting time complexities of the algorithms do not comprise any extra exponential

[2] Technically, in this case, the extraction of the executable remainder is part of the proposition propagation process, therefore there is one $|\mathcal{P}| \cdot |Act| \cdot |\mathcal{L}|$. The other $|\mathcal{P}| \cdot |Act| \cdot |\mathcal{L}|$ is for the proposition back-propagation similarly related to extraction of the subplan $\mathcal{P}_{\text{suf}}$.

dependency on any of the parameters. The additional terms are always polynomial. Consistently with [48], the asymptotic worst-case complexity is also never reduced, which is anticipated result of the analysis. The idea of the presented plan repair techniques in general is of lowering δ by simplifying the inner planning process with help of reuse of parts of the original plan. Since δ is in O_{MAP} in two exponent terms, such idea is *positively supported by the analysis* as well.

The results of the time complexity analysis of the repair algorithms are not in conflict with the stated hypotheses. Hypothesis 2 targets taking only a subset of $\mathcal{A}$ which can in effect lower the tree-width ω if the remaining agents are less coupled. In Hypothesis 3, the length δ of the inner repair plan is targeted, as in the cases of problems with actions having long dependency trees, it is theorized that fixing the problem sooner will require smaller δ than solving it later possibly with longer reverting plan of a bigger δ. In the last Hypothesis 4, smaller δ should be achieved by possibly short repair plans, where no reverting is caused by overusing of the original plans $f \in F, g \in G, f+g > m$ and no unnecessarily long repair plans are needed provided that $f+g < m$.

4.5.3 Communication Complexity of MA-Plan

For the study of communication complexity of presented multiagent plan repair algorithms, firstly, the communication complexity of the inner planning process has to be known. Unfortunately, the work of Brafman and Domshlak [7] formally tackle only time complexity.

The communication complexity of the ATC algorithm can be derived from space complexity which was studied in [10]. The analysis will use the Big-O notation similarly to the previous section. To distinguish time and communication complexity, the Big-O will use superscript c for communication complexity O^c.

Size of each message in ATC is $\max_{\alpha \in \mathcal{A}} |D_\alpha|^{sep}$, where *sep* is size of a maximal separator size in tree decomposition of the CSP. The size of the separator is in bucket-trees [10] the tree-width ω, therefore a worst case size of one message is $\max_{\alpha \in \mathcal{A}} |D_\alpha|^{\omega}$. Maximal number of messages communicated O^c_{cc} in a bucket-tree CSP solver is double the number of arcs (two messages for each arc in tree of n vertex graph), therefore $2(n-1)$, where n is the number of the buckets. The buckets represent the CSP variables (with constraints in them resembling the principle of tree-decomposition), therefore the number of the buckets is the number of agents in the coordination constraint (cc). Since the internal planning constraints (ipc) are represented as unary constraints, they do not require any communication. All together, it gives the communication complexity of the planner as

$$O^c_{cc} = 2(n-1) \cdot \max_{\alpha \in \mathcal{A}} |D_\alpha|^{\omega} = O^c(\varepsilon n (n\delta q)^{\delta\omega+\epsilon} + \epsilon') = O^c(n(n\delta q)^{\delta\omega+\epsilon}) = O^c_{MAP},$$

where $\epsilon = \omega$ is dominated by $\delta\omega$ in the exponent, $\varepsilon = 2$ is a polynomial coefficient and ϵ' is dominated by the first polynomial term. The communication complexity of planning using DisCSP for coordination is therefore not exponentially dependent on number of agents, it is not dependent

on the complexity of the individual planning $\mathcal{I}$ and it has no direct exponential dependence on the size of the original planning problem $|\Pi|$, similarly to the time complexity. The communication complexity is bounded by one exponential term in the number of the coordination points and tree-width of the agent interaction graph

$$\exp(\delta\omega).$$

4.5.4 Communication Complexity of the Plan Repair Algorithms

The communication complexity of the plan repair algorithms will be derived by an equal process as in the case of the time complexity. It builds on the derived complexity of the inner planning of MA-Plan which is in the case of communication $O^c(n(n\delta q)^{\delta\omega+\epsilon}) = O^c_{MAP}$.

Equally to the time complexity, replanning is in general the same as planning from the perspective of communication complexity. The only difference is in the initial state. That means n, q and ω are the same and δ depends on the particular initial state without any general guarantees on its change during replanning.

All proposed plan repair algorithms need a synchronization broadcast as the agents has to be aware of new plan repair process in case of failure in a private fact. Such broadcast is an additive factor $O^c_{sync} = O(n)$ as the messages are sent to all agents.

The Back-on-Track-Repair needs the synchronization broadcast and an inner planning process with disjunctive set of goals $\{s_0, \ldots, s_m\} \cup S_g$. The retrieval of a state s_j (see Algorithm 4.1) and concatenation with the suffix is done individually by each agent. Therefore the communication complexity is

$$O^c_{sync} + O^c_{MAP} = O^c(n + n(n\delta q)^{\delta\omega+\epsilon}) = O^c_{BoT}.$$

The Simple-Lazy-Repair needs a distributed simulation of execution together with extraction of the executable remainder of $\mathcal{P}_{[k..\infty]}$. In a worst case, such process requires application and broadcast of each public action in the plan. There can be maximally $|\mathcal{P}|$ of those actions and the effects of each action can be maximally of size of the language $|\mathcal{L}|$, therefore the communication complexity of the remainder extraction is $O^c_{rem} = O^c(n \cdot |\mathcal{L}| \cdot |\mathcal{P}|)$. With help of O^c_{rem}, the communication complexity of Simple-Lazy-Repair can be derived as

$$O^c_{sync} + O^c_{MAP} + O^c_{rem} = O^c(n + n(n\delta q)^{\delta\omega+\epsilon} + n \cdot |\mathcal{L}| \cdot |\mathcal{P}|) = O^c_{SLazy}.$$

The final state s_{lazy} after the simulated execution of the executable remainder do not have to be explicitly communicated as each agent knows its part of the state s_{lazy} and can start the consecutive planning process with its part of s_{lazy}.

Equally to the time complexity, the communication complexity of the Repeated-Lazy-Repair is in

worst case the same as the complexity of the Simple-Lazy-Repair, since in worst case Repeated-Lazy uses Simple-Lazy. That gives

$$O^c_{SLazy} = O^c_{RLazy}.$$

In the Generalized-Repair, the process is separable to repeated sub-processes of generating and checking of a repair strategy. The communication complexity will be firstly derived for one such sub-process. The two proposition propagation procedures, each in the worst case use the same principle as the simulation of executable remainder and are used with one inner planning, therefore

$$O^c(n(n\delta q)^{\delta\omega+\epsilon} + 2n \cdot |\mathcal{L}| \cdot |\mathcal{P}|) = O^c_{G1}.$$

Equally to the time complexity, the sub-process with the complexity O^c_{G1} can be used in worst case $|F| \cdot |G|$ times. If a sound solution is found the agents has to inform each other, therefore there is beside the initial synchronization a termination synchronization of $O^c(n^2)$, although the local plans has not to be communicated as they are later executed by the particular agents. The communication complexity of the Generalized-Repair is

$$\begin{aligned} O^c_{sync} + O^c(|F| \cdot |G|) \cdot O^c_{G1} + O^c(n^2) &= \\ O^c(n + |\mathcal{P}|^2 \cdot n(n\delta q)^{\delta\omega+\epsilon} + 2n \cdot |\mathcal{L}| \cdot |\mathcal{P}|^3 + n^2) &= O^c_{GEN}, \end{aligned}$$

where $|F| \cdot |G| = O(|\mathcal{P}|^2)$, as the index vectors can parametrize at most all combinations of indices to the original plan $\mathcal{P}$ by Definition 17.

The analyzed communication complexities $O^c_{BoT}, O^c_{SLazy}, O^c_{RLazy}$ and O^c_{GEN} do not bring any new terms exponentially dependent on any of the parameters. Therefore the communication complexity of all the proposed plan repair algorithms remains exponential only in the factor of number of coordination points δ in the inner repair plan and tree-width ω of the agent interaction graph, *i.e.*, $\exp(\delta\omega)$. This result is anticipated as the communication complexity is usually proportional to the time complexity as sending messages requires a computational time.

The resulting communication complexities of the plan repair algorithms concur with the proposed hypotheses, similarly as in the case of the time complexity. Hypothesis 1 states that the communication overhead is lowered by plan repair producing more preserving repairs in comparison to replanning. Since the communication complexity of replanning is exponentially dependent on δ this hypothesis is supported by the analysis as far as at least one coordination point is spared, because decreasing the exponential factor by one $\exp((\delta - 1)\omega)$ dominates any additional polynomial factors added by the plan repair techniques. This is true only, if the problems are tightly coordinated $\omega \gg 0$. If it be to the contrary, the exponential factor is negligible even if δ is not

decreased by the preservation of the repair, formally $\exp(\delta\omega) \to 1$ iff $\delta \to 0$ or $\omega \to 0$.

The arguments used for the last three hypothesis copies those in the time complexity analysis. Hypothesis 2 targets taking only a subset of $\mathcal{A}$, which can in effect lower the tree-width ω if the remaining agents are less coupled, and therefore lower the communication complexity. In Hypothesis 3, the length δ of the inner repair plan should be minimized if failures of actions with long dependency trees are fixed as soon as possible. In Hypothesis 4, smaller δ should be achieved by possibly short repair plans by appropriate reusing of the original plan.

4.6 Implementation

The plan repair algorithms were implemented for further experimental evaluation, verification and validation in high-fidelity simulation. All proposed algorithms stand on MA-Plan which makes the implementation of the planner key part of the repair algorithms. The following sections summarize *implementation details* of the used multiagent planner, new fixes and optimizations done in the planner required for successful verification of the repair algorithms and implementation of the particular plan repair algorithms as described and analyzed theoretically in the previous sections.

4.6.1 Multiagent Planner

Implementation of the multiagent planner used in this thesis denoted as MA-Plan was proposed by Nissim et al. in [51]. The implementation was publicly available, therefore it was used and the repair algorithms were built on it. Both the planner and the repair algorithms were written in the Java language[3]. As mentioned in the previous sections, the coordination subproblem in the planner is defined as a Distributed Constraint Satisfaction Problem (DisCSP) and it is solved by a DisCSP solver. The inner planning subproblem is defined as a classical planning problem with landmarks and it is solved by a heuristic search planner.

The algorithm used as the DisCSP solver is a customized *Asynchronous-Backtracking* (ABT) [57] with *Forward-Checking* (FC) [70] heuristics. Since the efficiency of the DisCSP solver implementation is in the planner crucial, it was implemented and adapted in the original codes from scratch not using any available DisCSP solvers. To improve its efficiency, it uses a specific process of local selection of a value for a variable representing the agent's individual plan. The process is based on a heuristic called *Least-Action-Landmarks-Added* (LALA) [51]. The heuristic prefers CSP values with lower number of landmarks for the inner planning process. The idea behind this is to make the search easier for the inner planner and for further coordination as prospectively less public preconditions has to be fulfilled by other agents. The drawbacks of the heuristics manifest in problems with local plans, where no affecting of facts both in initial state and goal states leads to no solution.

[3] http://java.com/en/

If the number of such values in the domain is large, it can significantly decrease the efficiency of the planner.

After selection of a value by the LALA heuristics, it is added into the *Current Partial Assignment* (CPA) of running ABT. Consecutively, the CPA is passed to a next agent as ABT prescribes. A heuristics used to select the next agent prefers unassigned agents achieving most goals leveraging the *most constrained principle*.

The process preparing the CSP values in the domain is based on an approach using generate-and-test principle for all possible time positions of the actions got from the inner planner and testing this timed actions together with their precondition requirements against the current CPA. The requirement combinations (each requirement for each action) are added only based on previous (already added) requirements, which means that combinations with invalid previous requirements are not generated and tested.

As the inner planner, a *Best-First Search* algorithm is used with the *Fast-Forward* relaxation heuristic, helpful actions heuristic and adapted landmark heuristic. Particularly, the implementation is an adapted JavaFF planner[4]. One of the adaptations is that the planner uses *internal projection* of the public actions. It means, the actions are strip of public facts (in all their $\mathsf{pre}(a)$, $\mathsf{add}(a)$ and $\mathsf{del}(a)$ sets). That is why an agent's inner planner is able to create a plan even if some of the public actions will be applicable only after applying other agent's action effects. If the inner planner returns a valid plan, it is used in the generate-and-test process of the timed action sequences effectively prescribing part of the CSP domain of one agent. When it is not possible to generate from such plan a CSP value, because the actions of the plan cannot be matched to any combination of the time points, the inner planner is run again with all such forbidden action sequences from previous runs. Another adaptation of the planner is that it operates with public actions required by preconditions from the preceding requirements in form of landmarks. This principle forces the inner planner to always include required public actions, as the landmarks are by definition required in every sound plan [59].

From the perspective of the flow of the algorithm, the planning process passes four distinguishable phases, (i) centralized preparation of a DisCSP instance, (ii) initialization of solving process for the DisCSP solver in a `Goal-Agent`, (iii) decentralized solving of the DisCSP problem and (iv) decentralized finalization of the DisCSP solving process.

In the first centralized phase, operators and facts are parsed from a domain file and a problem instance file respectively (both are described in PDDL [45]). The problem definition is enriched by a clause describing what objects are agents in the planning problem. Subsequently, all operators are grounded to actions (unfeasible action are ignored) according to facts form the problem definition. Actions are distributed among the agents according to one of its parameters containing name of one of the agents. Similarly to the actions, related atoms $p \in \mathcal{L}$ are assigned to particular agents

[4] http://www.inf.kcl.ac.uk/staff/andrew/JavaFF/

as well. The atoms are assigned by the actions they are used in. After the process of action and atom assignment, the public and private atoms are marked. All internal atoms are initially treated as private, a public atom is an atom, which is in an internal set of atoms of more then one agent (*i.e.*, it can be affected or required by two different agents, see Section 3.1). Similarly, all actions are initially treated as private, a public action contains at least one public atom (whether in preconditions or in effects). After this initialization the agents are prepared to receive first messages in the DisCSP solving process.

In the second phase, a special `Goal-Agent` is created and initialized. The agent contains only one `goal-action` containing atoms of the goal term as preconditions. The agent is selected as a first current agent for the DisCSP solving process. As the agent contains all goal atoms (even such, which would be without the `Goal-Agent` private) it generates requirements for all regular agents supplying a part of the goal term. The CSP domain for the goal agent has size $l = \delta n$, where n is number of the agent. Each goal atom is public, because the preconditions of the `goal-action` are goal atoms. Therefore the CSP domain generator creates all possible time positions of each public proposition. That means the domain size of the goal agent is $l^{|S_G|} = (\delta n)^{|S_G|}$. This exponential dependency on the number of goals was not considered in the theoretical analysis of [7] and caused significant efficiency downgrade of the algorithm implementation.

The next phase is the decentralized process of DisCSP solving which follows the ABT algorithm. There are several specifics of the solver. A fundamental one is that the DisCSP domains are generated during the solving process based on the results of the local planner (the process corresponds to the internal planning constraint mentioned in previous section). Current agent working on the CPA generates a local plan using the heuristic search with internal projection of the used actions. A local plan consists of actions fixed in time points and has to satisfy current atom requirements (preconditions of actions of previous agents) form the CPA. For such a plan, a new atom requirements ensue from public actions in the plan and these new requirements are added into the CPA. Afterward, the CPA is send to the next agent. If the last agent can satisfy all the requirements a solution is found, otherwise the partial solution is backtracked and the previous agent tries to generate new alternative local plan, generates new precondition requirements and the process continues. If the backtracking process returns to the `Goal-Agent` and there are no other possible requirements, no solution can be found.

In the final decentralized phase the agents know their local plans (if the multiagent plan exists) and they can execute them in a distributed manner.

4.6.2 Planner Improvements

Since the DisCSP-based planner was used as the inner planner in the plan repair algorithms, its reliability and stability was intensively tested on a wide spectrum of variations of the planning problems generated by the randomized failures. This set was much larger, than the number of the

original planning problems the planner was tested on by its authors. Such stress-testing revealed several problems in the planner which required fixing and additional optimizations during the work on the experiments and thereby for this dissertation. Three essential fixed parts were:

Public/private factorization overlap In specific cases, the routines for factorization of the planning problem generated facts, which were both private and public. Practically, such facts ended both in a public fact set and private fact set of one agent. According to the MA-STRIPS definition in [7] (see Section 3.1), each fact should be either public or private. Unfortunately, fixing this issue caused several other problems in other parts of the planner which depended on the bug. Practically if an fact was both public and private, it was treated by both the cooperative parts of the algorithm and by the inner planning part. In a sense, making a fact both private and public is more robust as the fact is treated by both the parts, however it have a negative impact on efficiency in worst case increasing the exponential complexity term by increasing length of the coordination points δ.

The only supplier of a public fact In the original implementation, the method for obtaining an internal projection of a public action was removing the precondition facts even if the planing agent was the only supplier of such fact. This made the planner incomplete, since if an agent is the only supplier of a fact in precondition, its inner planner has to plan for it and therefore it must not be removed during the projection. Additionally, this fix increased the computational efficiency of the planner $2\times$.

New and complete landmark generator The generator of the landmark sequences from the fact requirements induced by other agents' public actions was incomplete in a sense it did not return all combinations of possible supplying actions. In a case of two or more concurrent requirements by different agents at one time point, the generator did not return all possible supplying actions of the planning agent. In the original implementation all the tested planning problems did not contain such situations, and therefore this problem did not appear. By extending the set of testing domains by a tightly coordinated combinatorially intense problems (the COOPERATIVE PATHFINDING domain, see Section 5.1) this problem exhibited, as the planner was not able to solve such problems. The landmark generator was redesigned and implemented from scratch. Currently it is based on a principle of a generative vector containing an ordered set of action lists with all possible actions at the time points. Such vector is used for successive generation of possible sequences of the actions, *i.e.*, defining the possible landmark sequences. The initial implementation of the algorithm was done in Clojure[5] for more straightforward and flexible testing and followingly rewritten to Java because of computational efficiency.

[5]Clojure is a dynamic programming language strongly influenced by Lisp that runs on the Java Virtual Machine (http://clojure.org).

Several parts of the planner were optimized, increasing the scalability of the planner both by means of the number of agents and the length of the resulting plans:

Output plan length optimization The output of the original implementation of the planner was in form of the pairs of public actions at time points and the inner plans related to the particular DisCSP values representing the plans. As the plan repair algorithms required the plans in the matrix form, an extraction process was implemented. The problem with the resulting plan was that it did not considered empty actions bounded by the $n\delta$ limit which is correct from the perspective of the complexity bounds, but it is wasting in a sense of the plan repair. Therefore the extraction process was supplemented by a polynomial shortening algorithm which moves all actions to as soon as possible time points and preserves the dependencies among them. Such process does not jeopardy the soundness and complexity bounds of the planner and spared considerable number of empty steps in the plan (which was empirically verified).

DisCSP domain reordering The DisCSP domains were originally generated from the sooner time points to the later time points, *i.e.*, in a chronological order, however the goal-agent's requirements restrained the last possible time point in the possible public action sequences. That caused the DisCSP domain generator generated large amounts of values (the public action sequences based on the inner planning process) which were later unusable. Turning the process around, *i.e.*, staring the generative process from the last time points, increased efficiency of the planner about 10× in tightly coordinated problems with higher numbers of agents.

Inner planning depth optimization The original implementation of the adapted *Best-First Search* (BFS) algorithm from JavaFF searched for a plan with any number of public actions. The search was limited only by visiting all reachable states. However the search can be effectively limited by δ, as it represents the maximal number of coordination points (see Section 3.1). Limiting the BFS by pruning plans with more public actions than δ increased efficiency of the planer about 30× in tightly coordinated problems with higher numbers of agents.

Inner planning cache As the inner planning process is often called with the same parameters in case of backtracking in the ABT algorithm, the process was wrapped into a caching procedure. The cache was implemented as an associative map with keys defined by the parameters of the inner planner and with a value of the resulting plan. Addition of the local plan cache increased efficiency of the planner about 4× in tightly coordinated problems with higher numbers of agents.

Fixed fact optimization A lot of checking of the fact requirements in the generate-and-test pro-

cess for the CSP domains was omitted by pruning of checks of all actions in the inner plans only if they were not trying to delete a fixed fact. A fixed fact does not change any action. This optimization increased efficiency of the planner about 2× in highly tightly coordinated problems.

Removing NOOP actions The planner originally worked with action sets of the agents each containing a NOOP action ϵ, however as the public action pairs (a, t) define implicitly not only position of the actions, but also positions of ϵ actions, they could be removed. Removing NOOPs from the action sets improved efficiency of the planner about 5× in tightly coordinated domains.

In a result the fixes and optimizations increased computational efficiency of the planner in a best case about 24000× in tightly coordinated domains. The increase diminish with simpler problems, lower coupling and lower numbers of agents and increase with more complex problems. From an empirical observation, the efficiency increase was exponential in case of the *Inner planning cache*. In the other cases the exponential dependency was not confirmed, therefore the dependency was rather only polynomial.

Two extensive optimizations remained for a future work. The first one comprise rewriting of the process preparing the CSP values. In the original implementation the process is based on the generate-and-test principle which forces the planner to generate large amounts of unused domain values ahead. This optimization was already implemented, but needs more testing. The process was completely rewritten to lazy routines preparing each CSP value just when it is required by the ABT algorithm and implemented as a functional version of a generative structure for possible plans based on a grammar-like structure. The idea of the underlying principle was described in [29]. Currently, the implementation needs an exhaustive testing and debugging as the changes affected most of the planner codes especially of the DisCSP solver implementation.

The other future-work optimization is in the internal projection routines. Generally, the problem is that the internal projection removes all information about preconditions and effects of public facts from the public actions, since the facts can be achieved by other agents. This means the inner planner has to almost blindly generate all possible orderings of public actions in the plan. Private facts of such actions (and prospectively private actions) are the only constraining information for the planner and the CSP domain generator, if such even exist. To improve on this, the internal projection has to be altered to preserve as most complete as possible public information and iteratively decrease amount of this information. This principle could be described as a multiagent heuristics presuming that solving a problem locally by lower number of agents is always better than rely blindly on help from all other agents.

4.6.3 Multiagent Plan Repair Process and Algorithms

The multiagent planning, executing, monitoring and repair process has two phases. In the first phase, for a given domain a multiagent 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 multiagent team, or a centralized observer, upon a failure detection a plan repair algorithm is invoked. In turn, to find particular repair plans, the MA-Plan algorithm is invoked as specified by the plan repair algorithms introduced in the previous sections. The plan repair algorithms are technically wrappers around the planner and invoke it if the fixing part of the plan is required.

A scheme listed in Algorithm 4.5 shows the pseudo-code of the process. Since complete information is assumed, there is no difference between a decentralized and a centralized monitoring, hence for clarity, the algorithm instantiates the centralized version of monitoring. 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. Technically, the simulator of the execution is a Clojure program with a multiagent plan as the input and a sequence of states as an output. Since this process is straightforwardly describable as a functional process and the computational efficiency of the execution is not an issue, Clojure was a fine fit.

At this point, it is not explicitly articulated what a failure amounts to, since this can be application specific. Moreover, the implementation of the failures is designed as a modular element, which enabled testing of various types of failures not only because of the conducted experiments, but also because of easier debugging of the plan repair algorithms. From the theoretical point of view, 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 repair 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 4.5 does not necessarily terminate. Termination of of the scheme relies on two factors. Firstly, it is the termination property of the underlying multiagent planner invoked by the plan repair algorithms. 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

Algorithm 4.5 Plan execution and monitoring scheme.

Input: An initial multiagent planning problem $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ and vectors F and G of indices provided that the Generalized Repair algorithm is used.

```
1: P = MA-Plan(Π)
2: if P = χ then return fail
3: k = 1
4:
5: repeat
6:    agents perform P[k]
7:    if failure detected then
8:       retrieve the current state s from the environment
9:       P = Repair((Π, P, s, k), [F, G])
10:      k = 1
11:   else
12:      k = k + 1
13:   end if
14: until P = χ or k > |P|
```

failures sooner than the previous repair was fully executed. In a consequence, this would lead to a gradual prolongation of the executed plan so that it will never reach the end of its execution. Informally, for such domains, the Algorithm 4.5 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 [53] 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 4.5 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, Simple-Lazy and Generalized plan repair approaches 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 multiagent planner invocations and in a consequence save a large amount of communication overhead.

Chapter 5

Experimental Evaluation

The algorithms were evaluated in various experiments designed to verify the stated hypotheses. Since the techniques were designed as domain-independent, the particular domains were chosen to cover various types of planning challenges, especially the domains spread from fully coordinated to uncoordinated problems. This chapter describes the experimental setup and the results of the experiments together with discussion and conclusions.

5.1 Domains

The experiments were conducted on four planning domains. Three of the domains originate in the standard singleagent IPC planning benchmarks [25]. Similarly to the evaluation of the MA-Plan implementation in [51], the experiments were mostly based on domains straightforwardly modifiable to the multiagent setting: LOGISTICS, ROVERS, and SATELLITES. Additionally, the set of IPC-based domains was extended by a well-known coordination domain COOPERATIVE PATHFINDING. The domain is hard from the perspective of combinatorial complexity of required coordination, therefore only small instances were used which the MA-Plan was able to plan in reasonable time.

Instances of the LOGISTICS problems are about transporting packages among 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 T1 and T2 and one airplane A. There are two cities, each with one storage depot (d1 and d2) and one airport (a1 and a2). The trucks can move m(from, to) only within their cities, between one depot and one airport. The airplane can fly f(from, to) among all airports in the environment, but cannot land at the depots. All vehicles can load l(package, location) and unload u(package, 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 multiagent plan solving this particular instance is $\mathcal{P}^{log3} =$

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

The coordination frequency for such problem is $cf(\Pi^{log3}) = \frac{4}{9} = 0.\bar{4}$, as $\delta = 4$, the 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 understand as tightly coordinated, since 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, denoted in the experiments as LOGISTICS PAR, for 5 and 6 agents involves two parallel logistics subproblems. The separation of the classical and parallel logistics was necessary, because larger logistics problems with more agents were more combinatorially complex and the planner was not able to solve them.

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 $\mathsf{c(s,r,i)}$ 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 $\mathsf{p(s/r/i)}$. The rock and soil analysis can be executed provided that the rover is at the appropriate 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 appropriate 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 $\mathsf{R1}$, $\mathsf{R2}$ and $\mathsf{R3}$. A solution $\mathcal{P}^{rov3} =$

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

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

The SATELLITES domain 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 one of predefined target directions. Secondly, each imaging instrument has to be powered, switched on and calibrated before it can take an image $\mathsf{t(i)}$ in one of predefined modes. A solution of one of the experimental instance Π^{sat3} using three satellites $\mathsf{S1}$, $\mathsf{S2}$ and $\mathsf{S3}$ is

$\mathcal{P}^{sat3} =$

$$\begin{array}{l} \mathsf{S1}: \\ \mathsf{S2}: \\ \mathsf{S3}: \end{array} \left(\begin{array}{cc} \cdots & \mathsf{t(i1)} \\ \cdots & \mathsf{t(i2)} \\ \cdots & \mathsf{t(i3)} \end{array} \right)_{\text{3 private actions}}.$$

In this case, the coordination frequency is $cf(\Pi^{sat3}) = \frac{0}{3} = 0$, 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 domain is not a standard benchmark from IPC, therefore the particular problems used in the experiments are depicted in Figure 5.1.1 . The goal for the robots is to move $\mathsf{m(from, to)}$ to other positions usually occupied by the other robots. A representative problem Π^{cp3} contains three robots R1, R2 and R3 and a solution plan $\mathcal{P}^{cp3}$ is

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

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 fully coordinated problems.

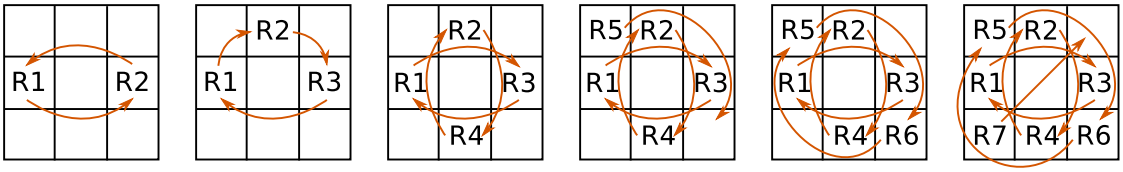


Figure 5.1.1: Instances of COOPERATIVE PATHFINDING problems with 2 to 7 agents (problems with 5–7 agents are used only in Chapter 7). Depicted robot positions are in the initial states and the arrows points at cells they have to move to.

5.2 Metrics

Four metrics were used to evaluate the measurements:

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

planning time is the measured cumulative time consumed by the underlying MA-Plan planner

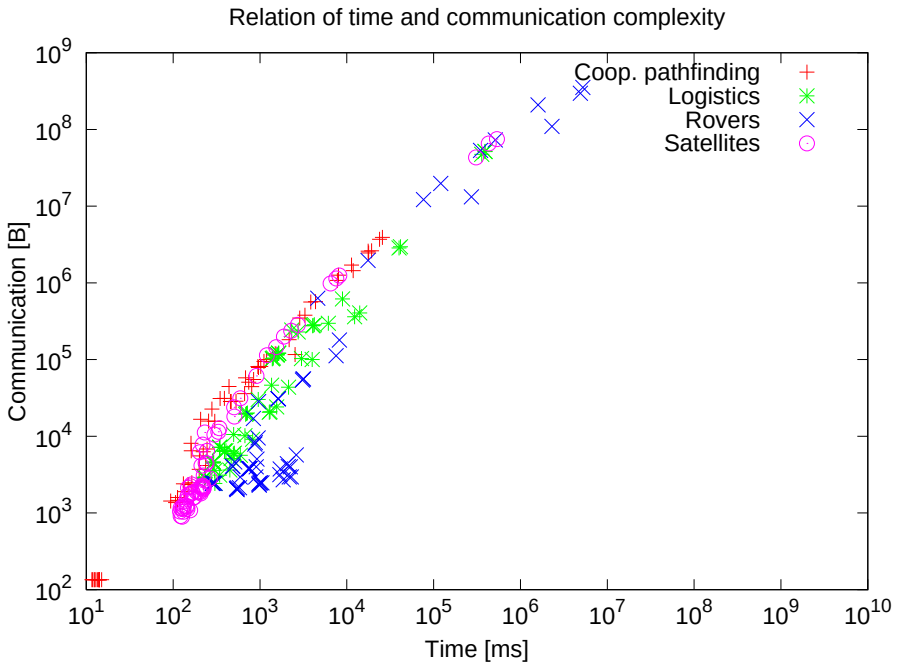


Figure 5.2.1: Relation between communicated bytes and computation time required for solving the plan repair problems.

used for generating initial and repair plans,

repair 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 and plan repair processes. That is messages generated by the DisCSP solver in the MA-Plan planner.

To account for differences in essential computational and communication complexity of the domains, a relationship experiment between these two measures was conducted. Figure 5.2.1 depicts the results and demonstrates that there is no essential discrepancy between the computational and communication complexity of the plan repair solutions. That means, the following presented results are not biased by problems extremely hard in time and simple in communication or *vice versa*.

5.3 Failure Types

Two types of plan failures were considered: *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 state, which obviates 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 immediately by one of the plan repair algorithms.

An *action failure* is simulated by not-execution of some of the individual agent's 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. Since the agents do not know which action failed, a simple solution to only locally find the right actions is not possible.

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

5.4 Experimental Setup and Process

The implementation of the experimental setup is based on a centralized simulator of the environment integrating the multiagent domain-independent planner MA-Plan. The individual agents are initialized, 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 plan repair process together with the algorithms was implemented as described in Section 4.6.3 in Java and Clojure languages. The detection if a generated failure is repairable was done by the Fast-Forward planner [24]. It was used the original implementation *FF-v2.3* of the planner in C which is publicly available from its webpage[1]. The planner was run from Java as an additional process. To parametrize the planner, a temporary PDDL file with the particular replanning problem was created. After planner finished, the textual results were parsed back in the Java process. Each plan repair experimental run was parametrized by:

- identifier of the plan *repair algorithm,*

[1] http://fai.cs.uni-saarland.de/hoffmann/ff.html

  - the additional parametrization of the Generalized-Repair was represented in the code under different identifiers,

- the *planning domain* in PDDL,
- the original *planning problem* in PDDL,
- the *failure generator* parameters (the failure probability P and the perturbation amount$c$ in form of number of fact for addition c^+ and for removal c^-),
  - the definition which of the failure generators to use was in the code,
- a *seed* number for the random generator (for reproducible debugging).

An output from one experiment was added as one row into a resulting file, incrementally creating a matrix of the results. The experimental runs were iteratively run from a Bash[2] shell script for easy and flexible configuration of each experimental batch and for independent crash testing outside of the Java Virtual Machine.

The finalized matrix after all experimental runs formed an input for an Octave[3] script. The script statistically processed the data and either generated a data-set for Gnuplot[4] graph renderer or directly generated the resulting graphs or data tables.

5.5 Results and Discussion

To validate the presented hypotheses four sets of experiments were prepared. The first set validated generally how much beneficial is preserving plan repair in contrast to replanning from scratch and additionally validated what are boundaries of the hypothesis. The other three experiments built on the results of the first one and deepens the study on how particular multiagent plan repair techniques and particular parametrization perform in different planning domains.

The following sections are organized using one pattern which firstly describes what plan repair algorithms or their parametrization were used in the experiments and consecutively what are the results with discussion how the results concur with the stated hypotheses.

5.5.1 More Preserving Repairs

The first hypothesis targets the fundamental question of the research in this dissertation whether preserving multiagent plan repair can gain substantial computational and communication efficiency in contrast to replanning from scratch and ideally with what limitations.

[2] http://www.gnu.org/software/bash/

[3] GNU Octave is an open source interpreted language, primarily intended for numerical computations, compatible with Matlab, http://www.gnu.org/software/octave/.

[4] http://www.gnuplot.info/

Used Algorithms

To evaluate validity of Hypothesis 1, the multiagent planning problems were tested against a plan repair algorithm implementing replanning from scratch and two of the repair algorithms Back-on-Track-Repair (BoT, see Algorithm 4.1) and Repeated-Lazy-Repair (RLazy, see Algorithm 4.3) introduced in the previous chapter.

Individual experimental measurements were parametrized by the plan failure probability P and each problem instance was executed 5–10 times with various random seeds. The resulting data are, in the figures, presented with the natural distribution. The box-plot 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 repair method and replanning from scratch (normalized at the 100% level). Since the MA-Plan planner was used 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.

Efficiency problems of the original MA-Plan implementation (in [51]) limited the experiments to plans with maximally six coordination points per agent. Additionally, the Back-on-Track-Repair algorithm could not leverage disjunctive goal form (see construction of Π_{back} in Algorithm 4.1) and this was emulated by an iterative process testing all term conjunctions in a sequence and thus resulting in multiple runs of MA-Plan instead of a single run with disjunctive goal.

Results and Discussion

The first batch of experiments directly targeted validation of Hypothesis 1:

> *Multiagent plan repair is expected to generate lower communication overhead in tightly coordinated domains.*

LOGISTICS and COOPERATIVE PATHFINDING, as coordinated domains with dynamics of the simulated environment modeled as action failures, were suitable to provide required insights. Table 5.1 shows results for a fixed failure probability $P = 0.3$ and Figures 5.5.1, 5.5.2 and 5.5.3 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 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. That

Domain	Agents	Repair 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 5.1: 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.

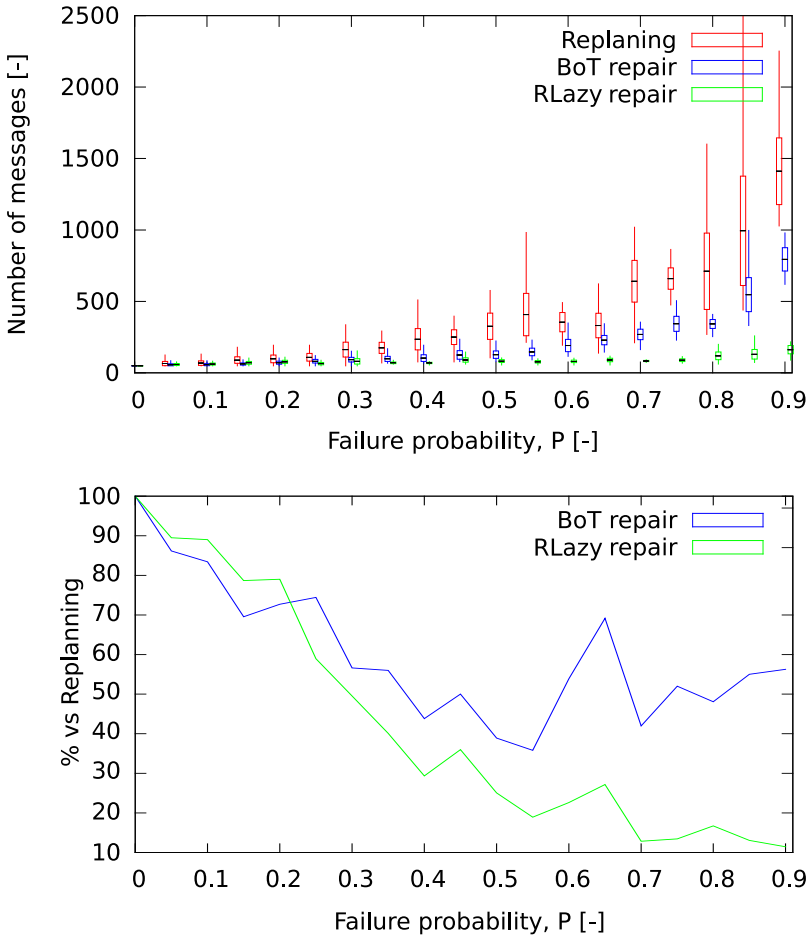


Figure 5.5.1: Experimental results of the communication metrics for LOGISTICS domain with 3 agents and action failures.

means, the plan repair algorithms scale better than replanning from scratch. The trends in Figures 5.5.1, 5.5.2 and 5.5.3 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* the first 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 overhead. Plan repair scales better with higher

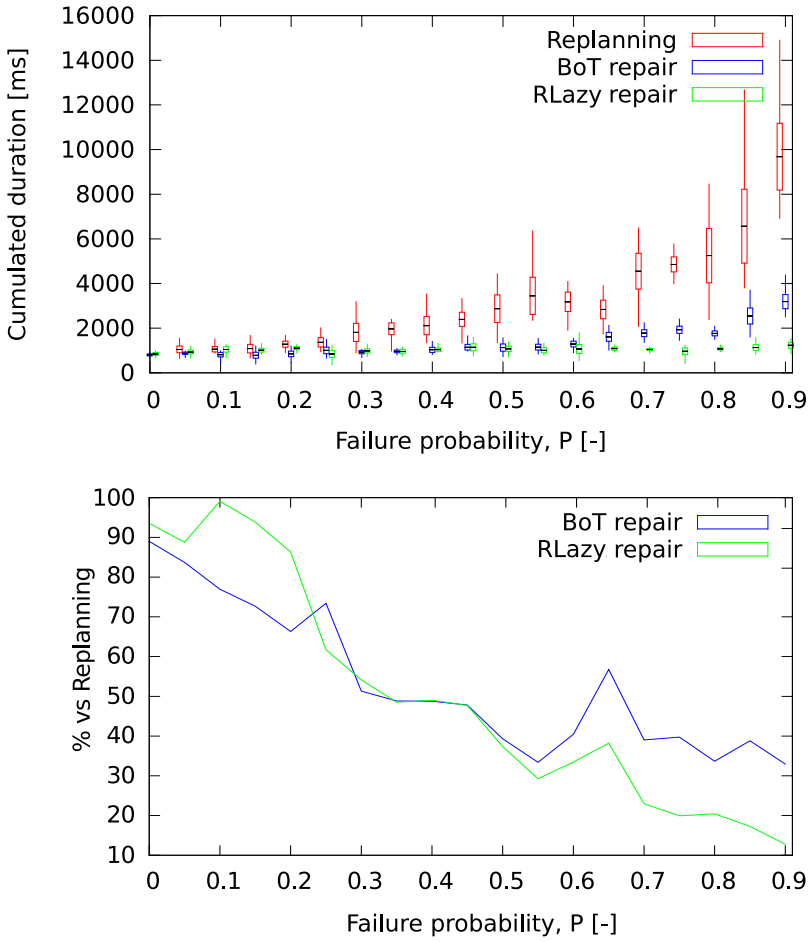


Figure 5.5.2: Experimental results of the planning time metrics for LOGISTICS domain with 3 agents and action failures.

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 5.5.1 depicts these results.

The second batch of experiments focused on boundaries of validity of the positive result presented above. In particular, the condition on the coordination tightness and feasibility of failures were validated. The auxiliary hypothesis states:

With decreasing coordination frequency of the planning domain, the communication efficiency gains of repair techniques should decrease. For loosely coordinated domains

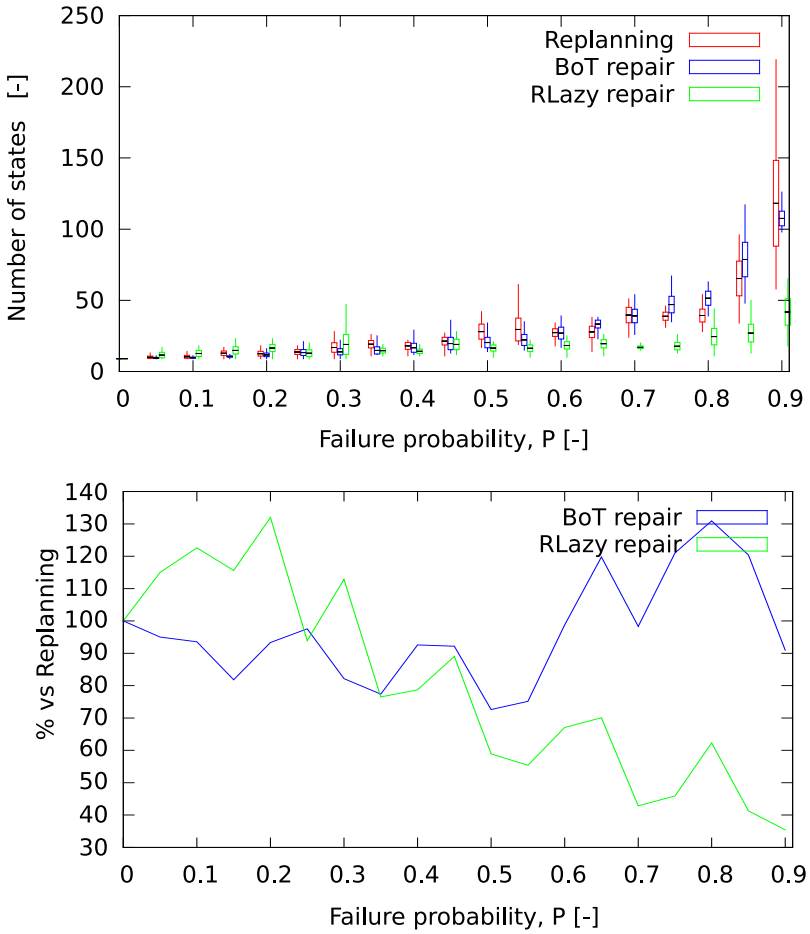


Figure 5.5.3: Experimental results of the execution length metrics for LOGISTICS domain with 3 agents and action failures.

> *the communication efficiency of plan repair should be on-par with that of the replanning approach.*

To validate the auxiliary hypothesis the experiments were run with ROVERS as a loosely coordinated and SATELLITES as an uncoordinated planning problem. The results in Table 5.1 shows that the plan repair 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 5.5.4 shows similar results for various failure probabilities P. The presented results support the auxiliary hypothesis.

The third batch of experiments targeted the perturbation magnitude of the plan failures. The second auxiliary hypothesis states:

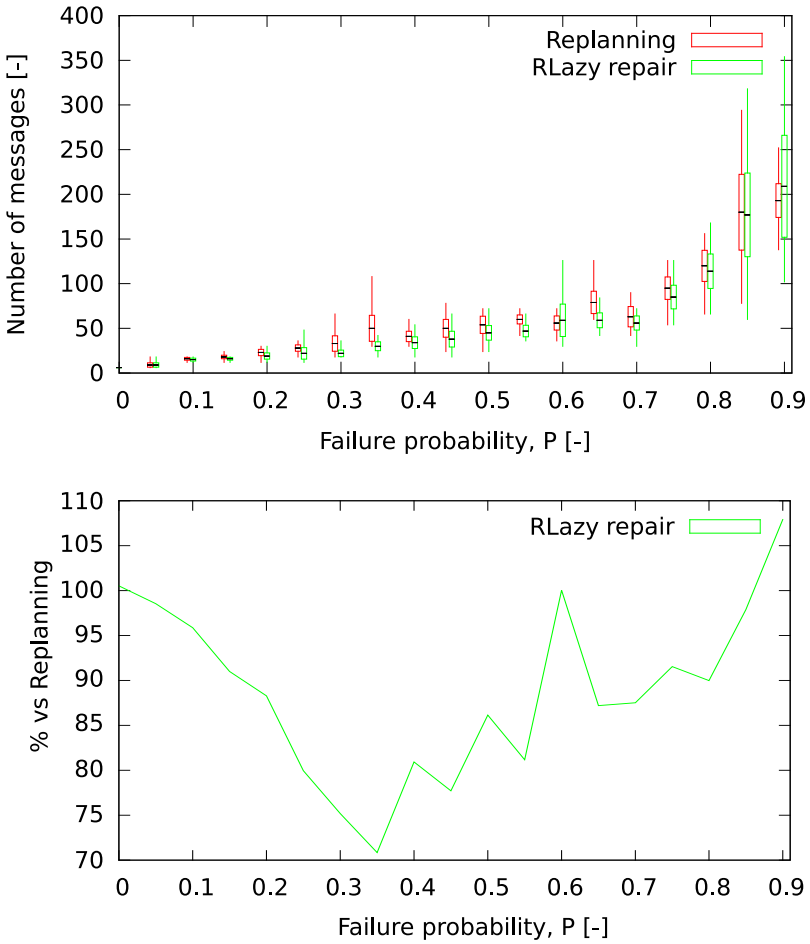


Figure 5.5.4: Experimental results for ROVERS domain with 3 agents and action failures.

Communication efficiency gain of plan repair 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, the LOGISTICS experiment was modified to simulate state perturbations as the model of the environment dynamics. Figure 5.5.5 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

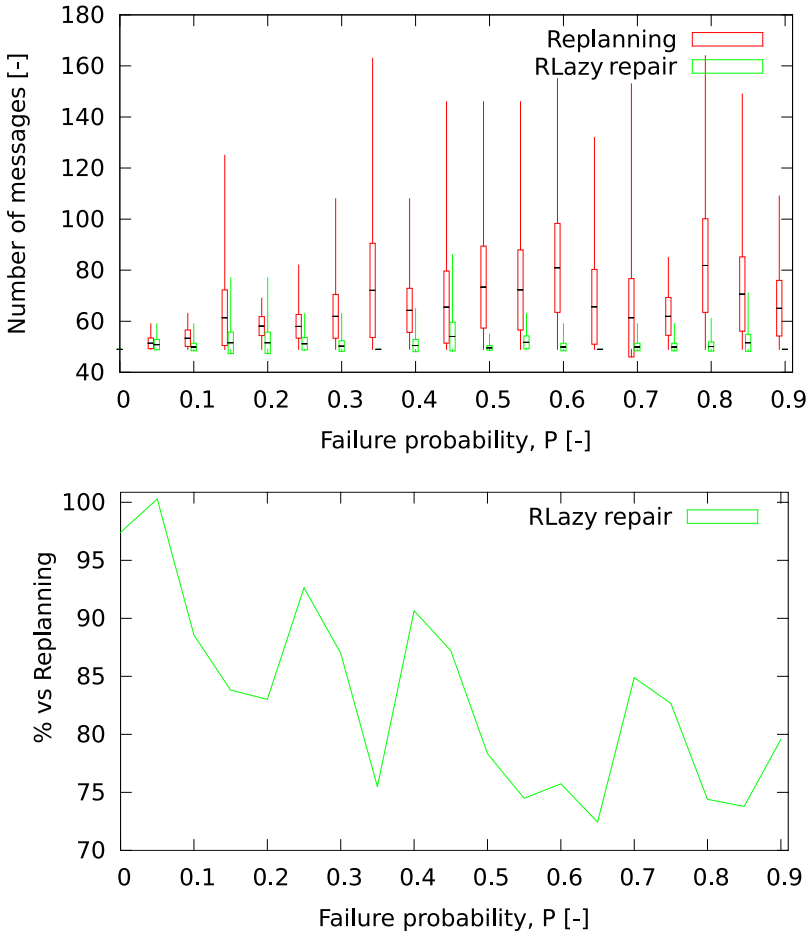


Figure 5.5.5: Experimental results for LOGISTICS domain with 3 agents and state perturbations with $c = 1$.

adding another one. As the chart shows, under random perturbations the plan repair technique lost its improvement against replanning. For stronger perturbations with $c = 2, 3, 4$ (not shown in the figure), the ratio between plan repair 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 repair algorithms. Table 5.1 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.

5.5.2 Number of Repairing Agents

Regardless of the theoretical results presented in [7] showing that the computational complexity of DisCSP-based multiagent planning is not exponentially dependent on the number of the agents, in practical experiments, there is a substantial dependence of this number on required communication and computational effort. This set of experiments analyzes this relation.

Used Algorithms

To validate Hypothesis 2, an extensive set of plan repair algorithms based on the Generalized-Repair algorithm was used. The algorithms used parametrization to act as fully suffix (Back-on-Track-Repair like) and fully prefix (Simple-Lazy-Repair like) approaches respectively and formed three main groups: one *without* agent count minimization, and two *with* agent count 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 a preference between agent minimization, size of preservation of the original plan and bound on the maximal length of the newly generated repair 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 4.4 was extended with an iteration starting only with a set of agents providing at least one action, which can contribute to the repair plan by a required proposition(s), *i.e.*, support part of $S_g \ominus \mathcal{P}_{\text{suf}}$. If such team of agents was not able to solve the plan repair problem, the team was 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 did not exist and the team was still not containing all the agent from $\mathcal{A}$, a random agent was added into the team and the process continues.

Results and Discussion

The experiments were conducted in all presented experimental domains and for all achievable combinations of agent counts. That gave twelve domain and problem instances. Each of the group contained six variances of the algorithms giving 216 experiments with the problem instances. Each of the experiments was averaged over 5 measurements with different random seeds.

Figure 5.5.6 shows results of the first batch of experiments. The first group of repair algorithms not minimizing the number of involved agents (red color) is in most measurements in both

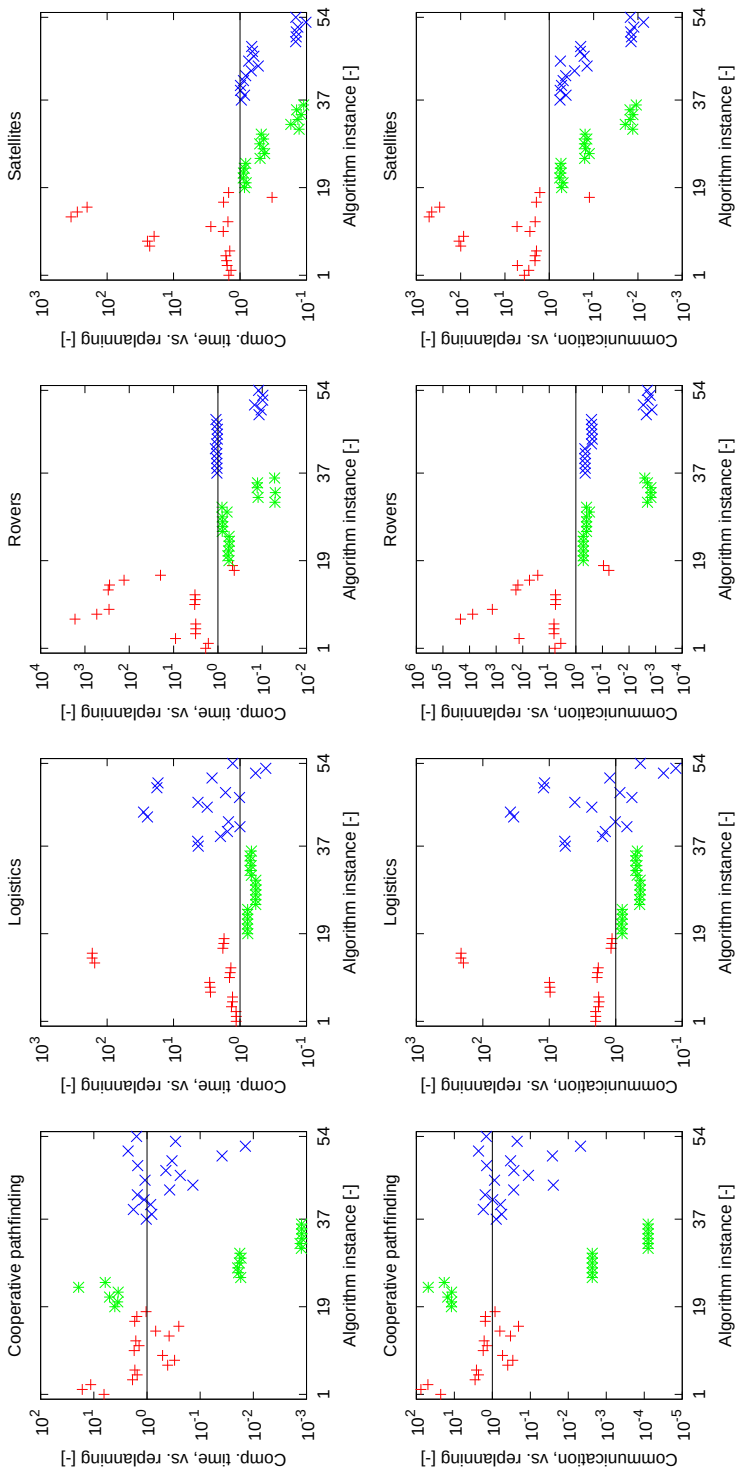


Figure 5.5.6: Comparison of various plan repair algorithms in proportion to replanning (black line at $y = 1$) with failure probability $P = 0.3$. Each point represent a mean of several runs of one of the particular repair algorithms. The red group contains plan repair algorithms using only the full 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.

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

Figure 5.5.7: A multiagent 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.

computational and communication metrics worse than the baseline replanning algorithm (such measurements are over the black line representing replanning). 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 repair 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 coordinated domains decreasing the communication and computational overheads with decreasing coordination of the domains. However in tighter coordinated domains the agent minimizing prefix-based algorithms fall behind the replanning baseline. In LOGISTICS domain, only 33% of the algorithms were better by means of communication overheads and only 18% by means of computational overheads. With increasing coordination the approach lose more. These results *support the second 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 repair. In the LOGISTICS domain the separation between the best prefix-based and the best suffix-based plan repair algorithm is about a half an order of magnitude in favor of the prefix-preserving approach. On the other hand, in COOPERATIVE PATHFINDING, suffix approaches gain an order and more.

5.5.3 Repair of Long-term Dependencies

The intuition behind the third 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 the 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 described in this section were conducted to validate this concept.

Used Algorithms

The most straightforward approach here is to compare the two plan repair algorithms reusing the whole original plan either as a prefix or as a suffix. These algorithms are again modification of the plan repair part of Algorithm 4.4 such that there is no iteration over various $f \in F$ and $g \in G$, but only two fixed values. The *pure prefix algorithm* (Lazy like) uses fixation $f = \{m\}, G = \{0\}$ and the *pure suffix algorithm* (BoT like) uses fixation $F = \{0\}, G = \{m\}$.

Furthermore, to be able to demonstrate the behavior and to explain the results, recall details on the LOGISTICS domain. The problem used in the experiments was described in Section 5.1 as the representative problem Π^{log3}. It contains two trucks $\mathsf{T_1}$ and $\mathsf{T_2}$ and an airplane A, two storage depots $\mathsf{d_1}$ and $\mathsf{d_2}$ and airports $\mathsf{a_1}$ and $\mathsf{a_2}$. The trucks move $\mathsf{m}(\mathsf{from}, \mathsf{to})$ only within their cities and the airplane flies $\mathsf{f}(\mathsf{from}, \mathsf{to})$ only among airports. All vehicles can load $\mathsf{l}(\mathsf{package}, \mathsf{location})$ and unload $\mathsf{u}(\mathsf{package}, \mathsf{location})$ one package p at any of the locations. The trucks start at their depots and the airplane at $\mathsf{a_1}$. The goal is to transport the package from $\mathsf{d_1}$ to $\mathsf{d_2}$. An optimal multiagent plan solving this particular instance is depicted with time steps in the matrix form in Figure 5.5.7.

Results and Discussion

To validate Hypothesis 3, the pure prefix-preserving and pure suffix-preserving repair algorithms were run in all testing domains. Ratio of successful repairs of these two repair algorithms against replanning was measured by means of computation time. In Figure 5.5.8, the results of these experiments are summarized.

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 and 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) and therefore the private dependencies are never longer than four actions.

Multiagent 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. Although 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).

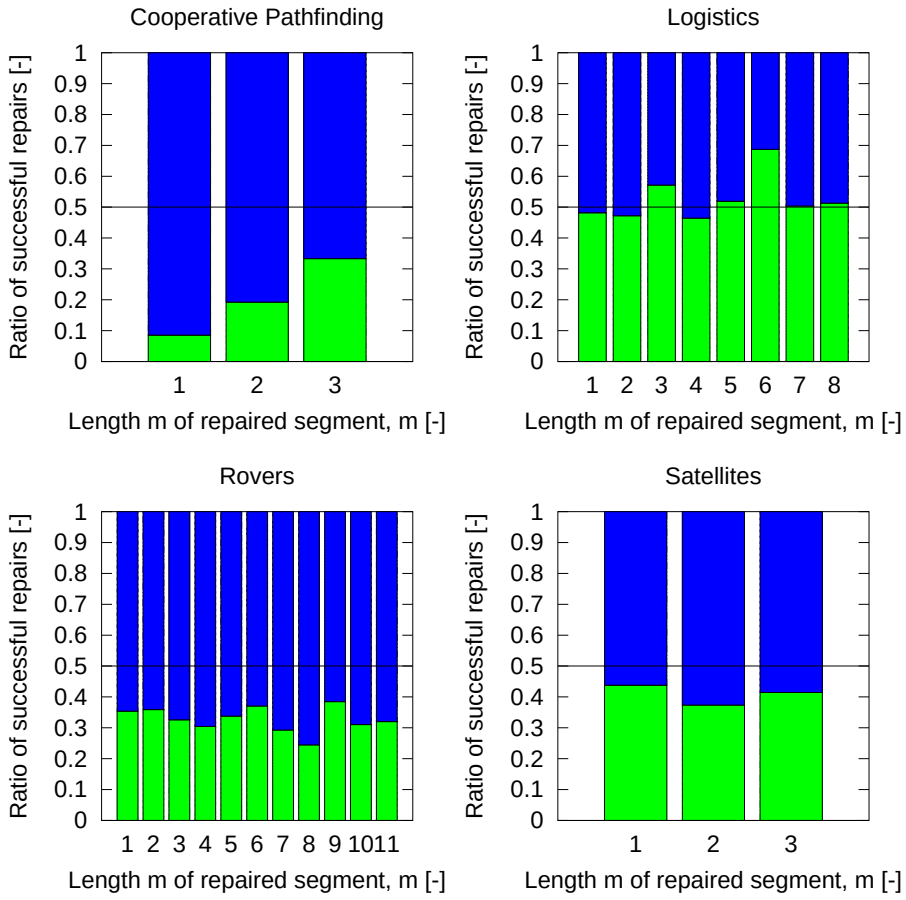


Figure 5.5.8: Comparison of success ratio against replanning between prefix-preserving (blue, Lazy) and suffix-preserving (green, Back-on-Track) plan repair algorithms.

In such repair problem, even if one of the leading actions in a private component fail, lazy approach solves nearly the complete problem only by reusing the original plan. More precisely, it reuses the original solution for the rest of the private components and all the public actions except one of the failed agent. As the results show, the prefix-based repair is always better then the suffix-based and the ratio between these two is rather 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 actions in the example plan and occupy the complete length of it. As the results show in Figure 5.5.8, there are two distinctive peaks where the suffix repair outperforms the prefix repair, additionally with a increasing trend. The first one is for repair plans of length $m = 3$ and the other one is for $m = 6$.

As presented in Figure 5.5.7, these lengths correspond to the package handover points in the plan, more precisely, to 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 the pure lazy approach (similarly to the Lazy approaches) causes the package is left in the last vehicle and the rest of the team finishes the executable remainder of the plan, which in principle means the vehicles are moving, but they are not transporting the package. On the other hand, in the same circumstances, the suffix-preserving 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 $m = 3$ (the $\overline{\mathsf{u}(\mathsf{p},\mathsf{a}_2)}$ action), 2 agents are needed to do lazy repair, because firstly the executable remainder of the original plan is reused to the last state without the package and than the planner has to be used to generate repair plan $\mathcal{P}^*$reverting all the moves and planning to one of the goal states again. 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 pure suffix approach (similarly to Back-on-Track) generates only 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 suffix repair.

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 repair algorithm gains against the prefix one.

The results of these experiments, namely of LOGISTICS and COOPERATIVE PATHFINDING, *moderately support* the third hypothesis. In Chapter 7, the conclusions on this hypothesis are extended with usage of distributed forward-search planner.

5.5.4 Repair Appropriately Reusing the Original Plan

A fundamental principle behind the presented plan repair algorithms can be described as action ordering preservation or, in other words, reuse of parts of former plans. It is not intuitively clear what is a good strategy how to reuse the original plan, moreover in relation to a particular planning domain. The experiments conducted in this sections provides insights into this issue.

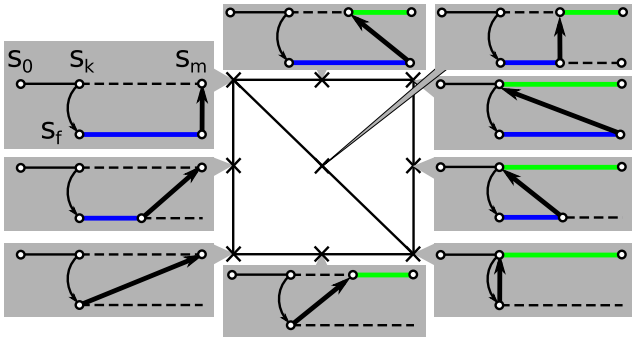


Figure 5.5.9: Scheme of a two-dimensional space representing plan repair algorithms preserving different parts of the original plan and reusing it in different ways. The blue segments represent prefix-based reuse and the green ones the suffix-based reuse. The notable states are: initial state s_0, last 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. For the sake of brevity, other states are not present in the diagrams.

Used Algorithms

A battery of plan repair algorithms was prepared to validate Hypothesis 4. In these experiments, the algorithms were modified in a sense how and how much they reuse the original plan. Such modifications led to a two-dimensional discrete space of different plan repair algorithms, as depicted in Figure 5.5.9, 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 a context of execution of the original plan. The diagrams start with an environment 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 $s_m \in S_G$. However, during execution of an action following a state s_k, an action 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 (more precisely one of the defined goals), the agents use one of the plan repair algorithms, which under the condition of perfect execution, would transform the world from s_f to a $s_m \in S_G$.

In Figure 5.5.9, 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 $F = \{m\}$. The other dimension represent number of actions reused as suffix of the final repair plan, that is fixing the iteration parameter $G = \{m\}$. In the presented scheme, $\mathcal{P}_{\text{pre}}$ from the Algorithm 4.4 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 reuse the same original plan, the space is always a square with a side of the length m.

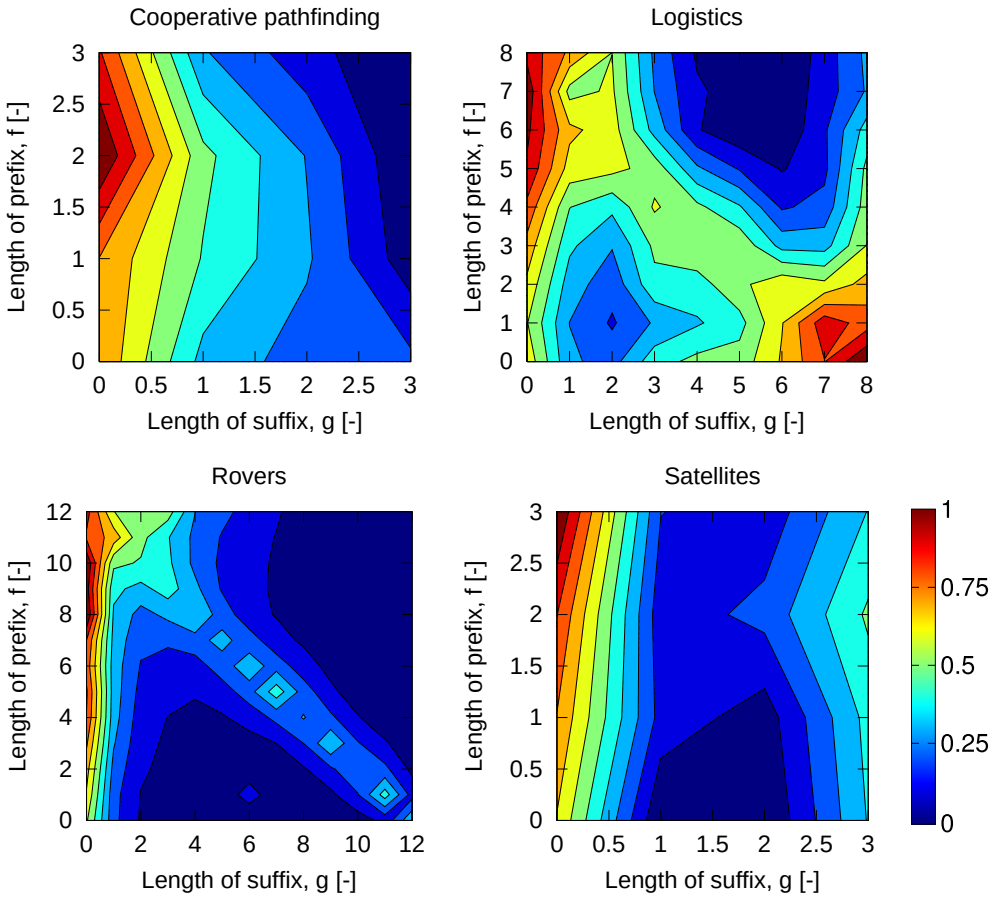


Figure 5.5.10: The maps present prefix (f on y-axis) *vs.* suffix (g on x-axis) preserving repair algorithms by a success rate against replanning in the repair time for all domains with three agents and $P = 0.3$. Red color represent algorithms, which were more often faster then replanning. The top-left to bottom right diagonal represent algorithms neither overusing or underusing the original plan.

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 remainder 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. Therefore it generates a full overlap of the prefix and suffix reuses.

Beside the extremes, the $(0, m), (1, m-1), ..., (m-1, 1), (m, 0)$ diagonal in the space is important

from perspective of the ongoing discussion as well. All the algorithms lying on this diagonal reuse each of the actions of the original plan exactly once in the original order. Meaning, the original plan is neither overused nor underused. Formally:

Definition 20. (*m-normal plan repair*) Let $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ be a multiagent plan repair problem, then an algorithm R is a *m-normal plan repair*, iff R solves the problem Σ by a multiagent 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 fourth and last hypothesis, a randomized sampling of the algorithm space was used to search for more successful algorithms lying on the m-normal repair diagonal by the hypothesis. The results are present in Figure 5.5.10.

Firstly, the sampling experimental process measured for each encountered repair problem the computation time of a corresponding replanning algorithm. After this base-line measurement, a tested repair 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 results. During the experimental execution and plan repair, different lengths of the original plan were used, that is 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 length of the initial multiagent plan generated.

As the resulting maps show, the hypothesis clearly holds for rather coordinated domains with longer plans (LOGISTICS, and ROVERS). In the fully coordinated domain of COOPERATIVE PATHFINDING, the diagonal is also present, but because of considerably short repaired plans, it degenerated. In the experiment with SATELLITES, the diagonal is not present. In the results before the merge (not presented in the figure), there was no apparent pattern, *i.e.*, the particular maps for different m contained shapes without any obvious relation.

These results *support* Hypothesis 4 with an auxiliary observation, that the effect is decreasing as the coupling of the domain decreases.

Chapter 6

Validation in Multiagent Simulation

In the previous chapters, the formal and experimental evaluation of the plan repair techniques was presented. The evaluation provided a strong evidence that plan repair benefits in lowering both computation and communication overheads under the expressed conditions and improves efficiency in continuous process of execution and repairing in dynamic environments in contrast to replanning from scratch. In this chapter, a demonstration of usage of plan repair in high-fidelity simulated world is provided. Such demonstration validates proposed techniques in the sense of the introductory motivation for plan repair in environments close to real-world.

A key challenge in deployment of such algorithms with initially theoretical design and synthetic evaluation is a software process allowing deployment and integration with the target system. A development process for such task was designed and tailored for tactical missions. The term *tactical mission* will denote a class of problems where a multi-robotic team carry out various tasks of information gathering, supporting disaster relief operations or assist in humanitarian missions [60, 38].

In the rest of the chapter, the development process will be described and explained, a software toolkit engineered with respect to the proposed process will be introduced and finally a deployment of a plan repair technique will conclude the chapter.

6.1 Development Process

The proposed development process is based on the *Simulation-aided Design of Multiagent Systems* (SADMAS) methodology [58, 26]. The core principle of the SADMAS methodology is an iterative development process supported by approximated validation using testbeds of increasing fidelity.

The goal of the process is a successful, cost-efficient deployment of the application on the target system, typically a hardware platform. The iterative process of the application development is based on the feedback from approximated testing. The extent of approximation can be described in two dimensions: *level of abstraction* (how much is the target system simplified) and *scope of abstraction* (which parts of the target system are simplified). In result, the initial system consisting of highly abstract algorithms is iteratively transformed, with increasing level of detail in each step, into a system deployable on the target hardware platform.

At the beginning, the algorithms are theoretically designed and evaluated in synthetic environments, described using general mathematical structures such as graphs as the plan repair algorithms in the previous chapters were. However, right from the start, the experiments are performed on the algorithms within the framework of the *target* simulation system. This means that the interfaces between the control algorithm and the simulated environment must be flexible enough to allow easy redeployment of the algorithm to higher-fidelity simulation environments. After validating and verifying the algorithm in a synthetic environment, parts of the simulation can be extend or replaced and the algorithm is re-validated in a simulation containing more aspects of the target environment, *i.e.*, having a lower level of abstraction.

Occasionally, after the abstraction of the simulation environment has been decreased, the tested algorithm has to be conservatively adapted. A *conservative adaptation* of an algorithm is an adaptation that preserves all the desired mathematical properties (*e.g.*, soundness, completeness) for the price of possibly newly added domain-specific constraints on the validity of these properties. The final sum of such adaptations results in a theoretically-backed algorithm applicable in highly detailed simulated environments. The mathematical properties of the algorithm stay valid under the constraints introduced by the applied conservative adaptations.

6.1.1 Environment Model

Such development process requires a simulator that offers high flexibility in terms of scenario storyboards that can be constructed. In particular, one should be able to freely choose the simulation entities that form a simulation instance (this requirement is related mainly to the scope of abstraction in SADMAS) and the levels of detail on which are the entities simulated (this requirement is closely related to the level of abstraction in SADMAS) to enable successive adaptation of the tested algorithms.

A fundamental part of the simulation platform is a model of the virtual environment. To satisfy the above-stated requirements, the platform distinguishes between the description of the *simulated state* and the *state controllers* which animate the simulated world.

The state of the environment is represented by sets of state variable containers called *state storages* (see Figure 6.1.1). Each state storage is responsible for holding a specific part of the

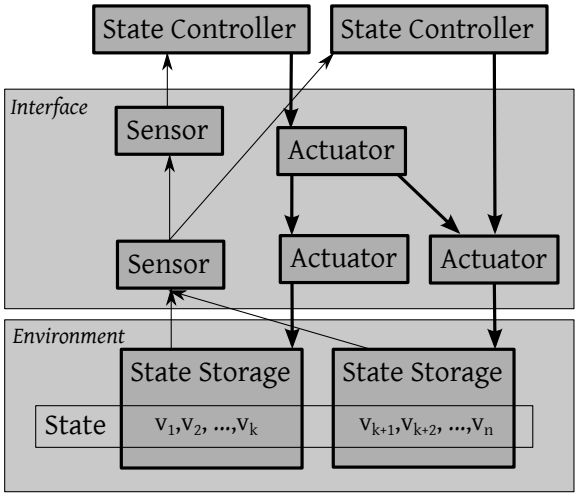


Figure 6.1.1: An example of a simulated environment, described by state variables $v_1, ..., v_n$ divided into two state storages. The state controllers (*e.g.*, agents) perceive and act in the environment through a set of sensors and actuators respectively. The top-level sensor and the top-level actuator act as a high-level abstraction for the low-level sensors and actuators (*e.g.*, autopilot actuator on top; yoke and pedals actuators in the bottom).

current state, *i.e.*, all the state storages together constitute the full description of the current state of the simulated environment.

State controllers can be both (a) the control algorithms tested in the simulated environment, specifically for context of this work plan repair algorithms and (b) program logic describing the mechanics of the simulated environment. The state controllers interact with the state of the environment indirectly through a set of interfaces called sensors and actuators. A sensor is an interface through which a particular part of the environment state can be read. Analogically, an actuator is an interface used to change a part of the environment state. Sensors and actuators are the only components that can directly access the state storages.

There are no *a priori* restrictions on what can a state controller model be, a controller can be a mechanism simulating physical laws of the environment (*e.g.*, application of the gravity force to all simulated entities having mass), a simple reactive algorithm (*e.g.*, simulation of swarm systems), or a complex deliberative algorithm (*e.g.*, cognitive cooperating agents). The elements of the environment having no associated controllers remain fixed in their initial state. These can be the shape of the landscape, buildings, bridges, etc.

Furthermore, the sensors and actuators are not strictly limited to have a state controller on one side and a state storage on the other. A sensor or an actuator can be connected to other sensors or actuators, effectively forming an interface network. Such approach to the design of simulated environments leads to a significant cost reduction on implementation and debugging of

the individual experimental scenarios, as the interface network can be flexibly reconfigured and the implementations of the individual sensors and actuators can be reused.

6.1.2 Simulation Process

The environment model has to be accompanied by a functional part, describing the behavior of the simulation. In general, an experimental validation requires statistical results from a large number of simulation runs. To ensure properties of the tested algorithms during the adaptation process, the simulation platform has to facilitate construction of experiment suites allowing execution of *reproducible experiments*.

While most of the abstract mathematical algorithms are well analyzed and strongly experimentally evaluated, it is much more challenging to design, run (and debug) replicable experiments in complex, high-fidelity robotic simulations involving dynamic entity behaviors and emergent behavioral phenomena. Large-scale simulations involve various aspects of nondeterminism, which can lead to non-reproducible simulation runs. Such factors include parallel and random processes, as well as the limitations of the underlying hardware, such as CPU scheduling or memory swapping, etc. To ensure reproducibility of experimental runs, the simulator has to follow the concept of *in vitro* simulation. That is a simulation that controls all the aspects of the modeled system. Besides controlling the evolution of the simulated world, the simulator must also have an ability to control the execution (*i.e.*, suspend and later resume) of the validated control algorithms. Further, the simulator has to be immune to the race conditions and different results of process scheduling on the underlying computational infrastructure. Finally, any random processes involved in the simulation must be also under the control of the simulator, so that the same sequences of random events are generated in any two runs of the same experiment.

The need to execute large numbers of reproducible simulation runs turned out to hinge on the speed of simulation execution and the ability to make the runs deterministic on demand. To tackle this issue, departure from the classical exclusive model of centralized discrete time ticks and adoption of the *event-based simulation mechanism* is required. This allows the system to disrespect real-time constraints of the wall-clock ticking mechanism and run the simulation as fast as possible given the available computational hardware resources (memory and CPU). The main advantage of this approach is that the time periods containing no simulation events can be skipped and thus the simulation runs significantly faster. However, at the same time the resulting simulator still features the ability to run at real-time simulation speed (for demonstration purposes or for hardware-in-the-loop experiment).

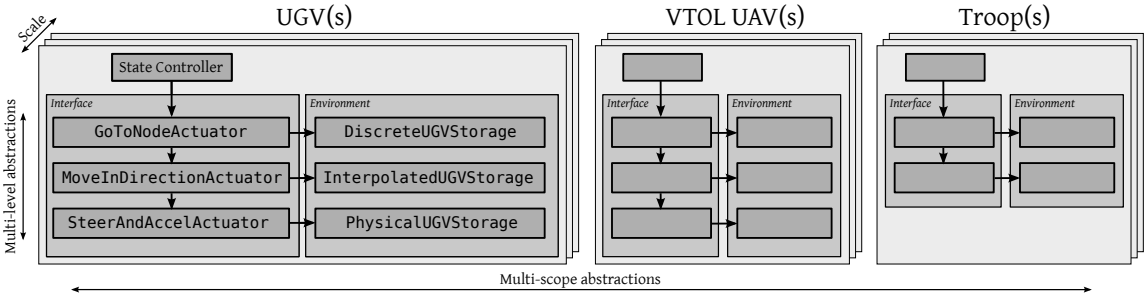


Figure 6.1.2: State storages and related actuators for description of three levels of abstraction for UGVs, three levels of abstraction for VTOL UAVs and two levels for simulated troops. The scope of abstraction is demonstrated using three different types of simulation entities, which can be variably used together. There is one *Interface* block and one *Environment* block divided into three views from perspectives of the particular simulated entities.

6.1.3 Example of a Multilevel and Multiscope Abstractions

The model based on state storages, sensors, actuators, and loosely coupled controllers offers high flexibility. In result, a programmer can add and remove new types of simulated entities easily (scope of abstraction) and easily switch between different types of simulation modes (level of abstraction) for individual entities. Moreover, as expected from a multiagent simulation, the model also offers scalability in terms of numbers of simulated entities.

For instance (see Figure 6.1.2), three types of abstractions can be defined for unmanned ground vehicles (UGVs) used in the simulation and represent them by three separate state storages `DiscreteUGVStorage`, `InterpolatedUGVStorage`, and `PhysicalUGVStorage`. The first one describes the current state of a UGV by a node on a street graph. The second one enriches the by-node description by a position vector (x, y) representing the position of a UGV on a 3D mesh of the ground surface. The last abstraction extends the state further with a description of a fully dynamic state comprising position (x, y, z), velocity $(\dot{x}, \dot{y}, \dot{z})$, acceleration $(\ddot{x}, \ddot{y}, \ddot{z})$ and rotational components $(\varphi, \theta, \psi), (\dot{\varphi}, \dot{\theta}, \dot{\psi}), (\ddot{\varphi}, \ddot{\theta}, \ddot{\psi})$. To control the state stored in these storages, three actuators `GoToNodeActuator`, `MoveInDirectionActuator`, and `SteerAndAccelerateActuator` are used. One can implement an actuator to control the respective state storage directly, but it is also possible to implement an actuator to control storages indirectly through other actuators. In practice, such coupling will result in an algorithm that recursively translates the higher-level control to lower-level control. For example, if a UGV state controller based on the node-to-node mode of navigation is required to drive a physically simulated UGV, it uses the following actuator sequence: `GoToNodeActuator` $\rightarrow$ `MoveInDirectionActuator` $\rightarrow$ `SteerAndAccelerateActuator`. It is obvious now that any of the presented state storages can be used as long as the controlling algorithm uses only the top-most actuator, *i.e.*, `GoToNodeActuator`. In effect, a high-level algorithm controlling a UGV

can be designed only on node-to-node basis using `GoToNodeActuator`, but it can be immediately tested in all prepared levels of abstraction (discrete, interpolated, physical).

To extend the scope of the simulation, simulated vertical take-off and landing unmanned aerial vehicles (VTOL UAVs) are added. Similarly to the simulated UGVs, VTOLs have three levels of abstraction represented by three state storages and three related actuators. The actuator sequence follows the same pattern as in UGVs.

The last example enriches the simulated environment with entities representing troops. Here, only two levels of abstraction are created. They are represented by two state storages `DiscreteTroopStorage` and `DirectedTroopStorage`. The first level of abstraction is similar to `DiscreteCarStorage` (representing the position of a trooper in terms of street graph nodes), the latter describes the ground position and the heading angle (x, y, φ) of a trooper. To control the troops new `WalkToNodeActuator` and `MoveAndTurnActuator` has to be created, as `GoToNodeActuator` cannot be reused in place of `WalkToNodeActuator,` since the UGV actuator uses a different control logic to simulate the movement (although the input parameters and the results are identical for both the actuators—both the UGVs and the troops move from one node to another—for the UGV, the duration of the movement can be computed from the engine power, for the trooper the duration of the movement can be, for instance, a function of the weight of the personal gear carried).

From this point, there are separate components for a UGV, a VTOL and a trooper in the model of environment. In a simulation run, these components can be used separately (only UGVs or only troops) or can be mixed together (*e.g.,* troops following a car). Moreover, different levels of abstraction of various components can be mixed together (for instance, a transportation UGV using trajectory interpolation representing a convoy is followed by physically simulated vehicles representing UGVs accompanied by troops having position and direction representing the support squad protecting the convoy against adversaries moving on node-to-node basis blocking junctions on the street map).

6.2 Multiagent Toolkit Alite

The development process together with the requirements on the simulator led to the architecture described in the previous section. The strong emphasis on the flexibility of the interfaces between the controllers and the simulated environment requires equally flexible software tools able to to help with the implementation of such a simulation system. Alite is a software toolkit that provides such support out of the box.

Alite[1] *[ˈeɪlaɪt]* is a software toolkit simplifying implementation and construction of (not only) multiagent simulations and multiagent systems. It stands on technologies related to the ecosystem around the Java Virtual Machine and it is mostly written in Java. The objectives of the toolkit are

[1] http://alite.agents.cz/

to provide a highly modular, flexible and open set of functionalities supporting rapid prototyping and fast implementation of multiagent applications, mainly focusing on highly scalable and complex simulated environments. The guiding principles underlying the Alite design are (i) modularity, so that the system does not commit a developer to a specific definition of concepts such as *agent*, *environment* and (ii) composability, so that the various components of the toolkit can be put together in a rapid and flexible manner. In result, Alite can be seen as a collection of highly refined functional elements providing clear and simple APIs, allowing a programmer to put together relatively complex multiagent simulation scenarios rapidly.

Alite addresses the problem of multiagent platform resilience in the face of the need to incorporate various *a priori* unknown future requirements by variability in composition of functional elements. The number of possible combinations allows for construction of a wide spectrum of structurally different multiagent applications. This feature distinguishes Alite from the pre-designed frameworks such as Jade[2], Cougaar[3] and Aglobe[4] multiagent platforms. As multiagent application's requirements evolve, the requirements on the agent platform itself are changing. Alite does not provide *a single platform for all*, but rather offers an efficient way to build a platform that fits the specific needs of the multiagent application under development. The application can make use of one or more functional elements available in Alite toolkit.

Among others, Alite provides an implementation of building blocks introduced in Section 6.1.1. An application developer can put together different parts from Alite toolkit to implement a multiagent simulation platform that targets specific requirements of the application in question. In particular, it is designed to facilitate implementation of simulations that adopt the *in vitro* principle and the event-based simulation mechanism as described in Section 6.1.2. Furthermore, Alite contains functional blocks supporting (i) inter-agent communication, (ii) configuration and initialization, and (iii) visualization. The communication package provides an easy-to-use interface that can be integrated with a number of message passing channels. For the tactical mission simulator, a message passing mechanism implemented using the event-based simulation is used. The configuration and initialization uses the dynamic programming language Groovy[5] to configure the parameters and initialize the initial state of the simulation in a concise and flexible manner. Such a flexibility allows a programmer to experiment with structurally different simulation scenarios, a must-have for a successful adoption of the presented development and deployment process. Finally, the state of the simulated world can be displayed using the 2D/3D visualization component. This component is designed as fully separable from the simulation core, therefore if the visualization component is turned off, there is no efficiency burden or negative influence caused. The tactical mission simulator was released as a standalone package under an open source license[6].

[2] http://jade.tilab.com/
[3] http://www.cougaar.org/
[4] http://agents.felk.cvut.cz/aglobe/
[5] http://http://groovy.codehaus.org/
[6] http://jones.felk.cvut.cz/redmine/projects/tacticalenvironment/wiki

The power of Alite's loosely coupled design has shown its benefits during the construction of a multiagent simulator of distributed tactical missions described in this chapter. Beside the main focus on the plan repair algorithms is this thesis, a number of tailor-made domain-specific components integrated with the Alite infrastructure enabled to transparently combine and validate other AI algorithms and control programs written in agent-oriented programming language Jazzyk [52] in a complex environment simulation comprising realistic physical simulation of rigid-body models based on JBullet simulator[7]. Another Alite-backed multiagent application of physically simulated UGVs for the domain of multi-gent cooperation and coordination in complex urban environments has been presented in [66].

6.3 Usage of Plan Repair in a Tactical Mission

Multiagent plan repair should be one of possible approaches needed to plan activities for a team of agents in the dynamic environment. Such activities can be described as multiagent planning as there is a group of simulated robotic assets in the environment which are cooperative by definition of the mission and require coordinated plans to perform their parts in the mission. The plans can be influenced by a dynamism caused by other entities in the simulation which are not in the robotic team. Such dynamism requires techniques which can handle them. Plan repair is one of such techniques.

In the particular tactical mission used in the validation experiment, a goal is to extract a VIP in a hostile urban area (village) by a allied forces. The topology of the village is known *a priori* from a satellite recon. There are troops supported by a heterogeneous team of autonomous robotic assets (two micro VTOLs, two small VTOLs and one UGV), see Figure 6.3.1. The environment dynamism represented by a unknown movement of hostile forces and unknown reactions of the allied forces involved cause failures in plan execution (particularly action failures as defined in Section 5.3). The autonomous units has to adapt (repair their plans) accordingly to be consistent with the behavior of the allied forces. The adaptation should take into account the high-level goals given to the robotic team which is are presumed to be compatible with goals of the rest of the unit (particularly, the robotic team has to provide surveillance of the extraction area). The robotic assets are controlled by a multiagent planner and plan repair algorithm based on the Back-on-Track technique presented in Section 4.1. The positions and numbers of the hostile forces are not known *a priori*.

[7] for more information on the physics simulation see JBullet (http://jbullet.advel.cz/) – a Java port of Bullet Physics Library (http://bulletphysics.org)

Figure 6.3.1: The ground forces (green) supported by the robotic team (2 small VTOLs – orange view prisms, 2 small VTOLs – red view cones, 1 UGV – brown four-wheeled car) in the simulated urban environment.

6.4 Deployment of Plan Repair into Tactical Simulation

According to Section 6.1, the deployment process iteratively adapts an plan repair algorithm towards the full high-fidelity simulator of a tactical environment. Three adaptation steps are depicted in Figure 6.4.1. The adaptation steps respected the decreasing level of abstraction and incremental adding of the three types of assets respect the scope of abstraction.

A *first adaptation step* begun with a theoretical design of a plan repair algorithm general enough to cover the dynamism of the presented tactical mission. The initial design used the Back-on-Track principles, although the implementation started from scratch to precisely follow the presented development process.

First tests and experiments were carried out in a synthetic grid-based environment based on the LOGISTICS planning domain (see Section 5.1). The used domain, coined as CRATES-CRANES, contained mobile cranes which could move in a grid environment and packages each with a goal target position. Each of the grid positions could be accessed only by one crane with an exception of several handover points. Each crane could carry only one package at a time. The cranes had the same three actions `move`, `load`, and `unload` as vehicles in LOGISTICS.

In the *second adaptation step*, the verified algorithm from the previous paragraph was used in a multiagent planning domain representing the heterogeneous team of robotic assets. The assets had to permanently cover the allied troops and not collide with each other. The initial plan of each asset was to move to the extraction point of the VIP and than then provide an area surveillance there. The problem was formalized with a set of tactical-level actions.

(a) (b) (c)

Figure 6.4.1: Levels of abstraction used for design, verification and validation of the algorithm providing plan repair ability for robotic assets: (a) synthetic environment used for the theoretical design, verification and validation of the plan repair algorithm, (b) adaptation of the algorithm to a simplified dynamic model of small VTOLs providing support for the ground team, and (c) example of reconnaissance actions carried out by the VTOLs in the integrated mission.

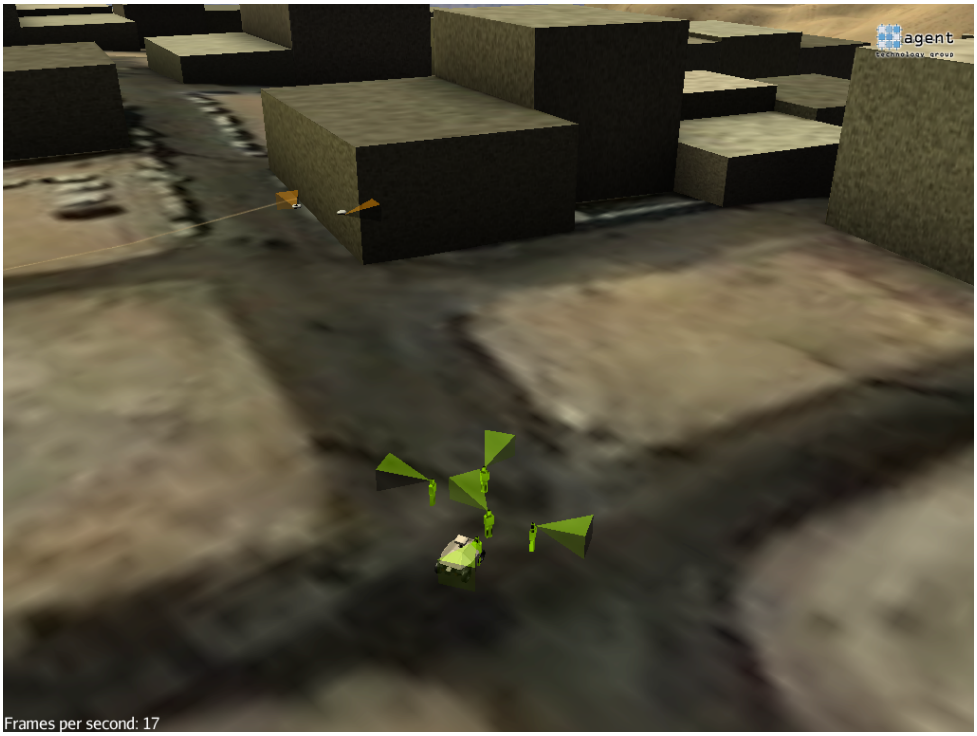


Figure 6.4.2: Two micro VTOLs (with orange view prisms) providing street recon actions in front of the allied team.

The used actions for the planning problem were:

noop The action represents doing nothing, *i.e.*, wait. The action has no preconditions and it is contained in action sets of all planning agents.

move-to The asset is moved from a current position to a new position. The action has no preconditions and it is contained in action sets of all planning agents.

personal-cover The action moves the asset above a designated position of a allied troop and orients it oppositely to provide backward visibility for the covered troop. The action is present only in action sets of the micro VTOLs.

street-recon The action moves and orients the asset as it can look through a requested street (see Figure 6.4.2). The action is present only in action sets of the micro VTOLs.

street-surveillance The action moves the asset along a requested street.

crossing-surveillance The action positions the asset above a crossing and makes it to observe the area around. The action is present only in action sets of the small VTOLs and UGVs.

There are three used conditions:

colliding The condition holds if an action is spatio-temporally colliding with action(s) of other assets.

covering The condition holds if an action causes an allied troop is covered (the asset is in a proximity).

allAlliedCovered The condition holds if all allied troops are covered (in the proximity of all troops is at least one supporting robotic asset).

The plan-repair mechanism is solving the problems caused by the unpredictable movement of the troops. The first move-to action directing the assets to the extraction point in the initial plan has the precondition covering which implies the action cannot be executed as the movement would violate the covering precondition by flying away from the team. Therefore, the precondition causes a failure of the plan. To solve the failure, the adapted Back-on-Track algorithm plans an action solving the problem by execution the personal-cover action for one of the allied troops. Usage of the action solves the failure and makes allAlliedCovered holding and thus the other assets can use other possible actions (surveillance and reconnaissance). The basic conflict avoidance among the assets is caused by the colliding precondition, because of which the assets are always planning repair actions at distinct positions.

Adaptation of the plan repair algorithm to the VTOLs in the domain of tactical support led to an introduction of a restricting condition on the algorithm. During the process of environment

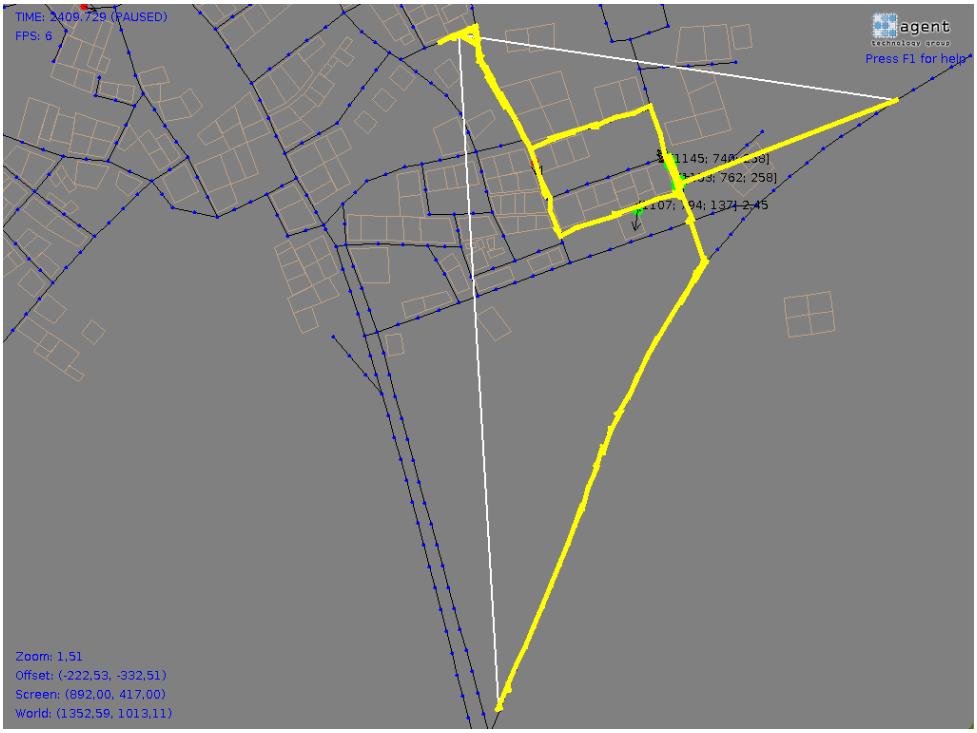


Figure 6.4.3: An individual plan of one of the micro VTOLs. The white lines represent the initial mission plan (move to the extraction point and move the final point). The yellow lines depict a spatial representation of the repaired plan of the asset.

abstraction, a problem with computational tractability arise and thus the maximum depth of the algorithm's search tree had to be limited. Such a change conditioned the soundness and completeness of the algorithm to limited time horizons (equivalent to the coordination length of the resulting plans δ), as the conservative adaptation of the development process dictates.

Finally, in the *third adaptation step*, the constrained version of the algorithm was used in the integrated mission to provide visual support for the troops in the field.

6.5 Results and Discussion

The initial mission plan and the iteratively repaired result of one of the small VTOLs is depicted in Figure 6.4.3.

The plan preserving multiagent technique based on Back-on-Track approach was successfully used in a dynamic high-fidelity environment using the presented development process and was able to direct a team of simulated robotic assets providing a support tactical mission in real-time. The

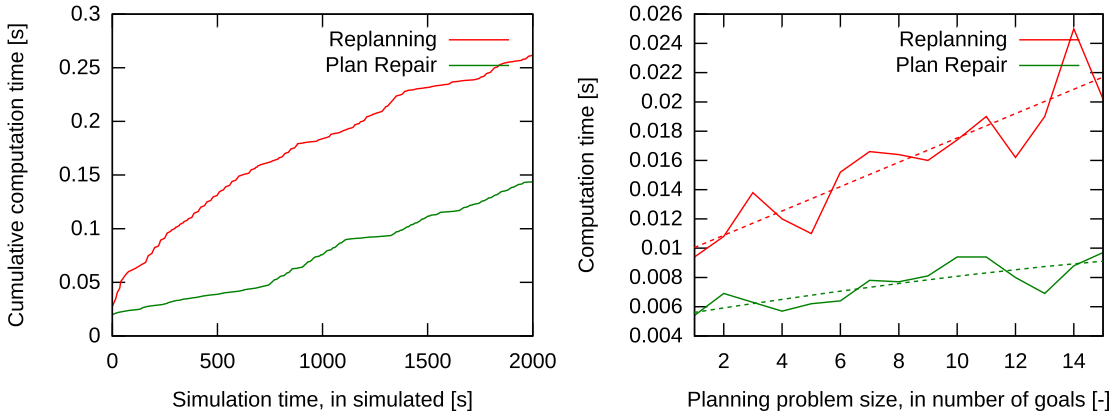


Figure 6.5.1: Experimental comparison of a plan repair technique based on the Back-on-Track approach and replanning from scratch in the high-fidelity simulation of a tactical mission. The results are for one autonomous micro VTOL providing cover for the ground team. The figures show (a) growth of cumulative computation time required for fixing of the original plan during the simulation and (b) dependency of required computation time for fixes from start to 500 simulated seconds w.r.t. number of the goals in the planning problem which relates to the size of the planning problem. The dashed curves are second order polynomial interpolation functions based on the least-squares-error method. Both are nearly linear.

adaptation of the algorithm for the domain of tactical support led to an introduction of a restricting condition on the depth of the search tree to limit the computational complexity of the search. The plan repair mechanism also addressed the problems caused by the uncertain movement of the supported troops and demonstrated its usability in continual planning scenarios. In this particular usage, the plan repair technique resembles techniques of reactive planning, but with an advantage of a permanently sound plan to the goal, which is usually impossible in reactive techniques.

Experimental results focusing on complexity comparison of the deployed plan repair algorithm in high-fidelity simulation are summarized in Figure 6.5.1. The results are shown for one support micro VTOL. Since in this case the communication complexity is the same for both plan repair and replanning, as both the algorithms solve only short horizon coordination problems, the results focus only on computation complexity.

The first experimental results (a) show a cumulative time which grow during the simulation as the plan repair or replanning algorithms has to fix action failures because of the moving ground team. The replanning from scratch (red curve) gets simpler during the execution, since the replanning problem successively requires shorter plans from the failed state s_f to the final goal in S_G. Hence the computation time grows more at the beginning, than near to the end. The com-

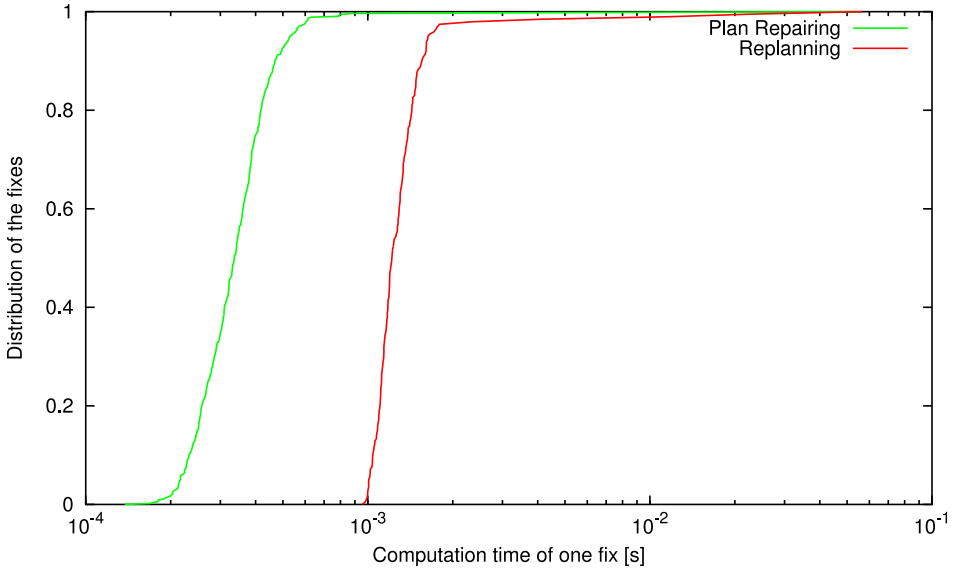


Figure 6.5.2: Distribution of time durations required for one fixing of a failure. The experimental setting is the same as in Figure 6.5.1

putation time required by the repair algorithm is significantly lower and the slope is similar to the simplest replanning problems near the end of the simulation. In this particular domain, the plan repair approach required approximately 2× less computation time than replanning from scratch. This difference would grow with grow of the planning problem size. To confirm this hypothesis experimentally, the other results show relation between number of goals in the planning problem, which corresponds to the size of the planning problem and computation time required by plan repair and replanning from beginning of the simulation to arbitrarily picked point in the simulation of 500 simulated seconds. The chart shows more steeper growth for the replanning than for plan repair. Particularly, the interpolated dashed lines are second-order polynomials derived by the least-squares-error method. For replanning, it is

$$+0.00017x^2 + 0.84x + 9.3,$$

and for plan repair

$$-0.0073x^2 + 0.36x + 5.2.$$

Both the relations are therefore nearly linear as the x^2 coefficients are in both cases negligible. The linear coefficients substantiate that plan repair is less dependent on the size of the planning problem than the replanning from scratch. Since the planning problem in this case is a variation

on path-finding, the growth of replanning is also polynomial and not exponential, as it would be in case of more complex domains.

To conclude the experimental part of the validation, in Figure 6.5.2, it is presented a distribution of all particular plan repairs and replans during the simulation sorted according to their durations. The results show that the fastest plan repair is in order of magnitude faster than the fastest replanning. The plan repair processes requires typically $100 \sim 300\mu s$ and the slowest repairs took $\sim 900\mu s$. The duration of the replanning process is typically $1000 \sim 1800\mu s$ and there is about 5% of replanning fixes that took $1800 \sim 55000\mu s$, the slowest replanning took $\sim 60ms$. Since these results correspond to successful fixings of the plan, in this particular scenario, it gives an answer to a question how would be covered the ground team by the robotic VTOL, if the available time for its decision making would decrease. The results can be understand as a relative measure which means that the team would begin to be uncovered with an order of magnitude shorter allowed decision time by plan repair the than by replanning from scratch.

Chapter 7

Validation with Forward-search Multiagent Planner

In Chapter 5, several experiments were impossible to carry out because of the limitations of the DisCSP-based planner. The particular issues with the planner were also identified by the authors of the planner recently. Improvements of scalability were proposed in [50, 49] by leaving the DisCSP-based approach and moving to a successful principle in classical planning—A^* or a variation on *Best-First Search* (BFS) with highly informed automatically derived heuristics.

To validate the properties of the proposed plan repair algorithms with another multiagent planning approach, a domain-independent multiagent planner, based on the principles presented in [50], was designed and implemented. The key properties of the planner summarized in adjectives are *Multiagent Distributed Lazily Asynchronous (MADLA)*, therefore the MADLA Planner.

This chapter describes the planner both w.r.t. the algorithmic design and key implementation details. Afterward, a selected set of the experiments from the previous sections were replicated using the new planner. The results are discussed and concluded.

7.1 Design of the Planner

The DisCSP-based domain-independent deterministic multiagent planner from [51] proposed a first step in the research field of planning algorithms solving problems based on the MA-STRIPS formalization. This approach precisely followed the ideas in [7], however exposed a couple of issues making the approach incomparable in efficiency with current state-of-the-art implementations of classical planners. One of the issues was bad scalability with growing length of the coordination part of the resulting plans δ. That is one of the reasons why the MADLA Planner was from the beginning designed with state-of-the-art principles from singleagent planning in mind and did not

used the DCSP-based planning approach. Particularly, it uses a forward-search using Best-First Search with an domain-independent heuristics both extended to the distributed multiagent setting.

The formal definitions from Section 4.6.3 has to be slightly extended for further explanation of the MADLA Planner. An α-internal (that is private for agent α) and public subset of all facts $\mathcal{L}$ will be denoted as $\mathcal{L}^{\alpha-\text{int}}$ and $\mathcal{L}^{\text{pub}}$ respectively where $\mathcal{L}^{\alpha-\text{int}} = \mathsf{atoms}(\alpha) \setminus \bigcup_{\beta \in \mathcal{A} \setminus \alpha} \mathsf{atoms}(\beta)$ and $\mathcal{L}^{\text{pub}} = \mathsf{atoms}(\alpha) \setminus \mathcal{L}^{\alpha-\text{int}}$. Facts relevant only to one agent α are denoted as $\mathcal{L}^{\alpha} = \mathcal{L}^{\alpha-\text{int}} \cup \mathcal{L}^{\text{pub}}$ and a *projection* of a state s^{α} to an agent α is a subset of a global state s containing only public facts and α-internal facts, formally $s^{\alpha} = s \cap \mathcal{L}^{\alpha}$. The set of *public actions* of an agent α is defined as $\alpha^{\text{pub}} = \{a \mid a \in \alpha, \mathsf{atoms}(a) \cap \mathcal{L}^{\text{pub}} \neq \emptyset\}$ and *internal actions* as $\alpha^{\text{int}} = \alpha \setminus \alpha^{\text{pub}}$. The symbol a^{α} will denote a projection of action $a \in \beta, \beta \neq \alpha$ for agent α, *i.e.*, action stripped of all other agents' propositions, formally $\mathsf{atoms}(a^{\alpha}) = \mathsf{atoms}(a) \cap \mathcal{L}^{\alpha}$.

Multiagent Best-First Search

Similarly to the proposed solution in [50], in the approach behind the MADLA Planner, each agent comprise its own OPEN and CLOSED lists and runs its own search in the state space using only the agent's atoms $\mathcal{L}^{\alpha}$. Since the search is based on expansion of states by applicable actions, the only usable actions are $a \in \alpha$ of the particular searching agent α and projections of the public actions from the other agents $a^{\alpha} \in \bigcup_{\beta \in \mathcal{A} \setminus \alpha} \beta$.

Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ be a multiagent planning problem with $\mathcal{A} = \alpha_1, \ldots, \alpha_n$ (see Definition 2). All agents begin with their projections of the initial state s_0^{α} as the first state in their OPEN lists. In parallel, the agents expand states from their OPEN lists ordered by evaluations of the states $f = g + h$ where one part is its cost g from the initial state s_0^{α} (*i.e.*, number of preceding actions, as each action's cost is fixed to $c = 1$) and a heuristic estimate h which will be explained in further paragraphs. If the examined state is not a goal and therefore neither the solution, the process of expansion firstly moves the expanded state from the OPEN list to the CLOSED list and subsequently searches for all actions of the expanding agent applicable in that state. Such actions are applied to the state and the resulting states are heuristically evaluated and added to the OPEN list (states already in the CLOSED list are ignored).

The principle described in the previous paragraph precisely corresponds to a couple of classical BFSes run by more agents in parallel. To extend the algorithm to the Multiagent Best-First Search (MA BFS), a message has to be sent to all agents, in a case a state was expanded by a public action. On receiving of such message, it has to be added to the OPEN list of the receiving agent. This straightforward principle causes the search passes to other agents if an action prospectively influencing atoms they consider was reached by one of their team members. In effect, the search process is distributed among the agents and in one agent's pass both the internal plans and the coordination public actions are searched for. Algorithm 7.1 presents the principle in a pseudo-code.

Algorithm 7.1 Multiagent Best-First Search.

```
1: 𝒪 ← OPEN list (of state ordered by their values f)
2: 𝒞 ← CLOSED list (of states)
3: 𝒪 ←initial state
4: repeat
5:   if 𝒪 ≠ ∅ then
6:     s ←poll(𝒪)
7:     if s ∉ 𝒞 then
8:       if s is goal then
9:         return solution(s)
10:      end if
11:      𝒞 ← 𝒞 ∪ s
12:      compute heuristic h for s
13:      compute value f = g + h for s
14:      if s was reached by public action then
15:        broadcast s to other agents
16:      end if
17:      𝒪 ← 𝒪∪expand(s)
18:    end if
19:  end if
20: until false
```

Multiagent Heuristics

The key part of the MA BFS algorithm is computation of heuristic estimate for the expanded states. The idea of a heuristic is to provide as precise as possible estimation of distance from the current state to a goal state. The tradeoff of a heuristic is the more precise the estimate is the more demanding it is on computation and communication. Since the cost of actions is fixed to $c = 1$, the estimated distance is counted in a number of actions.

In contrast to the planning process, the heuristic estimation has not to be complete in the sense that if there exists a plan the heuristic has to return a value (it can return $h = \infty$ meaning the distance is unknown). This allows to design heuristics completely unaware of the fact there are other agents and atoms as well as actions other than that of the estimating agent. Such heuristics were proposed and used in [50]. Since the heuristics are based on actions owned only by one agent the public actions has to be projections, they are denoted as *projected heuristics* h^α.

More challenging heuristic estimates do not ignore the other agents and in extreme tries to come up with the same result as a heuristic would compute in centralized version of the planning problem. Such approach was designed during the research work behind this thesis and proposed in [62]. In multiagent planning, such heuristics have to be, however, computed distributively using communication among the agents. Such heuristics will be denoted as global *multiagent heuristics* h.

The MADLA Planner was designed with various heuristics both projected and multiagent.

Currently, all the heuristics in the planner are based on the relaxation principle. Relaxation is a way of simplifying a planning problem by removing some constraints. In planning, a relaxation is typically obtained by removing delete effect of actions. Solution of such relaxed planning problem is a *relaxed plan*, which can be used to estimate the cost of a plan in the original problem, *e.g.*, the *Fast-Forward (FF)* heuristic estimation is based on the length of the relaxed plan. A classical technique for finding the relaxed plan is to build a *Relaxed Planning Graph (RPG)*. RPG is a graph representing reachability of facts and applicability of actions in the relaxed problem. Such RPG can be used to find a relaxed plan by backward search through the graph by effects of actions providing support for preconditions of later ones.

Currently, these particular heuristics are used in the MADLA planner:

***Projected Fast-Forward (FF)* heuristics** h^{α}_{FF} is the approach proposed in [50] applied on the FF heuristic. In this heuristic, no communication is required.

***Multiagent Fast-Forward (MAFF)* heuristics** h_{MAFF} was designed for the MADLA Planner and recently published in [62]. It was proven to be equal to the centralized FF heuristic in terms of the resulting heuristic estimate.

***Multiagent Set-additive (MASET+)* heuristics** $h_{\mathrm{MASET+}}$**,** instead of building a relaxed plan and computing the FF heuristic, this heuristic sums the number of actions in each layer of the relaxed planning graph, therefore providing computing a variation of the Set-additive heuristic [28].

The presented heuristics both in theory and practice influence the efficiency of the planning process as a whole. The zero communication requirements are the big benefit of the projected FF heuristics h^{α}_{FF}, as the heuristic is computed strictly locally by one agent. The caveat of the heuristic is that it is uninformed w.r.t. inter-agent coordination and provides no hints for the search beyond the boundary of the state space of one agent. The MAFF heuristics h_{MAFF} is highly informed and therefore directs the search quickly towards the goals. The drawback of this heuristic is that in practice the distributed implementation can be hardly done efficiently and therefore computation of one estimation slows down the planning agents. The last presented MASET+ heuristic $h_{\mathrm{MASET+}}$ is reasonably informed and can be implemented efficiently utilizing only agents required in the build of the relaxed plan. In the next sections, the particular implementation decisions and experimental results will demonstrate these properties empirically.

7.2 Implementation of the Planner

As mentioned before, the current status of the planner can be denoted as an experimental prototype. The implementation is done in Java and although the planner is targeting a fully distributed model

running on more dedicated computers, currently the distribution is only emulated by each agent running on its thread in one Java Virtual Machine. The communication among the agents is mediated by an additional thread running over a message queue which is shared among the agents and controls delivery of sent messages. Each of the agents has an additional queue dedicated to message receiving which allows the agents to simultaneously receive messages and run their planning processes.

The prototype uses data structures based on state-of-the-art singleagent planners, however in most cases not thoroughly optimized yet. The state description is based on *Finite Domain Representation (FDR) [46]* currently only with binary domains, using the `true` and `false` values of the variables represented as a linked hash-map. An optimized form of FDR should replace the variables and their respective values with integer arrays in future.

The heuristic estimator is designed as a modular component enabling flexible experimentation with different heuristics. Since all used heuristics are relaxation-based, the structure representing Relaxed Planning Graphs (RPGs) is reused in all implemented estimators. The structure uses an extended form of description of a state using hash map with values as lists enabling description of more simultaneously holding atoms, as required by the relaxation principle. The RPGs are built iteratively from initial state to a goal-satisfying state or to a fixed-point (no additional atoms are added by further application of all applicable actions). The relaxed plan (either based on Fast-Forward principle or Sat-additive principle) is extracted by back-search through the RPG.

In the projected Fast-Forward heuristic h^{α}_{FF}, each agent prepares its own RPG using only its actions. The public ones are projections, effectively stripping prospective private atoms of other agents. This building process is asynchronous by definition, as each agent builds RPG on its own. Similarly to the construction phase, the extraction phase of the relaxed plan is implemented as an asynchronous process as well.

The Multiagent Fast-Forward heuristics h_{MAFF} is implemented as a distributed synchronous algorithm. When an agent needs an heuristic estimate, it informs all other agents about the estimated state. All agents build its parts of the global RPG called *Agent Relaxed Planning Graphs (ARPGs)*. The ARPGs use actions of their respective building agents (similarly to the projection heuristics) additionally with projections of other agents' public actions which they used in their ARPGs. Note that this parallel building of a global RPG in form of more complement ARPGs requires an synchronized termination detection process. The resulting relaxed plan is then extracted in a parallel way with a final merging process started by the agent initiating the state heuristic evaluation.

Since such synchronous process congests the complete multiagent system, as all agents has to participate on heuristic estimation of all other agents, recently a lazy asynchronous approach was designed and implemented. The lazy asynchronous implementation is based on the good parts of the previous ones. Firstly, the algorithm starts only with the estimation requesting agent which builds

		orig.	BFS + h^{α}_{FF}			BFS + h_{MAFF}					BFS + lazy async. $h_{\mathrm{MASET+}}$				
	$\lvert\mathcal{A}\rvert$		$t[s]$	v	c_{s}	$t[s]$	v	c_{s}	c_{r}	c_{h}	$t[s]$	v	c_{s}	c_{r}	c_{h}
CP	3	–	0.4	99	97	6.3	168	166	6.6k	39	0.5	61	59	72	30
	5	–	88	25k	25k	–	–	–	–	–	10	1.1k	1.1k	120	736
	7	–	–	–	–	–	–	–	–	–	38	6.2k	6.2k	169	248
LOG	4	0.6	0.6	1.5k	847	2.5	505	272	10k	50	0.3	276	159	32	51
	6	38.5	6.2	17k	7.4k	–	–	–	–	–	1.1	461	217	52	100
ROV	2	1.4	–	–	–	88	378	4	25k	4	18	461	3	72	3
	3	7.9	–	–	–	–	–	–	–	–	182	1.4k	9	108	10
SAT	4	1.2	32	32k	369	16	941	27	9.8k	22	0.8	441	24	3	21
	6	4.4	–	–	–	–	–	–	–	–	5.4	1.4k	57	6	55
	8	–	–	–	–	–	–	–	–	–	31	3.4k	115	8	109

Table 7.1: Results for the originally used DCSP-based planner (column orig.) used in the previous chapters and three relaxation heuristics in the MADLA Planner. $|\mathcal{A}|$ is a number of agents in the problem, t is duration of the search in seconds, v is a number of visited states, c_{s} is a number of search messages (each of size of a state), c_{r} is a number of messages building (A)RPGs (each of size of a projected public action) and c_{h} is a number of messages for the heuristic estimate (each of size of a partial relaxed plan for h_{MAFF} or partial cost estimation for $h_{\mathrm{SET+}}$). The domains are COOPERATIVE PATHFINDING (CP), LOGISTICS (LOG), ROVERS (ROV) and SATELLITES (SAT). Runs denoted as – did not finish in the 10 minutes limit.

its projected ARPG and incorporates the external public actions based on a cached ARPG from before of the planning process (such ARPG can be build by the synchronized building algorithm). With such ARPG the initiating agent starts the relaxed plan extraction routine. The routine can, however, require a heuristic estimation for one of the supplied external public actions. Such estimation is out of reach of the current agent, therefore the supplying agent is requested for its estimate by a message. Such process can recursively pass to other agents. Technically, the algorithm is a distributed recursion with branching as described in [20].

The Multiagent Set-additive heuristics $h_{\mathrm{MASET+}}$ uses this asynchronous and lazy approach and as Table 7.1 shows, it provides most efficient multiagent planning in MADLA planner in comparison to the two previous heuristics and also the originally used DCSP-based planner (the better results are only in ROVERS, where the DCSP-based planner gains because of highly-optimized Fast-Forward planner used internally for the private plans). Furthermore, the results support the informal hypotheses on the heuristics properties summarized in the previous section. The projected h^{α}_{FF} is less informed as the numbers of visited states v are mostly the highest. The h_{MAFF} heuristic is more informed (lower v), but the computation is rather computationally heavy (worse t and $c_s + c_r + c_h$). The lazy asynchronous $h_{\mathrm{MASET+}}$ heuristic's results are the best in general.

The complete implementation of the MADLA Planner can be wrapped up in a sequence of steps beginning with the input of the planner and ending with a resulting plan as (the description is only

schematic and does not contain particular loops):

1. Reading of the input PDDL file and an additional file describing which objects in planning the problem definition are represented by agents.
2. Basic grounding and filtering process of the facts and actions.
3. Initialization of the agents with factorized planning problem.
4. In the case of $h_{\mathrm{MASET}+}$, preparation of the initial ARPGs used later in the lazy heuristic estimates.
5. Synchronized start of the search.
   - Running processes of the Best-First Search by the agents.
   - Possibly parallel computation of the heuristic estimates directing the distributed search.
6. After detection of a found goal, state distributed termination of the search and writing the resulting plan to the output.

The planner was supplemented by a couple of helper routines and classes to provide flexible and easy to use experimental suite, which in turn showed its strengths also in using the planner for the final experimental validation of the plan repair algorithms.

7.3 Plan Repair with Forward-search Multiagent Planner

The MADLA planner was used as the core planning component in the plan repair techniques presented in Chapter 4 to provide validation of the plan repair principles with different planning approach than the originally used DCSP-based planner from [51].

Since the plan repair algorithms were from the first steps designed with a modularity of the planner in mind, replacing the planner did not require to change the design of the repair approaches and no big changes in the implementation. Since both implementations of the planner and the repair algorithms use data structures for describing states, actions, facts and other planning concepts, the operations on these structures used by the repair algorithms had to be reimplemented around the data structures used by the MADLA planner. The biggest part was a de-linerization algorithm used to parallelize the sequential outputs from the MADLA planner. The actions are during the process partitioned into queues for each agent and a graph of partial orderings among dependent actions of the plan is generated. Afterward, the actions are iteratively taken from the queues only if there are no other dependent actions. If an agent cannot add its action because of some dependencies, it uses the empty action ϵ.

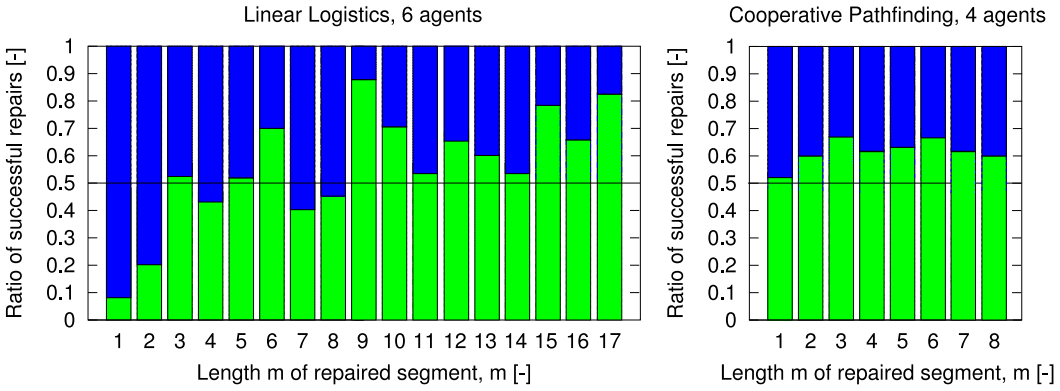


Figure 7.4.1: Comparison of success ratio between prefix-preserving (blue, Lazy) and suffix-preserving (green, Back-on-Track) plan repair algorithms against replanning. The results extend the comparison in Figure 5.5.8.

All four designed plan repair algorithms were used on the same domains and in a subset of experiments presented in Chapter 5 with the same experimental setting. The experiments were selected with two respects: (a) to provide enough evidence that different planner does not disapprove the results w.r.t. the presented hypotheses and (b) if possible, to extend the results because of the better efficiency of the planner. Next section reports on the results and discusses the differences in the conclusions of the plan repair techniques with a distributed forward-search planner.

7.4 Results and Discussion

The first batch of experiments targeted extended validation of Hypothesis 1:

> *Multiagent plan repair is expected to generate lower communication overhead in tightly coordinated domains.*

The results with the originally used DisCSP-based planner from Section 5.5.1 strongly supported the hypothesis. The results with the MADLA planner are presented in Table 7.2. The form of the results copies Table 5.1. The results show both the same planning problems which were used with the original planner and additionally several new results which were impossible to measure with the DisCSP-based planner MA-Plan.

In all presented results, the plan repair techniques using the MADLA planner outperform replanning from scratch in the communication metrics. Additionally, less planning time was always required for the plan repair than for the replanning. Similarly to the results with the DisCSP-based planner, more efficient plan repair in environments with failing actions and $P = 0.3$ is the

Domain	$\|\mathcal{A}\|$	Repair time [ms]			No. of messages [-]			Communication [kB]			Exec. length [-]		
		BoT	RLazy	Replan	BoT	RLazy	Replan	BoT	RLazy	Replan	BoT	RLazy	Replan
LOGISTICS	2	38.7	34.7	56.8	26.9	18.4	28.9	1.9	1.7	2.5	7.5	10.2	9.3
	3	93.3	149.6	128.2	108.5	149.6	113.5	11.7	20.1	15.7	11.7	30.3	13.8
	4	194.9	150.7	184.3	194.9	150.7	184.3	35.2	50.1	63.7	11	15.5	11.8
LOG. (PAR)	5	2295.0	114.8	247.5	6423.2	179.3	338.8	2.8MB	51.7	125.1	11.7	12.8	10.8
	6	3570.0	667.5	1109.4	5134.4	805.5	1134.5	3.3MB	439.5	761.3	11.4	29.8	15.9
LOG. (LIN)	**5**	**434.9**	**351.2**	**586.3**	**1041.2**	**354.0**	**611.1**	**401.4**	**160.9**	**271.1**	**21.3**	**49.2**	**28.0**
	7	**6191.8**	**1799.8**	**2705.6**	**13158**	**988.8**	**1517.9**	**3.6MB**	**423.3**	**668.8**	**29.7**	**99.5**	**37.2**
	9	**94418**	**9025.0**	**9680.8**	**109k**	**2740.1**	**2862.7**	**11MB**	**893.3**	**1.4MB**	**37.1**	**206.5**	**48.7**
COOP. PF.	2	60.9	127.6	65.0	46.0	110.3	51.1	8.5	33.4	16.5	4.8	6.7	4.7
	3	61.0	78.9	120.0	36.5	43.3	82.2	8.2	20.6	58.4	2.6	2.8	3.3
	4	559.9	14133	1651.7	277.0	4831.3	826.1	111.4	892.9	1.3MB	9.4	9.4	7.7
	5	**4167.6**	**22316**	**>300k**	**1316.0**	**7427.6**	–	**2MB**	**3.5MB**	–	**18.0**	**29.8**	–
ROVERS	2	5001.0	2518.1	5044.9	680.9	247.5	219.5	378.9	90.3	140.5	16.8	24.1	15.8
SATELLITES	2	53.6	22.6	24.3	48.5	4.9	5.8	3.0	0.7	1.0	4.8	5.5	4.5
	4	2524.1	111.0	100.3	2213.0	20.2	23.6	1.3MB	8.9	14.1	5.2	7.0	5.3
	6	29082	493.2	836.3	5467.3	24.1	88.1	7.1MB	20.1	122.0	4.7	6.1	4.7
	8	**>300k**	**2057.0**	**8142.8**	–	**27.8**	**272.6**	–	**45.5**	**661.0**	–	**5.9**	**5.0**

Table 7.2: Results of experiments for all domains with probability $P = 0.3$ and action failures using the MADLA planner. The highlighted cells are the best results for a particular domain and a particular metrics. The bolded results are extended experiments possible to measure because of the MADLA planner. The LOG. (LIN) problems are based on a linear chain of handovers by the logistics fleet (in contrast to the parallel logistics LOG. (PAR), where the handovers are parallel). The problems with the dash were were unsolvabe in a limit of 5 minutes.

Repeated-Lazy approach. It is outperformed by the Back-on-Track only in the fully coordinated COOPERATIVE PATHFINDING. In a consequence, the experiments validate that the proposed plan repair algorithms are able to work with another multiagent planning approach and reaffirmed validity of the first hypothesis.

Validity of Hypotheses 2 and 4 was successfully confirmed in the core experiments of the work, however because of scalability problems with the DisCSP-based planner, Hypothesis 3 could not be conclusively verified. The idea of the third hypothesis is:

> *If an action fails and it has potentially a lot of future dependencies, trying to fix it as soon as possible is rather better idea, than ignore it and try to repair it later.*

To conclude on the last not yet fully supported Hypothesis 3, the MADLA planner was used in an extended version of the experiment presented in Section 5.5.3. Recall that the pure prefix-preserving and pure suffix-preserving repair algorithms were run in the testing domains. Ratio of successful repairs of these two repair algorithms against replanning was measured by means of computation time.

The results are depicted in Figure 7.4.1. The situation in the LOGISTICS domain copies the previous results in Figure 5.5.8, the first two peaks related to the handover points in the plan are for repair lengths 3 and 6 equal to the results using the MA-Plan planner. The rising trend and the peaks continue even for lengths 9 to 17. With the increasing length of the repaired part of the plan, the peaks are slightly decreasing in contrast to the step 9, still in the longer repaired parts the suffix-preserving repair (Back-on-Track) outperforms the prefix-preserving one (Lazy).

The experiment in the COOPERATIVE PATHFINDING resemble the previous results in the first three lengths of the repaired segments. The same trend is apparent, but the absolute values differ. Since the MADLA planner is not so much sensitive to the number of goal facts, the positive effect of the Back-on-Track repair exhibits in the domains with long action dependencies event for small m. For the longer repaired segments the suffix-preserving approach keeps its lead.

The results update the previous conclusions of the third hypothesis such that it is *substantially supported* by the experimental results.

Chapter 8

Conclusion

The problem of multiagent plan repair brought a variety of research challenges comprising appropriate formalism design, algorithm design, theoretical analysis, implementation work, experimental evaluations, verifications and validations.

The first part of the thesis proposes a formalization for multiagent plan repair problems based on state-of-the-art multiagent planning formalization. An important newly defined property of a multiagent plan is its coordination frequency which determines the amount of coordination required for an optimal solution of a multiagent planning problem. Leveraging this concept, planning problems can be categorized as uncoordinated, loosely coordinated, tightly coordinated or fully coordinated, or the coordination can be precisely expressed as a numeric value.

The core of the thesis proposes four plan repair algorithms transforming a multiagent repair problem to a problem of multiagent planning. Such usage of a multiagent planner in the algorithms has both advantages and disadvantages (similarly to compilations of special planning problems into classical planning). The key advantage is that there is no need for repetition of work on the combinatorial and search algorithms which can be reused from other solutions of planning problems. Another advantage is the planner can be replaced by a better one to improve efficiency of the outer algorithm, *e.g.*, a plan repair algorithm. The disadvantages are a possible bias of the implementation details of the used planner inappropriate for the particular plan repair algorithms or other outer algorithms and an impossibility to tailor the planner for such algorithms as plan repair.

The last proposed plan repair algorithm was designed as a generalization of previous two and revealed relations between them w.r.t. the set of proposed hypotheses. The soundness and completeness of the algorithms were proven. Additionally, time and communication complexity of the algorithms were analyzed. The results comply with conclusions from the literature and the stated hypotheses of the thesis. Large portion of work was dedicated to implementation both of the used

multiagent planner and the plan repair process with the proposed algorithms. Overview of the implementation details reports especially on successes w.r.t. to the computational efficiency.

The algorithms were evaluated in various types of experiments targeting the stated hypotheses. In all cases, the hypotheses were supported, in several cases with additional limitations and specifications. Based on the experimental results, a summary of heuristic approaches can be sated in a form of simply usable advices decreasing computation or communication overheads during repair of multiagent plans by plan reuse. These advices can be used for various plan repair approaches targeting systems with planning agents. The advices are:

- *Prefer preserving plan repair over replanning from scratch if the domains are tightly coordinated and the divergence of the failed states stays close to the ideal execution.*
- *Prefer smaller numbers of involved agents in the plan repair process.*
- *Prefer suffix-preserving repair techniques (Back-on-Track) when repairing failures with long dependencies especially among different agents.*
- *Prefer m-normal plan repair algorithms, that means try to reuse each of the actions from the original plan precisely once.*

To validate the fundamental principle of the proposed plan repair algorithms, one of them was after an adaptation deployed into a high-fidelity simulated environment of a support tactical mission. The results showed that this can be done if the targeted planning problem can be appropriately formalized. The deployment experiment showed that plan repair based on preservation of parts of the original plan is thriving in a short-repair loop and acts similarly as reactive planning, but with permanently sound plan to the goal of the planning problem or particularly to the objective of the tactical mission. The proposed software engineering methodology aiming at deployment of various complex algorithms into a high-fidelity environments was verified on an adapted Back-on-Track algorithm and showed its usability to transpose initially theoretical algorithms to environments close to the real world.

Finally, description of a newly designed multiagent planner based on a distributed forward-search principle was presented. The planner was used to validate the plan repair algorithms with another planning technique and verified the proposed hypotheses with a different planning approach than the used in the core experiments of the thesis.

Majority of the thesis was covered by five publications. Formalization, plan repair algorithms and first experiments were initially published in [31, 33]. The theoretical extension with deeper experimental study was presented in [34]. Generalization of the suffix-based and prefix-based plan repair approaches with experimental evaluation was published in [32]. The deployment scheme was proposed in [39].

8.1 Directions for Future Research

During the work on the thesis several problems appeared as interesting directions for future research. The key directions are summarized in this section.

Firm multiagent planner The second validation chapter described a new prototypical multiagent planner. The results of the planner especially w.r.t. its time efficiency are promising, but a lot of work is still required to be done. The most notable extensions are to distribute heuristics as landmarks, helpful actions, and similar. Probably more challenging, but also more interesting in the sense of future research would be to design special multiagent heuristics.

Robust multiagent planning As this thesis was focused on an unknown failure model, the state-of-the-art approaches to planning with uncertainty was not comparable with the approaches proposed here. An interesting question is, however, how to modify planning based on *Markov Decision Processes* (MDPs) to be able to describe probabilities parametrized by previous plans. Such approach could bring the area of multiagent plan repair presented in this dissertation closer to research in the field of (distributed) MDPs.

Extended application of plan repair In Chapter 6, one of the proposed plan repair techniques was deployed and tested in a simulated environment of a multi-robotic team. The last and probably most interesting future direction from perspective of practical applications is to experiment with plan repair on real robots and search for efficient multiagent plan repair techniques for real-world robotic deployment.

8.2 Thesis Achievements

This section summarizes the contribution of the thesis to the state-of-the-art. Achieved improvements are the following:

1. *Novel formalization of problems of multiagent domain-independent plan repair.*
   The dissertation proposes a novel formalization based on a state-of-the-art MA-STRIPS framework for formal description of multiagent problems. Besides the core formalism, measures for plan difference and coordination tightness of multiagent planning problems were defined.

2. *Formal and algorithmic definition of four novel multiagent plan repair algorithms.*
   Four novel plan repair algorithms for domain-independent multiagent planning were proposed both formally and algorithmically. The algorithms are based on reuse of parts of the original plan in form of a prefix or suffix or both. The last algorithm is generalization of the previous two and provides insights into possible interconnection of those principles.

3. *Proofs of soundness and completeness of the proposed algorithms.*
   The proposed plan repair algorithms were theoretically studied and the results of the study were presented as proofs of their completeness and soundness. The proofs used the proposed formalization and algorithm description and were provided with auxiliary conditions under which they are valid.

4. *Time and communication complexity analysis of the proposed algorithms.*
   Besides the soundness and completeness analysis, the algorithms were theoretically analyzed from the perspective of computational and communication complexity. The analysis used the description of the algorithms and resulted in a complexity formulas in the Big-O notation for all four proposed algorithms both for computational and communication complexity.

5. *Experimental evaluation of the proposed plan repair algorithms.*
   The plan repair algorithms were thoroughly verified in an exhaustive set of experiments targeting various aspects of multiagent plan repair. The key ones compared plan repair with replanning from scratch in a wide spectrum of planning domains bounded by a domain requiring complete coordination of the agents and on the other hand by a domain requiring no coordination at all.

6. *Novel methodology for deployment of algorithms to high-fidelity simulations.*
   To allow deployment of the plan repair approaches to environments close to real-world, a software engineering development process was designed and proposed in the thesis. This process was tailored to tactical missions which were also used for the first validation of one of the proposed multiagent plan repair approaches.

7. *Validation of plan repair in high-fidelity simulation of a tactical mission.*
   One of the proposed plan repair techniques was tested with the proposed deployment process into a high-fidelity simulation to show the core principle of plan repair reusing parts of the original plan is vital for real-world usage and even for reactive planning.

8. *Design and implementation of a novel distributed forward-search multiagent planner.*
   Since the thorough experimental results were done with one specific state-of-the-art planning technique based on Distributed Constraint Satisfaction Problem solving the other validation of the repair approach was done with a novel design and implementation of a distributed multiagent forward-search planner. The newly designed planner was more efficient and therefore was used to extend the results of the previous experiments.

9. *Additional validation of plan repair with the proposed multiagent forward search planner.*
   The new planner was used to validate the results from the thorough experiments and extended the conclusions with additional insights, which were not possible to find out with the original planner.

8.3 Selected Related Publications

This section summarizes author's selected publications related to the content of the thesis.

Articles in journals and book chapters (5):

A. Komenda, P. Novák, and M. Pěchouček. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications.* 2013. ISSN 1084-8045 (to appear).

A. Komenda, J. Vokřínek, M. Čáp, M. Pěchouček. Developing Multiagent Algorithms for Tactical Missions Using Simulation. *IEEE Intelligent Systems.* Volume 28(1), pages 42–49. 2013. ISSN 1541-1672.

P. Novák, A. Komenda, M. Čáp, J. Vokřínek, M. Pěchouček. Simulated Multi-robot Tactical Missions in Urban Warfare. In *Multiagent Systems and Applications Volume 1: Practice and Experience,* pages 147–183, Berlin: Springer, 2013. ISBN 978-3-642-33322-4.

A. Komenda, J. Vokřínek, and M. Pěchouček. Plan representation and execution in multi actor scenarios by means of social commitments. *Web Intelligence and Agent Systems.* Volume 9(2), pages 123–133, 2011. ISSN 1570-1263.

J. Vokřínek, A. Komenda, and M. Pěchouček. Abstract architecture for task-oriented multi-agent problem solving. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Volume 41(1), pages 31–40, 2011. ISSN 1094-6977.

In proceedings (15):

A. Komenda, P. Novák, and M. Pěchouček. How to Repair Multi-agent Plans: Experimental Approach. In *Distributed and Multi-agent Planning (DMAP) Workshop of 23rd International Conference on Automated Planning and Scheduling (ICAPS'13).* 2013.

M. Štolba, and A. Komenda. Fast-Forward Heuristic for Multiagent Planning. In *Distributed and Multi-agent Planning (DMAP) Workshop of 23rd International Conference on Automated Planning and Scheduling (ICAPS'13).* 2013.

K. Durkota, and A. Komenda. Deterministic Multiagent Planning Techniques: Experimental Comparison (Short paper). In *Distributed and Multi-agent Planning (DMAP) Workshop of 23rd International Conference on Automated Planning and Scheduling (ICAPS'13).* 2013.

A. Komenda, R. N. Lass, P. Novák, W. C. Regli, and M. Pěchouček: Scalable and robust multi-agent planning with approximated DCOP. In *Proceedints of 6th International Workshop on*

Optimisation in Multi-Agent Systems, OPTMAS 2013, workshop affiliated with AAMAS 2013. May 6-7 2013.

A. Komenda, P. Novák, and M. Pěchouček. Decentralized Multi-agent Plan Repair in Dynamic Environments (Extended Abstract). In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems.* County of Richland: IFAAMAS, pages 1239–1240. 2012.

P. Novák, A. Komenda, V. Lisý, B. Bošanský, M. Čáp, M. Pěchouček. Tactical Operations of Multi-Robot Teams in Urban Warfare (Demonstration). In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems.* County of Richland: IFAAMAS, Volume 3, pages 1473–1474. 2012.

A. Komenda, and P. Novák. Multi-agent Plan Repair. In *Proceedings Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities, Proceedings of IJCAI 2011 Workshop.,* pages 1-6., Menlo Park, California: AAAI Press, 2011.

J. Vokřínek, A. Komenda, and M. Pěchouček. Agents towards vehicle routing problems. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 773–780, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

J. Vokřínek, A. Komenda, and M. Pěchouček. Cooperative agent navigation in partially unknown urban environments. In *PCAR '10: The Third International Symposium on Practical Cognitive Agents and Robots. Proceedings of the AAMAS-10 Workshops.*, 2010.

J. Vokřínek, A. Komenda, and M. Pěchouček. Decommitting in multi-agent execution in non-deterministic environment: Experimental approach. In C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, editors, *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15, 2009, Volume 2*, pages 977–984. IFAAMAS, 2009.

J. Vokřínek, A. Komenda, and M. Pěchouček. Relaxation of social commitments in multi-agent dynamic environment. In *ICAART 2009 - Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 520–525. INSTICC Press, 19-21 January 2009.

A. Komenda, J. Vokřínek, M. Pěchouček, G. Wickler, J. Dalton, and A. Tate. I-globe: Distributed planning and coordination of mixed-initiative activities. In *KSCO '09: Knowledge Systems for Coalition Operations 2009*, Chilworth Manor, Southampton, UK, Mar-Apr 2009.

G. Wickler, A. Komenda, M. Pěchouček, A. Tate, and J. Vokřínek. Multi-agent planning

with decommitment. In *KSCO '09: Knowledge Systems for Coalition Operations 2009*, Chilworth Manor, Southampton, UK, Mar-Apr 2009.

A. Komenda, J. Vokřínek, M. Pechoucek, G. Wickler, J. Dalton, and A. Tate. Distributed planning and coordination in non-deterministic environments (demo). In *Proceedings of 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15, 2009, Volume 2*, pages 1401–1402. IFAAMAS, 2009.

A. Komenda, M. Pěchouček, J. Bíba, and J. Vokřínek. Planning and re-planning in multi-actors scenarios by means of social commitments. In *Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT/ABC 2008), Volume 3*, pages 39–45. IEEE, October 2008.

Bibliography

[1] Ambros-Ingerson, J., and Steel, S. Integrating planning, execution and monitoring. *AAAI* (1988), 83–88.

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

[3] Barták, R., Salido, M. A., and Rossi, F. Constraint satisfaction techniques in planning and scheduling. *Journal of Intelligent Manufacturing 21*, 1 (2010), 5–15.

[4] Barták, R., and Toropila, D. Solving sequential planning problems via constraint satisfaction. *Fundam. Inf. 99*, 2 (Apr. 2010), 125–145.

[5] Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. The complexity of decentralized control of markov decision processes. *Math. Oper. Res. 27*, 4 (Nov. 2002), 819–840.

[6] Brafman, R. I., and Domshlak, C. Factored planning: How, when, and when not. In *AAAI* (2006), AAAI Press, pp. 809–814.

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

[8] Bresina, J. L., and Morris, P. H. Mixed-initiative planning in space mission operations. *AI Magazine 28*, 2 (2007), 75–88.

[9] Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T. andSmith, B., Fisher, F., Barrett, T., Steb-

BINS, G., AND D., T. ASPEN - automating space mission operations using automated planning and scheduling. In *Proceedings of International Conference on Space Operations (SpaceOps 2000)*.

[10] DECHTER, R. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

[11] DECKER, K., AND LESSER, V. Generalizing the Partial Global Planning Algorithm. *International Journal on Intelligent Cooperative Information Systems 1*, 2 (June 1992), 319–346.

[12] DES JARDINS, M., DURFEE, E. H., ORTIZ, C. L. J., AND WOLVERTON, M. A survey of research in distributed, continual planning. *AI Magazine 20*, 4 (1999), 13–22.

[13] DES JARDINS, M., AND WOLVERTON, M. Coordinating a distributed planning system. *AI Magazine 20*, 4 (1999), 45–53.

[14] DO, M. B., AND KAMBHAMPATI, S. Planning as constraint satisfaction: solving the planning graph by compiling it into csp. *Artif. Intell. 132*, 2 (Nov. 2001), 151–182.

[15] DOHERTY, P., AND KVARNSTRÖM, J. Talplanner: A temporal logic-based planner. *AI Magazine 22*, 3 (2001), 95–102.

[16] DURFEE, E. H. Distributed problem solving and planning. In *A Modern Approach to Distributed Artificial Intelligence*, G. Weiß, Ed. The MIT Press, San Francisco, CA, 1999, ch. 3.

[17] EROL, K., HENDLER, J., AND NAU, D. S. Htn planning: complexity and expressivity. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 2)* (Menlo Park, CA, USA, 1994), AAAI'94, American Association for Artificial Intelligence, pp. 1123–1128.

[18] ESTLIN, T., CASTANO, R., ANDERSON, R., GAINES, D., FISHER, F., AND JUDD, M. Learning and planning for mars rover science. In *In Proc. of IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modeling, Planning, Learning, and Communicating* (2003), Morgan Kaufmann Publishers.

[19] Fikes, R., and Nilsson, N. 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.

[20] Gafni, E., and Rajsbaum, S. Recursion in distributed computing. In *Proceedings of the 12th international conference on Stabilization, safety, and security of distributed systems* (Berlin, Heidelberg, 2010), SSS'10, Springer-Verlag, pp. 362–376.

[21] Gerevini, A., and Serina, I. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *AIPS* (2000), S. Chien, S. Kambhampati, and C. A. Knoblock, Eds., AAAI, pp. 112–121.

[22] Gupta, S., Bourne, D., Kim, K., and Krishnan, S. Automated process planning for sheet metal bending operations. *Journal of Manufacturing Systems 17(5)* (1998), 338–360.

[23] Helmert, M., and Domshlak, C. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of ICAPS'09* (2009), A. Gerevini, A. E. Howe, A. Cesta, and I. Refanidis, Eds., AAAI.

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

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

[26] Jakob, M., Pěchouček, M., Novák, P., Čáp, M., and Vaněk, O. Towards incremental development of human-agent-robot applications using mixed-reality testbeds. *IEEE Intelligent Systems, Special Issue on HART: Human-Agent-Robot Teamwork* (2011).

[27] Kambhampati, S., and Srivastava, B. Universal classical planner: An algorithm for unifying state-space and plan-space planning. *New Directions in AI Planning* (1995), 261–271.

[28] Keyder, E., and Geffner, H. Set-Additive and TSP heuristics for planning with action costs and soft goals. In *Proceedings of ICAPS Workshop on Heuristics for Domain-Independent Planning* (2007).

[29] Komenda, A., Lass, R. N., Novák, P., Regli, W. C., and Pěchouček, M. Scalable and robust multi-agent planning with approximated DCOP. In *Proceedints of 6th International Workshop on Optimisation in Multi-Agent Systems, OPTMAS 2013, workshop affiliated with AAMAS 2013* (May 6-7 2013).

[30] Komenda, A., and Novák, P. Multi-agent plan repairing. In *Proceedings of Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities IJCAI-DMPOUW Workshop* (2011), pp. 1–6.

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

[32] Komenda, A., Novák, P., and Pěchouček, M. How to repair multi-agent plans: Experimental approach. In *Proceedings of Distributed and Multi-agent Planning (DMAP) Workshop of 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)* (2013).

[33] Komenda, A., Novák, P., and Pěchouček, M. Decentralized multi-agent plan repair in dynamic environments. *CoRR abs/1202.2773* (2012).

[34] Komenda, A., Novák, P., and Pěchouček, M. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications* (2013).

[35] Komenda, A., Pěchouček, M., Bíba, J., and Vokřínek, J. Planning and re-planning in multi-actors scenarios by means of social commitments. In *Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT/ABC 2008)* (october 2008), vol. 3, IEEE, pp. 39–45.

[36] Komenda, A., Vokřínek, J., and Pěchouček, M. Plan representation and execution in multi actor scenarios by means of social commitments. *Web Intelligence and Agent Systems 9*, 2 (2011), 123–133.

[37] Komenda, A., Vokřínek, J., Pěchouček, M., Wickler, G., Dalton, J., and Tate, A. Distributed planning and coordination in non-deterministic environments. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2* (Richland, SC, 2009), AAMAS '09, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1401–1402.

[38] Komenda, A., Vokřínek, J., Pěchouček, M., Wickler, G., Dalton, J., and Tate, A. I-Globe: Distributed planning and coordination of mixed-initiative activities. In *Proceedings of Knowledge Systems for Coalition Operations (KSCO 2009)* (2009).

[39] Komenda, A., Vokřínek, J., Čáp, M., and Pěchouček, M. Developing multiagent algorithms for tactical missions using simulation. *Intelligent Systems, IEEE 28*, 1 (2013), 42–49.

[40] Krogt, R. V. D., and Weerdt, M. D. Plan repair as an extension of planning. *. of the Int. Conf. on Automated Planning* (2005).

[41] Krogt, R. V. D., and Weerdt, M. D. Plan repair using a plan library. *Proceedings of the Belgium-Dutch Conference on* (2005).

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

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

[44] Lopez, A., and Bacchus, F. Generalizing graphplan by formulating planning as a csp. In *Proceedings of the 18th international joint conference on Artificial intelligence* (San Francisco, CA, USA, 2003), IJCAI'03, Morgan Kaufmann Publishers Inc., pp. 954–960.

[45] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. PDDL – the planning domain def-

inition language – Version 1.2. *Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control* (1998).

[46] Nau, D., Ghallab, M., and Traverso, P. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[47] Nau, D. S. Current trends in automated planning. *AI Magazine 28*, 4 (2007), 43–58.

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

[49] Nissim, R., Apsel, U., and Brafman, R. I. Tunneling and decomposition-based state reduction for optimal planning. In *ECAI* (2012), L. D. Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. J. F. Lucas, Eds., vol. 242 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 624–629.

[50] Nissim, R., and Brafman, R. I. Multi-agent a* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3* (Richland, SC, 2012), AAMAS '12, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1265–1266.

[51] Nissim, R., Brafman, R. I., and Domshlak, C. A general, fully distributed multi-agent planning algorithm. In *Proceedings of AAMAS* (2010), pp. 1323–1330.

[52] Novák, P. *Jazzyk: A Programming Language for Hybrid Agents with Heterogeneous Knowledge Representations*. Springer-Verlag, Berlin, Heidelberg, 2009, pp. 72–87.

[53] Novák, P., and Jamroga, W. Agents, Actions and Goals in Dynamic Environments. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011* (2011), T. Walsh, Ed., IJCAI/AAAI, pp. 313–318.

[54] Novák, P., Komenda, A., Čáp, M., Vokřínek, J., and Pěchouček, M. Simulated multi-robot tactical missions in urban warfare. In *Multiagent Systems and Applications*. Springer, 2012, pp. 147–183.

[55] Oliehoek, F., and Vlassis, N. Dec-POMDPs and extensive form games: equivalence of models and algorithms. IAS technical report IAS-UVA-06-02, Intelligent Systems Lab, University of Amsterdam, Amsterdam, The Netherlands, Apr. 2006.

[56] pa So, Y., and Durfee, E. H. Designing tree-structured organizations for computational agents. *Computational and Mathematical Organization Theory 2* (1996), 219–246.

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

[58] Pěchouček, M., Jakob, M., and Novák, P. Towards simulation-aided design of multi-agent systems. In *Post-proceedings of the eighth international workshop on programming multi-agent systems, ProMAS 2010, LNAI, Vol. 6599* (2010), Springer-Verlag.

[59] Richter, S., Helmert, M., and Westphal, M. Landmarks revisited. In *AAAI* (2008), D. Fox and C. P. Gomes, Eds., AAAI Press, pp. 975–982.

[60] Siebra, C., and Tate, A. I-Rescue: A Coalition Based System to Support Disaster Relief Operations. In *Proceedings of The Third International Association of Science and Technology for Development (IASTED) International Conference on Artificial Intelligence and Applications (AIA-2003)* (September 2003).

[61] Spalzzi, L. A survey on case-based planning. *Artif. Intell. Rev. 16*, 1 (Sept. 2001), 3–36.

[62] Štolba, M., and Komenda, A. Fast-forward heuristic for multiagent planning. In *Proceedings of Distributed and Multi-agent Planning (DMAP) Workshop of 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)* (2013).

[63] Vokřínek, J., Komenda, A., and Pěchouček, M. Decommitting in multi-agent execution in non-deterministic environment: Experimental approach. In *AAMAS '09: Proceedings of the eight international joint conference on Autonomous agents and multiagent systems* (2009).

[64] Vokřínek, J., Komenda, A., and Pěchouček, M. Relaxation of social commitments in multi-agent dynamic environment. In *Proceedings of International Conference on Agents and Artificial Intelligence (ICAART09)* (19-21 January 2009), Springer.

[65] Vokřínek, J., Komenda, A., and Pěchouček, M. Agents towards vehicle routing problems. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1* (Richland, SC, 2010), AAMAS '10, International Foundation for Autonomous Agents and Multiagent Systems, pp. 773–780.

[66] Vokřínek, J., Komenda, A., and Pěchouček, M. Cooperative agent navigation in partially unknown urban environments. In *PCAR '10. Proceedings of the AAMAS-10 Workshops.* (May 2010), pp. 46–53.

[67] Vokřínek, J., Komenda, A., and Pěchouček, M. Abstract architecture for task-oriented multi-agent problem solving. *IEEE Transactions on Systems, Man, and Cybernetics, Part C 41*, 1 (2011), 31–40.

[68] Wickler, G., Potter, S., and Tate, A. Using I-X process panels as intelligent To-Do lists for agent coordination in emergency response. *International Journal of Intelligent Control and Systems (IJICS), Special Issue on Emergency Management Systems* (2006).

[69] Wickler, G., Pěchouček, M., Komenda, A., Vokřínek, J., and Tate, A. Multi-agent planning with decommitment. In *Proceesings of Knowledge Systems for Coalition Operations (KSCO 2009)* (2009).

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