CZECH TECHNICAL UNIVERSITY IN PRAGUE

DOCTORAL THESIS

Multi-Agent Planning by Plan Set Intersection

Author:
RNDr. Jan TOŽIČKA

Supervisor:
prof. Michal PĚCHOUČEK

Supervisor Specialist:
Dr. Antonín KOMENDA

AI Center
Department or Computer Science
Faculty of Electrical Engineering

Ph.D. Programme: *Electrical Engineering and Information Technology*
Branch of Study: *Information Science and Computer Engineering*

Prague, January 2017

Declaration of Authorship

I declare that this thesis was composed by me, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

RNDr. Jan TOŽIČKA

"We are the recorders and reporters of facts – not the judges of the behaviors we describe."

Alfred Charles Kinsey

Czech Technical University in Prague

Abstract

Faculty of Electrical Engineering
Department or Computer Science

Doctor of Philosophy

Multi-Agent Planning
by Plan Set Intersection

by RNDr. Jan TOŽIČKA

Coordination of a team of cooperative agents and their activities towards fulfillment of goals is described by multi-agent planning. For deterministic environments, where agents are not willing to share all their knowledge, the MA-STRIPS model provides minimal extension from classical planning. MA-STRIPS exactly prescribes what information can be freely communicated between the agents and what information has to be kept private such that the shared or individual goals can be still achieved.

This thesis proposes a novel multi-agent planning approach which distributively intersects local plans of the agents towards a global solution of the multi-agent planning problem. This core principle builds on local compilation to a classical planning problem and compact representation of the local plans in the form of Finite State Machines. The efficiency of the resulting planner is further boosted up by distributed delete-relaxation heuristic, an approximative local plan analysis, and reduction of agents' internal problems.

The planning approach is analysed theoretically, in particular we prove its completeness and soundness. Experimental evaluation shows its applicability in a full privacy setting where only public information can be communicated and in less restricted privacy settings. At a recent international competition of distributed multi-agent planners, the proposed planner showed top performance when compared with other existing state-of-the-art multi-agent planners.

Acknowledgements

This work would not be possible without the support of my supervisor prof. Michal Pěchouček and my supervisor specialist Dr. Antonín Komenda. 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. Jan Jakubův who was my closest coworker without whom the latest research results could not be of the quality they are now. This thesis is written in first-person plural form to pay tribute to the teamwork and to this fruitful collaboration.

Contents

Declaration of Authorship iii

Abstract vii

Acknowledgements ix

1 Introduction 1
- 1.1 Problem Statement . . . 3
  - 1.1.1 Thesis Objectives . . . 4
- 1.2 Contributions and Accomplishments . . . 6
- 1.3 Organization . . . 8

2 Related Work 9
- 2.1 Multi-Agent Planning . . . 11
  - 2.1.1 Multi-Agent Planning Domain Description Languages . . . 12
  - 2.1.2 Multi-Agent Planners . . . 14

3 Formal Foundations of Multi-Agent Planning 17
- 3.1 Multi-Agent Planning Problem . . . 17
- 3.2 Privacy Classification of Facts and Actions . . . 19
- 3.3 Local Planning Problems . . . 21

4 Planning by Plan Set Intersection 25
- 4.1 Planning with External Actions . . . 25
- 4.2 Generic Planner . . . 29
- 4.3 Planning State Machines (PSM) . . . 31
  - 4.3.1 Basics of Planning State Machines . . . 31
  - 4.3.2 Extending a PSM with Solutions . . . 33
  - 4.3.3 Public Planning State Machines . . . 33
  - 4.3.4 Intersection of Public PSMs . . . 39
- 4.4 Multi-agent Planning with Complete PSMs . . . 41

5 PSM Planner 45
- 5.1 Internal Dependencies of Actions . . . 47
  - 5.1.1 Dependency Graphs . . . 48
  - 5.1.2 Publicly Equivalent Problems . . . 51
  - 5.1.3 Simple Action Dependencies . . . 52
  - 5.1.4 Dependency Graph Reductions . . . 53

5.1.5 Planning with Dependency Graphs . . . 58
5.1.6 Privacy Leakage Analysis . . . 60
5.2 Initial Relaxed Plan Landmark . . . 63
5.3 Generating New Plans . . . 64
5.4 Guiding Plan Search Using Public PSMs . . . 65
5.4.1 Plan Verification and Analysis . . . 67
5.5 Practical STRIPS Extensions . . . 70
5.5.1 From STRIPS to PDDL, and Back Again . . . 70
5.5.2 Internal Goals . . . 71

6 Experiments **73**
6.1 PSM-R and PSM-V Experimental Results . . . 73
6.1.1 Benchmark Domains . . . 74
6.1.2 Overall Benchmark Results . . . 76
6.1.3 Communication Overhead Evaluation . . . 79
6.2 PSM-VR Privacy Experiments . . . 80
6.2.1 Privacy Classifications . . . 80
6.2.2 Privacy in Benchmark Domains . . . 80
6.2.3 Privacy Benchmarks . . . 84
6.3 PSM-D Experimental Results . . . 86
6.3.1 Domain Analysis . . . 87
6.3.2 Experimental Results . . . 88

7 Conclusions **95**
7.1 Thesis Achievements . . . 96
7.2 Selected Related Publications . . . 98

Bibliography **101**

List of Figures

3.1 MA-STRIPS privacy classification of facts and actions. 19

4.1 A motivation example for computing PSM public projection. 35
4.2 Example of computing PSM public projection. 37
4.3 The complete PSM of agent `Truck` (Example 9). 42
4.4 Public projections of complete PSMs of agents from Example 9. 43
4.5 Intersection of public PSMs from Figure 4.4 representing all possible public solutions. 43

5.1 Graphical illustration of reduction operations (R1)–(R4). 54
5.2 Example of dependency graph containing all knowledge which can leak during *logistics* planning. 62

6.1 *Left*: Sequential and parallel times needed to solve task as a function of problem size. *Right:* Portion of time an agent spends on verification of other agents' plans. 78
6.2 Number of iterations and amount of communication of PSM variants. . 79
6.3 Performance of PSM-VR with different privacy classifications measured by the number of iterations and amount of communication. . . . 85
6.4 Comparison of planning times of PSM-VR algorithm without and with internal problem reduction. 88

List of Tables

2.1 Categorization of multi-agent planning by the number and the role of agents. . . . 10

6.1 Number of problems solved by the compared planners. Privacy classification follows FMAP and thus the results are not directly comparable with MA-STRIPS planners. . . . 77

6.2 Comparison of run times on selected problems solved by all the planners. Times are in seconds, PSM variants have number of iterations in parenthesis. . . . 78

6.3 Percentage of internal facts (left) and actions (right) in benchmark domains with respect to different privacy classifications. Values are averages over problems in each domain. . . . 84

6.4 Number of MA-STRIPS problems solved by the compared planners: RDFF, GPPP, and PSM-VR. . . . 85

6.5 Problem coverage of PSM-VR on benchmark domains with different privacy classifications. Number of problems in each domain is in parenthesis. . . . 86

6.6 Results of the analysis of internal dependencies of public actions in benchmark domains. . . . 87

6.7 PSM-VR and PSM-VRD: Overall coverage solved problem instances at CoDMAP competition. . . . 89

6.8 PSM-VR and PSM-VRD: IPC score over the plan quality at CoDMAP competition. . . . 90

6.9 PSM-VR and PSM-VRD: IPC Agile score over the planning time at CoDMAP competition. . . . 91

6.10 PSM-VR and PSM-VRD: Sum of times (in seconds) needed to solve selected problems at CoDMAP competition. . . . 92

List of Abbreviations

BRP	**B**est **R**esponse **P**lanning
CoDMAP	**C**ompetition **o**f **D**istributed and **M**ulti-**A**gent **P**lanners
CSP	**C**onstraint **S**atisfaction **P**roblem
CSTRIPS	**C**oncurrent **STRIPS**
DFS	**D**eterministic **F**inite **S**tate Machine
DPGM	**D**istributed **P**lanning through **G**raph **M**erging
FMAP	**F**orward **M**ulti-**A**gent **P**lanning
FSM	**F**inite **S**tate **M**achine
GPS	**G**eneral **P**roblem **S**olver
IPC	**I**nternational **P**lanning **C**ompetition
MAD-A*	**M**ulti-**A**gent **D**istributed **A***
MAP	**M**ulti-**A**gent **P**lanning
MA-PDDL	**M**ulti-**A**gent **PDDL**
MAPL	**M**ulti-**A**gent **P**lanning **L**anguage
MA-STRIPS	**M**ulti-**A**gent **STRIPS**
NADL	**N**on-deterministic **A**gent **D**omain **L**anguage
NFS	**N**on-deterministic **F**inite **S**tate Machine
PDDL	**P**lanning **D**omain **D**efinition **L**anguage
PSM	**P**lanning **S**tate **M**achine
SAT	Boolean **Sat**isfiability Problem
STRIPS	**St**anford **R**esearch **I**nstitute **P**roblem **S**olver

Dedicated to my future (A)I.

Chapter 1

Introduction

"As long as water flows, or grass grows upon the earth,
or the sun rises to show your pathway, . . . "

James Monroe

Planning is the process of reasoning about a current situation and then organizing one's own activities in order to achieve a desired goal. Automated planning tries to reproduce the same behavior algorithmically. Similarly to planning being an essential part of human everyday life, automated planning is an important part of artificial intelligence. Research began in the 1950s and GENERAL PROBLEM SOLVER (GPS), the first implemented solver, was created in 1959 (Newell, Shaw, and Simon, 1959).

We will focus on a restricted part of automated planning called classical planning and its extension to multi-agent planning. It has been shown, that classical planning belongs among the hardest problems in AI (Bylander, 1994; Mundhenk et al., 2000).

Planning can be targeted to solve many different problems, e.g. robot motion planning, manipulation planning, communication planning, etc. The obvious approach would appear be to create a *domain-specific* planner for each of these problems and to create and optimize techniques to solve it as efficiently as possible. In this thesis, we chose a different path. We focus on *domain-independent planning* which allows to specify any such a domain in a more or less general language. Only the generality of used language limits to which problems a domain-independent planner can be applied. This approach has several advantages. Obviously, the application of a domain-independent planner to a specific problem is less costly than a development of a new domain-specific planner. Also, studying the domain-independent planning allows us to better understand the planning process itself and thus to design more

general algorithms which actually form the next step towards fully-fledged artificial intelligence.

Classical planning can be naturally extended towards multiple planning entities that act in a shared environment and coordinate their activities in order to achieve common goals. Multi-agent planning with a deterministic model describes such problems and proposes techniques to solve them. The agents are obliged to keep information about their individual abilities private, they are not allowed to communicate it with other agents. Consider an application domain in which several logistic companies have to cooperate to fulfill complex transportation tasks, which cannot be managed by any of the companies individually. Although the companies have to cooperate, they still need to keep their know-how secret, for example their modes of transportation or local routes. In such a situation, the companies represented by planning agents would not benefit from any competitive behavior as the objective is common for all the companies, but they still have to keep parts of their knowledge private because of their local competition.

1.1 Problem Statement

The previous section introduced the problem of multi-agent planning where agents are defined by actions they can perform. Each action can change owner's internal state, or state of the environment, or both. This simple observation allows to naturally define which actions are considered *private*, and thus not shared with other agents, and which actions need to be *public* to allow inter-agent cooperation. MA-STRIPS (Brafman and Domshlak, 2008) defines actions *private* if they perceive and affect internal state only. The remaining actions are defined to be *public*. Under these settings, agents are willing to communicate their public actions with each other, although without details how these actions affect their internal states. Hence, we can suppose that every agent advertises its public actions before any calculation is performed.

Similarly, in the presented logistics scenario, each company advertises information about cities from which and to which it can transport the goods, but it avoids to share the details about the means of transport, detailed price calculations, etc. In the case, one trusted entity has all this private information, the problem of transporting a package that requires cooperation of multiple logistic companies can be easily described as classical planning problem. For classical centralized solution of this problem we could implement well-known A* (Hart, Nilsson, and Raphael, 1968) algorithm with appropriate heuristics and easily solve the problem. Nevertheless, the assumption of trusted entity, with whom all agents are willing to share their private knowledge, is not realistic.

In more realistic case, where no trusted entity is present, the agents need to find a solution without sharing their private knowledge. Natural approach to solve such a problem would be to take a solution of a previous problem, i.e. A*, and adapt it to new settings. Similar algorithms exist (Nissim and Brafman, 2012a; Jezequel and Fabre, 2012), and their execution directly corresponds to the execution of A* algorithm on the whole problem. In this case, the multi-agent nature of the problem just forms an obstruction that can be overcome using different techniques of distributed systems and cryptography.

Our approach to multi-agent planning is different. We see the natural distribution of the problem between participating agents as an opportunity to create new type of planner. Such a planner could profit from inherent multi-agent properties of distributed parallel execution and natural problem fragmentation with explicit definition of which information can be shared between the agents. This lead us to the main question of this thesis:

> *Is there more natural approach to multi-agent planning than distribution of existing centralized planning algorithms?*

In the real world, a company from described logistic scenario would create its plan while considering what other companies can achieve. Once a plan is created, it needs to be confirmed by all participating companies, which check whether they can perform their part of the plan. If all companies agree, they can sign the contract and start with the execution of the plan. If any company cannot perform required task, different plan has to be created, while a lesson from this failed attempt should be learned. This principle, where one planning entity proposes plans containing other agents' actions while having the right amount of knowledge about these actions, seems to be a promising idea for creation of a new multi-agent planner. The problem, this thesis solves, is to design and implement a competitive distributed multi-agent planner following this elementary idea. We summarize the objectives of this thesis in the following subsection.

1.1.1 Thesis Objectives

Following the main question introduced in the introduction of this section we can describe four main objectives of this thesis and one *bonus*, nice but hard to achieve, objective as follows:

(1) *Analysis of multi-agent coordination.*
Unlike in classical planning, no agent is aware of the whole problem. Therefore, agents need to coordinate their actions possibly affecting other agents. This needs to be formally described and analyzed from the perspective of how this coordination problem can be used for multi-agent planning.

(2) *Design novel coordination-centric multi-agent planner.*
After the analysis of the coordination problem, we aim to design a multi-agent planner that exploits properties of the coordination problem. Human approach to solve this coordination problem, where one person tries to plan actions for the others, should be integrated in the designed planner. The separation of coordination problem and agents' local problems should be the main idea of the designed planner.

(3) *Planner implementation.*
The designed planner will be implemented. We prefer to build our new multi-agent planner on top of an existing classical planner. This approach would take profit of the steady progress in classical planning.

(4) *Planner evaluation.*
Implemented planner will be experimentally compared with state-of-the-art multi-agent planners using a common set of benchmark problems. Such a benchmark set needs to be created and it should be based on existing IPC benchmarks for classical planning.

(Bonus) *Learned lesson for classical planning?*
During the steps described above, is there anything what can contribute to the research in classical planning? Possibly the distribution of the problem, or the approach separating local problems from the coordination problem?

1.2 Contributions and Accomplishments

This thesis is a compilation of author's several previous publications coauthored by Jan Jakubův and Antonín Komenda.

In this thesis, we present an extensibility-based multi-agent planning algorithm which utilizes Finite State Machines to compactly represent sets of plans. We call this representation *Planning State Machines* (PSM). PSMs not only allow us to compactly represent (even infinite) sets of plans by a finite structure but mainly allow us to effectively implement operations crucial for our planning algorithms. The main idea of PSM-based planner was briefly sketched in (Tožička, Jakubův, and Komenda, 2014) and the formal development including proofs of soundness and completeness was published in (Tožička et al., 2016). This basic planner is extended by an extensibility approximation using a type system checker for process calculi (Jakubův, Tožička, and Komenda, 2015), by a method of a distributed relaxed heuristic used for example in MADLA planner (Štolba and Komenda, 2014), and by a possibility to share some information about private knowledge (Tožička, Jakubův, and Komenda, 2015a). This gives rise to a multi-agent planner outperforming the state-of-the-art planners.

We use classical multi-agent benchmark domains found in the literature to evaluate our planner. We provide a comprehensive domains description and we compare experimental results with other state-of-the-art planners. Finally, we analyze the benchmark domains from the point of view of different privacy classifications. Different privacy classifications differ in facts explicitly revealed to other agents. While other planners are usually designed with a fixed privacy classification in mind, we show that our planner can be easily adjusted to work with various privacy classifications. Hence we provide the user the freedom to choose what is public, and we can directly compare our planner with other planners. Furthermore, we show that a restricted public knowledge can even improve planner performance.

The main contribution of this thesis is the design of a novel PSM planner accompanied with formal proofs and several heuristics, that improve planner efficiency. Presented experiments show, that the PSM planner outperforms the best state-of-the-art planners. This result has been independently confirmed by the Competition of Distributed and Multi-Agent Planners collocated with ICAPS 2015 where the PSM planner won at *Coverage* and *Quality* scores of distributed track (Štolba, Komenda, and Kovács, 2016).

A paper describing the PSM planner with novel reductions of private knowledge (Tožička, Jakubův, and Komenda, 2016) has been awarded the Best Student Paper Award at ICAART 2016[1]. We have also extended these reductions to classical planning and demonstrated their impact on existing state-of-the-art planners. Paper

[1] `http://www.icaart.org/PreviousAwards.aspx`

describing this research has been accepted as a full paper to the main track of the most prestigious planning conference ICAPS 2016 (Tožička et al., 2016).

1.3 Organization

This thesis is composed of three main parts. In the first part, we introduce the state of the art in multi-agent planning in Chapter 2 and we describe formal model of the multi-agent planning in Chapter 3.

The second part of the thesis presents the proposed multi-agent planner. Chapter 4 describes a general scheme of the planner and introduces *planning state machines* which allow its effective implementation. Chapter 5 extends the planner scheme by several heuristics and provides details of its implementation.

The last part of the thesis is devoted to experimental evaluation of the proposed planner (Chapter 6) and conclusions (Chapter 7).

Chapter 2

Related Work

"Seek wisdom, not knowledge. Knowledge is of the past, wisdom is of the future."

Lumbee Proverb

After a brief overview of classical planning methods, this chapter summarizes published research on multi-agent planning in Section 2.1. Then, we look on formalisms extending single agent PDDL to multi-agent planning problems focusing mainly on definition of privacy of agent knowledge in Section 2.1.1. Finally, Section 2.1.2 provides an overview of different approaches to the solution of multi-agent planning problem.

In classical planning, the plan synthesis process is typically a systematic search in either space of states, plans, or a combination of both (Ghallab, Nau, and Traverso, 2004).

State-space planners explore search space directly corresponding to the transition system of the planning problem. Nodes are thus states and arcs correspond to actions. Solution of the planning problem is then any path between the initial state and any goal state. Most common approaches to implement a state-space planner are based either on *forward search*, or *backward search*, or some combination of both approaches. Different approach offers hierarchical planning where the plan is searched in hierarchy of abstractions rather than original states (Sacerdott, 1973).

Plan-space search space contains nodes representing sequences of actions. Arcs then correspond to elementary refinements of the plan (e.g., adding, or removing some action in order to fulfill another goal fact, or to fix some of plans inconsistencies). Search starts with a state representing an empty plan, and it ends as soon as a state representing a valid solution is found. An obvious disadvantage of plan space is that it is infinite unlike the corresponding state space. However, a popular principle of *least commitment planning* allowed to create effective planners (Tate, 1976).

By \ For	**single agent**	**mutliple agents**
single agent	classical, single agent planning	centralized multi-agent planning
mutliple agents	distributed / factored planning	distributed multi-agent planning

TABLE 2.1: Categorization of multi-agent planning by the number and the role of agents.

Plan-space planning also seems to be well suitable for multi-agent planning (Ghallab, Nau, and Traverso, 2004) and we build a multi-agent planner inspired by plan-space planning later in this thesis.

A different approach to planning represents Graphplan (Blum and Furst, 1995) that creates an effective structure called *planning graph* which contains states that could possibly be reachable from the initial state and once a goal state is reached it searches for a valid plan within this structure. SATPLAN (Kautz, Mcallester, and Selman, 1996) translates the constraints represented by the planning graph into a set of clauses and uses general SAT solver to find a solution. The solution of SAT problem is then translated back to the solution of the original planning problem. Improved version of SATPLAN (Kautz, Selman, and Hoffmann, 2006) won 1st prize among optimal planners at the deterministic track of IPC 2006 competition.

Since the computational complexity of classical planning is intractable in the worst case, the use of heuristics is inevitable to achieve acceptable planner efficiency. Among the published heuristics, *merge-and-shrink* heuristic (M&S) (Helmert et al., 2007) is the most interesting one for our work. M&S firstly creates atomic projections of the problem. Then, it subsequently *merges* these projections and *shrinks* them to fit in the memory. The resulting abstraction is used to estimate the price of the solution of the original problem. We use similar approach to the *merging* step of the algorithm when combining local solutions of multiple agents (see Chapter 4 for details).

Another heuristics we use in our planner is *delete relaxation* heuristic (Bonet and Geffner, 2001). This heuristic creates a relaxed problem where it just ignores all delete effects of actions. The cost of a solution of this relaxed problem is then used as an estimate of the cost of the solution of the original problem. In our planner, we use the solution of the relaxed problem as a first approximate solution of the whole problem. This approach helps agents to direct their local searches to a common global solution (Section 5.2).

2.1 Multi-Agent Planning

The problem of multi-agent planning is understood in several different ways by AI researchers. Following (de Weerdt and Clement, 2009), we can generally define it as a group of planning agents which create a plan for a group of executing agents. Planning and executing agents can be the same entities, or either of these groups can be represented by a single agent. Table 2.1 summarizes four different cases. Classical planning belongs to the case when a single agent plans for a single agent. When we distribute this process among multiple agents, we move towards planning by multiple agents for a single executing agent, which is known as distributed or factored planning. Many domains of classical planning contain multiple entities whose actions are planned; logistics, for example, contains multiple vehicles transporting packages. When the actions for these entities are created centrally, i.e., by a single entity, we talk about centralized multi-agent planning. Once we distribute these actions between the executing agents, which coordinate their actions, but each of them plans actions for itself to achieve its own goals, we get to *distributed multi-agent planning*. In the following text, multi-agent planning (MAP) describes the distributed multi-agent planning where agents collaborate on a common goal, unless it is stated otherwise.

MAP problems can be classified by the level of cooperation between the agents. Brafman et al. in (Brafman and Domshlak, 2008) defines the coupling of an problem as a tree-width of the agent interaction graph. More abstractly, we say that a problem is *loosely coupled* when there is low level of inter-agent interaction, whereas *tightly coupled* problems contain agents with rich interaction graphs.

The most used multi-agent planning model MA-STRIPS (Brafman and Domshlak, 2008) prescribes a privacy scheme defining which information has to stay public. Although the original motivation was to analyze the computational complexity of the multi-agent planning problem, most of the planners in the literature stick to this particular definition. The most notable exception is the FMAP planner (Torreño, Onaindia, and Sapena, 2014) which allows to mark public information in the planning problem description. Other formalisms describing various multi-agent extensions of PDDL are described in Section 2.1.1. The planner proposed in this thesis does not require any specific privacy definition and thus can be compared with both types of privacy definitions as shown in Chapter 6.

The concept of private knowledge is the most important aspect of multi-agent planning. It can factor a planning problem and thus positively affect the complexity of the planning process (Brafman and Domshlak, 2008). Nevertheless, multi-agent planners usually do not target this particular facet of the problem. Distributed MA-STRIPS multi-agent planners in literature can be roughly separated to three groups

by privacy preservation. Most of the planners follows concept of privacy by *information obfuscation* (Nissim and Brafman, 2012b; Borrajo, 2013) or *information aggregation* (Štolba and Komenda, 2014; Torreño, Onaindia, and Sapena, 2014). With information obfuscation, agents are allowed to communicate private information with other agents as far as the information is obfuscated such that only the owning agent can understand it (for example, the name of an action is replaced by a hash code). With information aggregation, the information is aggregated such that only the owning agent knows all details (for example, a summed up cost of private actions can be send to other agents). An exception is the GPPP planner (Maliah, Shani, and Stern, 2014) providing *full privacy* by communicating only public information. Our approach also provides full privacy. Especially in the contrast to the obfuscation principle, we can reduce the size of plan space because privacy preservation acts as natural abstraction of the problem from perspective of particular agents. Communication of abstracted plans thus also decreases the amount of communicated data.

2.1.1 Multi-Agent Planning Domain Description Languages

STRIPS and PDDL are two standard languages to describe deterministic single-agent planning problems. Nevertheless, there is no similar standard in the multi-agent planning. There are several attempts to create such a standard. Following paragraphs provide a closer look on these standards in chronological order.

NON-DETERMINISTIC AGENT DOMAIN LANGUAGE (NADL) (Jensen and Veloso, 2000) allows to describe multi-agent non-deterministic domains. It distinguishes two types of agents. *System* agents are controllable entities for which the planner creates a plan. The plans are executed synchronously and it is assumed that each action has unit duration. On the contrary, *environment* agents cannot be controlled by the planner and thus they represent non-deterministic changes in the environment which cannot be controlled by *system* agents. Description of each agent contains a list of actions it can execute.

Boutilier et al. (Boutilier and Brafman, 2001) extended STRIPS by description of *concurrent interacting actions* to allow to plan for multi-agent teams with possibly colliding actions. Description of each action is extended to contain a *concurrent* action list specifying which actions must be, and which cannot be, executed concurrently. Together with *conditional effects*, the actions can result in different effects depending on which actions are executed concurrently by other agents. This allows to naturally describe *joint* actions, i.e. actions that require synchronized cooperation of several agents to achieve a goal which cannot be achieved by actions of single agents.

MULTI-AGENT PLANNING LANGUAGE (MAPL) (Brenner, 2003) extends PDDL to allow the description of RoboCup Rescue search and rescue scenarios (Kitano et al., 1999). In these scenarios, agent cannot perceive the complete state of the

world. In MAPL, this is simulated by a special value *unknown* which can be assigned as a variable value. Colliding actions, that cannot be executed synchronously, are described by *read-write locks* on state variables. Non-deterministic, or a priori unknown, duration of an action is described as an interval in the action specification. MAPL then introduces special actions for inter-agent communication. Therefore, communication acts can be incorporated directly into agents' plans to allow explicit plan synchronization. Communicative actions explicitly describe what is being communicated between the agents and thus this approach allows to control the level of private knowledge leakage.

CONCURRENT STRIPS (CSTRIPS) (Oglietti and Cesta, 2004) extends STRIPS to deal with concurrent actions. Each agent is described as a collection of *concurrent threads* for modeling possible concurrent activities. Each concurrent thread is represented by its own set of actions and thus the usage of an action inherently describes which agent operates it without the necessity to explicitly specify it as it is usual in other domain descriptions. However, this CSTRIPS is defined on the model level only and has never been used by any planner.

MA-STRIPS (Brafman and Domshlak, 2008) is a natural extension of STRIPS extending the main idea of CSTRIPS to partition the actions between the agents. In MA-STRIPS, the STRIPS set of actions is replaced by a set of disjoint sets of actions, where each set of actions represents the abilities of a single agent. MA-STRIPS also distinguishes between public information, which can be freely communicated between the agents, and private information, which describes agent's internal functioning and which should not be communicated in scenarios with privacy protection requirements. Public facts are defined as facts appearing in actions of multiple agents and these actions are then defined as public. The complement forms private facts, and actions, respectively. We use MA-STRIPS privacy definitions (see Chapter 3 for details) through this thesis and we provide a detailed analysis of its consequences for multi-agent planning domains in Section 6.2.2.

MA-PDDL (Kovács, 2012) is an extension of PDDL 3.1 which allows to specify the owner of each action using field `agent` in the definition of the action. MA-PDDL allows to define different goals for different agents and positive and negative interferences between their concurrent actions. Nevertheless, MA-PDDL does not allow to manually specify what facts are public/private, and thus it is not suitable for description of domains with privacy preservation.

FMAP (Torreño, Onaindia, and Sapena, 2014) uses separate domain and problem PDDL files for each agent. This approach allows easy and natural distribution of planning process. PDDL language is extended with `shared-data` field which allows to specify which predicates are shared with which agents. However, this approach does not always allow to define fact privacy classification as defined by MA-STRIPS. FMAP also refers to a state-of-the-art planner which uses this domain description. We compare our designed planner with FMAP in Chapter 6.

None of the existing PDDL extensions allows to express both FMAP (public facts specified by a list of public predicate names) and MA-STRIPS (see Chapter 3) privacy definitions. Nevertheless, our proposed planner allows to finely tune the amount of knowledge shared among the agents up to the level of single facts and thus it can work with any definition. Section 6.2.2 describes in details the differences in these privacy specifications for different multi-agent domains.

2.1.2 Multi-Agent Planners

The first multi-agent planner for MA-STRIPS called PLANNING FIRST (Nissim, Brafman, and Domshlak, 2010) transforms the inter-agent coordination problem into a CSP problem in combination with local planning. In generated CSP problem, each agent is represented by a single variable whose values represent all possible sequences of its public actions bounded by the maximal length. CSP constraints then specify which sequences can be combined into a valid global solution and which sequences are also locally valid. Following the principle of iterative deepening, the maximal length of sequences is iteratively increased until a solution is found. However, this does not guarantee the optimality of the solution, but minimizes the maximal number of one agent's public actions. PLANNING FIRST planner shows very good performance for loosely coupled problems.

MAD-A* (Nissim and Brafman, 2012a; Nissim and Brafman, 2012b) is a distributed multi-agent adaptation of well-known A* algorithm. Each agent maintains its own search space and explores it using its own actions. Expanded states are broadcasted to all other agents which then use their actions to expand it too creating possible distinct children states. The messages communicated between the agents contain full description of a state, i.e., it includes public part of the state and also all private parts of all the agent. Since the actions of an agent need to access only the public part of a state and its own private part, the private parts of other agents can be encrypted to hide the private information. Unlike A*, the fact, that a goal state has been selected for the expansion, does not guarantee that optimal solution has been found. Therefore, once an agent expands a goal state, a verification procedure that checks the optimality of the solution is performed.

SELFISH-MAD-A* (Nissim and Brafman, 2013) extends MAD-A* algorithm to self-interested agents. In such a scenario, each agent tries to maximize its reward while cooperating on planning task with other agents. Selfish-MAD-A* uses a distributed implementation of Vickery-Clarke-Groves approach (Vickrey, 1961; Clarke, 1971; Groves, 1973) to mechanism design. Centralized trusted bank assures the distribution of payments after receiving partial costs after removing an agent from the plan from all the agents. Although agents share some information derived from

their private knowledge with the centralized bank, the algorithm preserves similar level of privacy as the original MAD-A*.

DPGM (Pellier, 2010) is a multi-agent extension of Graphplan (Blum and Furst, 1995). Each agent creates its own planning graph and extracts local *threats* and *promotions*, i.e. negative and positive interactions between agents, and exchanges them with other agents. Received knowledge is merged into local planning graphs and every agent extracts local solutions. The coordination problem of these local solutions is described using CSP constraints and solved as a CSP problem. If the coordination of local solutions fails, different local solution are extracted and coordinated. If no other local solutions are available, the local planning graphs are extended by a new layer implementing iterative deepening search, which assures the completeness of the algorithm. DPGM implementation (Durkota and Komenda, 2013) shows good performance for loosely coupled problems while it is outperformed by MAD-A* on tightly coupled problems.

FMAP (Torreño, Onaindia, and Sapena, 2014) is a representative of a multi-agent partial ordered planner. It applies multi-agent A* heuristic search to explore the space of partial plans. Selected partial plan is *refined* using embedded forward-chaining partial order planner. FMAP shows state-of-the-art performance in problems of all levels of coupling, and thus we compare our proposed planner with FMAP in experimental evaluation (see Chapter 6 for detailed results).

BRP (Jonsson and Rovatsos, 2011) represents a best-response planning method for improving an existing plan created by different planning method. Computing the best response is formulated as a modified local planning problem and thus can be solved by any classical planner. The authors also define *congestion planning problems* which correspond to *congestion games* (Rosenthal, 1973) and they show that BRP is guaranteed to result in a Nash equilibrium for this type of planning problems.

μ-SATPLAN (Dimopoulos, Hashmi, and Moraitis, 2010) is a multi-agent extension of classical SATPLAN where agents compute their plans sequentially. First agent creates a solution to his local problem and sends it to next agent. After a plan solving the problem from the perspective of several agents is received, an agent creates a new solution for his local problem which does not collide with the received plan and sends it to the next agent. SATPLAN (Kautz, Selman, and Hoffmann, 2006), which uses planning graphs for the translation of the planning problem into SAT problem, is used as underlying local planning system. The presented experiments show that the algorithm works well for two agents. Nevertheless, given the nature of the algorithm, it does not scale well for larger number of agents.

Multi-agent planning can also be seen as a specific form of *factored planning* (Amir and Engelhardt, 2003). Factored planning tries to decompose a planning problem into possibly independent subproblems. Solving these subproblems scales linearly with the size of the domain and in the worst case exponentially with the size only of the largest subproblem and interactions among subproblems. Obviously, the catch

is that not all planning problems can be factored enough to benefit from such efficiency gain. In (Brafman and Domshlak, 2006), causal graphs (Bacchus and Yang, 1994) of the planning domains are used to identify when factorization is computationally beneficial. A practical algorithm based on this result and on principle of decomposition trees (Darwiche and Hopkins, 2001) was proposed in (Kelareva et al., 2007). This principle can be also seen as a variation on localized planning (Lansky and Getoor, 1995). The difference between multi-agent planning and factored planning is that in multi-agent planning the factorization is fixed and given by agent abilities.

DISTOPLAN (Fabre et al., 2010) pioneered the idea of planning by means of Finite State Machines (FSM) containing local solutions. DISTOPLAN aims at (optimal) factored planning where all the information can be shared between the agents, i.e., no privacy is achieved. Although the results show good performance only for few problems which factor well, we have extended the idea of using FSM to represent a set of local solutions into *planning state machine* (PSM; see Section 4.3 for details). Additionally, we use a principle of intersection of the PSMs to effectively filter out unfeasible combination of plans of different agents. This approach proved to be efficient even for problems that cannot be easily factored and it also provides high level privacy preservation.

Besides representation of local plans as totally or partially ordered sequences of actions, a compact representation of set of local plans utilizing various types of Finite State Machines was proposed in aforementioned (Fabre et al., 2010) and our recent work (Tožička, Jakubův, and Komenda, 2014). In (Tožička et al., 2014), we have proposed notions of *external actions* and public plan *extensibility*. When planning with external actions, agents are informed about public actions of other agents. Hence they are able to plan actions *for* other agents. However, external actions are striped of private information and thus it can happen that an agent plans an external action inappropriately. The notion of extensibility allows to recognize plans where external actions are used correctly. In (Tožička, Jakubův, and Komenda, 2014), we have used extensibility with PSMs to outline a generic scheme of multi-agent planners further elaborated in Chapter 5.

Chapter 3

Formal Foundations of Multi-Agent Planning

"Be still and the Earth will speak to you."

Navajo Proverb

Similarly as in classical planning, we assume a planning model based on extension of STRIPS (Fikes and Nilsson, 1971) compactly representing a deterministic transition system. The multi-agent extension, described in Section 3.1, follows the principles proposed in MA-STRIPS by Brafman and Domshlak in (Brafman and Domshlak, 2008). Agent capabilities are described as a finite repertoire of agent's STRIPS actions. MA-STRIPS defines privacy classification of facts and action, which we describe in details in Section 3.2. The agent actions possibly affect only parts of the environment thus inducing local planning problems of the particular agent. Local planning problems are described in Section 3.3. Therefore, this (partial) "separation of concerns" of the agents keeps the private information local. It also helps to increase efficiency of the planning process by hiding parts irrelevant for other agents.

In this thesis, the agents are *cooperative* and *coordinated* and they concurrently plan and execute their local plans in order to achieve a joint goal. The environment wherein the agents act is *classical* with *deterministic* actions.

3.1 Multi-Agent Planning Problem

An MA-STRIPS *planning problem* Π is a quadruple $\Pi = \langle P, \{\alpha_i\}_{i=1}^{n}, I, G\rangle$, where P is a set of facts, α_i is the set of actions of i-th agent, $I \subseteq P$ is an initial state, and $G \subseteq P$ is a set of goal facts. Selector functions $\mathsf{facts}(\Pi)$, $\mathsf{agents}(\Pi)$, $\mathsf{init}(\Pi)$, and $\mathsf{goal}(\Pi)$ are defined so that $\Pi = \langle \mathsf{facts}(\Pi), \mathsf{agents}(\Pi), \mathsf{init}(\Pi), \mathsf{goal}(\Pi)\rangle$ holds for any problem Π.

An *action* an agent can perform is a quadruple containing unique action id and three subsets of $\mathsf{facts}(\Pi)$ which in turn denote the set of *preconditions*, the set of *add effects*, and the set of *delete effects*. Action ids are arbitrary atomic objects and we always consider ids to be unique within a given problem. Selector functions $\mathsf{id}(a), \mathsf{pre}(a)$, $\mathsf{add}(a)$, and $\mathsf{del}(a)$ are defined so that $a = \langle \mathsf{id}(a), \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$ holds for any action a. Moreover let $\mathsf{eff}(a) = \mathsf{add}(a) \cup \mathsf{del}(a)$.

An *agent* is identified with its capabilities, in other words, the i-th agent $\alpha_i = \{a_1, \ldots, a_m\}$ is determined by a finite set of actions it can preform in the environment. We use metavariable α to range over agents from Π. A *planning state* s is a finite set of facts and we say that fact p *holds in* s, or that p *is valid in* s, iff $p \in s$. When $\mathsf{pre}(a) \subseteq s$ then *state progression* function γ is defined classically as $\gamma(s, a) = (s \setminus \mathsf{del}(a)) \cup \mathsf{add}(a)$.

Example 1 *Throughout the paper, we shall use the following running example concerning a small logistic company. The company owns two transport vehicles (*`plane` *and* `truck`*) and operates three locations (*`prague`*,* `brno`*, and* `ostrava`*). A* `plane` *can travel from* `prague` *to* `brno` *and back, while a* `truck` *provides connection between locations* `brno` *and* `ostrava`*. The company receives a delivery job to transport the* `crown` *from* `prague` *to* `ostrava`*. The company manager needs to plan tasks for the vehicle operators so that the delivery job is done.*

This delivery problem can be expressed using MA-STRIPS *model as follows. Actions* `fly`(*loc*$_1$, *loc*$_2$) *and* `drive`(*loc*$_1$, *loc*$_2$) *describe movement of* `plane` *and* `truck` *respectively. Actions* `load`(*veh*, *loc*) *and* `unload`(*veh*, *loc*) *describe loading and unloading of* `crown` *by a given vehicle at a given location.*

We define two agents `Plane` *and* `Truck`*. The agents are defined by sets of executable actions as follows.*

$$
\begin{aligned}
\mathit{Plane} = \{\ & \texttt{load(plane,prague)}, \texttt{load(plane,brno)}, \\
& \texttt{unload(plane,prague)}, \texttt{unload(plane,brno)} \\
& \texttt{fly(brno,prague)}, \texttt{fly(prague,brno)},\ \} \\
\mathit{Truck} = \{\ & \texttt{load(truck,brno)}, \texttt{load(truck,ostrava)}, \\
& \texttt{unload(truck,brno)}, \texttt{unload(truck,ostrava)} \\
& \texttt{drive(ostrava,brno)}, \texttt{drive(brno,ostrava)},\ \}
\end{aligned}
$$

Aforementioned actions are defined using facts `at`(*veh*, *loc*) *to describe possible vehicle locations, and facts* `in`(`crown`, *loc*) *and* `in`(`crown`, *veh*) *to describe positions of* `crown`.

$\mathsf{facts}(a)$	$=$	$\mathsf{pre}(a) \cup \mathsf{add}(a) \cup \mathsf{del}(a)$	*facts of action* a
$\mathsf{facts}(\alpha)$	$=$	$\bigcup_{a \in \alpha} \mathsf{facts}(a)$	*facts of agent* α
$\mathsf{pub\text{-}facts}(\Pi)$	$=$	$\bigcup_{\alpha \neq \beta} (\mathsf{facts}(\alpha) \cap \mathsf{facts}(\beta))$	*public facts of* Π $(\alpha, \beta \in \mathsf{agents}(\Pi))$
$\mathsf{int\text{-}facts}(\alpha)$	$=$	$\mathsf{facts}(\alpha) \setminus \mathsf{pub\text{-}facts}(\Pi)$	*facts internal for agent* α
$\mathsf{rel\text{-}facts}(\alpha)$	$=$	$\mathsf{facts}(\alpha) \cup \mathsf{pub\text{-}facts}(\Pi)$	*facts relevant for agent* α
$\mathsf{pub\text{-}actions}(\alpha)$	$=$	$\{a \in \alpha : \mathsf{eff}(a) \cap \mathsf{pub\text{-}facts}(\Pi) \neq \emptyset\}$	*public actions of agent* α
$\mathsf{int\text{-}actions}(\alpha)$	$=$	$\alpha \setminus \mathsf{pub\text{-}actions}(\alpha)$	*internal actions of agent* α

FIGURE 3.1: MA-STRIPS privacy classification of facts and actions.

We omit action ids in examples when no confusion can arise. For example, we have the following.

$$\begin{aligned}
\mathtt{fly}(\mathit{loc}_1, \mathit{loc}_2) = \langle &\{\mathtt{at}(\mathtt{plane}, \mathit{loc}_1)\}, \\
&\{\mathtt{at}(\mathtt{plane}, \mathit{loc}_2)\}, \\
&\{\mathtt{at}(\mathtt{plane}, \mathit{loc}_1)\}\rangle \\
\mathtt{load}(\mathit{veh}, \mathit{loc}) = \langle &\{\mathtt{at}(\mathit{veh}, \mathit{loc}), \mathtt{in}(\mathtt{crown}, \mathit{loc})\}, \\
&\{\mathtt{in}(\mathtt{crown}, \mathit{veh})\}, \\
&\{\mathtt{in}(\mathtt{crown}, \mathit{loc})\}\rangle
\end{aligned}$$

The initial state and the goal are given as follows.

$$I = \{\mathtt{at}(\mathtt{plane}, \mathtt{prague}), \mathtt{at}(\mathtt{truck}, \mathtt{brno}), \mathtt{in}(\mathtt{crown}, \mathtt{prague})\}$$
$$G = \{\mathtt{in}(\mathtt{crown}, \mathtt{ostrava})\}$$

⊠

The goal reflects the delivery requirement.

3.2 Privacy Classification of Facts and Actions

In MA-STRIPS multi-agent planning, each fact is classified either as *public* or as *internal* out of computational or privacy concerns. MA-STRIPS specifies this classification as follows. A fact is *public* when it is mentioned by actions of at least two different agents. A fact is *internal for agent* α when it is not public but mentioned by some action of α. A fact is *relevant for* α when it is either public or internal for α. Relevant facts contain all the facts which agent α needs to understand, because other facts are

internal for other agents and thus not directly concerns α. Formal definitions and notations used throughout the thesis are presented in the upper parts of Figure 3.1.

It is possible to extend the set of public facts to contain additionally some facts that would be internal by the above definition. This is important for the experimental evaluation because some multi-agent planners use different facts classification. It is an advantage of our planner that it can be used with different facts classification because (1) we provide a user the freedom to choose what is public, and (2) we can directly compare our planner with planners that use different classifications. The only requirement for our planner is that every fact shared by at least two agents is public. Furthermore, it is common in literature (Nissim and Brafman, 2012b) to require that all the goals are public. An MA-STRIPS problem with internal goals can be easily transformed to an equivalent problem without internal goals (see Section 5.5.2) and thus we omit internal goals in formal presentation. Then $\mathsf{pub\text{-}facts}(\Pi)$ is defined as the minimal superset of the intersection from the definition that satisfies $G \subseteq \mathsf{pub\text{-}facts}(\Pi)$. In this thesis we suppose $G \subseteq \mathsf{pub\text{-}facts}(\Pi)$ and also another simplification common in literature (Brafman and Domshlak, 2008) which says that α_i are pairwise disjoint[1].

Example 2 *In our running example,* $\mathtt{in(crown, brno)}$ *is the only fact shared by the two agents. As we require* $G \subseteq \mathsf{pub\text{-}facts}(\Pi)$ *we have the following facts classification.*

$$\begin{aligned}\mathsf{pub\text{-}facts}(\Pi) &= \{\mathtt{in(crown, brno)}, \mathtt{in(crown, ostrava)}\}\\ \mathsf{int\text{-}facts}(\mathtt{Plane}) &= \{\mathtt{at(plane, prague)}, \mathtt{at(plane, brno)},\\ &\qquad \mathtt{in(crown, prague)}, \mathtt{in(crown, plane)}\}\end{aligned}$$

⊠

MA-STRIPS further extends this classification of facts to actions as follows. An action is *public* when it has a public effect, otherwise it is *internal*. Strictly speaking, MA-STRIPS defines an action as public whenever it mentions a public fact even in a precondition (that is, when $\mathsf{facts}(a) \cap \mathsf{pub\text{-}facts}(\Pi) \neq \emptyset$). However, our method of multi-agent planning does not rely on synchronization of public preconditions, and hence we can allow actions with only public preconditions to be internal. For our

[1] This rules out *joint actions*. Any MA-STRIPS problem with joint actions can be translated to an equivalent problem without joint actions. However, a solution that would take advantage of joint actions is left for future research.

planner it is enough to know that internal actions do not *modify* public state. Formal definitions and notations are presented in the lower part of Figure 3.1.

3.3 Local Planning Problems

In MA-STRIPS problems, agent actions are supposed to manipulate a shared global state when executed. In multi-agent planning with external actions, a *local planning problem* is constructed for every agent α. Each local planning problem of α is a classical STRIPS problem containing α's own actions together with information about public actions of other agents. These local planning problems allow us to divide an MA-STRIPS problem to several STRIPS problems which can be solved separately by a classical planner. This thesis describes a way how to find a solution of an MA-STRIPS problem but it does not address the question of *execution* of a plan in some real-world environment.

The *projection* $F \triangleright \alpha$ of set of facts F to agent α is the restriction of F to the facts relevant for α. Hence projection removes from F facts not relevant for α and thus it represents F as understood by agent α. The *projection* $a \triangleright \alpha$ of action a to agent α removes from a facts not relevant for α, again representing a as seen by α. The projections are formally defined as follows.

Definition 1 *Given* Π*, let* F *be an arbitrary set* $F \subseteq \mathsf{facts}(\Pi)$ *of facts and let* a *be an action from* Π*. The* projection $F \triangleright \alpha$ of F to $\alpha \in \mathsf{agents}(\Pi)$*, and the* projection $a \triangleright \alpha$ of action a to α *are defined as follows.*

$$\begin{aligned} F \triangleright \alpha &= F \cap \mathsf{rel\text{-}facts}(\alpha) \\ a \triangleright \alpha &= \langle \mathsf{id}(a), \mathsf{pre}(a) \triangleright \alpha,\ \mathsf{add}(a) \triangleright \alpha,\ \mathsf{del}(a) \triangleright \alpha \rangle \end{aligned}$$

The action projection is extended to sets of actions element-wise. ⊠

Note that $a \triangleright \alpha = a$ when $a \in \alpha$. Hence projection to α alters only actions of agents other than α. Also note that action ids are preserved under projection.

Example 3 *In our example we have the following.*

$$
\begin{aligned}
\texttt{fly(prague,brno)} \triangleright \textit{Plane} &= \texttt{fly(prague,brno)} \\
\texttt{fly(prague,brno)} \triangleright \textit{Truck} &= \langle \emptyset, \emptyset, \emptyset \rangle \\
\texttt{load(truck,brno)} \triangleright \textit{Plane} &= \langle \{\texttt{in(crown,brno)}\}, \emptyset, \\
&\qquad \{\texttt{in(crown,brno)}\} \rangle \\
\texttt{unload(truck,ostrava)} \triangleright \textit{Plane} &= \langle \emptyset, \{\texttt{in(crown,ostrava)}\}, \emptyset \rangle
\end{aligned}
$$

⊠

In multi-agent planning with *external actions*, every agent α is from the beginning equipped with projections of public actions of other agents. These projections, which we call external actions, describe how agent α sees effects of public actions of other agents. In a local planning problem, an agent needs external actions so that he can create a plan which contains also public actions of other agents. The set of actions in a local planning problem of agent α simply contains actions of agent α together with external actions of α. Now it is easy to define a *local planning problem* $\Pi \triangleright \alpha$ of agent α also called *projection of* Π to α as a classical STRIPS problem. The set of facts P and the initial state I are restricted to those facts relevant for α. There is no need to restrict the goal G because all the goal facts are public and thus relevant for all the agents. A formal definition follows.

Definition 2 *Given an* MA-STRIPS *problem* Π*, the* local planning problem $\Pi \triangleright \alpha$ *of agent* α *is defined for every* $\alpha \in \mathsf{agents}(\Pi)$ *as the classical* STRIPS *problem*

$$\Pi \triangleright \alpha = \langle \mathsf{facts}(\Pi) \triangleright \alpha,\ \alpha \cup \mathsf{ext\text{-}actions}(\alpha),\ \mathsf{init}(\Pi) \triangleright \alpha,\ G \rangle$$

where the set $\mathsf{ext\text{-}actions}(\alpha)$ *of* external actions of agent α *is defined as follows.*

$$\mathsf{ext\text{-}actions}(\alpha) = \bigcup_{\beta \neq \alpha} (\mathsf{pub\text{-}actions}(\beta) \triangleright \alpha) \qquad \textit{(for all } \beta \in \mathsf{agents}(\Pi)\textit{)}$$

⊠

Example 4 *In our example, all the actions arranging vehicle movements are internal. Public actions are only the actions providing package manipulation at public locations* `brno` *and* `ostrava`*. Hence the set* $\mathsf{pub\text{-}actions}(\textit{Plane})$ *contains actions* `load(plane,brno)` *and*

`unload(plane, brno)` *while* pub-actions(`Truck`) *is as follows:*

$$\{\ \texttt{load(truck, brno)}, \texttt{unload(truck, brno)},\ \texttt{load(truck, ostrava)}, \texttt{unload(truck, ostrava)}\ \}$$

Hence ext-actions(`Truck`) *has 2 actions and* ext-actions(`Plane`) *has 4 actions. This yields the local problem* $\Pi \triangleright$ `Plane` *with 10 actions and the problem* $\Pi \triangleright$ `Truck` *with 8 actions.* ⊠

Chapter 4

Planning by Plan Set Intersection

"One finger cannot lift a pebble."

Hopi Proverb

In this chapter, we show how *local planning problems* defined in Section 3.3 can be used to solve multi-agent planning problems. Section 4.1 introduces important properties of solutions of local planning problems and their relation to the solution of the whole problem. Section 4.2 proposes a general multi-agent planner scheme that is built on these properties. Then, we describe data structure that allows to effectively represent the generated plans, called *planning state machines* (PSMs), and thus effective implementation of the general planner scheme in Section 4.3. Finally, we present an algorithm allowing to compute all solutions of given multi-agent planning problem in Section 4.4.

4.1 Planning with External Actions

We would like to solve agent local problems separately and compose local solutions to a global solution of Π. However, not all local solutions can be easily composed to a solution of Π. Concepts of *public plans* and their *extensibility* help us to recognize local solutions which are suitable to this aim.

A *plan* π is a sequence of actions $\langle a_1, \ldots, a_k \rangle$. A plan π defines an order in which the actions are executed by their unique owner agents. It is supposed that independent actions can be executed in parallel. A *solution* of Π is a plan π whose execution

transforms the initial state I to the state s such that $G \subseteq s$. A *local solution* of agent α is a solution of the local planning problem $\Pi \triangleright \alpha$. Let $\mathsf{sols}(\Pi)$ and $\mathsf{sols}(\Pi \triangleright \alpha)$ denote the sets of all the solutions of MA-STRIPS problem Π and all the local solutions of α respectively.

Example 5 *Let us consider the following plans.*

$$\begin{aligned}
\pi_0 = \langle\ & \texttt{load(plane,prague)},\ \texttt{fly(prague,brno)},\\
& \texttt{unload(plane,brno)},\ \texttt{load(truck,brno)},\\
& \texttt{drive(brno,ostrava)},\ \texttt{unload(truck,ostrava)}\ \rangle\\
\pi_1 = \langle\ & \texttt{unload(truck,ostrava)} \triangleright \textit{Plane}\ \rangle\\
\pi_2 = \langle\ & \texttt{unload(plane,brno)} \triangleright \textit{Truck},\ \texttt{load(truck,brno)},\\
& \texttt{drive(brno,ostrava)},\ \texttt{unload(truck,ostrava)}\ \rangle
\end{aligned}$$

It is easy to check that π_0 is a solution of our example MA-STRIPS *problem Π. Plan π_1 is a solution of $\Pi \triangleright$ `Plane` because projection* $\texttt{unload(truck,ostrava)} \triangleright \textit{Plane}$ *of* `Truck`*'s public action simply produces the goal state out of the blue. Finally, $\pi_2 \in \mathsf{sols}(\Pi \triangleright$ `Truck`$)$.* ⊠

A *public plan* σ is a plan that contains only public actions. A public plan can be seen as a solution outline that captures execution order of public actions while ignoring agents internal actions. A public plan can be safely sent to any agent because it contains only public information. In order to avoid confusions between public and external versions of the same action, we formally define public plans to contain only public action *id*s. For a plan π of Π (or a plan of $\Pi \triangleright \alpha$) we define the *public projection* $\pi \triangleright \star$ of π as the sequence of all public action *id*s from π preserving their order. Public projection of a plan thus removes any internal actions from π. Formal definition follows.

Definition 3 *A* public plan σ *is a sequence of public action* ids*. Given a plan π of Π (or of $\Pi \triangleright \alpha$), the public projection $\pi \triangleright \star$ of π is defined to be the public plan $\pi \triangleright \star = \langle \mathsf{id}(a) : a \in \pi$ and $a \in \mathsf{pub\text{-}actions}(\Pi) \rangle$. Public projection is extended to sets of plans element-wise.* ⊠

Example 6 *In our example we know that* $\pi_0 \in \mathsf{sols}(\Pi)$ *and* $\pi_1 \in \mathsf{sols}(\Pi \triangleright \texttt{Plane})$ *and* $\pi_2 \in \mathsf{sols}(\Pi \triangleright \texttt{Truck})$*. Thus we can construct the following public solutions.*

$$\begin{aligned}
\pi_0 \triangleright \star &= \langle \mathsf{id}(\texttt{unload}(\texttt{plane}, \texttt{brno})), \mathsf{id}(\texttt{load}(\texttt{truck}, \texttt{brno})), \\
&\qquad \mathsf{id}(\texttt{unload}(\texttt{truck}, \texttt{ostrava}))\rangle \\
\pi_1 \triangleright \star &= \langle \mathsf{id}(\texttt{unload}(\texttt{truck}, \texttt{ostrava}))\rangle \\
\pi_2 \triangleright \star &= \langle \mathsf{id}(\texttt{unload}(\texttt{plane}, \texttt{brno})), \mathsf{id}(\texttt{load}(\texttt{truck}, \texttt{brno})), \\
&\qquad \mathsf{id}(\texttt{unload}(\texttt{truck}, \texttt{ostrava}))\rangle
\end{aligned}$$

Note that $\pi_0 \triangleright \star = \pi_2 \triangleright \star$ *and also note that we have omitted the projection operator (*$\triangleright$*) because ids are preserved under projection.* ⊠

From every solution π of Π (or of $\Pi \triangleright \alpha$) we can construct a uniquely determined public plan $\sigma = \pi \triangleright \star$. On the other hand, for a single public plan σ there might be more than one, or none, solutions with public projection σ. A public plan σ is called *extensible* when there is a solution of Π with public projection σ. Similarly, when there is a solution of $\Pi \triangleright \alpha$ with public projection σ, then σ is called α*-extensible*. Extensible public plans give us an order of public actions which is acceptable for all the agents. Thus extensible public plans are very close to solutions of Π and it is relatively easy to construct a solution of Π once we have an extensible public plan. Hence our algorithms will aim at finding extensible public plans. The following formally defines public plan extensibility.

Definition 4 *Let* σ *be a public plan of* Π*.*

$$\begin{aligned}
\sigma \textit{ is } &\text{extensible} \quad \textit{iff} \quad \exists \pi \in \mathsf{sols}(\Pi) : \pi \triangleright \star = \sigma \\
\sigma \textit{ is } &\alpha\text{-extensible} \quad \textit{iff} \quad \exists \pi \in \mathsf{sols}(\Pi \triangleright \alpha) : \pi \triangleright \star = \sigma
\end{aligned}$$

⊠

Example 7 *In our example we can see that* $\pi_0 \triangleright \star$ *is extensible because it was constructed from the solution of* Π*. For the same reason we see that* $\pi_1 \triangleright \star$ *is* `Plane`*-extensible and* $\pi_2 \triangleright \star$ *is* `Truck`*-extensible. It is easy to see that* $\pi_2 \triangleright \star$ *is also* `Plane`*-extensible. However,* $\pi_1 \triangleright \star$ *is not* `Truck`*-extensible because* `Truck` *needs to execute other public actions prior to* `unload(truck, ostrava)`*.* ⊠

The following proposition states the correctness of the multi-agent planning with external actions. It establishes the relationship between extensible and α-extensible

plans. Its direct consequence is that to find a solution of Π it is enough to find a local solution $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ which is β-extensible for every other agent β. A constructive proof follows.

Theorem 1 ((Tožička et al., 2014)) *Public plan σ of Π is extensible if and only if σ is α-extensible for every agent $\alpha \in \mathsf{agents}(\Pi)$.*

Proof. *($\Rightarrow$). When σ is extensible then there is $\pi \in \mathsf{sols}(\Pi)$ such that $\pi \triangleright \star = \sigma$. Let α be arbitrary but fixed. Let us construct plan π_α of $\Pi \triangleright \alpha$ from π by removing internal actions of agents other than α, and by applying projection to the remaining actions obtaining $\pi_\alpha = \langle a \triangleright \alpha : a \in \pi$ and $a \in \mathsf{pub\text{-}actions}(\Pi) \cup \mathsf{int\text{-}actions}(\alpha)\rangle$. Clearly $\pi_\alpha \triangleright \star = \sigma$ because π_α preserves the order of public actions. To prove $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ we first observe that no action b internal for $\beta \neq \alpha$ can change state s of Π in a way observable by α, that is, $\gamma(s, b) \triangleright \alpha = s \triangleright \alpha$. Hence the sequence of states (of Π) which proves $\pi \in \mathsf{sols}(\Pi)$ can be easily transformed to a sequence of states of $(\Pi \triangleright \alpha)$ which proves $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$. Thus σ is α-extensible.*

($\Leftarrow$) For every agent α_i, σ is α_i-extensible and thus there is some local solution π_i such that $\pi_i \in \mathsf{sols}(\Pi \triangleright \alpha_i)$ and $\pi_i \triangleright \star = \sigma$. When more than one local solutions exist, we can choose an arbitrary from them. Now we construct a solution π of Π from local solutions π_i's as follows. We split each π_i at the positions of public actions from σ and we join the corresponding internal parts of different plans together. Internal actions of different agents cannot interact through a shared fact (otherwise this fact would be public and these actions would be public too) and thus we can join different internal parts in any order, preserving only the order of actions of individual agents. Then we construct π of Π from σ by translating ids from σ to corresponding actions of Π and by adding the joined parts between corresponding public actions in σ. Clearly $\pi \triangleright \star = \sigma$ and $\pi \in \mathsf{sols}(\Pi)$. Hence σ is extensible. □

Example 8 *We have seen previously that $\pi_2 \triangleright \star$ is `Truck`-extensible and also `Plane`-extensible. Hence we know that there is some solution of Π even without knowing π_0. Furthermore the proof of Theorem 1 shows how to reconstruct the solution. On the other hand, we know that $\pi_1 \triangleright \star$ is not `Truck`-extensible and thus $\pi_1 \triangleright \star$ is not extensible.* ⊠

4.2 Generic Planner

Theoretic foundations described in previous section directly provide the main idea of our new planner. We can reformulate Theorem 1 to a functional description of the planner, as stated by Corollary 2.

Corollary 2 *Given* Π*, it holds that*

$$\mathsf{sols}(\Pi) \triangleright \star = \bigcap_{\alpha \in \mathsf{agents}(\Pi)} (\mathsf{sols}(\Pi \triangleright \alpha) \triangleright \star)$$

Obviously, the generation of the whole set $\mathsf{sols}(\Pi \triangleright \alpha)$ is impractical for an effective planner. Nevertheless, the agents can try to calculate the intersection using any subset of $\mathsf{sols}(\Pi \triangleright \alpha)$. If the intersection is nonempty, it contains a solution of Π. Otherwise, agents can add more local solutions and try again, until the whole $\mathsf{sols}(\Pi \triangleright \alpha)$ is provided by each agent. This generic algorithm is listed in Algorithm 1.

Algorithm 1: Generic algorithm showing planning by plan set intersection.

```
1  Function GenericMAPlanner(Π ▷ α) is
2      Φ_α ← ∅;
3      loop
4          π ← generate new solution of Π ▷ α;
5          Φ_α ← Φ_α ∪ {π ▷ ⋆};
6          exchange Φ_α with other agents;
7          Φ ← ⋂_{β∈agents(Π)} Φ_β;
8          if Φ ≠ ∅ then
9              return Φ;
10         end
11     end
12 end
```

The basic idea behind the multi-agent planning algorithm (Algorithm 1) described in this thesis is based on Corollary 2 and it can be described briefly as follows. Every agent α from the MA-STRIPS problem Π executes its own planning loop perhaps on a different machine. It keeps generating new solutions of its local planning problem $\Pi \triangleright \alpha$ and announces their public projections to all other agents. Hence the set Γ^*_α of public plans generated so far by α is known by all the agents. Once there is

a single public plan σ generated by all the agents, we can stop the algorithm yielding σ as the public solution of Π. This is because every plan generated by agent β is automatically β-extensible and hence σ is extensible by Theorem 1.

Algorithm `GenericMAPlanner()`, executed by each agent α, starts with an empty set of plans Φ_α. Then, it sequentially generates new solutions π of local problem $\Pi \triangleright \alpha$ and adds their public projections to the plan set. After adding new plan, the plan sets are exchanged between the agents and they compute the intersection. Alternatively, one trusted agent can be chosen to compute the intersection to avoid repeating computations. Once there is a single public plan $\pi \triangleright \star$ generated by all the agents, the intersection of agents' plan sets is nonempty, and thus we can stop the algorithm yielding Φ as a set containing some solutions of Π. Algorithm `GenericMAPlanner()` is sound and complete as stated by Theorem 3.

Theorem 3 (Completeness and Soundness) *Let Π be an* MA-STRIPS *planning problem such that $\mathsf{sols}(\Pi) \neq \emptyset$. Let Φ be a result of* `GenericMAPlanner`$(\Pi \triangleright \alpha)$ *for an arbitrary agent α. Then $\Phi \neq \emptyset$ and $\Phi \subseteq \mathsf{sols}(\Pi) \triangleright \star$. Moreover, if the underlying planner (1) is complete, (2) optimal (with respect to length of generated plan) and (3) allows to generate different plans, then the algorithm always terminates.*

Proof. *Let π be a solution of Π. As a result of completeness and optimality and of the underlying planner, agent α will generate, in the worst case, all the solutions of $\Pi \triangleright \alpha$ up to the length of π. These must include a solution with the public projection $\pi \triangleright \star$. As this hold for every agent α, the intersection of their respective public plan sets must be non-empty and the algorithm terminates. This ensures algorithm completeness and termination. The soundness of the algorithm directly follows from the properties of intersection, Theorem 1.* □

This generic algorithm provides the main structure of a planner. Nevertheless, it is necessary to fill in two white places in order to allow its implementation. Firstly, in all but the most trivial domain, the $\mathsf{sols}(\Pi \triangleright \alpha)$ is a countably infinite set and thus it is necessary to design an effective data structure for representing it, such that it allows an effective implementation of required operations on it. We propose to use *Planning State Machines* defined and discussed in Section 4.3. Secondly, algorithm `GenericMAPlanner()` requires an underlying classical planner, which is complete, optimal, and which allows to sequentially generate different solutions of a given

problem. While first two requirements are met by most of the state-of-the-art planners, the latter one demands a change in the planner functionality. We describe how we modified the FastDownward planner in Section 5.3 in Chapter 5 devoted to the planner implementation.

4.3 Planning State Machines (PSM)

In this section, we utilize finite state machines to effectively represent sets of plans (or public plans) of a STRIPS problem mentioned in the above algorithm description. These finite state machines, which we call *planning state machines* (PSM), are described in Section 4.3.1. PSMs allow us to effectively implement operations which are crucial for our multi-agent planning algorithm. These operations are (1) adding a new solution to an existing PSM (Section 4.3.2), (2) computing a public projection of a PSM (Section 4.3.3), and (3) intersecting public projections of PSMs (Section 4.3.4).

4.3.1 Basics of Planning State Machines

Finite state machines (Hopcroft, Motwani, and Ullman, 2006) are widely used in computer science for manifold purposes. In this section, we utilize state machines to recognize and compute solutions of STRIPS and MA-STRIPS planning problems. To achieve this we use the set of planning actions A as an alphabet while planning states (sets of facts) become states of our *planning state machine* (PSM). PSM state-transitions δ simply resembles planning state progression function γ. Hence a PSM accepts words over A, that is, plans.

For our purposes, a deterministic finite state machine (DFS) is defined as a tuple $\langle \Sigma, S, s_0, \delta, F \rangle$ where Σ is a finite alphabet, S is a finite set of states, $s_0 \in S$ is an initial state, δ is a complete state-transition function ($\delta : S \times \Sigma \to S$), and $F \subseteq S$ is a set of accepting states. A non-deterministic finite state machine (NFS) is a tuple $\langle \Sigma, S, s_0, \delta, F \rangle$ much like a DFS but the state-transition function is non-deterministic, that is, $\delta : S \times \Sigma \to \mathcal{P}(S)$. For a DFS (for an NFS respectively), the state-transition function δ can be naturally extended to $\delta^\star : S \times \Sigma^\star \to S$ (respectively to $\delta^\star : S \times \Sigma^\star \to \mathcal{P}(S)$) where $\Sigma^\star$ is the set of all finite words over alphabet Σ. A word $w \in \Sigma^\star$ is *accepted* by a DFS when $\delta^\star(s_0, w) \in F$. A word $w \in \Sigma^\star$ is *accepted* by an NFS when $\delta^\star(s_0, w) \cap F \neq \emptyset$.

Definition 5 *A* planning state machine (PSM) *of a* STRIPS *problem* $\Pi = \langle P, A, I, G\rangle$ *is a DFS* $\Gamma = \langle \Sigma, S, I, \delta, F\rangle$ *where*

(1) alphabet Σ *is the set action ids* $\Sigma = \{\mathsf{id}(a) : a \in A\}$,

(2) states are sets of facts ($S \subseteq \mathcal{P}(P)$) *with* $I \in S$,

(3) transitions satisfy that $\delta(s, \mathsf{id}(a)) = s'$ *implies* $\gamma(s, a) = s'$,

(4) and accepting states are $F = \{s \in S : G \subseteq s\}$.

Let $\mathsf{accept}(\Gamma)$ *denote the set of all plans accepted by* Γ. ⊠

In general, a PSM does not need to contain all possible planning states of Π because S is only required to be a subset of $\mathcal{P}(P)$ which contains I. However, the following *soundness* result can be trivially proved for any PSM.

Lemma 4 *Let* Γ *be a PSM of a* STRIPS *problem* Π. *Then* $\mathsf{accept}(\Gamma) \subseteq \mathsf{sols}(\Pi)$.

Proof. *Follows directly from Definition 5 and definition of* γ. □

The opposite inclusion does not necessarily hold. However, for a given STRIPS problem Π we can easily construct a *complete* PSM Γ which accepts all the solutions of Π. A complete PSM needs to contain all the possible transitions and all (reachable) planning states. A complete PSM can be constructed by a breath-first search starting from the initial state and by adding all reachable planning states together with all possible transitions. Of course this construction is highly ineffective but it shall be used below to demonstrate basic operations on PSMs. The following defines a complete PSM.

Definition 6 *A PSM* $\Gamma = \langle \Sigma, S, I, \delta, F\rangle$ *of* $\Pi = \langle P, A, I, G\rangle$ *is* complete *when*

(1) $S = \mathcal{P}(P)$, *and*

(2) transitions additionally satisfy that $\gamma(s, a) = s'$ *implies* $\delta(s, \mathsf{id}(a)) = s'$ *whenever* $\gamma(s, a)$ *is defined.* ⊠

Hence a complete PSM accepts every solution of Π, formally as follows.

Lemma 5 *For a complete PSM* Γ *of* Π *it holds that* $\mathsf{accept}(\Gamma) = \mathsf{sols}(\Pi)$.

Proof. *Follows directly from Lemma 4 and the definition of* γ. □

4.3.2 Extending a PSM with Solutions

The first operation we define on PSMs is extending an existing PSM with a new solution π. The operation is denoted $\Gamma \oplus \pi$ and its result is an extended PSM which accepts all the plans as Γ and additionally π. The operation is implemented simply by traversing π and by adding corresponding states and transitions to Γ. The following constructive definition suggests an implementation linear in the size of the added solution. Note that in the definition we consider δ to be a set of triples, writing $\langle s, a, s' \rangle \in \delta$ instead of $s' \in \delta(s, a)$.

Definition 7 *Let a PSM $\Gamma = \langle \Sigma, S, I, \delta, F \rangle$ of Π and a solution $\pi = \langle a_1, \ldots, a_n \rangle$ of Π be given. Denote $s_0 = I$ and $s_i = \gamma(s_{i-1}, a_{i-1})$ for $0 < i \leq n$. The PSM $\Gamma \oplus \pi$ is defined as follows.*

$$\Gamma \oplus \pi = \langle \Sigma, S \cup \{s_0, \ldots, s_n\}, I, \delta \cup \{\langle s_{i-1}, \mathsf{id}(a_{i-1}), s_i \rangle : 0 < i \leq n\}, F \cup \{s_n\} \rangle$$

⊠

The operation $\oplus$ can extend the set of accepted plans by more than π. However, the following lemma states that the PSM $\Gamma \oplus \pi$ accepts all the plans as Γ, and additionally other plans including π. Additionally accepted plans other than π do not cause any problem because Lemma 4 ensures that every additionally accepted plan is a solution of Π. Important is that $\mathsf{accept}(\Gamma \oplus \pi) \subseteq \mathsf{sols}(\Pi)$ holds.

Lemma 6 *Let Π be a classical* STRIPS *problem, let Γ be a PSM of Π, and let $\pi \in \mathsf{sols}(\Pi)$. Then $\Gamma \oplus \pi$ is correctly defined and $\mathsf{accept}(\Gamma) \cup \{\pi\} \subseteq \mathsf{accept}(\Gamma \oplus \pi)$.*

Proof. *Let us prove that $\Gamma \oplus \pi$ is correctly defined as specified by Definition 5. Properties (1)-(3) are trivial. Property (4) is satisfied because $\pi \in \mathsf{sols}(\Pi)$ and hence $G \subseteq s_n$ where s_n is the last state from Definition 7. It follows from Definition 7 that both PSMs are defined on the same alphabet, and that Γ is a sub-automaton of $\Gamma \oplus \pi$. Hence clearly $\mathsf{accept}(\Gamma) \subseteq \mathsf{accept}(\Gamma \oplus \pi)$. Moreover the sequence of states $s_0, \ldots, s_n$ from Definition 7 proves that $\pi \in \mathsf{accept}(\Gamma \oplus \pi)$. Hence the claim.* □

4.3.3 Public Planning State Machines

Previous sections define a planning state machine Γ which effectively represents the set $\mathsf{accept}(\Gamma)$ of plans. From Γ, we would like to compute the corresponding set of

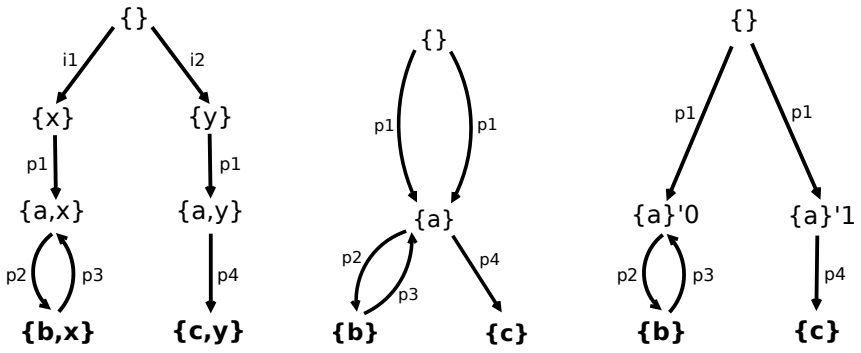
public plans, that is, the set $\mathsf{accept}(\Gamma)\triangleright\star = \{\pi\triangleright\star : \pi \in \mathsf{accept}(\Gamma)\}$. In this section we achieve this by transforming PSM Γ to a *public planning state machine* which (1) accepts exactly the aforementioned set of public plans and (2) contains only public information. We call this operation the *public projection of PSM* Γ and we denote it $\Gamma\triangleright\star$.

Public PSMs will be exchanged among agents during our multi-agent planning algorithm. Therefore, out of privacy concerns, it is essential that public PSMs contain only public information. A first attempt to construct a public PSM from PSM Γ would be to treat internal actions as ε-transitions and eliminate them from Γ using standard algorithm. The standard algorithm to eliminate ε-closures simply "bridges" ε-transitions with new transitions. As for internal facts contained within states, a first attempt is simply to delete them. Let us consider the example PSM Γ_1 from Figure 4.1 (left). After eliminating internal transitions and after deleting internal facts from states we obtain the PSM Γ_2 (Figure 4.1, middle). Unfortunately, Γ_2 also accepts the plan $\langle \mathsf{p1}, \mathsf{p2}, \mathsf{p3}, \mathsf{p4}\rangle$ which is not a public projection of any plan accepted by Γ_1. The problem is that two different states of Γ_1, namely $\{\mathsf{a}, \mathsf{x}\}$ and $\{\mathsf{a}, \mathsf{y}\}$, were merged in Γ_2 after removing internal facts x and y. To solve this problem we introduce integer marks to distinguish states which would otherwise became equal after removing internal facts. This is demonstrated by PSM Γ_3 (Figure 4.1, right). It is easy to check that Γ_3 accepts exactly public projections of the plans accepted by Γ_1. Also note that Γ_3 is non-deterministic because of the non-deterministic transitions from the initial state. Hence public projection can introduce non-determinism.

In order to formally define public PSMs we need to define *public projections of states* and *actions*. The public projection $F\triangleright\star$ of a set of facts F is simply the restriction of F to public facts. The public projection $a\triangleright\star$ of action a restricts facts in a to public facts preserving action id.

Definition 8 *Let* Π *be an* MA-STRIPS *problem. Let* F *be an arbitrary set* $F \subseteq \mathsf{facts}(\Pi)$ *and let* a *be an action from* Π*. The* public projection $F\triangleright\star$ of F*, and the* public projection $a\triangleright\star$ of *action* a *are defined as follows.*

$$\begin{aligned} F\triangleright\star &= F \cap \mathsf{pub\text{-}facts}(\Pi) \\ a\triangleright\star &= \langle \mathsf{id}(a), \mathsf{pre}(a)\triangleright\star,\ \mathsf{add}(a)\triangleright\star,\ \mathsf{del}(a)\triangleright\star\rangle \end{aligned}$$



(Γ_1) (Γ_2) (Γ_3)

FIGURE 4.1: A motivation example for computing PSM public projection. We suppose a context where pn are public and in internal actions, and where a, b, c are public and x, y internal facts. Accepting states are marked bold. The initial state is $\{\}$.

Public projection is extended to sets of actions element-wise. ⊠

The previous discussion explained why public PSMs need to contain integer-labeled states and why public PSMs need to be non-deterministic. Hence a public PSM of an MA-STRIPS problem Π is an NFS with the following properties.

Definition 9 *A* public PSM *of an* MA-STRIPS *problem* Π *is an NFS* $\Gamma^* = \langle \Sigma, S, I_0, \delta, F \rangle$ *where*

(1) the alphabet is $\Sigma = \{\mathsf{id}(a) : a \in \mathsf{pub\text{-}actions}(\Pi)\}$,

(2) states are integer-labeled sets of public facts ($S \subseteq \mathcal{P}(\mathsf{pub\text{-}facts}(\Pi)) \times \mathbb{N}$*),*

(3) the initial state is $I_0 = \langle \mathsf{init}(\Pi) \triangleright \star, 0 \rangle$ *and* $I_0 \in S$,

(4) transitions satisfy that $\langle s', i' \rangle \in \delta(\langle s, i \rangle, \mathsf{id}(a))$ *implies* $\gamma(s, a \triangleright \star) = s'$,

(5) and $\langle s, i \rangle \in F$ *implies* $\mathsf{goal}(\Pi) \subseteq s$.

Let $\mathsf{accept}(\Gamma^*)$ *denote the set of all plans accepted by* Γ^*. ⊠

Now we describe the public projection algorithm to compute $\Gamma \triangleright \star$ from Γ. It is motivated by the standard ε-elimination algorithm (Hopcroft, Motwani, and Ullman, 2006, Chapter 2.5) extended with integer-mark introduction and public projection of states. For every state s, the *internal closure set* $\mathsf{int\text{-}closure}_\Gamma(s)$ contains all the states reachable from s by internal transitions only. The set $\mathsf{int\text{-}closure}_\Gamma(s)$ can be computed by a DFS and the following definition gives its semantics. We omit the index Γ when no confusion can arise.

Algorithm 2: Algorithm to compute the public projection $\Gamma \triangleright \star$ of PSM Γ.

```
1  Function PublicProjection(Γ) is
2      ⟨Σ, S, I, δ, F⟩ ← Γ;
3      Σ0 ← {id(a) : a ∈ pub-actions(Π)};
4      I0 ← ⟨I ▷ ⋆, 0⟩;
5      S0 ← {I0};
6      ρ ← ∅;                    // initialize state renaming, ρ : S → P(pub-facts(Π)) × ℕ
7      ρ(I) ← I0;                                          // set value of ρ(I) to I0
8      foreach s ∈ (S \ {I}) do
9          ρ(s) ← ⟨s ▷ ⋆, |S0|⟩;                    // |S0| increases with every iteration
10         S0 ← S0 ∪ ρ(s);
11     end
12     δ0 ← ∅;                    // initialize new transitions, δ0 : S0 × Σ0 → P(S0)
13     F0 ← ∅;
14     foreach s ∈ S do                                   // for every original state of Γ
15         {r1, ..., rk} ← int-closure(s);
16         foreach id ∈ Σ0 do
17             δ0(ρ(s), id) ← {ρ(δ(ri, id)) : 0 < i ≤ k};
18         end
19         if int-closure(s) ∩ F ≠ ∅ then
20             F0 ← F0 ∪ {ρ(s)};                        // mark ρ(s) as an accepting state
21         end
22     end
23     Γ* ← ⟨Σ0, S0, I0, δ0, F0⟩;
24     return Γ*;
25 end
```

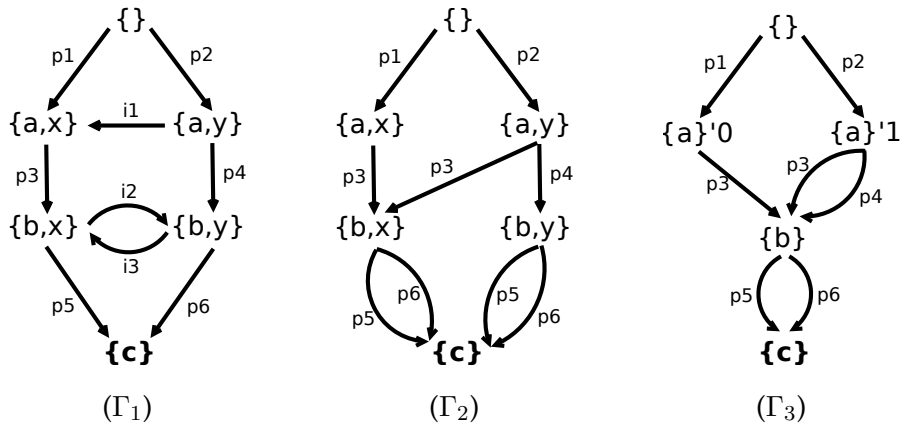
Definition 10 *Given PSM Γ of agent local problem $\Pi \triangleright \alpha$, an* internal closure *of s_0, denoted* int-closure(s_0)*, is the least set of states such that*

(1) $s_0 \in$ int-closure(s_0)*, and*

(2) whenever $s \in$ int-closure(s_0) *for some s then for all* $a \in$ int-actions(α) *it holds that* $\delta(s, \mathsf{id}(a)) \in$ int-closure(s_0).

In other words, the set int-closure(s_0) *contains s_0 and all the states reachable from s_0 by transitions corresponding to internal actions.* ⊠

Once internal closures are computed for every state of Γ, the public projection algorithm proceeds as described by Algorithm 2. The role of the state renaming ρ is to translate states of Γ to states of the public projection. For every state s of Γ, the renaming defines the state $\rho(s)$ in the constructed public PSM consisting of the public projection of s and a unique integer mark. The second foreach cycle which starts at line 14 takes care of "bridging" of internal transitions. When state s' is



(Γ_1) (Γ_2) (Γ_3)

FIGURE 4.2: Example of computing PSM public projection. We suppose a context where pn are public and in internal actions, and where a, b, c are public and x, y internal facts. Accepting states are marked bold. The initial state is $\{\}$.

reachable from s in Γ by (zero or more) internal transitions followed by one public transition $\mathsf{id}(a)$, then $\Gamma \triangleright \star$ will contain a transition from $\rho(s)$ to $\rho(s')$ labeled with $\mathsf{id}(a)$. Finally the condition at line 19 marks accepting states.

The PSM public projection algorithm can be alternatively explained on the example from Figure 4.2. PSM Γ_1 (Figure 4.2, left) is the input PSM. PSM Γ_2 (Figure 4.2, middle) is obtained from Γ_1 by eliminating internal transitions. PSM Γ_3 (Figure 4.2, right) is obtained from Γ_2 by public projection of states and by marks introduction. Note that Γ_3 additionally compresses Γ_2 by unifying states with equal public projection which has equal sets of outgoing transitions. This is an optimization implemented in our planner but omitted from formal presentation. Another optimization is to remove states unreachable from the initial state and to remove states from which no accepting state is reachable. None of the above optimizations affects the semantics.

The following definition defines $\Gamma \triangleright \star$ as the result of Algorithm 2 and formally states algorithm correctness.

Definition 11 *Let $\Pi \triangleright \alpha$ be a local problem of agent α and let Γ be a PSM of $\Pi \triangleright \alpha$. The* public projection *of Γ, denoted $\Gamma \triangleright \star$, is the result of Algorithm 2.* ⊠

Lemma 7 *Let $\Pi \triangleright \alpha$ be a local problem of agent α and let Γ be a PSM of $\Pi \triangleright \alpha$. Then $\Gamma \triangleright \star$ is a public PSM of Π and $\mathsf{accept}(\Gamma \triangleright \star) = \mathsf{accept}(\Gamma) \triangleright \star$.*

Proof. *Let $\Gamma^* = \Gamma \triangleright \star$ and let δ_Γ be the transition function of Γ. Let us prove the inclusions ($\subseteq$) and ($\supseteq$) separately.*

($\subseteq$) Let $\sigma \in \mathsf{accept}(\Gamma^*)$ and let $\sigma = \langle id_1, \ldots, id_n \rangle$. Let $s_0, \ldots, s_n$ be the sequence of states of Γ^* which proves $\sigma \in \mathsf{accept}(\Gamma^*)$. Now we can sequentially process these actions and construct a sequence of action ids π' such that $\pi' \in \mathsf{accept}(\Gamma)$ and $\pi' \triangleright \star = \sigma$ as follows. Thanks to the integer labels, we can unambiguously translate every state of Γ^* to the state of Γ using function ρ^{-1}. Let $t_i = \rho^{-1}(s_i)$ for $0 \leq i \leq n$. We start with empty π'. We know that the transition from s_{i-1} to s_i labeled by id_i has been added to Γ^* by line 17 of Algorithm 4. Hence there is state r of Γ such that $r \in \mathsf{int\text{-}closure}(t_{i-1})$ and $\delta_\Gamma(r, id_i) = t_i$. Hence there has to be a (possibly empty) sequence of internal action ids $\langle id'_1, \ldots, id'_l \rangle$ which proves $r \in \mathsf{int\text{-}closure}(t_{i-1})$. We simply append $\langle id'_1, \ldots, id'_l, id_i \rangle$ to π'. In this way, we construct π' by sequential processing of all action ids from σ. We know that s_0 is the initial state of Γ^* and also that t_0 is the initial state of Γ. It holds that $\pi' \triangleright \star = \sigma$ because all the actions from σ were added to π' in the right order and the additionally added actions are internal. To prove the claim it is now enough to check that $\pi' \in \mathsf{accept}(\Gamma)$. When t_n is an accepting state of Γ, we are done. Otherwise, s_n is marked as an accepting state of Γ^* by line 19 and therefore there exists some accepting state r' of Γ such that $t_n \in \mathsf{int\text{-}closure}(r')$. Finally, we append internal action ids which prove $t_n \in \mathsf{int\text{-}closure}(r')$ to π'. Thus $\pi' \in \mathsf{accept}(\Gamma)$ and hence the claim.

($\supseteq$) Let $\pi \in \mathsf{accept}(\Gamma)$ and let $\sigma = \pi \triangleright \star$. We simulate the plan $\pi = \langle id_1, \ldots, id_n \rangle$ in the state space of Γ. We shall show that this simulation directly corresponds to the simulation of σ in Γ^*. Let $s_0, \ldots, s_n$ be the sequence of states of Γ which proves $\pi \in \mathsf{accept}(\Gamma)$. Clearly the initial state of Γ (that is, s_0) is translated by ρ to the initial state of Γ^*. For a transition from s_{i-1} to s_i labeled by id_i in Γ we distinguish two following two cases. Either (1) id_i is public or (2) internal. If id_i is public, it is trivially added by line 17 to Γ^* and thus we can follow the corresponding transition in Γ^*. If id_i is internal, we find the first transition with public action $\delta(s_{j-1}, id_j) \to s_j, j > i$. Note that all internal actions are removed when doing a public projection. Thus, we can proceed similarly to the previous case having virtual transition from $\delta(s_{i-1}, id_i) \to s_j$ with the only difference that now the needed transition is added because $s_j \in \mathsf{int\text{-}closure}(s)$. It can happen that no such index j exists, i.e., the plan π ends with a sequence of internal actions. In that case, the state $\rho(s_{i-1})$ is going to be added to the goal states at line 19. □

4.3.4 Intersection of Public PSMs

The previous section describes how to compute the public projection of a PSM. Suppose we have two public PSMs of an MA-STRIPS problem Π. This section describes how to compute an intersection of two public PSMs which is a public PSM which accepts the plans accepted by both the original PSMs.

There is a standard algorithm (Hopcroft, Motwani, and Ullman, 2006, Theorem 4.8) to compute an intersection of two arbitrary NFSs. The standard algorithm defines an intersection of NFS Γ_1^* and NSF Γ_2^* as a new NFS whose set of states is the Cartesian product of states of Γ_1^* and Γ_2^*. The intersection of NFSs contains a transition between two states when there are corresponding transitions in both the original NFSs Γ_1^* and Γ_2^*. This standard algorithm, however, needs to be adjusted because the standard algorithm applied to public PSMs would not yield a correctly defined public PSM. The reason is that the structure of states in a public PSM is fixed.

We want to compute an intersection of two public PSMs (of the same MA-STRIPS problem Π). We can take advantage of the fact that both public PSMs are defined on the same set of states. Moreover a transition from state $\langle s, i\rangle$ labeled by action a uniquely determines s' in the destination state $\langle s', i'\rangle$. Hence we do not need to define the set of states in an intersection as a Cartesian product but we can use integer-labeled public states and only adjust integer marks appropriately. Thus an intersection of two public PSMs will be a public PSM. To combine integer marks we can use arbitrary but fixed injective function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ such that $0 \cdot 0 = 0$. A classical example is the Cantor pairing function[1].

The following defines intersection of public PSMs Γ_1^* and Γ_2^*, and proves its correctness. The set of states of an intersection PSM is constructed using the Cantor pairing function as follows. Whenever there is a state $\langle s, i\rangle$ in Γ_1^* and also a state $\langle s, j\rangle$ in Γ_2^* then the intersection PSM contains the state $\langle s, i \cdot j\rangle$. Hence every state of the intersection PSM corresponds to uniquely determined states in Γ_1^* and Γ_2^*. The state transition function of an intersection PSM emulates transition functions of both input public PSMs. A state in the intersection PSM is accepting when both the corresponding states are accepting in Γ_1^* and Γ_2^*. Finally note that the Cantor pairing function is not commutative and thus the intersection operation is commutative up

[1] $i \cdot j = \frac{(i+j)(i+j+1)}{2} + j$

to the integer marks. Nevertheless, all possible resulting public PSMs are equal with respect to the set of accepted plans.

Definition 12 *Let $\Gamma_1^* = \langle \Sigma, S_1, I, \delta_1, F_1 \rangle$ and $\Gamma_2^* = \langle \Sigma, S_2, I, \delta_2, F_2 \rangle$ be two public PSMs of an* MA-STRIPS *problem Π. Let $\cdot$ be the Cantor pairing function. The* intersection *of Γ_1^* and Γ_2^* is a public PSM $\Gamma_0^* = \langle \Sigma, S_0, I, \delta_0, F_0 \rangle$ of problem Π where*

(1) $S_0 = \{\langle s, i \cdot j \rangle : \langle s, i \rangle \in S_1$ and $\langle s, j \rangle \in S_2\}$, and

(2) $\langle s', i' \cdot j' \rangle \in \delta_0(\langle s, i \cdot j \rangle, id)$ iff $\langle s', i' \rangle \in \delta_1(\langle s, i \rangle, id)$ and $\langle s', j' \rangle \in \delta_2(\langle s, j \rangle, id)$,

(3) and $F_0 = \{\langle s, i \cdot j \rangle : \langle s, i \rangle \in F_1$ and $\langle s, j \rangle \in F_2\}$.

The intersection of Γ_1^* and Γ_2^* is denoted $\Gamma_1^* \cap \Gamma_2^*$. ⊠

Lemma 8 *The intersection $\Gamma_1^* \cap \Gamma_2^*$ of two public PSMs Γ_1^* and Γ_2^* of Π is a correctly defined public PSM of Π and the following holds.*

$$\mathsf{accept}(\Gamma_1^* \cap \Gamma_2^*) = \mathsf{accept}(\Gamma_1^*) \cap \mathsf{accept}(\Gamma_2^*)$$

Proof. *Let $\Gamma_1^* = \langle \Sigma, S_1, I, \delta_1, F_1 \rangle$ and $\Gamma_2^* = \langle \Sigma, S_2, I, \delta_2, F_2 \rangle$. Let $\Gamma_0^* = \Gamma_1^* \cap \Gamma_2^*$. Let us first prove that the $\Gamma_1^* \cap \Gamma_2^*$ is a correctly defined PSM of* MA-STRIPS *problem Π as specified in Definition 9. Properties (1) to (3) are trivially fulfilled. Property (4) is proved by Definition 12 (2) and property (5) by Definition 12 (3). Now let us prove the inclusions ($\subseteq$) and ($\supseteq$) separately.*

($\subseteq$) Let $\sigma = \langle id_1, \ldots, id_n \rangle$ be a public plan such that $\sigma \in \mathsf{accept}(\Gamma_0^*)$. Let $\langle s_0, l_0 \rangle, \ldots, \langle s_n, l_n \rangle$ be the sequence of states which proves $\sigma \in \mathsf{accept}(\Gamma_0^*)$. Thanks to the distinctiveness property of an injective function $\cdot$ we can find i_k and j_k such that $l_k = i_k \cdot j_k$ for every $0 \leq k \leq n$. It holds that $\langle s_n, i_n \rangle \in F_1$ and $\langle s_n, j_n \rangle \in F_2$ by Definition 12 (3). Hence the sequence of states $\langle s_0, i_0 \rangle, \ldots, \langle s_n, i_n \rangle$ proves that $\sigma \in \mathsf{accept}(\Gamma_1^*)$ by Definition 12 (2). Similarly $\sigma \in \mathsf{accept}(\Gamma_2^*)$. Hence the claim.

($\supseteq$) Let $\Gamma_0^* = \langle \Sigma, S_0, I, \delta_1, F_0 \rangle$. Let $\sigma = \langle id_1, \ldots, id_n \rangle$ be a public plan such that $\sigma \in \mathsf{accept}(\Gamma_1^*) \cap \mathsf{accept}(\Gamma_2^*)$ Let $\langle s_0, i_0 \rangle, \ldots, \langle s_n, i_n \rangle$ be the sequence of states which proves $\sigma \in \mathsf{accept}(\Gamma_1^*)$ and let $\langle q_0, j_0 \rangle, \ldots, \langle q_n, j_n \rangle$ be the sequence of states which proves $\sigma \in \mathsf{accept}(\Gamma_2^*)$. We know that $\langle s_0, i_0 \rangle = \langle q_0, j_0 \rangle = I$. Hence it is easy to check by Definition 9 (4) that $s_k = q_k$ for all $0 \leq k \leq n$. Also we know that $\langle s_n, i_n \rangle \in F_1$ and $\langle s_n, j_n \rangle = \langle q_n, j_n \rangle \in F_2$. Hence $\langle s_n, i_n \cdot j_n \rangle \in F_0$ by Definition 12 (3). Now the sequence of

states $\langle s_0, i_0 \cdot j_0 \rangle, \ldots, \langle s_n, i_n \cdot j_n \rangle$ *proves that* $\sigma \in \mathsf{accept}(\Gamma_0^*)$ *by Definition 12 (2). Hence the claim.* □

4.4 Multi-agent Planning with Complete PSMs

The results from the previous sections now make it easy to introduce Algorithm 3 to compute all public solutions of a given MA-STRIPS problem Π. For every agent α, the algorithm computes the complete PSM of $\Pi \triangleright \alpha$ and its public projection. All the public PSMs are then intersected and their intersection is returned as a result. Theorem 9 states that the intersection contains exactly all public solutions of Π.

Algorithm 3: Multi-agent planning algorithm with complete PSMs

```
1 Function PsmPlanComplete(Π) is
2     foreach α ∈ agents(Π) do
3         Γ_α ← complete PSM of Π ▷ α;          // BFS or DFS search
4         Γ*_α ← Γ_α ▷ ⋆;
5     end
6     return ⋂_{α∈agents(Π)} Γ*_α;
7 end
```

Theorem 9 *Let* Π *be an* MA-STRIPS *problem and* $\Gamma^* = \texttt{PsmPlanComplete}(\Pi)$. *It holds that* $\mathsf{accept}(\Gamma^*) = \mathsf{sols}(\Pi) \triangleright \star$.

Proof. *Let* Γ_α *denote the complete PSM of* $\Pi \triangleright \alpha$ *and* $\Gamma_\alpha^* = \Gamma_\alpha \triangleright \star$. *Let us prove the inclusions* ($\subseteq$) *and* ($\supseteq$) *separately.*

($\subseteq$) *Let* $\sigma \in \mathsf{accept}(\Gamma^*)$. *Then* $\sigma \in \mathsf{accept}(\Gamma_\alpha^*)$ *for every* α *by Lemma 8. Hence for every* α *there is* $\pi_\alpha \in \mathsf{accept}(\Gamma_\alpha)$ *such that* $\sigma = \pi_\alpha \triangleright \star$ *by Lemma 7. By Lemma 5* $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ *and thus* σ *is* α*-extensible for every* α. *Hence* σ *is extensible by Theorem 1.*

($\supseteq$) *Let* $\pi \in \mathsf{sols}(\Pi)$ *and* $\sigma = \pi \triangleright \star$. *Hence* σ *is extensible and thus also* α*-extensible for every* α *by Theorem 1. Hence for every* α *there is* $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ *with* $\pi_\alpha \triangleright \star = \sigma$. *Clearly* $\pi_\alpha \in \mathsf{accept}(\Gamma_\alpha)$ *by Lemma 5 and thus* $\sigma \in \mathsf{accept}(\Gamma_\alpha^*)$ *by Lemma 7. Thus* $\sigma \in \mathsf{accept}(\Gamma_\alpha^*)$ *for all* α *and hence the claim by Lemma 8.* □

Example 9 *In this example we demonstrate complete PSMs, public projection, and PSM's intersection on our running Example 1. The complete PSM of agent* `Truck` *is presented in Figure 4.3. The* black node *represents the initial state and the* gray nodes *are goal states.*

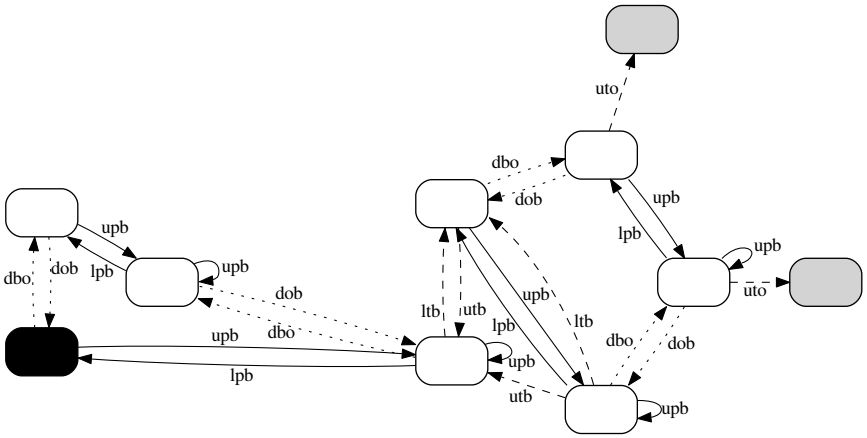


FIGURE 4.3: The complete PSM of agent `Truck` (Example 9).

Dotted edges *represent internal actions,* dashed edges *public actions, and* solid edges *external actions. To improve clarity, we shorten action labels by the first letters of involved objects, for example, the action* `fly(prague,brno)` *is shortened as "fpb", and so on. We also remove edges outgoing goal states and we omit state labels which can be easily filled in using state progression function* γ.

The complete PSM of agent `Plane` *is too large for presentation (containing 32 states and 72 transitions). However, its public projection is shown together with the public projection of* `Truck`*'s public PSM in Figure 4.4. Note how public projection decreases number of states and transitions. The intersection of both public PSMs is shown in Figure 4.5. Note that the intersection PSM represents infinite set of all possible public solutions by a finite structure.*

⊠

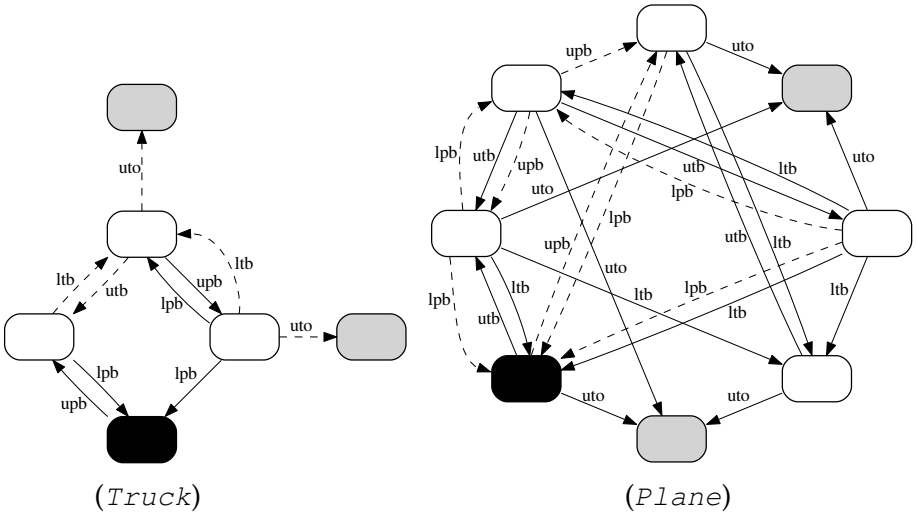


FIGURE 4.4: Public projections of complete PSMs of agents from Example 9.

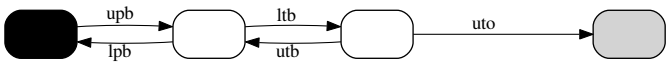


FIGURE 4.5: Intersection of public PSMs from Figure 4.4 representing all possible public solutions.

Chapter 5

PSM Planner

"Details emerge more clearly as the fractal curve is redrawn."

Michael Crichton, Jurassic Park

This chapter describes the main contribution of this thesis. We describe how to use *Planning State Machines* together with Algorithm 1 to effectively solve multi-agent MA-STRIPS problems. Section 4.4 already introduced planning algorithm which is correct and complete, and its advantage is that it computes all the public solutions of a given MA-STRIPS problem. However, its time and space complexity renders it unusable for more complex problems. In this section, we extend the iterative Algorithm 1 using PSMs for plan set representation and provide several practical improvements to make it usable in practice.

The multi-agent planning algorithm is described in Algorithm 4. This algorithm also outlines the content of this chapter. It starts by reduction and sharing some internal dependencies between public actions at Line 2. This optional step is described in Section 5.1 and algorithm containing this extension is called PSM-D. Algorithm continues by computing relaxed solution of the problem at Line 3 and incorporates this solution into its local problem to direct the first generation of local solution. This optional step is described in Section 5.2 and algorithm containing this extension is called PSM-R.

Then the algorithm continues by the main loop (lines 6–17). One execution this loop is called one *iteration* of the algorithm. By a new plan in the first step inside the loop (Line 7), we mean a plan that was not generated in any of the previous loop

Algorithm 4: Distributed multi-agent planning algorithm.

```
1  Function PsmPlanDistributed(Π ▷ α) is
2      reduce and share dependency graphs ;                    // Section 5.1
3      π‘ ← (relaxed solution of Π) ▷ ⋆ ;                       // Section 5.2
4      incorporate π‘ into local Π ▷ α;
5      Γ_α ← empty PSM;
6      loop
7          generate new π_α ∈ sols(Π ▷ α);                     // Section 5.3
8          Γ_α ← Γ_α ⊕ π_α;
9          Γ*_α ← Γ_α ▷ ⋆;
10         announce public PSM Γ*_α to other agents;
11         receive/update public PSMs of other agents;
12         Γ* ← ⋂_{β∈agents(Π)} Γ*_β;        // intersection of public PSMs of all agents
13         if accept(Γ*) ≠ ∅ then
14             return Γ*;
15         end
16         incorporate other agents Γ*_β into local Π ▷ α;     // Section 5.4
17     end
18 end
```

iterations. To achieve this we propose a technique based on known diverse planning techniques. Details are provided in Section 5.3. The variable Γ_α keeps a PSM representing all the plans generated so far. The new plan π_α is added to Γ_α and public PSM Γ^*_α is computed using Algorithm 2. Then all public PSMs are exchanged among the agents. In our implementation, this is synchronization step when the executing agent might need to wait for other agents to finish their computations of public PSMs. However, an alternative non-blocking implementation where the executing agent only updates public PSMs currently available is also possible. Then the intersection Γ^* of public PSMs of all the agents is computed and possibly returned as a result. The intersection is computed by every agent to avoid a centralized component. In the last step of the loop, public PSMs of other agents are incorporated into the local planning problem $\Pi \triangleright \alpha$ using landmarks and action costs. This step, described in details in Section 5.4, is optional and can be skipped. An optional improvement of this mechanism based on plan verification, is described in Section 5.4.1 and called PSM-V. Section 5.5 mentions several implementation problems that we think worth to mention.

As we already mentioned in previous paragraphs, Algorithm 4 contains three optional improvements: PSM-D allowing to reduce and share internal dependencies,

PSM-R creating relaxed solution to direct the search in the first iteration, and PSM-V providing a plan verification during directing next searches. Obviously, these extensions can be combined and then their names combine intuitively. PSM-VRD then represents the ultimate planner containing all three extensions. Experimental evaluation of each extension and the whole planner are provided in Chapter 6.

Let us conclude the introduction of this chapter by Theorem 10, which states the soundness and completeness of Algorithm 4.

Theorem 10 (Completeness and Soundness) *Let* Π *be an* MA-STRIPS *problem such that* $\mathsf{sols}(\Pi) \neq \emptyset$. *Let* Γ^* *be a result of* $\texttt{PsmPlanDistributed}(\Pi \triangleright \alpha)$ *for an arbitrary agent* α. *Then* $\mathsf{accept}(\Gamma^*) \neq \emptyset$ *and* $\mathsf{accept}(\Gamma^*) \subseteq \mathsf{sols}(\Pi) \triangleright \star$. *Moreover, if the underlying planner (1) is complete, (2) optimal (with respect to length of generated plan) and (3) allows to generate different plans, then the algorithm always terminates.*

Proof. *The proof of this theorem directly follows the proof of Theorem 3, while considering the properties of PSM stated by Lemmas 4–8.* □

5.1 Internal Dependencies of Actions

One of the benefits of planning with external actions is that every agent can plan separately its local problem which involves planning of actions for other agents (external actions). Other agents can then only verify whether a plan generated by another agent is α-extensible for them. A disadvantage of this approach is that agents have only a limited knowledge about external actions because internal facts are removed by public projection. Thus it can happen that an agent plans external actions inappropriately in a way that the resulting public plan is not α-extensible for some agent α.

We try to overcome the limitation of partial information about external actions by quipping agents with additional information about external actions without revealing internal facts. Section 5.1.1 describes *dependency graphs* which are used as a formal ground for our analysis of public and external actions. This allows to define *publicly equivalent problems* and *internally independent problems* in Section 5.1.2 and *simply dependent problems* in Section 5.1.3. Section 5.1.4 provides a set of reductions

allowing to simplify possibly complex dependency graphs. Finally, Section 5.1.5 describes how Algorithm 4 can be extended by planning with dependency graphs. We conclude this section by analysis of privacy leakage during planning with dependency graphs in Section 5.1.6.

5.1.1 Dependency Graphs

Local planning problem $\Pi \triangleright \alpha$ of agent α contains information about external actions provided by the set $\mathsf{ext\text{-}actions}(\alpha)$. The idea is to equip agent α with more information described by a suitable structure. A *dependency graph* is a structure we use to encapsulate information about public actions which an agent shares with other agents.

Dependency graphs are known from literature (Jonsson and Bäckström, 1998; Chrpa, 2010). In our context, a *dependency graph* Δ is a bipartite directed graph defined as follows.

Definition 13 *A* dependency graph Δ *is a bipartite directed graph whose nodes are actions and facts. We write* $\mathsf{actions}(\Delta)$ *and* $\mathsf{facts}(\Delta)$ *to denote action and fact nodes respectively. Given the nodes, graph* Δ *contains the following three kinds of edges.*

$$
\begin{aligned}
(a \to f) \in \Delta \textit{ iff } f \in \mathsf{add}(a) &\qquad \textit{(}a \textit{ produces } f\textit{)} \\
(f \to a) \in \Delta \textit{ iff } f \in \mathsf{pre}(a) \setminus \mathsf{del}(a) &\qquad \textit{(}a \textit{ requires } f\textit{)} \\
(f \dashrightarrow a) \in \Delta \textit{ iff } f \in \mathsf{pre}(a) \cap \mathsf{del}(a) &\qquad \textit{(}a \textit{ consumes } f\textit{)}
\end{aligned}
$$

Additionally, a fact can be marked as initial in Δ*. The set of states marked as initial is denoted* $\mathsf{init}(\Delta)$.

Hence edges of a dependency graph Δ are uniquely determined by the set of nodes. Note that action nodes are themselves actions, that is, triples of fact sets. These action nodes can contain additional facts other than fact nodes $\mathsf{facts}(\Delta)$. We use dependency graphs to represent internal dependencies of public actions. Dependencies determined by public facts are known to other agents and thus we do not need them in the graph as fact nodes. From now on we suppose that $\mathsf{facts}(\Delta)$ contains no public facts as fact nodes. Action nodes, however, can contain public facts in their public actions.

Definition 14 *Let an* MA-STRIPS *problem* Π *be given. The* minimal dependency graph $\mathsf{MD}(\alpha)$ *of agent* $\alpha \in \mathsf{agents}(\Pi)$ *is the dependency graph uniquely determined by the following set of nodes.*

$$\begin{aligned} \mathsf{actions}(\mathsf{MD}(\alpha)) &= \mathsf{pub\text{-}actions}(\Pi) \\ \mathsf{facts}(\mathsf{MD}(\alpha)) &= \emptyset \\ \mathsf{init}(\mathsf{MD}(\alpha)) &= \emptyset \end{aligned}$$

Hence $\mathsf{MD}(\alpha)$ has no edges as there are no fact nodes. Thus the graph contains only separated public action nodes. Furthermore, the set $\mathsf{ext\text{-}actions}(\alpha)$ of external actions of agent α can be trivially expressed as follows.

$$\mathsf{ext\text{-}actions}(\alpha) = \bigcup_{\beta \neq \alpha} (\mathsf{actions}(\mathsf{MD}(\beta)) \triangleright \star)$$

Thus we see that dependency graphs can carry the same information as provided by $\mathsf{ext\text{-}actions}(\alpha)$.

Definition 15 *The* full dependency graph $\mathsf{FD}(\alpha)$ *of agent* α *contains all the actions of* α *and all the internal facts of* α*.*

$$\begin{aligned} \mathsf{actions}(\mathsf{FD}(\alpha)) &= \alpha \\ \mathsf{facts}(\mathsf{FD}(\alpha)) &= \mathsf{int\text{-}facts}(\alpha) \\ \mathsf{init}(\mathsf{FD}(\alpha)) &= \mathsf{init}(\Pi) \cap \mathsf{int\text{-}facts}(\alpha) \end{aligned}$$

Hence $\mathsf{FD}(\alpha)$ contains all the information known by α. By publishing $\mathsf{FD}(\alpha)$, an agent reveals all his internal knowledge which might be a potential privacy risk. On the other hand, other agents are by $\mathsf{FD}(\alpha)$ provided the most precise information about dependencies of public actions of α. Every plan of another agent, computed with $\mathsf{FD}(\alpha)$ in mind, is automatically α-extensible. Thus we see that dependency graphs can carry dependencies information with a varied precision.

Dependency Graph Collections

A dependency graph represents information about public actions of one agent. Every agent needs to know information from all the other agents. We use *dependency*

graph collections to represent all the required information. A *dependency graph collection* $\mathcal{D}$ of an MA-STRIPS problem Π is a set of dependency graphs which contains exactly one dependency graph for every agent of Π. We write $\mathcal{D}(\alpha)$ to denote the graph of α. We write $\mathsf{actions}(\mathcal{D})$, $\mathsf{facts}(\mathcal{D})$, and $\mathsf{init}(\mathcal{D})$ to denote in turn all the action, fact, and initial fact nodes from all the graphs in $\mathcal{D}$.

Definition 16 *Given problem* Π*, we can define the* minimal collection $\mathsf{MD}(\Pi)$ *and the* full collection $\mathsf{FD}(\Pi)$ *as follows.*

$$\begin{aligned} \mathsf{MD}(\Pi) &= \{\mathsf{MD}(\alpha) : \alpha \in \mathsf{agents}(\Pi)\} \\ \mathsf{FD}(\Pi) &= \{\mathsf{FD}(\alpha) : \alpha \in \mathsf{agents}(\Pi)\} \end{aligned}$$

Later we shall show some interesting properties of the minimal and full collections.

Local Problems and Dependency Collections

In order to define local problems informed by $\mathcal{D}$, we need to define facts and action projections which preserve information from $\mathcal{D}$. We use symbol $\triangleright_{\mathcal{D}}$ to denote *projections accordingly to* $\mathcal{D}$. Recall that the public projection $a \triangleright \star$ of action a is the restriction of the facts of a to $\mathsf{pub\text{-}facts}(\Pi)$. The *public projection* $a \triangleright_{\mathcal{D}} \star$ *of action* a *accordingly to* $\mathcal{D}$ is the restriction of the facts of a to $\mathsf{pub\text{-}facts}(\Pi) \cup \mathsf{facts}(\mathcal{D})$. Public projection is extended to sets of actions element-wise. Furthermore, *external actions of* α *according to* $\mathcal{D}$, denoted $\overline{\mathsf{ext\text{-}actions}}_{\mathcal{D}}(\alpha)$, contain public projections (according to $\mathcal{D}$) of actions of other agents. In other words, $\overline{\mathsf{ext\text{-}actions}}_{\mathcal{D}}(\alpha)$ carries all the information published by other agents for agent α. It is computed as follows.

$$\overline{\mathsf{ext\text{-}actions}}_{\mathcal{D}}(\alpha) = \bigcup_{\beta \neq \alpha} (\mathsf{actions}(\mathcal{D}(\beta)) \triangleright_{\mathcal{D}} \star)$$

This equation describes distributed computation of $\overline{\mathsf{ext\text{-}actions}}_{\mathcal{D}}(\alpha)$ where every other agent β separately computes published actions, applies public projection, and sends the result to α.

In order to define a local planning problem of agent α which would take information from $\mathcal{D}$ into consideration, we need to extract from $\mathcal{D}$ facts and initial facts of other agents. Below we define sets $\overline{\mathsf{facts}}_{\mathcal{D}}(\alpha)$ and $\overline{\mathsf{init}}_{\mathcal{D}}(\alpha)$ which contain those facts and initial facts published by other agents, that is, all the facts from $\mathcal{D}$ except of the

facts of α.

$$\begin{aligned} \overline{\mathsf{facts}}_{\mathcal{D}}(\alpha) &= \mathsf{facts}(\mathcal{D}) \setminus \mathsf{facts}(\mathcal{D}(\alpha)) \\ \overline{\mathsf{init}}_{\mathcal{D}}(\alpha) &= \mathsf{init}(\mathcal{D}) \setminus \mathsf{init}(\mathcal{D}(\alpha)) \end{aligned}$$

Now we are ready to define *local planning problems according to* $\mathcal{D}$ which extends local planning problems by the information contained in $\mathcal{D}$.

Definition 17 *Let* Π *be* MA-STRIPS *problem. The* local problem $\Pi \rhd_{\mathcal{D}} \alpha$ of agent $\alpha \in \mathsf{agents}(\Pi)$ accordingly to $\mathcal{D}$ *is the classical* STRIPS *problem* $\Pi \rhd_{\mathcal{D}} \alpha = \langle P_0, A_0, I_0, G_0 \rangle$ *where*

(1) $P_0 = \mathsf{facts}(\Pi \rhd \alpha) \cup \overline{\mathsf{facts}}_{\mathcal{D}}(\alpha)$,

(2) $A_0 = \alpha \cup \overline{\mathsf{ext\text{-}actions}}_{\mathcal{D}}(\alpha)$,

(3) $I_0 = \mathsf{init}(\Pi \rhd \alpha) \cup \overline{\mathsf{init}}_{\mathcal{D}}(\alpha)$, *and*

(4) $G_0 = \mathsf{goal}(\Pi)$.

We can see that a local problem $\Pi \rhd_{\mathcal{D}} \alpha$ according to $\mathcal{D}$ extends the local problem $\Pi \rhd \alpha$ by the facts and actions published by $\mathcal{D}$.

Let us consider following two boundary cases of dependency collections: $\mathsf{MD}(\Pi)$ and $\mathsf{FD}(\Pi)$. Given an MA-STRIPS problem Π, we can construct local problems using the minimal dependency collection $\mathsf{MD}(\Pi)$. It is easy to see that $\Pi \rhd_{\mathsf{MD}(\Pi)} \alpha = \Pi \rhd \alpha$ for every agent α. With the full dependency collection $\mathsf{FD}(\Pi)$ we obtain equal projections, that is, $\Pi \rhd_{\mathsf{FD}(\Pi)} \alpha = \Pi \rhd_{\mathsf{FD}(\Pi)} \beta$ for all agents α and β. Moreover, local solutions equal MA-STRIPS solutions, that is, $\mathsf{sols}(\Pi \rhd_{\mathsf{FD}(\Pi)} \alpha) = \mathsf{sols}(\Pi)$ for every α.

5.1.2 Publicly Equivalent Problems

We have seen that dependency collections can provide information about internal dependencies with a varied precision. Given two different collections, two different local problems can be constructed for every agent. However, when the two local problems of the same agent equal on public solutions, we can say that they are equivalent because their public solutions are equally extensible.

In order to define equivalent collections, we first define public equivalence on planning problems. Two planning problems Π_0 and Π_1 are *publicly equivalent*, denoted $\Pi_0 \simeq \Pi_1$, when they have equal public solutions. Formally as follows.

$$\Pi_0 \simeq \Pi_1 \quad \Leftrightarrow \quad \mathsf{sols}(\Pi_0) \rhd \star = \mathsf{sols}(\Pi_1) \rhd \star$$

Public equivalence can be extended to dependency graph collections as follows. Two collections $\mathcal{D}_0$ and $\mathcal{D}_1$ of the same MA-STRIPS problem Π are *equivalent*, written $\mathcal{D}_0 \simeq \mathcal{D}_1$, when for any agent α, it holds that the local problems $\Pi \triangleright_{\mathcal{D}_0} \alpha$ and $\Pi \triangleright_{\mathcal{D}_1} \alpha$ are publicly equivalent. Formally as follows.

$$\mathcal{D}_0 \simeq \mathcal{D}_1 \quad \Leftrightarrow \quad (\Pi \triangleright_{\mathcal{D}_0} \alpha) \simeq (\Pi \triangleright_{\mathcal{D}_1} \alpha) \text{ (for all } \alpha)$$

Example 10 *Given an* MA-STRIPS *problem* Π*, with the full dependency collection* $\mathsf{FD}(\Pi)$ *we can see that* $\Pi \simeq \Pi \triangleright_{\mathsf{FD}(\Pi)} \alpha$ *holds for any agent. Hence to find a public solution of* Π *it is enough to solve the local problem (accordingly to* $\mathsf{FD}(\Pi)$*) of an arbitrary agent. The same holds for any dependency collection* $\mathcal{D}$ *such that* $\mathcal{D} \simeq \mathsf{FD}(\Pi)$*. Note that* $\mathcal{D}$ *can be much smaller and provide less private information than the full dependency collection.* ⊠

The above definitions allow us to recognize problems without any internal dependencies which we can define as follow.

Definition 18 *An* MA-STRIPS *problem* Π *is* internally independent *when*

$$\mathsf{MD}(\Pi) \simeq \mathsf{FD}(\Pi)$$

In order to solve an internally independent problem, it is enough to solve the local problem $\Pi \triangleright \alpha$ of an arbitrary agent. Any local public solution is extensible which makes internally independent problems easier to solve because there is no need for interaction and negotiation among the agents. Later we shall show how to algorithmically recognize internally independent problems. The following formally captures the above properties.

Lemma 11 *Let* Π *be an internally independent* MA-STRIPS *problem. Then* $(\Pi \triangleright \alpha) \simeq \Pi$*.*

Proof. $(\Pi \triangleright \alpha) \simeq (\Pi \triangleright_{\mathsf{MD}(\Pi)} \alpha) \simeq (\Pi \triangleright_{\mathsf{FD}(\Pi)} \alpha) \simeq \Pi$ □

5.1.3 Simple Action Dependencies

Let us consider dependency collections without internal actions, that is, collections $\mathcal{D}$ where $\mathsf{actions}(\mathcal{D})$ contains no internal actions. When $\mathcal{D}$ is published, then no agent publishes actions additional to $\mathsf{ext\text{-}actions}(\alpha)$ which is desirable out of privacy concerns. Furthermore, the plan search space of $\Pi \triangleright_{\mathcal{D}} \alpha$ is not increased when

compared to $\Pi \triangleright \alpha$. Even more, every additionally published fact in $\mathcal{D}$ providing a valid dependency prunes the search space. Action dependencies captured by collections without internal actions can be expressed by requirements on the order of actions in a plan. This further abstracts the published information providing privacy protection. Thus it seems reasonable to publish dependency collections without internal actions.

Simply Dependent Problems

The following defines simply dependent MA-STRIPS problems, where internal dependencies of public actions can be expressed by a dependency collection free of internal actions.

Definition 19 *An* MA-STRIPS *problem* Π *is* simply dependent *when there exists* $\mathcal{D}$ *such that* $\mathsf{actions}(\mathcal{D})$ *contains no internal actions and* $\mathcal{D} \simeq \mathsf{FD}(\Pi)$.

Suppose we have a simply dependent MA-STRIPS problem and a dependency collection $\mathcal{D}$ which proves the fact. In order to solve Π, once again, it is enough to solve only one local problem $\Pi \triangleright_{\mathcal{D}} \alpha$ (of an arbitrary agent α).

Lemma 12 *Let* Π *be a simply dependent* MA-STRIPS *problem. Let* $\mathcal{D}$ *be a dependency collection which proves that* Π *is simply dependent. Then* $(\Pi \triangleright_{\mathcal{D}} \alpha) \simeq \Pi$ *holds for any agent* $\alpha \in \mathsf{agents}(\Pi)$.

Proof. $(\Pi \triangleright_{\mathcal{D}} \alpha) \simeq (\Pi \triangleright_{\mathsf{FD}(\Pi)} \alpha) \simeq \Pi$ □

The above method requires all the agents to publish the information from $\mathcal{D}$. However, the information does not need to be published to all the agents as it is enough to select one *trusted* agent and send the information only to him. Hence it is enough for all the agents to agree on a single trusted agent.

5.1.4 Dependency Graph Reductions

Recognizing simply dependent MA-STRIPS problems might be difficult in general. That is why we define an approximative method which can provably recognize some simply dependent problems. We define a set of reduction operations on dependency graphs and we prove that the operations preserve relation $\simeq$. Then we apply the

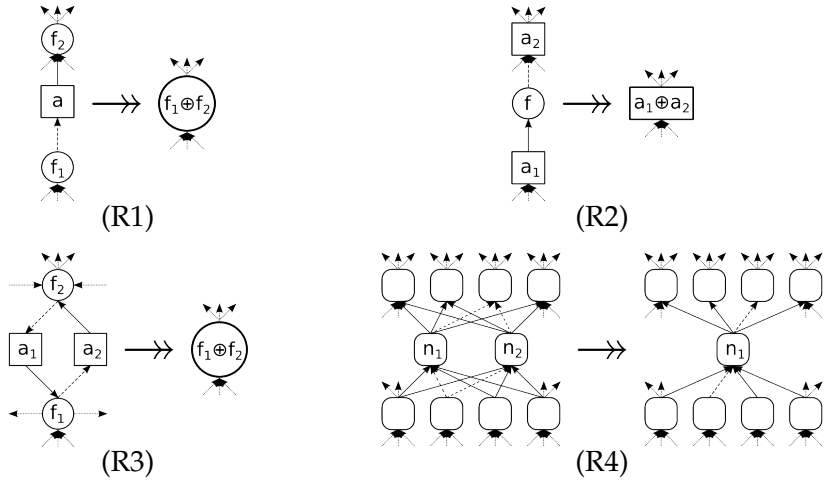


FIGURE 5.1: Graphical illustration of reduction operations (R1)–(R4). Circles represent fact nodes and rectangles represent action nodes. Rounded boxes in (R4) represent any node (either fact or action node).

reductions repeatedly starting with $\mathsf{FD}(\Delta)$ obtaining a dependency graph which can not be reduced any further. This is done by every agent. When the resulting graphs contain no internal actions, then we know that the problem is simply dependent. Additionally, when the resulting graphs contain no internal facts, then we know that the problem is independent.

Our previous work (Tožička, Jakubův, and Komenda, 2015a) was restricted to problems where $\mathsf{pre}(a) = \mathsf{del}(a)$ holds for every action a. This impractical limitation is removed here. We still restrict our attention to problems where $\mathsf{del}(a) \subseteq \mathsf{pre}(a)$ holds for every action a. This is not considered limiting because a problem not meeting this requirement can be easily transformed to a permissible equivalent problem. All presented benchmarks follow this limitation and thus this translation does not affect presented results.

Finally, to abstract from the set of initial facts of a dependency graph Δ, we introduce to the graph a special *initial action* $\langle \emptyset, \mathsf{init}(\Delta), \emptyset \rangle$. We suppose that every dependency graph has exactly one initial action and hence we do not need to remember the set of initial facts. The initial action is handled as *public* even when it has no public effect. Both definitions of dependency graphs are trivially equivalent but the one with an initial action simplifies the presentation of reduction operations.

We proceed by informal descriptions of dependency graph reductions. The formal definition is given below. The operations are depicted in Figure 5.1.

(R1) Remove Simple Action Dependency. If some internal action has only one delete effect and one add effect and there is no other action depending on the consumed fact (f_1) we can merge both facts into one and remove that action.

(R2) Remove Simple Fact Dependency. If some fact is the only effect of some action and there is only one action that consumes this effect without any side effects, we can remove this fact and merge both actions.

(R3) Remove Small Action Cycle. In many domains, there are reversible internal actions that allow transitions between two (or more) states without any other preconditions. All these states can be merged into a single state and the actions changing them can be omitted.

(R4) Merge Equivalent Nodes. If two nodes (facts or actions) equal on incoming and outgoing edges, then we can merge these two nodes. Mostly this is not present directly in the domain description but this structure might appear when we simplify a dependency graph using other reductions.

(R5) Remove Invariants. After several reduction steps, it can happen that all the delete effects on some fact are removed and the fact is always fulfilled from the initial state. This happens, for example, in *Logistics*, where the location of a vehicle is internal knowledge and can be freely changed as described by reduction (R3). Once these cycles are removed, only one fact remains. The remaining fact represents that the vehicle is *somewhere*, which is always true. This fact can be freely removed from the dependency graph.

In order to formally define the above reductions we first define operator $[F]_{f_1 \to f_2}$ which renames fact f_1 to f_2 in the set of facts $F \subseteq P$.

$$[F]_{f_1 \to f_2} = \begin{cases} F & \text{if } f_1 \notin F \\ (F \setminus \{f_1\}) \cup \{f_2\} & \text{otherwise} \end{cases}$$

Similarly, we define operator $[F]_{\text{-}f} = F \setminus \{f\}$ which removes fact f from the set of facts F. These operators are extended to actions (applying the operator to preconditions, add, and delete effects) and to action sets (element-wise). The operators can be further extended to dependency graphs, where $[\Delta]_{\text{-}f}$ is the dependency graph determined by $[\mathsf{actions}(\Delta)]_{\text{-}f}$ and $[\mathsf{facts}(\Delta)]_{\text{-}f}$. Finally, for two actions a_1 and a_2

we define the merged action $a_1 \oplus a_2$ as the action obtained by unifying separately preconditions, add, and delete effects of both the actions.

The following formally defines *reduction relation* $\Delta_0 \rightarrow \Delta_1$ which holds when Δ_0 can be transformed to Δ_1 using one of the reduction operations.

Definition 20 *The* reduction relation $\Delta_0 \rightarrow \Delta_1$ *on dependency graphs is defined by the following four rules.*

(R1) *Rule (R1) is applicable to* Δ_0 *when*

(1) Δ_0 *contains edges* $(f_1 \dashrightarrow a \rightarrow f_2)$,

(2) a *is internal, and*

(3) there are no other edges from/to a*, and*

(4) there are no other edges from f_1.

Then $\Delta_0 \rightarrow \Delta_1$ *where* Δ_1 *is defined as* $\Delta_1 = [\Delta_0]_{f_1 \rightarrow f_2}$. *The initial action is preserved.*

(R2) *Rule (R2) is applicable to* Δ_0 *when*

(1) Δ_0 *contains edges* $(a_1 \rightarrow f \dashrightarrow a_2)$,

(2) there are no other edges from/to f*, and*

(3) there are no other edges from a_1*, and*

(4) a_2 *has no other delete effects, and*

(5) a_2 *is internal action.*

Then $\Delta_0 \rightarrow \Delta_1$ *where* Δ_1 *is given by the following.*

$$\mathsf{actions}(\Delta_1) = \{[a_1 \oplus a_2]_{-f}\} \cup (\mathsf{actions}(\Delta_0) \setminus \{a_1, a_2\})$$
$$\mathsf{facts}(\Delta_1) = [\mathsf{facts}(\Delta_0)]_{-f}$$

If a_1 *is the initial action of* Δ_0 *then the new merged action becomes the initial action of* Δ_1. *Otherwise, the initial action is preserved.*

(R3) *Rule (R3) is applicable to* Δ_0 *when*

(1) Δ_0 *contains edges* $(f_1 \dashrightarrow a_1 \rightarrow f_2)$*, and*

(2) Δ_0 *contains edges* $(f_2 \dashrightarrow a_2 \rightarrow f_1)$*, and*

(3) a_1 *and* a_2 *are both internal, and*

(4) there are no other edges from/to a_1 *or* a_2*.*

Then $\Delta_0 \to \Delta_1$ *where* Δ_1 *is given by the following.*

$$\begin{aligned} \mathsf{actions}(\Delta_1) &= [\mathsf{actions}(\Delta_0) \setminus \{a_1, a_2\}]_{f_2 \to f_1} \\ \mathsf{facts}(\Delta_1) &= [\mathsf{facts}(\Delta_0)]_{f_2 \to f_1} \end{aligned}$$

The initial action is preserved as it is public.

(R4) *Rule (R4) is applicable to* Δ_0 *when* Δ_0 *contains two nodes* n_1 *and* n_2 *(either action or fact nodes) such that*

(1) nodes n_1 *and* n_2 *have equal sets of incoming and outgoing edges, and*

(2) n_1 *and* n_2 *are not public actions.*

Then $\Delta_0 \to \Delta_1$ *where, in the case* n_1 *and* n_2 *are actions,* Δ_1 *is given by the following.*

$$\begin{aligned} \mathsf{actions}(\Delta_1) &= \{n_1 \oplus n_2\} \cup (\mathsf{actions}(\Delta_0) \setminus \{n_1, n_2\}) \\ \mathsf{facts}(\Delta_1) &= \mathsf{facts}(\Delta_0) \end{aligned}$$

When n_1 *or* n_2 *is the initial action of* Δ_0 *then the new merged action becomes the initial action of* Δ_1*. Otherwise, the initial action is preserved.*

In the case n_1 *and* n_2 *are facts,* $\Delta_1 = [\Delta_0]_{n_2 \to n_1}$*.*

(R5) *Let* a_{init} *be the initial action of* Δ_0*. Rule (R5) is applicable to* Δ_0 *when there exists fact* f *such that*

(1) Δ_0 *contains edge* $(a_{\mathit{init}} \to f)$*, and*

(2) Δ_0 *contains no edge* $(f \dashrightarrow a)$ *for any* a*.*

Then $\Delta_0 \to \Delta_1$ *where* Δ_1 *is defined as* $\Delta_1 = [\Delta_0]_{-f}$*. The initial action of* Δ_1 *is* $[a_{\mathit{init}}]_{-f}$*.*

The following defines *reduction equivalence relation* $\Delta_0 \sim \Delta_1$ as a reflexive, symmetric, and transitive closure of $\to$. In other words, Δ_0 and Δ_1 are *reduction equivalent* when one can be transformed to another using the reduction operations. Dependency collections $\mathcal{D}_0$ and $\mathcal{D}_1$ are *reduction equivalent* when graphs of corresponding agents are reduction equivalent.

Definition 21 *Dependency graphs* reduction equivalence relation, *denoted* $\Delta_0 \sim \Delta_1$, *is the least reflexive, symmetric, and transitive closure generated by the relation* $\rightarrow$.

Given MA-STRIPS *problem* Π, *dependency collections* $\mathcal{D}_0$ *and* $\mathcal{D}_1$ *of* Π *are* reduction equivalent, *written* $\mathcal{D}_0 \sim \mathcal{D}_1$, *when* $\mathcal{D}_0(\alpha) \sim \mathcal{D}_1(\alpha)$ *for any agent* $\alpha \in \mathsf{agents}(\Pi)$.

The following theorem formally states that reduction operations preserves public equivalence.

Theorem 13 *Let* Π *be an* MA-STRIPS *problem and let* $\mathsf{pre}(a) \subseteq \mathsf{del}(a)$ *hold for any internal action. Let* $\mathcal{D}_0$ *and* $\mathcal{D}_1$ *be dependency collections of problem* Π. *Then* $\mathcal{D}_0 \sim \mathcal{D}_1$ *implies* $\mathcal{D}_0 \simeq \mathcal{D}_1$.

Proof. [Proof sketch] It can be shown that none of the reduction operations changes the set of public plans $\mathsf{sols}(\mathcal{D}_0(\alpha)) \triangleright \star$ of any agent $\alpha \in \mathsf{agents}(\Pi)$. Therefore repetitive application of reductions assures that $\mathcal{D}_0 \simeq \mathcal{D}_1$.

To avoid possible action confusion caused by value renaming, we suppose that actions are assigned unique ids which are preserved by the reduction, and that plans are sequences of these ids. □

The consequences of the theorem are discussed in the following section.

Recognizing Simply Dependent Problems

Let us have an MA-STRIPS problem Π where $\mathsf{pre}(a) \subseteq \mathsf{del}(a)$ holds for every internal action a. Suppose that every agent α can reduce its full dependency collection $\mathsf{FD}(\alpha)$ to a state where it contains no internal action. Then there is $\mathcal{D}$ such that $\mathcal{D} \sim \mathsf{FD}(\Pi)$ and hence $\mathcal{D} \simeq \mathsf{FD}(\Pi)$ by Theorem 13. Hence Π is simply dependent and its public solution can be found without agent interaction, provided all the agents allow to publish $\mathcal{D}$. Important idea here is that publicly equivalent dependency graphs do not need to reveal the same amount of sensitive information. Moreover when $\mathcal{D} \sim \mathsf{MD}(\alpha)$ then Π is independent and can be solved without any interaction and without revealing other than public information. This gives us an algorithmic approach to recognize some independent and simply dependent problems.

5.1.5 Planning with Dependency Graphs

This section describes how agents use dependency graphs in order to solve MA-STRIPS problem Π (Algorithm 6). At first, every agent computes the dependency

Algorithm 5: Compute the dependency graph to be published by agent α.

```
1  Function ComputeSharedDG(α) is
2      Δ_0 ← FD(α);
3      loop
4          if ∃Δ_1 : Δ_0 → Δ_1 then
5              Δ_0 ← Δ_1;
6          else
7              break;
8          end
9      end
10     if Δ_0 contains only public actions then
11         return Δ_0;
12     else
13         return MD(α);
14     end
15 end
```

Algorithm 6: Distributed planning with dependency graphs.

```
1 Δ ← ComputeSharedDG(α);
2 send Δ to other agents;
3 construct 𝒟 from other agent's graphs;
4 compute local problem Π ▷_𝒟 α;
5 continue Algorithm 4 with Π ▷_𝒟 α instead of Π ▷ α;
```

graph it is willing to share using function `ComputeSharedDG` described by Algorithm 5. Every agent α starts with the full dependency graph $\mathsf{FD}(\alpha)$ and tries to apply reduction operations repeatedly as long as it is possible. When the resulting reduced dependency graph Δ_0 contains only public actions, then the agent publishes Δ_0. Otherwise, the agent publishes only the minimal dependency graph $\mathsf{MD}(\alpha)$. Algorithm 5 clearly terminates for every input because every reduction decreases the number of nodes in the dependency graph. Hence the algorithm loop (lines 3–9 in Algorithm 5) can not be iterated more than n times when n is the count of nodes in $\mathsf{FD}(\alpha)$. Moreover, every reduction operation can be performed in a time polynomial to the size of the problem, and thus the whole algorithm is polynomial-time.

Once the shared dependency graph Δ is computed, Algorithm 6 continues by sending Δ to other agents. Then shared dependency graphs of other agents are received. This allows every agent to complete the dependency collection $\mathcal{D}$, and to construct the local problem $\Pi \triangleright_{\mathcal{D}} \alpha$. The rest of the planning procedure is the same as in the case of Algorithm 4.

The algorithm can be further simplified when all the agents succeed in reducing $\mathsf{FD}(\alpha)$ to an equivalent dependency graph without internal actions, that is, when Π is provably simply dependent. Then it is enough to select one agent to compute public solution of Π. When some agent α (but not all the agents) succeeds in reducing $\mathsf{FD}(\alpha)$ then every plan created by any other agent will be automatically α-extensible. Therefore, when exactly one agent fails in reducing $\mathsf{FD}(\alpha)$, then this agent can compute the solution of $\Pi \triangleright_{\mathcal{D}} \alpha$, and this solution is guaranteed to be extensible. More generally, when more agents fail, only these agents are required to continue in iterated negotiation as described by Algorithm 4.

5.1.6 Privacy Leakage Analysis

Although distributed planning with the dependency graphs trades exposition of (reduced) private information for efficiency, from perspective of the worst case, there is no extra private knowledge shared. In this section, we will support this claim by analysis of what information is leaked by sharing the reduced dependency graph Δ.

According to (Brafman, 2015), a multi-agent planning algorithm is *strongly privacy preserving* if no agent can deduce any information about private facts and private preconditions/effects of any action, beyond what can be deduced from the public projection of the planning problem and the public projection of the solution plan.

The shared reduced dependency graph Δ can be, in some cases, equivalent to the original dependency graph. In these cases, *renamed* internal facts are shared between the agents. Nevertheless other agents do not know, whether any reduction has been performed, and so they see a dependency graph which can represent an unlimited number of different dependency graphs with an *equivalent reduction*. The definition of strong privacy allows an agent to share any knowledge which can be deduced from own actions description, the public projection of other agents actions, and the public projection of the solution plan. Not much information can be deduced from the solution plan when solving a single problem. In fact, other agents can deduce only the information that given sequence of actions is possible and thus no action deletes preconditions of the immediately following action.

Suppose a situation, when we observe an agent α using a strongly privacy preserving algorithm in long term. The most extreme case is when we know all possible plans in which the agent can participate in all different problems and we also know

all the problems which are unsolvable for this agent, thus which public plans are not α-extensible. We can suppose that all these plans contain all agent's published actions and all their possible combinations. From this knowledge we could deduce what dependencies are between public actions appearing in these plans. Certainly, this deduced information will contain all the information contained in the shared dependency graph Δ. This gives us the first insight about the amount of information which is contained in the shared dependency graph. We can state that it contains information which could be eventually deduced if we follow the agent planning of all possible different planning tasks.

Let us now compare sharing of dependency graph with privacy which an agent leaks during a single run of PSM algorithm without sharing of dependency graphs in the worst case. Suppose that we have two agents α and β with PSM algorithm. During one iteration, both agents update their public PSMs and share it with the other agent to check whether there is a non-empty intersection. In the worst case, it can happen, that the agent α publishes all its possible plans, therefore Γ^*_α is complete and no longer changing, while the agent β still updates his Γ^*_β. Then, agent β can reconstruct all the agent's α dependencies between its public actions. Again, shared dependency graph Δ is certainly a subset of the knowledge β would deduce in this case.

Even though agents are not able to detect the presence of the internal fact from a single execution of the PSM algorithm, the agents can deduce the existence of this fact when cooperating longer time or when it takes long time before the agents find a common solution. For example, suppose that there is no solution at all. In that case, each agent publishes all his possible local solution and then announces that there are no more local solutions. Then, similarly to the previous case, each other agent can see whether some action has to always precede another action. If such case exists, the agent deduce that there is a fact or some other dependency between these actions.

Example 11 *In our running Example 1, agent* Plane *starts to sequentially generate following possible solutions – we show only* Plane*'s own actions and omit the actions planned for* Truck *agent, because only these actions can reveal some knowledge about its internal facts and actions.*

- $\langle\rangle$

FIGURE 5.2: Example of dependency graph containing all knowledge which can leak during *logistics* planning.

- $\langle \texttt{unload(plane,brno)} \rangle$
- $\langle \texttt{unload(plane,brno)}, \texttt{load(plane,brno)} \rangle$
- $\langle \texttt{unload(plane,brno)}, \texttt{load(plane,brno)}, \texttt{unload(plane,brno)} \rangle$
- ...
- $\langle \left(\texttt{unload(plane,brno)}, \texttt{load(plane,brno)} \right)^* \rangle$
- $\langle \left(\texttt{unload(plane,brno)}, \texttt{load(plane,brno)} \right)^*, \texttt{unload(plane,brno)} \rangle$

Even though there is an infinite number of different public plans, they can be compactly represented in finite structure (for example PSM *uses* Planning state machines *– see Section 5.5 for more details).*

From these plans we can observe:

- *all plans start with* `unload(plane,brno)` *action, and*
- *a* `load(plane,brno)` *action has to be between two* `unload(plane,brno)` *actions.*

Moreover, we can observe that, unless an empty plan is generated, there is always action `unload(plane,brno)` *before* `load(plane,brno)`*, nevertheless these actions are already dependent through a public fact* `in(crown,brno)` *and thus it does not imply any internal connection. Similarly, we can observer that between two* `load(plane,brno)` *actions has to be* `unload(plane,brno)` *action.*

These observations can be modeled by a dependency graph (Figure 5.2) containing one internal fact that is also present in the initial state. This fact is then consumed by action `unload(plane,brno)` *and produced by action* `load(plane,brno)`*.* ⊠

We can conclude that the PSM-D algorithm (i.e. PSM extended by sharing dependency graphs) does not publish any more internal information that the PSM algorithm in the worst case. In fact, the most significant difference between these two

algorithms is that PSM-D exposes exactly the internal information that could be exposed by PSM in the worst case and which can be described in the form of simple dependencies between the public actions.

5.2 Initial Relaxed Plan Landmark

The delete effect relaxation, where delete effects of actions are ignored, has proved its relevance both in STRIPS planning (Hoffmann and Nebel, 2001), and recently also in MA-STRIPS planning (Štolba and Komenda, 2014). It is known that to find a solution of a relaxed problem is an easier task than to find a solution of the original problem. There are distributed algorithms to find a solution of a relaxed MA-STRIPS problem using *distributed planning graphs* (Štolba and Komenda, 2013). Effective implementation using *exploration queues* can also be found in literature (Štolba and Komenda, 2014). All these algorithms respect privacy, that is, they do not reveal internal facts and actions to other agents.

We use a relaxed solution of MA-STRIPS problem Π to improve Algorithm 4 as follows. At first we compute some solution π of the relaxation of Π. We compute its public projection $\pi \triangleright \star$ which is a sequence of public action ids. We use this id sequence as an initial landmark in the first loop iteration of Algorithm 4. The sequence is integrated into $\Pi \triangleright \alpha$ in the same way as in Definition 23 (the public projection $\pi \triangleright \star$ can be seen as a public PSM which contains only $\pi \triangleright \star$). The same initial landmark is used by all the agents. When $\pi \triangleright \star$ is extensible then every agent α is likely to generate local solution π_α such that $\pi_\alpha \triangleright \star = \pi \triangleright \star$ in the first iteration. In that case the algorithm terminates directly in the first iteration causing a dramatic speed-up. Otherwise, the initial landmark is forgotten by all the agents and the algorithm continues by the second iteration as before. Practical impact of initial relaxed plan landmarks is experimentally evaluated in Section 6.1.

Example 12 *In our running Example 1, we can find a relaxed solution and compute its public projection. The shortest solution has the following public projection.*

$$\sigma = \langle\, \texttt{unload(plane,brno)},\ \texttt{load(truck,brno)},\ \texttt{unload(truck,ostrava)} \,\rangle$$

We can see that this public plan σ is extensible. When the agents use σ as landmarks in the first iterations, both of them succeed to generate a plan with public projection σ. Hence

the intersection of public PSMs is not empty and the algorithm terminates after the first iteration. ⊠

5.3 Generating New Plans

This section describes how to generate a plan of a classical STRIPS problem which differs from a set of plans provided as an input. This extension is implemented by *diverse planning* techniques. There exists many different approaches to diverse planning (Tožička et al., 2015). For the generation of different plans we take inspiration from *homotopy class constraints* (Bhattacharya, Kumar, and Likhachev, 2010). In our setting, homotopy classes of plans are naturally defined by plan public projections. That is, two plans π_1 and π_2 belong to same homotopy class iff $\pi_1 \triangleright \star = \pi_2 \triangleright \star$.

In our implementation, we have extended the FastDownward planner but the same technique can be used to extend any planner based on a state space search. The technique is based on the idea of augmented graphs (Bhattacharya, Kumar, and Likhachev, 2010). Every state is extended by a vector of numbers where each vector field corresponds to one of the forbidden plans. The i-th vector field value indicates whether the plan ending at this state is different from the i-th forbidden plan. Value -1 indicates that current plan differs while a non-negative number denotes the position in the corresponding forbidden plan. During action application at some state, we check whether the applied action equals the expected action at the next position in the forbidden plan. The initial state corresponds to the vector of zeros. During the state search, a plan is accepted as a solution only when the plan ends in a state with only -1 values.

Algorithm 4 starts with an empty set of forbidden plans. In every iteration, the generated plan is added to this set. This ensures that the algorithm generates a different plan in every iteration.

As an optimization, we use action costs to force the underlying planner to prefer internal actions to public actions, and public actions to external actions. These action costs[1] correspond to the knowledge an agent has about these actions. An agent has full information about its internal actions and as they do not affect other agents they should be used whenever possible. The agent also has full information about its

[1] We have chosen costs 10 for internal actions, 100 for public actions, and 1000 for external action. Nevertheless, the exact values are not important.

public actions but because they can affect other agents they should be used more carefully. Finally, the agent has only a limited information about external actions of other agents because some information can be pruned of by public projection. Hence external actions are the most expensive.

5.4 Guiding Plan Search Using Public PSMs

In every iteration of Algorithm 4, the agent receives public PSMs of all other agents. These PSMs contain information about plans found by other agents. Information in these PSMs can be used to guide a new plan generation so that the algorithm finds a solution faster. We incorporate information from public PSMs into the local planning problem by extending the problem with *soft-landmark* actions and by adjusting *action costs*. This problem extension influences plan search in the desired way. When agent α receives public PSM Γ^*_β of another agent β, we would like the local plan generator to prefer sequences of public actions suggested by Γ^*_β. This is because Algorithm 4 terminates only when all the agents generate the same public solution. Hence it is preferable to find a local solution $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ such that the public projection $\pi_\alpha \triangleright \star$ is contained in $\mathsf{accept}(\Gamma^*_\beta)$. We achieve this by extending $\Pi \triangleright \alpha$ with special landmark actions without affecting the set of solutions of $\Pi \triangleright \alpha$. These landmark actions basically duplicate actions from $\Pi \triangleright \alpha$ but have decreased action costs. The landmark actions have additional preconditions to ensure that landmark actions are used in the order given by Γ^*_β.

The process of extending local problem $\Pi \triangleright \alpha$ with Γ^*_β is sketched as follows. We extend the local problem with a set of fresh facts P_{marks} distinct from facts of $\Pi \triangleright \alpha$ where each fresh fact corresponds to a state of Γ^*_β. Hence we have bijection μ from states of Γ^*_β to P_{marks}. Let Γ^*_β contains a transition from s to s' labeled by action a. We then extend the local problem with a duplicate of a which (1) can be applied only in states where $\mu(s)$ is valid, and (2) additionally transforms fact $\mu(s)$ to $\mu(s')$. In this way landmark actions can be applied only in the order given by Γ^*_β. The following defines a landmark action.

Definition 22 *Let a be an action and let $from$ and to be two facts. The* landmark action $\mathsf{lm\text{-}act}(a, from, to)$ *is defined as follows.*

$$\mathsf{lm\text{-}act}(a, from, to) = \langle \mathsf{id}(a), \mathsf{pre}(a) \cup \{from\}, \mathsf{add}(a) \cup \{to\}, \mathsf{del}(a) \cup \{from\} \rangle$$

For action id id, the landmark action (w.r.t. agent α) is defined as

$$\mathsf{lm\text{-}act}_\alpha(id, from, to) = \mathsf{lm\text{-}act}(a, from, to)$$

where a is the uniquely determined action of $\Pi \triangleright \alpha$ with $\mathsf{id}(a) = id$. ⊠

Once we have added landmark facts and actions we just extend the initial state of the local problem with $\mu(I_0)$ where I_0 is the initial state of Γ^*_β. A complete extension of the local problem is formally defined as follows.

Definition 23 *Let Π be an* MA-STRIPS *problem and $\Gamma^* = \langle \Sigma, S, I', \delta, F \rangle$ a public PSM of Π. Let P_{marks} be a set of facts distinct from $\mathsf{facts}(\Pi)$ such that $|P_{marks}| = |S|$ and let μ be a bijection from S to P_{marks}. For $\alpha \in \mathsf{agents}(\Pi)$, the problem $\Pi \otimes_\alpha \Gamma^*$ is defined as the* STRIPS *problem $\langle P_0, A_0, I_0, G \rangle$ where*

(1) $P_0 = \mathsf{facts}(\Pi \triangleright \alpha) \cup P_{marks}$*, and*

(2) $A_0 = \mathsf{actions}(\Pi \triangleright \alpha) \cup \{\mathsf{lm\text{-}act}_\alpha(id, \mu(s), \mu(s')) : s' \in \delta(s, id)\}$*, and*

(3) $I_0 = \mathsf{init}(\Pi \triangleright \alpha) \cup \{\mu(I')\}$.

The problem $\Pi \otimes_\alpha \Gamma^*$ is called the local problem of α extended with Γ^*. ⊠

Note that described extension does not affect the goal state condition. There is a straightforward relationship between solutions of $\Pi \triangleright \alpha$ and solutions of $\Pi \otimes_\alpha \Gamma^*_\beta$. Clearly every solution of $\Pi \triangleright \alpha$ is a solution of the extended problem $\Pi \otimes_\alpha \Gamma^*_\beta$. On the other hand, every solution of $\Pi \otimes_\alpha \Gamma^*_\beta$ can be translated to the solution of $\Pi \triangleright \alpha$ by replacing landmark actions with original actions of $\Pi \triangleright \alpha$.

The crucial point is that landmark actions are assigned significantly decreased action costs so that the underlying planner prefers landmark actions to original actions. In our implementation the costs are set as follows. Suppose that $\Pi \triangleright \alpha$ is being extended with Γ^*_β. The cost of $\mathsf{lm\text{-}act}(a, from, to)$ is set to 1 if a is owned by α (the receiver of Γ^*_β) or if a is owned by β (the sender). Otherwise the cost 10 is used. The

reasoning behind this choice is that the owner (either α or β) of the action has full information about it and thus it is more likely to be used correctly.

Definition 23 can be easily adjusted so that it allows repeated extension of $\Pi \triangleright \alpha$. During planning, problem $\Pi \triangleright \alpha$ is extended with every $\Gamma^*_{\beta_i}$ received from other agents in the previous loop iteration, one by one. Hence the underlaying planner is launched with the problem

$$\Pi \otimes_\alpha \Gamma^*_{\beta_1} \otimes_\alpha \Gamma^*_{\beta_2} \otimes_\alpha \cdots \otimes_\alpha \Gamma^*_{\beta_m}$$

provided $\otimes_\alpha$ is left-associative. In the first iteration, the local problem $\Pi \triangleright \alpha$ is used without any extension.

Practical experiments revealed that the actions costs and landmark actions described in this section are crucial for a practical usage of Algorithm 4. For experimental evaluation see Section 6.1.

Section 5.4.1 describes a method PSM-V based on plan verification which allows to improve performance of multi-agent planning Algorithm 4.

5.4.1 Plan Verification and Analysis

Distributed PSM planner from Algorithm 4 uses public plans generated by other agents as landmarks to guide future plan search (see Section 5.4). However, it is desirable to use only extensible plans to guide plan search because non-extensible plans can not lead to a non-empty public PSMs intersection. Every generated plan should be verified by other agents in order to determine its extensibility. However, extensibility (or α-extensibility) checking is expensive and thus we propose only an approximative method of plan verification. We have firstly published an extension of a PSM planner with a method of plan verification with promising results in (Jakubův, Tožička, and Komenda, 2015). The rest of this section describes the method.

Firstly, we describe how to approximate α-extensibility of public plan σ. Given a public plan $\sigma = \langle id_1, \ldots, id_n \rangle$ we create a problem $\Pi \circledast_\alpha \sigma$ which is solvable iff σ is α-extensible. We extend the set of facts with fresh facts $P_{marks} = \{m_0, \ldots, m_n\}$. We add the action $\mathsf{lm\text{-}act}(a_i, m_{i-1}, m_i)$ for all $0 < i \leq n$ to the actions of $\Pi \circledast_\alpha \sigma$ (see Definition 22). The initial state of $\Pi \circledast_\alpha \sigma$ is extended with m_0 but, this time, the

last mark fact m_n is added to the goal of $\Pi \circledast_\alpha \sigma$. This ensures that any solution of $\Pi \circledast_\alpha \sigma$ contains all actions from σ in the right order, possibly interleaved with α's internal actions. Hence every solution of $\Pi \circledast_\alpha \sigma$ can be translated to a solution of $\Pi \triangleright \alpha$ (but not necessarily the other way round). The following formalizes construction of $\Pi \circledast_\alpha \sigma$ and proves its correctness.

Definition 24 *Let Π be a* MA-STRIPS *planning problem and $\alpha \in \mathsf{agents}(\Pi)$. Let $\sigma = \langle id_1, \ldots, id_n \rangle$ be a public plan of Π. Let $P_{marks} = \{m_0, \ldots, m_n\}$ be a set of facts distinct from $\mathsf{facts}(\Pi)$. The* α-extensibility check problem *of σ, denoted $\Pi \circledast_\alpha \sigma$, is the* STRIPS *problem $\langle P_{marks} \cup (\mathsf{facts}(\Pi) \triangleright \alpha), A, (\mathsf{init}(\Pi) \triangleright \alpha) \cup \{m_0\}, \mathsf{goal}(\Pi) \cup \{m_n\} \rangle$ where $A = \mathsf{int\text{-}actions}(\alpha) \cup \{\mathsf{lm\text{-}act}_\alpha(id, m_{i-1}, m_i) : 0 < i \leq n\}$.* ⊠

Theorem 14 ((Tožička et al., 2014)) *Let Π be an* MA-STRIPS *problem, let α be an agent of Π, and let σ be a public plan of Π. Then σ is α-extensible iff $\mathsf{sols}(\Pi \circledast_\alpha \sigma) \neq \emptyset$.*

Proof. *Both the implications are proved similarly by replacing public actions with their respective landmark actions ($\Rightarrow$) or the other way round ($\Leftarrow$).*

($\Rightarrow$) Let σ be α-extensible. Hence there is $\pi \in \mathsf{sols}(\Pi \triangleright \alpha)$ such that $\pi \triangleright \star = \sigma$. Clearly π contains only public actions in the order given by σ. The rest are internal actions of α. We construct π_0 from π by replacing public actions by their respective landmark actions. It is easy to verify that $\pi_0 \in \mathsf{sols}(\Pi \circledast_\alpha \sigma)$.

($\Leftarrow$) Let $\pi \in \mathsf{sols}(\Pi \circledast_\alpha \sigma)$. Clearly $\pi \triangleright \star = \sigma$. Let us construct π_0 be translating landmark actions back to their original actions. It still holds $\pi_0 \triangleright \star = \sigma$ and it is easy to check that $\pi_0 \in \mathsf{sols}(\Pi \triangleright \alpha)$. Hence σ is α-extensible. □

The first attempt to use the above construction of $\Pi \circledast_\alpha \sigma$ has been published in (Tožička et al., 2014). It tries to centrally generate public solutions σ and to verify that σ is α-extensible by every agent α. A roadblock of this attempt is that it is relatively hard for agent α to find out that σ is *not* α-extensible. Then $\mathsf{sols}(\Pi \circledast_\alpha \sigma) = \emptyset$ and it usually requires the underlaying planner used to solve $\Pi \circledast_\alpha \sigma$ to traverse the whole search space. That is because the state-of-the-art planners are optimized to find a solution of a given solvable problem and not to determine that the problem is not solvable. That is why we have proposed (Jakubův, Tožička, and Komenda, 2015) an approximate method to determine problem solvability.

Previously proposed approximation of $\Pi \circledast_\alpha \sigma$ solvability (Jakubův, Tožička, and Komenda, 2015) is done using generic process calculi type system scheme POLY✶

(Makholm and Wells, 2005; Jakubův and Wells, 2010). However, the same result can be achieved using *planning graphs* (Ghallab, Nau, and Traverso, 2004, Chapter 6) which we briefly describe here. We construct a complete relaxed planning graph of $\Pi \circledast_\alpha \sigma$, that is, the planning graph of $\Pi \circledast_\alpha \sigma$ with action delete effects removed. A planning graph with k layers can be constructed in time polynomial in k. Then, we examine the last fact layer of the constructed planning graph. Recall that $\Pi \circledast_\alpha \sigma$ contains fresh mark facts $P_{marks} = \{m_0, \ldots, m_n\}$. When mark m_i is valid in some planning state, it means that (1) public actions a_1,...,a_i from σ were already correctly used in the current plan and that (2) the next public action to be used is a_{i+1}. The result of the analysis is the maximum j such that m_j is in the last fact layer of the relaxed planning graph. This resulting j is interpreted as follows. When $j < n$ then clearly $\Pi \circledast_\alpha \sigma$ is unsolvable because m_n is a goal fact. Moreover the result j tells us that there is no way for an agent to follow the public plan σ up to the point where a_{j+1} can be applied. This gives us an approximation of a valid prefix of σ. On the other hand, $j = n$ does not necessarily implies that σ is α-extensible because the proposed method is only an approximation of $\Pi \circledast_\alpha \sigma$ solvability.

The above α-extensibility approximation allows the following plan verification procedure. When agent α generates a new plan π_α, it sends its public projection $\pi_\alpha \triangleright \star$ to all the other agents. Once other agent β receives $\pi_\alpha \triangleright \star$, it runs the above β-extensibility check and sends its result back to agent α (just after line 7 of Algorithm 4). Agent α collects analysis results from all the other agents and computes their minimum l. Plan π_α is then stripped so that only the first l public actions remain in it. This stripped plan is then used to extend PSM Γ_α in Algorithm 4. Hence only the stripped plan is used as a landmark to guide future plan search. In the next iteration, a plan with public projection different from π_α is required to be computed. When $\pi_\alpha \triangleright \star = \langle id_1, \ldots, id_n \rangle$ we can even further speed up convergence of Algorithm 4 by forbidding any plan with public prefix $\langle id_1, \ldots, id_l, id_{l+1} \rangle$ to be generated in the future. This does not affect completeness of Algorithm 4 (Theorem 10) because only provably non-extensible plans are forbidden.

Example 13 *In this example we demonstrate planning with plan analysis on our running Example 1. The* first iteration *is as follows. All the agents use an optimal planner in order to solve their local problems and thus agent* `Plane` *creates the simplest plan where the goal is reached by agent* `Truck` *alone. That is,* `Plane` *generates following plan (see also*

Example 5):

$$\langle \texttt{unload(truck,ostrava)} \rangle.$$

This plan does not pass through the verification process of agent `Truck` *because* `Truck` *needs to execute additional public actions prior to the last goal-reaching action. In the meanwhile, agent* `Truck` *generates a plan with following public projection.*

$$\sigma = \langle \texttt{unload(plane,brno)}, \texttt{ load(truck,brno)}, \texttt{ unload(truck,ostrava)} \rangle$$

This public plan is extensible and thus passes through the verification check. Landmark actions created from σ are added to `Plane`*'s local problem. The intersection of public PSMs after the first iteration is empty because* `Plane`*'s PSM is empty.*

In the second iteration, `Plane` *follows the landmarks from the above public plan σ and it succeeds by generating a plan with the public projection σ. At this point, public PSMs of both the agents contain σ and hence the algorithm terminates after the second iteration independently on the plan generated by* `Truck`*.* ⊠

5.5 Practical STRIPS Extensions

In this section, we describe several interesting problems encountered when implementing a PSM-based planner. Firstly, the theory is based on MA-STRIPS formalism which is based on STRIPS. Just like STRIPS, MA-STRIPS requires *grounded* problem specification which is not appropriate for real world problems. Section 5.5.1 describes how to move towards more convenient PDDL-like planning language which allows a compact representation by introduction of *parametric* actions. Then, in Section 5.5.2, we describe how we handle internal goals without the need to publish them.

5.5.1 From STRIPS to PDDL, and Back Again

STRIPS language is a formal language often used in automated planning theory. STRIPS also provides a formal base for MA-STRIPS. Nevertheless, it is not very practical for real world problems because it supports only grounded representation. Therefore, in practice and in benchmark tests, Planning Domain Definition

Language (PDDL) is typically used. PDDL supports predicates with typed parameters which allow to describe a full range of facts or actions by parametric statements. When we convert our running Example 1 into PDDL language, we can have only one parametric action `drive(`*`from`*`,`*`to`*`)` instead of multiple actions for different locations. But backward grounding of this parametric action can create instances which were not in the original domain. To get rid of these instances, new predicate `isRoad(`*`from`*`,`*`to`*`)` can be introduced. Such a predicate is never part of action effects and thus it is *constant* during execution of any plan. When we ground a PDDL problem, we can evaluate these predicates and we can omit action instances where the predicate evaluates to *false* because these instances can never be used. Using the same definition of public facts for PDDL as we described in MA-STRIPS, these constants would be public, because they appear as precondition of actions of different agents. Nevertheless, it is not necessary to communicate them because their evaluation never changes and thus every agent can has its own copy.

A conversion from PDDL to STRIPS, that is, *grounding*, is needed in two places. Firstly, we use it to compute the size of a problem because number of predicates and actions of the grounded problem better describes real complexity of the problem. More importantly, it is needed to compute MA-STRIPS representation of input problems because FMAP problems, which we use as benchmarks (see Section 6.1), are defined in the PDDL format. Hence input PDDL problem needs to be grounded so that we can use MA-STRIPS definition of fact privacy classification. FMAP benchmark problems define a separate domain and problem PDDL file for every agent. Firstly, we create a single agent problem by merging all agent domains and problems. Secondly, we use grounding algorithm implemented in FastDownward planner[2] to ground it. Then, we take all the facts used in the grounded problem and ground the original agents' problems to these facts.

5.5.2 Internal Goals

So far we have considered MA-STRIPS problems where all goal facts are public. Nevertheless, in some cases, agents, although cooperative, can have different internal goals and agents might not be willing to share these facts with other agents because

[2]See `http://www.fast-downward.org/`. Script `translate.py` creates an SAS representation of an input PDDL problem.

of privacy concerns. This section describes how to transform an MA-STRIPS problem with internal goals to an equivalent problem where the original internal goals can be kept internal.

The transformation extends each agent α with a public *confirmation action* which can be executed when all the goals of α are satisfied. The confirmation actions can be executed only at the end of a plan. The goal of the transformed problem expresses that all the agents confirmed their goals.

More formally, let Π be an MA-STRIPS problem where some or all goals are internal. Firstly, we introduce a fresh fact `planning` with the meaning that planning is in progress, that is, that no confirmation action (of any agent) has been used so far. Fact `planning` shall be initially valid in the initial state of the transformed problem and shall be deleted by the first confirmation action. Every action from Π shall be extended with precondition `planning`. Next we introduce fresh virtual goal facts $\texttt{done}_1, \ldots, \texttt{done}_n$. Fact $\texttt{done}_i$ means that the confirmation action of the i-th agent was used. The confirmation action $\mathsf{confirm}(\alpha, i)$ of agent the i-th agent α can be used when all the goals relevant for α (that is public goals and α's internal goals) are satisfied. Its add-effect is the virtual goal $\texttt{done}_i$ and it deletes `planning`, formally, $\mathsf{confirm}(\alpha, i) = \langle \mathsf{goal}(\Pi) \triangleright \alpha, \{\texttt{done}_i\}, \{\texttt{planning}\}\rangle$. The goal of the transformed MA-STRIPS problem is set to $\{\texttt{done}_1, \ldots, \texttt{done}_n\}$.

There is a direct correspondence between solutions of the original and of the transformed problem. Every solution of the transformed problem can be translated to a solution of the original problem by removing confirmation actions. Moreover the goal facts of the transformed problem can be freely published because they do not carry any confidential information.

Chapter 6

Experiments

"And as the Cherokee walked farther from his mountains,
he began to die."

Forrest Carter, The Education of Little Tree

This chapter describes experimental evaluation of proposed PSM planner (Algorithm 4) with its optional extensions. Variants PSM-R, PSM-V, and their combination PSM-VR are evaluated in Section 6.1. Section 6.2 focuses on different privacy classifications and it shows how the increase of privacy of facts and goals affects performance of our planner. Finally, Section 6.3 evaluates the impact of reduction and sharing of internal dependencies (variant PSM-VRD) using official results of CoDMAP'15 competition.

6.1 PSM-R and PSM-V Experimental Results

We have performed a set of experiments to compare our planners with another state-of-the-art multi-agent planners[1] and also to evaluate the impact of plan verification on planning times. We have decided to compare our planners with FMAP (Torreño, Onaindia, and Sapena, 2014), RDFF (Štolba and Komenda, 2014) and GPPP (Maliah, Shani, and Stern, 2014). All planners are compared on well defined problems taken

[1] All the tests were performed on a single PC, CPU Intel i7 3.40GHz with 8 cores, and memory limited to 8GB RAM.

from International Planning Competition (IPC) problems as published by FMAP authors. FMAP classifies facts as public or internal using a manual selection of public predicate names. On the other hand, RDFF and GPPP use privacy classification as defined by MA-STRIPS. In practice, FMAP public facts are a superset of MA-STRIPS public facts. Nevertheless, our PSM-based algorithms can handle both privacy classifications. In our experiments, we use exactly the same input files as the authors of FMAP used during its evaluation[2], and we also use the same time limit of 30 minutes for each problem.

The above mentioned state-of-the-art multi-agent planners are compared with several variants of our PSM-based planner. Variant PSM is the basic version described by Algorithm 4 in Chapter 5. Then, we have two extensions of this basic algorithm. Variant PSM-R uses initial relaxed plan landmarks described in Section 5.2. Variant PSM-V is the basic algorithm extended with the plan verification as described in Section 5.4.1. Finally, there is PSM-VR which combines both extensions into a solid planner that benefits from of all these features.

The experiments are organized as follows. Firstly, we describe benchmark domains in Section 6.1.1. Then, in Section 6.1.2, we compare the variants of our algorithm and compare it with FMAP planner which had the highest coverage between multi-agent planners at the time when experiments were executed. In Section 6.1.3, we further analyze the communication of the PSM variants and explore its connection with the number of iteration needed to solve a problem.

6.1.1 Benchmark Domains

We have performed experiments on 244 PDDL problems from following 10 domains. Following list of FMAP domains also describes which information is public:

Blocksworld is a multi-agent version of the classical planning problem where each of 4 agents represents one robotic arm that can move and stack blocks. All information in this domain is public.

Depots problems contain two types of agents. *Trucks* agents transport crates between hoists located in *depots*. These *depots* move crates to correct pallets. *Depots* problems contain from 5 to 12 agents. All information is public.

[2]We would like to thank the authors of FMAP for a kind support with their planner.

Driverlog problems contain from 2 to 8 agents representing *drivers* that operate several *trucks* to transport packages to required locations. All information is known by all agents. Nevertheless, some constants are internal (`link` and `path` representing roads and paths connecting different locations).

Elevators contain two types of agents representing *slow* and *fast elevators*. The goal is to transport passengers between floors. *Elevators* problems contain from 3 to 5 agents. Positions of passengers are always public, while elevator positions and the number of passengers in each elevator are internal. Passenger positions are changed by actions `board` and `leave` with natural meaning.

Logistics domain contains two types of agents, *trucks* and *planes*, transporting packages between cities. Loading and unloading of packages is performed by actions `load` and `unload`, respectively. A goal specifies only the final location of packages. A transportation task often requires cooperation of several agents. *Logistics* problems contain from 3 to 10 agents. The location of an agent is internal but the location of a package is public.

Openstacks problems contain a *manager* agent who handles product orders, and *manufacturer* agents who produce these ordered products. This problem is based on minimum maximum simultaneous open stacks combinatorial optimization problem. All information is public including goals specifying that the orders have been shipped.

Rovers problems contain from 1 to 8 *rovers*, each represented by one agent. The goal is to collect samples and communicate acquired data. Every *rover* is capable of fulfilling of an arbitrary goal but an agent has limited resources and thus it is necessary to decide which goal will be fulfilled by which agent. Sample locations and information about whether the data have been communicated is public.

Satellites problems contain from 1 to 12 *satellite* agents taking images in space. The `pointing` of a satellite and whether an image has been taken is public. Both can be included in a goal.

Woodworking domain contains 4 agents representing 7 machines in a production chain. All information is public.

Zenotravel domain contains from 2 to 8 agents representing *planes* with a limited fuel. The goal is to transport passengers between cities but it can also specify positions of some planes. Positions of passengers and planes are public. A fuel level is internal. Thus, all `fly` actions are public and only `refuel` actions are internal.

These problems, published by the authors of FMAP, are inspired by traditional benchmark problems of International Planning Competition (IPC)[3]. In FMAP, the privacy classification is defined as a list of predicates shared with other agent. It is therefore possible to specify that some knowledge is shared between only two agents, for example. However, in these problems all knowledge is always shared among all the agents. Goals are always public. Let us call this set of problems FMAP problems and refer to this level of privacy as FMAP privacy.

In FMAP problems, constants known to all agents are often considered to be internal. For example, every agent knows that there is a road between two cities but the agents do not know that other agents know it. Thus, for example in *Driverlog* domain, every *truck* publishes its `drive` action without precondition that there is a road connecting two cities. In the descriptions, when it is stated that "*All information is public*", the problem can contain these internal constants.

6.1.2 Overall Benchmark Results

Table 6.1 shows an overall coverage of solved problems. We can see that the FMAP has better results in most of the domains and also in the overall coverage when compared with basic PSM, PSM-R and PSM-V variants. Nevertheless, we can see that both PSM-R and PSM-V excel in few domains (PSM-R in *Elevators* and *Logistics*, and PSM-V in *Rovers*). PSM extended with both features – PSM-VR – keeps the benefits of these features and outperforms FMAP in the overall score.

A relaxed plan helps especially in *Elevators* and *Logistics* domains. In both domains the relaxed plan well captures the coordination points, that is, where the *passenger* (or *package*) will be transported by which agent (*elevator* or *truck*). The relaxed plan thus represents a solution outline which is then extended by each agent with internal actions. This allows to solve the task in a single iteration.

[3] http://ipc.icaps-conference.org/

Domain	FMAP	PSM	PSM-R	PSM-V	PSM-VR
Blocksworld (34)	19	**27**	26	26	26
Depots (20)	**6**	0	0	0	3
Driverlog (20)	**15**	10	13	14	14
Elevators (30)	**30**	1	**30**	4	**30**
Logistics (20)	10	0	**20**	0	**20**
Openstacks (30)	23	**30**	**30**	**30**	**30**
Rovers (20)	**19**	7	6	14	16
Satellite (20)	**16**	6	6	9	9
Woodworking (30)	22	**27**	**27**	**27**	**27**
Zenotravel (20)	**18**	17	**18**	17	**18**
Total (244)	178	125	176	141	**193**

TABLE 6.1: Number of problems solved by the compared planners. Privacy classification follows FMAP and thus the results are not directly comparable with MA-STRIPS planners.

Plan verification, represented by PSM-V variant, helped in *Driverlog* and *Rovers* problems where many solutions generated by an agent were unacceptable by some other agent. In *Driverlog*, this is caused by the privacy of constants defining the topology of the world (the predicates `link` and `path` describing connected locations). The convergence is improved by trimming out parts of plans which are impossible to fulfill, that is, a `drive` action between locations which are not connected by a road. In *Rovers*, the situation is similar – private constants describe abilities of each rover. When an agent requires some action from another agent which can not be performed then the verification allows to remove such a plan from landmarks so that other agents are not confused by it.

Table 6.2 compares run times needed to solve selected tasks solvable by all the PSM variants. We can see that all the PSM variants scale much better than FMAP, especially in the *Openstacks* problems which are all solved in the first iteration. PSM performs best in most domains. This is a result of the requirement that the selected problems have to be solvable by all the variants and the basic PSM does not need to spend time on relaxed plan creation or on verification.

Left graph of Figure 6.1 shows how much time it is needed to solve different problems by PSM-VR as a function of problem size. The problem size is calculated as a number of actions and facts of the grounded problem (described in Section 5.5.1). Right graph of Figure 6.1 shows the time spent during the verification of other

	FMAP	PSM	PSM-V	PSM-R	PSM-VR
Driverlog					
p-01	**0.6**	2.2 (2)	2.3 (2)	1.1 (1)	1.2 (1)
p-05	**1.8**	33.8 (9)	6.5 (3)	3.2 (2)	4.1 (2)
p-08	11.9	9.6 (3)	6.0 (2)	**4.2** (2)	5.4 (2)
p-10	**2.1**	3.0 (2)	4.3 (2)	3.5 (2)	4.8 (2)
p-13	16.2	13.7 (3)	14.6 (3)	**8.2** (2)	8.3 (2)
Openstacks					
p-01	1.4	1.7 (1)	**1.2** (1)	**1.2** (1)	1.4 (1)
p-06	9.7	1.8 (1)	**1.7** (1)	1.8 (1)	3.1 (1)
p-11	51.0	**1.8** (1)	2.3 (1)	3.3 (1)	5.7 (1)
p-16	171.0	**2.2** (1)	4.4 (1)	5.5 (1)	9.1 (1)
p-21	497.0	**2.4** (1)	6.4 (1)	8.7 (1)	14.1 (1)
p-26	N/A	**2.9** (1)	12.5 (1)	13.3 (1)	22.6 (1)
Woodworking					
p-01	2.7	**1.2** (1)	2.0 (1)	**1.2** (1)	2.1 (1)
p-06	200.3	4.0 (2)	10.2 (2)	**3.5** (2)	7.4 (2)
p-11	1.9	**1.2** (1)	1.4 (1)	**1.2** (1)	1.5 (1)
p-16	N/A	**3.2** (2)	5.7 (2)	3.3 (2)	5.7 (2)
p-21	**0.4**	1.2 (1)	1.3 (1)	1.2 (1)	1.5 (1)
p-26	N/A	**1.4** (1)	2.1 (1)	1.5 (1)	2.7 (1)

TABLE 6.2: Comparison of run times on selected problems solved by all the planners. Times are in seconds, PSM variants have number of iterations in parenthesis.

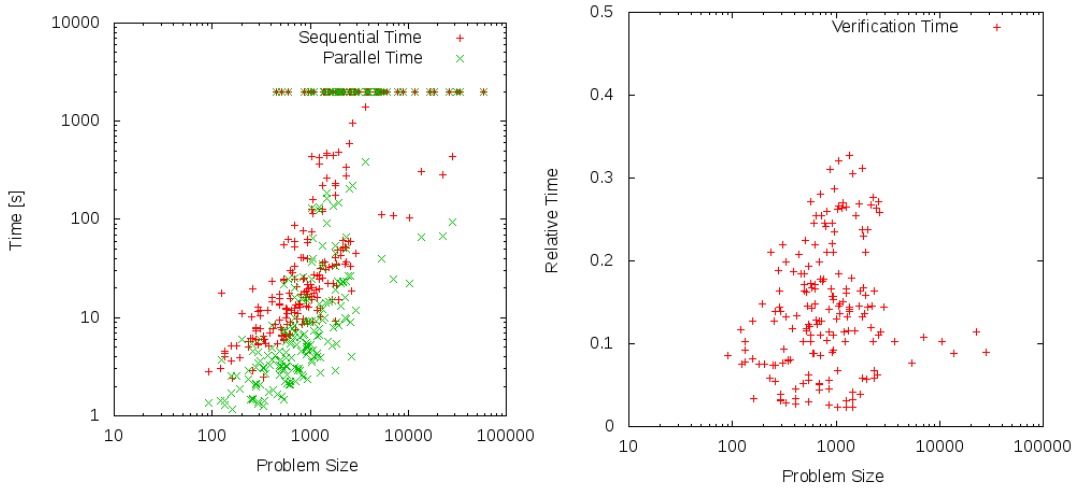


FIGURE 6.1: *Left*: Time needed to solve task as a function of problem size (log axis; time 2000 means not solved). *Right:* Portion of time an agent spends on verification of other agents' plans.

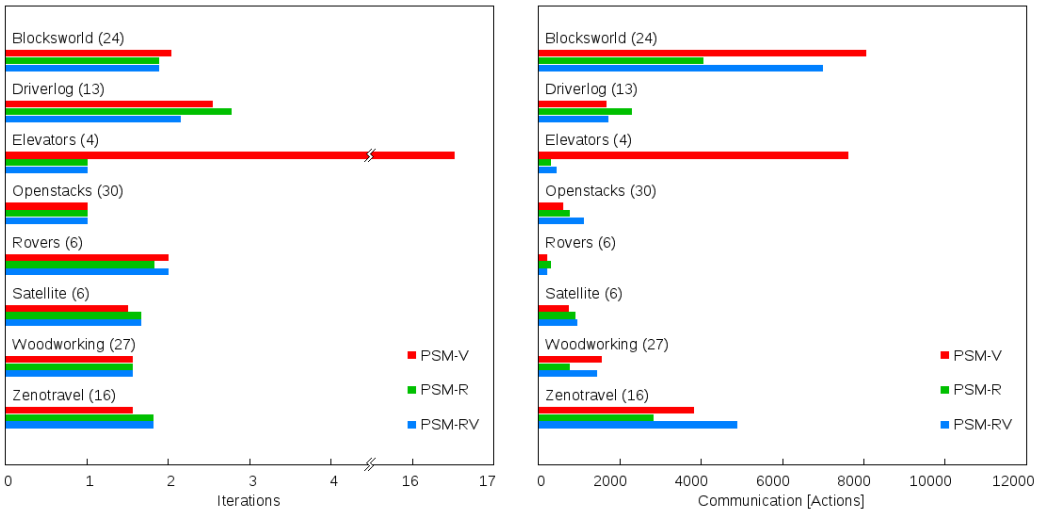


FIGURE 6.2: Number of iterations and amount of communication of PSM variants – always taken as average over problems solved by all methods in the graph (number of problems is appended to the domain name).

agents' plans. It shows that an agent spends less than one third of its computation time on verification. In average, it is approximately 14% of agent computation time. The relative time needed for verification is independent on the problem size and its grow/descent depends on a particular domain.

6.1.3 Communication Overhead Evaluation

Graph 6.2 shows an average number of iterations required to solve problems of selected domains (left), and the amount of communication among the agents (right). The averages are taken only for the problems solved by all three variants: PSM-R, PSM-V and PSM-VR. The numbers in parenthesis show how many problems is in this intersection (note that no *Depot* nor *Logistic* problem is solvable by PSM-R and thus the domains are not included). The performance measured by the number of iterations does not differ much over the domains (with the exception of *elevators* where the number of iterations of one problem is significantly decreased). The amount of communication is measured as the total number of actions sent among all agents. This includes the exchange of public plan projections used as landmarks and also queries for plan verification. As expected, we can see that verification requires additional communication but in several domains this increase is outweighed by the decrease in number of iterations needed to solve the task.

6.2 PSM-VR Privacy Experiments

In this section, we analyze different privacy classifications (Section 6.2.1) and we compare benchmark domains with respect to these classifications (Section 6.2.2). We experimentally evaluate the impact of privacy classifications on the performance of our best PSM-based planner PSM-VR. We compare PSM-VR with other state-of-the-art MA-STRIPS planners (Section 6.2.3).

6.2.1 Privacy Classifications

In Section 3.2, we have described MA-STRIPS privacy classification which defines the minimal amount of public information needed to be shared by different agents. MA-STRIPS requires facts used by actions of at least two different agents to be public. We have converted FMAP problems to MA-STRIPS problems (as described in Section 5.5.1) which reduces the amount of public information. Moreover, we allow agents to have internal goals which further increases privacy.

We now have four different privacy classifications for our benchmark problem set. First is the original FMAP privacy setting. Second, denoted FMAP$^{\ominus}$, is a variant of FMAP where those facts mentioned only by a single agent are made internal (which can make some goals internal). Third, denoted MA-STRIPS, is the MA-STRIPS privacy classification with public goals. Fourth, denoted MA-STRIPS$^{\ominus}$, is the MA-STRIPS classification which allows internal goals.

Let FMAP(Π) denote the percentage of private facts in problem Π with respect to FMAP classification, and similarly for the other three privacy classifications. It certainly holds that FMAP(Π) $\leq$ FMAP$^{\ominus}$(Π) and MA-STRIPS(Π) $\leq$ MA-STRIPS$^{\ominus}$(Π), and finally FMAP(Π) $\leq$ MA-STRIPS(Π).

6.2.2 Privacy in Benchmark Domains

Privacy of agent knowledge in different domains with these privacy classifications is following:

Blocksworld

FMAP: All information is public.

FMAP$^{\ominus}$: Information what each agent is holding is made internal (predicate `holding`). This predicate is never part of the goal and thus all goals are public.

MA-STRIPS: Facts that do not need to be shared are internal including all instances of predicate `holding`. Nevertheless, every action changes the position of a block which affects every other agent, and thus all the action are public.

MA-STRIPS$^{\ominus}$: Same as MA-STRIPS.

Depots

FMAP: All information is public.

FMAP$^{\ominus}$: The location of a *truck* is internal (predicate `at`). This predicate is never part of any goal and thus all goals are public.

MA-STRIPS: *Truck* locations and *truck* loads are always internal. The position of a crate is specified by predicate `on`. A crate can be either `on` a hoist or `on` a truck. When it is `on` a truck, the predicate is internal.

MA-STRIPS$^{\ominus}$: Same as MA-STRIPS.

Driverlog

FMAP: All information is known by each agent. Nevertheless, some constants are defined as internal (`link` and `path`).

FMAP$^{\ominus}$: *Driver* locations are internal (predicate `at`). Those parts of a goal specifying driver locations are internal.

MA-STRIPS: The location of a *driver* (predicate `at`) is internal if it is not part of a goal. Actions `walk` changing *driver*'s position are also internal unless they mention some public position.

MA-STRIPS$^{\ominus}$: Predicate `at` is always internal and the respective parts of goals are internal to some agent.

Elevators

FMAP: Positions of passengers are always public while *elevator* positions and the number of passengers in each *elevator* are internal.

FMAP$^{\ominus}$: Same as FMAP.

MA-STRIPS: Only passenger positions at floors accessible by two or more *elevators* and goal positions are public. Actions `board` and `leave` at these floors are public. All other information is internal.

MA-STRIPS$^{\ominus}$: Positions of passengers at floors accessible by only one *elevator* and respective `board/leave` actions are internal.

Logistics

FMAP: Agent locations are internal but the package location is public.

FMAP$^{\ominus}$: Same as FMAP.

MA-STRIPS: Locations of packages accessible to only one *truck* or *plane* are internal to appropriate agent. Actions `load` and `unload` at these locations are internal too.

MA-STRIPS$^{\ominus}$: Same as MA-STRIPS.

Openstacks

FMAP: All information is public including goals that the orders have been shipped.

FMAP$^{\ominus}$: Information about orders (predicates `waiting` and `shipped`), including all goals, is known to *manager* only. *Manufacturers* have empty goals.

MA-STRIPS: Instances of predicate `waiting` are internal. Goal facts `shipped` are public.

MA-STRIPS$^{\ominus}$: Same as FMAP$^{\ominus}$.

Rovers

FMAP: Locations of samples and whether the data have been communicated are public.

FMAP$^{\ominus}$: Same as FMAP.

MA-STRIPS: If a sample can be analyzed by only one *rover* then the location of this sample is agent's internal fact.

MA-STRIPS$^{\ominus}$: Same as MA-STRIPS.

Satellites

FMAP: Pointings of satellites and whether the image has been taken is public.Both are used by goals.

FMAP$^{\ominus}$: Pointings of satellites are internal (predicate `pointing`). This is occasionally internal part of a goal.

MA-STRIPS: Pointings of satellites are internal unless they appear in a goal.

MA-STRIPS$^{\ominus}$: Same as FMAP$^{\ominus}$.

Woodworking

FMAP: All information is public.

FMAP$^{\ominus}$: Constants and the internal state, specifying whether some board is loaded in the *highspeed saw* (predicate `in-highspeed-saw`), are internal.

MA-STRIPS: As in FMAP$^{\ominus}$. Moreover, *saw* agent always starts the production chain and thus the availability of resources is its internal knowledge (that is, predicate `available`). *Saw* agent has to perform at least two actions before it produces an intermediate product needed by other agents. Thus these initial actions are also internal (for example, action `load-highspeed-saw`).

MA-STRIPS$^{\ominus}$: Same as MA-STRIPS.

Zenotravel

FMAP: Positions of passengers and planes are public. A fuel level is internal. Thus, all `fly` actions are public and only `refuel` actions are internal.

FMAP$^{\ominus}$: Positions of planes are internal (predicate `at`). Thus, only positions of passengers are public. Therefore, actions `fly` and `zoom` (which is actually a shortcut for two `fly` actions) are internal.

MA-STRIPS: Same as FMAP$^{\ominus}$, except that goal facts that are public and the positions of passengers (predicate `in`) in cities reachable by only one plane, which are internal.

MA-STRIPS$^{\ominus}$: Similarly to FMAP$^{\ominus}$, positions of planes are always internal. Moreover, positions of passengers (predicate `in`) in cities reachable by only one plane are also internal.

Privacy classifications on benchmark domains, measured as a relative ratio of internal facts and actions, are demonstrated in Table 6.3.

	FMAP		FMAP$^{\ominus}$		MA-STRIPS		MA-STRIPS$^{\ominus}$	
Blocksworld	0%	0%	16%	0%	26%	0%	26%	0%
Depots	0%	0%	5%	6%	39%	6%	39%	6%
Driverlog	0%	0%	38%	10%	36%	8%	38%	10%
Elevators	33%	18%	33%	18%	74%	37%	74%	37%
Logistics	9%	10%	9%	10%	70%	32%	70%	32%
Openstacks	0%	0%	34%	0%	17%	0%	34%	0%
Rovers	70%	54%	70%	54%	77%	59%	77%	59%
Satellite	19%	3%	46%	79%	48%	76%	48%	79%
Woodworking	3%	0%	4%	0%	29%	2%	29%	2%
Zenotravel	19%	8%	35%	83%	60%	77%	61%	83%

TABLE 6.3: Percentage of internal facts (left) and actions (right) in benchmark domains with respect to different privacy classifications. Values are averages over problems in each domain.

6.2.3 Privacy Benchmarks

In Section 6.1 above we have compared our PSM-based planners with FMAP on problems with FMAP privacy classification. Now we compare our best PSM-VR with MA-STRIPS based algorithms RDFF and GPPP. Both planners, similarly to FMAP, are state-space search planners. RDFF (Štolba and Komenda, 2014) is based on distribution of A* algorithm with distributed heuristics with a variable number of agents involved in heuristic estimation. Greedy Privacy Preserving Planner (GPPP) (Maliah, Shani, and Stern, 2014) is based on an iterative deepening search in relaxed subproblems enhanced with landmarks. Table 6.4 shows that PSM-VR outperforms both state-of-the-art algorithms[4], even though GPPP solved more instances of *Depots*, *Satellite* and *Zenotravel* domains.

Table 6.5 shows overall results of PSM-VR algorithm for all privacy settings. Unfortunately, we are not aware of any planner that could be used to compare the performance on problems with internal goals. We can see that in some domains, increased privacy improves the performance of the planner, while in others, the performance decreases. For example, in *Satellites* problems, the most difficult goal is the final position pointing of a satellite (predicate `pointing`). If this goal is internal, then the complexity of the problem decreases. The opposite case is represented by *Openstacks* domain. When the goals are public then a producer (agent *manufacturer*) can

[4]We would like to thank Michal Štolba for evaluating RDFF and GPPP on our benchmarks.

Domain	RDFF	GPPP	PSM-VR
Blocksworld (34)	6.8	3	**26**
Depots (20)	6.2	**8**	6
Driverlog (20)	14	9	**15**
Elevators (30)	2.9	16†	**30**
Logistics (20)	5.8	**20**	**20**
Openstacks (30)	11.7	0‡	**30**
Rovers (20)	14.7	10	**18**
Satellite (20)	10.8	**16**	11
Woodworking (30)	5.6	0‡	**24**
Zenotravel (20)	6.1	**20**	13
Total (244)	84.6	102	**193**

TABLE 6.4: Number of MA-STRIPS problems solved by the compared planners: RDFF, GPPP, and PSM-VR. †Used version of the domain in GPPP experiments without action costs, consisting of 16 problems. ‡GPPP does not support action costs.

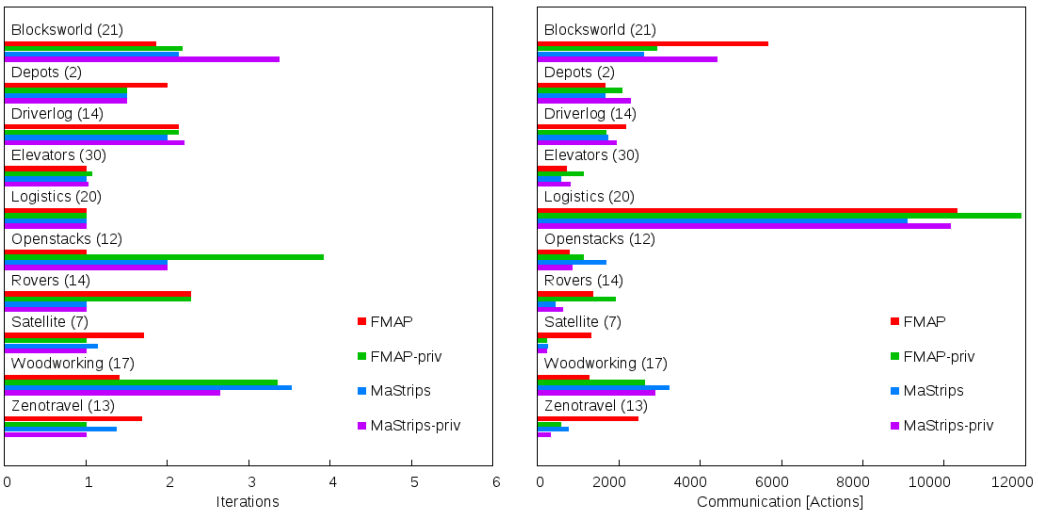


FIGURE 6.3: Performance of PSM-VR with different privacy classifications measured by the number of iterations and amount of communication. Values are averages over those problems solved by all the planners. Number of the problems considered for each domain is in parenthesis.

Domain	FMAP	FMAP$^{\ominus}$	MA-STRIPS	MA-STRIPS$^{\ominus}$
Blocksworld (34)	**26**	24	**26**	**26**
Depots (20)	3	4	**6**	4
Driverlog (20)	14	**15**	**15**	**15**
Elevators (30)	**30**	**30**	**30**	**30**
Logistics (20)	**20**	**20**	**20**	**20**
Openstacks (30)	**30**	12	**30**	26
Rovers (20)	16	14	**18**	**18**
Satellite (20)	9	**18**	11	17
Woodworking (30)	**27**	22	24	23
Zenotravel (20)	**18**	16	13	16
Total (244)	193	175	193	**195**

TABLE 6.5: Problem coverage of PSM-VR on benchmark domains with different privacy classifications. Number of problems in each domain is in parenthesis.

more easily create a plan fitting the requirements of a consumer (agent *manager*). Internal goals improve the performance with MA-STRIPS classifications, but the effect on FMAP classifications is the opposite. The best overall coverage is for MA-STRIPS$^{\ominus}$ with 195 solved problems out of 244 problems. This shows that increase of internal facts do not only support privacy but it can also improve performance.

6.3 PSM-D Experimental Results

For experimental evaluation of reduction of internal dependencies (Section 5.1) we use benchmark results from the CoDMAP'15 competition[5]. The benchmark set contains 12 domains with 20 problems per domain. Each agent has its own domain and problem files containing description of known facts and actions. Additionally, some facts/predicates are specified as private and thus should not be communicated to other agents. The privacy classification roughly corresponds to MA-STRIPS.

We firstly present analysis of internal dependencies and their reductions in Section 6.3.1. Then, in Section 6.3.2, we present results independently evaluated by organizers of the CoDMAP'15 competition.

[5]See `http://agents.fel.cvut.cz/codmap`

Domain	Facts / Public	Merge facts	Fact disclosure	Actions / Public	Success
Blocksworld	787 / 733	53	100 %	1368 / 1368	100 %
Depots	1203 / 1139	56	85 %	2007 / 2007	100 %
Driverlog	1532 / 1419	16	25 %	7682 / 7426	100 %
Elevators	509 / 343	43	29 %	2060 / 1767	70 %
Logitics	240 / 154	56	63 %	342 / 298	100 %
Rovers	2113 / 1251	31	3 %	3662 / 1555	13 %
Satellite	846 / 578	0	0 %	8839 / 914	1 %
Taxi	177 / 173	0	0 %	107 / 107	100 %
Wireless	1456 / 1421	25	73 %	1809 / 1809	100 %
Woodworking	1448 / 1425	7	27 %	4126 / 4126	100 %
Zenotravel	1349 / 1204	0	0 %	13516 / 2364	0 %

TABLE 6.6: Results of the analysis of internal dependencies of public actions in benchmark domains.

6.3.1 Domain Analysis

In this section, we present analysis of internal dependencies of public action in the benchmark problems.

We have evaluated internal dependencies of public actions within benchmark problems by constructing full dependency graph for every agent in every benchmark problem. We have applied Algorithm 5 to reduce full dependency graphs to an irreducible publicly equivalent dependency graph. The results of the analysis are presented in Table 6.6. The table columns have the following meaning. Column **(Facts / Public)** represents an average number of all facts, and of public facts respectively, in a domain problem. Column **(Merge facts)** represents an average size of $\mathsf{facts}(\Delta)$ in the resulting irreducible dependency graph. Column **(Fact disclosure)** represents the percentage of published merge facts with respect to all the internal facts. Column **(Actions / Public)** represents an average number of all actions, and of public actions respectively, in a domain problem. Column **(Success)** represents the percentage of agents capable of reducing their full dependency graph to a publicly equivalent graph without internal actions.

We can see that seven of the benchmark domains, namely *Blocksworld*, *Depots*, *Driverlog*, *Logistics*, *Taxi*, *Wireless*, and *Woodworking* were found simply dependent. All the problems in these domains can be solved by solving a local problem of a single agent. On the contrary, in most problems of domains *Rovers*, *Satellite*, and *Zenotravel*, none of the agents were able to reduce its full dependency graph so that

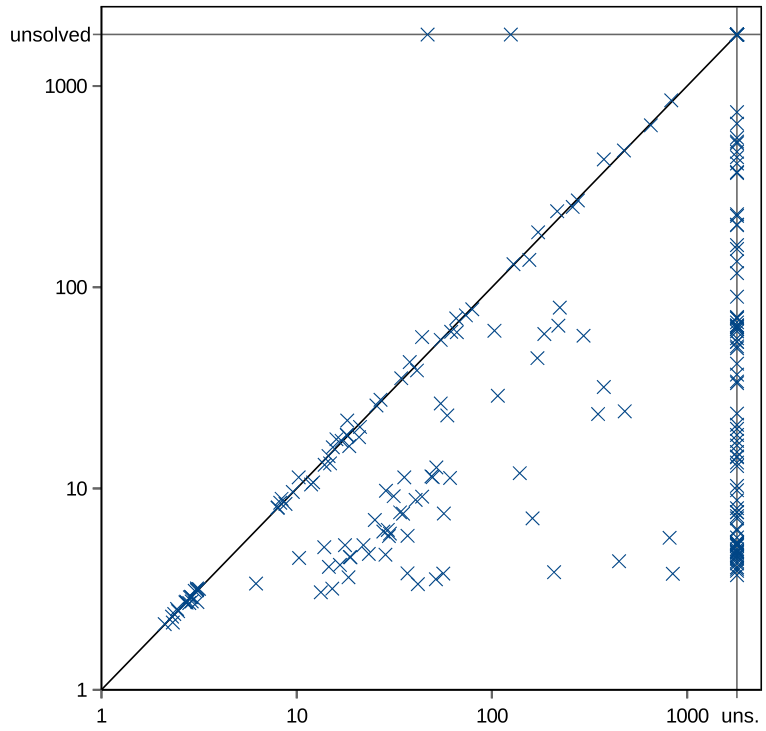


FIGURE 6.4: Comparison of planning times (in seconds) of PSM-VR algorithm without (X axis) and with (Y axis) internal problem reduction.

it contains no internal actions. Hence the agents in these domain publish only the minimal dependency graphs and hence the analysis does not help in solving them. Finally, in *Elevators* domain, some of the agents succeeded in reducing their full dependency graphs and thus the analysis can partially help to solve them.

6.3.2 Experimental Results

To evaluate the impact of dependency analysis, we use two variants of our PSM planner submitted to the CoDMAP'15 competition (Tožička, Jakubův, and Komenda, 2015b). Namely, we use planner PSM-VR (Section 5.2 and Section 5.4.1) and its extension with dependency analysis PSM-VRD (Section 5.1).

Figure 6.4 evaluates the impact of the dependency analysis on CoDMAP benchmark problems. For each problem, a point is drawn at the position corresponding to the runtime without dependency analysis (x-coordinate) and the runtime with dependency analysis (y-coordinate). Hence the points below the diagonal constitute improvements. Results show that the dependency analysis decreases overall planning time of PSM algorithm. We can see that in few cases the time increases which is

Domain	#	MAPLAN LMCUT ‡	MAPLAN MA-LMCUT ‡	MH-FMAP	MAPLAN FF+DTG	PSM-VR	PSM-VRD
Blocksworld	20	2	1	0	14	12	**20**
Depots	20	5	2	2	10	1	**16**
Driverlog	20	15	9	18	18	16	**20**
Elevators	20	2	0	**9**	**9**	2	5
Logitics	20	4	5	4	**16**	0	**16**
Openstacks	20	1	1	8	**18**	14	**18**
Rovers	20	2	4	18	**19**	13	13
Satellite	20	13	4	4	14	7	**17**
Taxi	20	19	14	**20**	19	9	**20**
Wireless	20	3	2	0	**4**	0†	0†
Woodworking	20	3	4	8	14	9	**19**
Zenotravel	20	6	6	16	**19**	16	16
Coverage	240	75	52	107	174	99	**180**

TABLE 6.7: Official results of CoDMAP multi-agent planner competition (`http://agents.fel.cvut.cz/codmap/results/`). Table shows overall coverage of solved problem instances. ‡This is optimal planner. †This post-submission domain was not supported by the planner parser. These results are presented with the consent of CoDMAP organizers.

caused by the time consumed by reduction process. Also, by publishing additional facts, the problem size can grow and thus it can become harder to solve.

Tables 6.7, 6.8, and 6.9 show official results of the CoDMAP competition. We can see that the dependency analysis significantly improved the performance of PSM-VR planner. Moreover, PSM-VRD achieved the overall best coverage in 8 out of 12 domains. As expected, the highest coverage directly corresponds to the success of dependency analysis. The table also shows results of two additional criteria comparing the quality (IPC Score) of solutions and the time (IPC Agile Score) need to find the solution. In both criteria PSM-VRD performed very well even though it was outperformed by MAPLAN-FF+DTG planner in the IPC Agile Score.

Table 6.10 shows times the planners needed to solve problems. Problems contain only those solved by all the planners. Some planners did not solved any problems of

Domain	#	MAPLAN LMCUT ‡	MAPLAN MA-LMCUT ‡	MH-FMAP	MAPLAN FF+DTG	PSM-VR	PSM-VRD
Blocksworld	20	2	1	0	7	11	**17**
Depots	20	5	2	2	6	1	**15**
Driverlog	20	15	9	**17**	12	14	16
Elevators	20	2	0	**8**	6	1	4
Logitics	20	4	5	4	13	0	**15**
Openstacks	20	1	1	8	**18**	9	12
Rovers	20	2	4	**18**	16	5	5
Satellite	20	**13**	4	4	10	6	**13**
Taxi	20	**19**	14	17	15	6	16
Wireless	20	3	2	0	**4**	0†	0†
Woodworking	20	3	4	7	13	8	**17**
Zenotravel	20	6	6	**15**	**15**	10	10
IPC Score	240	75	52	100	135	72	**140**

TABLE 6.8: Official results of CoDMAP multi-agent planner competition (`http://agents.fel.cvut.cz/codmap/results/`). Table shows IPC score over the plan quality Q (a sum of Q^*/Q over all problems, where Q^* is the cost of an optimal plan, or of the best plan found by any of the planners for the given problem during the competition). ‡This is optimal planner. †This post-submission domain was not supported by the planner parser. These results are presented with the consent of CoDMAP organizers.

Domain	#	MAPLAN LMCUT ‡	MAPLAN MA-LMCUT ‡	MH-FMAP	MAPLAN FF+DTG	PSM-VR	PSM-VRD
Blocksworld	20	1	0	0	**14**	5	**14**
Depots	20	3	1	1	9	0	**14**
Driverlog	20	10	4	11	**17**	7	14
Elevators	20	1	0	4	**8**	1	4
Logitics	20	3	2	2	13	0	**14**
Openstacks	20	0	0	3	**18**	7	8
Rovers	20	1	1	7	**19**	6	6
Satellite	20	10	2	1	**13**	3	12
Taxi	20	14	7	10	**19**	3	15
Wireless	20	**3**	2	0	2	0†	0 †
Woodworking	20	2	3	4	9	5	**18**
Zenotravel	20	5	4	10	**18**	8	8
Agile Score	240	52	27	52	**159**	45	127

TABLE 6.9: Official results of CoDMAP multi-agent planner competition (`http://agents.fel.cvut.cz/codmap/results/`). Table shows IPC Agile score over the planning time T (a sum of $1/(1+\log_{10}(T/T^*))$ over all problems, where T^* is the runtime of the fastest planner for the given problem during the competition). ‡This is optimal planner. †This post-submission domain was not supported by the planner parser. These results are presented with the consent of CoDMAP organizers.

Domain	#	MAPLAN LMCUT ‡	MAPLAN MA-LMCUT ‡	MH-FMAP	MAPLAN FF+DTG	PSM-VR	PSM-VRD
Depots	1	2	3	12	**1**	56	4
Driverlog	9	18	622	28	**14**	172	39
Rovers	1	1296	1396	47	**1**	15	13
Satellite	2	1634	496	17	**2**	22	22
Taxi	7	18	331	212	**11**	2525	32
Woodworking	1	4	5	25	**2**	29	6
Zenotravel	6	**11**	57	23	1806	54	53
Total	27	2983	2910	363	1836	2873	**168**

TABLE 6.10: Official results of CoDMAP multi-agent planner competition (`http://agents.fel.cvut.cz/codmap/results/`). Table shows sum of times (in seconds) needed to solve selected problems. Selected problems contain problems solved by all the planners. ‡This is optimal planner. These results are presented with the consent of CoDMAP organizers.

some domains and thus these domains were omitted. We can see that even though PSM-VRD did not solved any problem in the fastest time, it outperformed other planners in total. Reductions of internal dependencies allowed achieve more uniform planning times (especially, it significantly reduced long planning times of *Driverlog* and *Taxi* domains). In total, it allowed to improve the performance of PSM-VR more than 17-times.

Chapter 7

Conclusions

"Now I can eat well, sleep well and be glad.
I can go everywhere with a good feeling."

Geronimo

The problem of multi-agent domain-independent planning is a natural and realistic extension of classical planning and thus it has attracted a lot of attention among researchers recently. The research focuses on the whole range of related problems from the problem definition and languages for problem description to the implementation of various planners. So far, most of the planner implementations were based on distribution of existing centralized solvers for classical planning. In this thesis, we have decided to take a different approach. We designed a novel planner that focuses on inter-agent coordination and naturally allows to be executed in parallel by all the agents.

The first part of the thesis formally introduced the problem of privacy-preserving multi-agent domain-independent planning and related terminology. Then, we introduced a novel planner scheme combining coordination-space search with compilation to classical planning. We proposed a compact representation of large and possibly infinite sets of local plans in the form *Planning State Machines* (PSM). PSMs allowed us to effectively implement desired operations, namely PSM intersection and public projection. We have also shown how this planning approach can be used to find all solutions to the problem during a single iteration of communication between the agents.

The core of the thesis extends the proposed generic planning scheme with improvements and heuristics. We proposed several practical improvements to increase the effectiveness and coverage of the basic algorithm. Distributed delete-relaxation heuristic (PSM-R), approximative plan analysis (PSM-V), and reduction of internal dependencies (PSM-D), all proved to be effective in practice. Each of the improvements helps to solve a particular class of problems and we have shown that all the improvements can be combined together without impairing each other (PSM-VRD).

Finally, the last part of the thesis focuses on experimental evaluation of the planner and its improvements. These experiments revealed that the created PSM-VRD planner outperforms state-of-the-art planners. Moreover, this result was confirmed by an independent evaluation at the CoDMAP'15 competition.

Experimental results confirm that exploiting the problem of agent coordination is a viable approach to multi-agent planning.

7.1 Thesis Achievements

This section summarizes the achievements of this thesis. The following list represents published results that extended the state of the art in multi-agent planning:

(1) *Design of novel coordination-space distributed iterative multi-agent planner scheme.*
This thesis describes *planning by plan set intersection* and proposes a general iterative scheme to implement it.

(2) *Effective representation of sets of local solutions.*
Planning state machines are introduced to compactly represent even countably infinite sets of plans. Required operations are described and their properties are proved.

(3) PSM-VRD *planner implementation.*
Proposed scheme of the coordination-space planner is implemented and the proof of soundness and completeness is provided. Three extensions of the basic algorithm are introduced and analyzed in detail.

(4) *Novel graph representation of planning problem.*

This thesis defines novel graph representation of planning problem. This representation is used for representation of internal dependencies of public actions and their analysis.

(5) *Reductions of internal dependencies.*

Graph representation of internal dependencies allows us to design and implement *safe* reductions. These reductions are shown to enable the detection of problems that can be trivially solved and to significantly improve the time-efficiency of solving many other problems. Similar reductions have been successfully used in classical planning as a preprocessing step (Tožička et al., 2016).

(6) *Experimental evaluation of the planner.*

This thesis provides thorough experimental evaluation of the implemented planner and its variants on the state-of-the-art multi-agent planning problems and their comparison with other state-of-the-art multi-agent planners.

(7) *Privacy preservation analysis.*

The algorithm, together with its extensions is analyzed with respect to the protection of private information. Moreover, the thesis shows that PSM-VRD facilitates operation under various privacy specifications.

(8) *Winner of the CoDMAP'15 competition.*

The implemented planner participated at the CoDMAP'15 international competition of multi-agent planners, and won its distributed track.

(9) *Best Student Paper Award at ICAART'16*

The article describing reductions of internal dependencies has been awarded the Best Student Paper Award at ICAART'16 conference.

The described achievements demonstrate that all the objectives described in Section 1.1.1 were accomplished, while the results exceeded our expectations. This includes the *bonus* objective aiming to make a contribution to the research of classical planning, which was accomplished by polynomial reductions of planning problems (Tožička et al., 2016).

7.2 Selected Related Publications

This section provides an overview of the author's publications related to multi-agent planning. Moreover, it also lists the author's articles on diverse planning, which is problem closely related to the content of this thesis (especially to the generation of local solutions, see Section 5.3).

Multi-Agent Planning Articles

Articles in Impacted Journals

- Jan Tožička, Jan Jakubův, Antonín Komenda, Michal Pěchouček: Privacy-concerned multiagent planning. Knowl. Inf. Syst. 48(3): 581-618 (2016).

Articles in Proceedings

- Jan Tožička, Michal Štolba, Antonín Komenda: ϵ-Strong Privacy Preserving Multiagent Planner by Computational Tractability. ICAART 2017: (to appear).
- Jan Tožička, Jan Jakubův, Martin Svatoš, Antonín Komenda: Recursive Polynomial Reductions for Classical Planning. ICAPS 2016: 317-325.
- Michal Štolba, Jan Tožička, Antonín Komenda: Secure Multi-Agent Planning. PrAISe@ECAI 2016: 11:1-11:8.
- Michal Štolba, Jan Tožička, Antonín Komenda: Secure Multi-Agent Planning Algorithms. ECAI 2016: 1714-1715.
- Jan Tožička, Jan Jakubův, Antonín Komenda: Recursive Reductions of Internal Dependencies in Multiagent Planning. ICAART (2) 2016: 181-191.
- Jan Tožička, Jan Jakubův, Antonín Komenda: From Public Plans to Global Solutions in Multiagent Planning. EUMAS/AT 2015: 21-33.
- Jan Tožička, Jan Jakubův, Antonín Komenda: On Internally Dependent Public Actions in Multiagent Planning. DMAP 2015: 18-24.
- Jan Jakubův, Jan Tožička, Antonín Komenda: Multiagent Planning by Plan Set Intersection and Plan Verification. ICAART (2) 2015: 173-182.

- Jan Jakubův, Jan Tožička, Antonín Komenda: Using Process Calculi for Plan Verification in Multiagent Planning. ICAART (Revised Selected Papers) 2015: 245-261.

- Jan Tožička, Jan Jakubův, Antonín Komenda: Generating Multi-Agent Plans by Distributed Intersection of Finite State Machines. ECAI 2014: 1111-1112.

- Jan Tožička, Jan Jakubův, Karel Durkota, Antonín Komenda, Michal Pěchouček: Multiagent Planning Supported by Plan Diversity Metrics and Landmark Actions. ICAART (1) 2014: 178-189.

Diverse Planning Articles

Articles in Impacted Journals and Book Chapters

- Jan Tožička, Antonín Komenda: Diverse Planning for UAV Control and Remote Sensing. Sensors 16(12):2199 (2016).

- Jan Tožička, Jan Jakubův, Karel Durkota, Antonín Komenda: Extensibility Based Multiagent Planner with Plan Diversity Metrics. Trans. Computational Collective Intelligence 20: 117-139 (2015).

Articles in Proceedings

- Jan Tožička, David Šišlák, Michal Pěchouček: Planning of Diverse Trajectories for UAV Control Displays. AAMAS 2013: 1231-1232.

- Jan Tožička, Jan Balata, Zdeněk Mikovec: Diverse Trajectory Planning for UAV Control Displays. AAMAS 2013: 1411-1412.

- Jan Tožička, David Šišlák, Michal Pěchouček: Planning of Diverse Trajectories. ICAART (2) 2013: 120-129.

- Jan Tožička, David Šišlák, Michal Pěchouček: Diverse Planning for UAV Trajectories. ICAART (Revised Selected Papers) 2013: 277-292.

Bibliography

Amir, Eyal and Barbara Engelhardt (2003). "Factored Planning". In: *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI) 2003*, pp. 929–935.

Bacchus, Fahiem and Qiang Yang (1994). "Downward Refinement and the Efficiency of Hierarchical Problem Solving". In: *Artificial Intelligence* 71.1, pp. 43–100.

Bhattacharya, Subhrajit, Vijay Kumar, and Maxim Likhachev (2010). "Search-Based Path Planning with Homotopy Class Constraints." In: *Third Annual Symposium on Combinatorial Search (SOCS) 2010*. Ed. by Ariel Felner and Nathan R. Sturtevant. AAAI Press.

Blum, Avrim L. and Merrick L. Furst (1995). "Fast Planning Through Planning Graph Analysis". In: *Artificial Intelligence* 90.1, pp. 1636–1642.

Bonet, Blai and Hctor Geffner (2001). "Planning as Heuristic Search". In: *Artificial Intelligence* 129, pp. 5–33.

Borrajo, Daniel (2013). "Plan Sharing for Multi-Agent Planning". In: *Proceedings of Second Workshop on Distributed and Multi-Agent Planning (DMAP), Workshop of ICAPS 2013*, pp. 57–65.

Boutilier, Craig and Ronen I. Brafman (2001). "Partial Order Planning with Concurrent Interacting Actions". In: *Journal of Artificial Intelligence Research* 14, pp. 105–136.

Brafman, Ronen I. (2015). "A Privacy Preserving Algorithm for Multi-Agent Planning and Search". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI) 2015*, pp. 1530–1536.

Brafman, Ronen I. and Carmel Domshlak (2006). "Factored Planning: How, When, and When Not". In: *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence 2006*, pp. 809–814.

Brafman, Ronen I. and Carmel Domshlak (2008). "From One to Many: Planning for Loosely Coupled Multi-Agent Systems". In: *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS) 2008*. Vol. 8, pp. 28–35.

Brenner, Michael (2003). "Multiagent Planning with Partially Ordered Temporal Plans". In: *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI) 2003*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 1513–1514.

Bylander, Tom (1994). "The Computational Complexity of Propositional STRIPS Planning". In: *Artificial Intelligence* 69, pp. 165–204.

Chrpa, Lukáš (2010). "Generation of Macro-Operators via Investigation of Action Dependencies in plans". In: *Knowledge Engineering Review* 25.3, pp. 281–297.

Clarke, Edward H. (1971). "Multipart Pricing of Public Goods". In: *Public Choice* 11.1, pp. 17–33. ISSN: 1573-7101.

Darwiche, Adnan and Mark Hopkins (2001). "Using Recursive Decomposition to Construct Elimination Orders, Jointrees, and Dtrees". In: *Proceedings of Symbolic and Quantitative Approaches to Reasoning with Uncertainty, Sixth European Conference (ECSQARU) 2001*, pp. 180–191.

Dimopoulos, Yannis, Muhammad Adnan Hashmi, and Pavlos Moraitis (2010). "Extending SATPLAN to Multiple Agents". In: *Proceedings of the Thirtieth International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI) 2010*, pp. 137–150.

Durkota, Karel and Antonín Komenda (2013). "Deterministic Multiagent Planning Techniques: Experimental Comparison (Short paper)". In: *Proceedings of Second Workshop on Distributed and Multi-Agent Planning (DMAP), Workshop of ICAPS 2013*, pp. 43–47.

Fabre, Eric et al. (2010). "Cost-Optimal Factored Planning: Promises and Pitfalls". In: *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS) 2010*, pp. 65–72.

Fikes, Richard E. and Nils J. Nilsson (1971). "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Proceedings of the Second International Joint Conference on Artificial Intelligence (IJCAI) 1971*, pp. 608–620.

Ghallab, Malik, Dana S. Nau, and Paolo Traverso (2004). *Automated Planning: Theory and Practice*. Elsevier. ISBN: 9781558608566.

Groves, Theodore (1973). "Incentives in Teams". In: *Econometrica* 41.4, pp. 617–631.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. ISSN: 0536-1567.

Helmert, Malte et al. (2007). "Flexible Abstraction Heuristics for Optimal Sequential Planning". In: *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS) 2007*, pp. 176–183.

Hoffmann, Jörg and Bernhard Nebel (2001). "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research (JAIR)* 14, pp. 253–302.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0321455363.

Jakubův, Jan, Jan Tožička, and Antonín Komenda (2015). "Multiagent Planning by Plan Set Intersection and Plan Verification". In: *Proceedings of the Sevent International Conference on Agents and Artificial Intelligence (ICAART) 2015*, pp. 173–182.

Jakubův, Jan and Joe B. Wells (2010). "Expressiveness of Generic Process Shape Types". In: *Proceedings of the Fifth International Conference on Trustworthly Global Computing (TGC) 2010*. Vol. 6084. Lecture Notes in Computer Science (LNCS). Springer, pp. 103–119. ISBN: 978-3-642-15639-7.

Jensen, Rune M. and Manuela M. Veloso (2000). "OBDD-based Universal Planning for Synchronized Agents in Non-Deterministic Domains". In: *Journal of Artificial Intelligence Research* 13, pp. 189–226.

Jezequel, Loïg and Eric Fabre (2012). "A#: A Distributed Version of A* for Factored Planning". In: *Proceedings of the Fifty-First IEEE Conference on Decision and Control (CDC) 2012*, pp. 7377–7382.

Jonsson, Anders and Michael Rovatsos (2011). "Scaling Up Multiagent Planning: A Best-Response Approach". In: *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS) 2011*, pp. 114–121.

Jonsson, Peter and Christer Bäckström (1998). "Tractable Plan Existence Does Not Imply Tractable Plan Generation". In: *Annals of Mathematics and Artificial Intelligence* 22, pp. 281–296.

Kautz, Henry, David Mcallester, and Bart Selman (1996). "Encoding Plans in Propositional Logic". In: *Knowledge Representation and Reasoning*. Morgan Kaufmann, pp. 374–384.

Kautz, Henry A., Bart Selman, and Jörg Hoffmann (2006). "SatPlan: Planning as Satisfiability". In: *Abstracts of the Fifth International Planning Competition*.

Kelareva, Elena et al. (2007). "Factored Planning Using Decomposition Trees". In: *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI) 2007*, pp. 1942–1947.

Kitano, Hiroaki et al. (1999). "RoboCup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research". In: *IEEE International Conference on Systems, Man, and Cybernetics*. IEEE Computer Society, pp. 739–746.

Kovács, Dániel. L. (2012). "A Multi-Agent Extension of PDDL3.1". In: *Proceedings of the Third Workshop on the International Planning Competition (IPC), Workshop of ICAPS 2012*. Atibaia, So Paulo, Brazil, pp. 19–27.

Lansky, Amy L. and Lise Getoor (1995). "Scope and Abstraction: Two Criteria for Localized Planning". In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI) 1995*, pp. 1612–1619.

Makholm, Henning and Joe B. Wells (2005). "Instant Polymorphic Type Systems for Mobile Process Calculi: Just Add Reduction Rules and Close". In: *Proceedings of Fourteenth European Symposium on Programming (ESOP) 2005*. Vol. 3444. Lecture Notes in Computer Science (LNCS). Springer, pp. 389–407. ISBN: 3-540-25435-8.

Maliah, Shlomi, Guy Shani, and Roni Stern (2014). "Privacy Preserving Landmark Detection". In: *Proceedings of Twenty-First European Conference on Artificial Intelligence (ECAI) 2014*.

Mundhenk, Martin et al. (2000). "Complexity of Finite-horizon Markov Decision Process Problems". In: *Journal of ACM* 47.4, pp. 681–720. ISSN: 0004-5411.

Newell, Allen, John C. Shaw, and Herbert A. Simon (1959). "Report on a General Problem-Solving Program". In: *Proceedings of the International Conference on Information Processing*, pp. 256–264.

Nissim, Raz and Ronen Brafman (2012a). "Multi-Agent A* for Parallel and Distributed Systems". In: *Proceedings of Workshop on Heuristics and Search for Domain-Independent Planning, Workshop of ICAPS 2012*, pp. 43–51.

Nissim, Raz and Ronen I. Brafman (2012b). "Multi-Agent A* for Parallel and Distributed Systems". In: *Proceedings of Eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2012*. Valencia, Spain, pp. 1265–1266. ISBN: 0-9817381-3-3, 978-0-9817381-3-0.

— (2013). "Cost-Optimal Planning by Self-Interested Agents". In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI) 2013*. Bellevue, Washington: AAAI Press, pp. 732–738.

Nissim, Raz, Ronen I. Brafman, and Carmel Domshlak (2010). "A General, Fully Distributed Multi-Agent Planning Algorithm". In: *Proceedings of Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2010*, pp. 1323–1330.

Oglietti, Marcelo and Amedeo Cesta (2004). "CSTRIPS: Towards Explicit Concurrent Planning". In: *Proceedings of Ninth National Symposium of 'Associazione Italiana per l'Intelligenza Artificiale (AI*IA) 2004, Third Italian Workshop on Planning and Scheduling (IWP) 2004*. ISBN: 88-89422-09-2. Perugia, Italy.

Pellier, Damien (2010). "Distributed Planning through Graph Merging". In: *Proceedings of the Second International Conference on Agents and Artificial Intelligence (ICAART) 2010*, pp. 128–134.

Rosenthal, Robert W. (1973). "A Class of Games Possessing Pure-Strategy Nash Equilibria". In: *International Journal of Game Theory* 2.1, pp. 65–67. ISSN: 1432-1270.

Sacerdott, Earl D. (1973). "Planning in a Hierarchy of Abstraction Spaces". In: *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI) 1973*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 412–422.

Štolba, Michal and Antonín Komenda (2013). "Fast-Forward Heuristic for Multiagent Planning". In: *Proceedings of Second Workshop on Distributed and Multi-Agent Planning (DMAP), Workshop of ICAPS 2013*. Vol. 120, pp. 75–83.

Štolba, Michal and Antonín Komenda (2014). "Relaxation Heuristics for Multiagent Planning". In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS) 2014*, pp. 298–306.

Štolba, Michal, Antonín Komenda, and Dániel L. Kovács (2016). "Competition of Distributed and Multiagent Planners (CoDMAP)". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI) 2016*, pp. 4343–4345.

Tate, Austin (1976). *Project Planning Using a Hierarchic Non-linear Planner*. Department of Artificial Intelligence Research Report No 25, University of Edinburgh.

Torreño, Alejandro, Eva Onaindia, and Oscar Sapena (2014). "FMAP: Distributed Cooperative Multi-Agent Planning". In: *Applied Intelligence* 41.2, pp. 606–626.

Torreño, Alejandro, Eva Onaindia, and Oscar Sapena (2014). "A Flexible Coupling Approach to Multi-Agent Planning Under Incomplete Information". In: *Knowledge and Information Systems* 38.1, pp. 141–178.

Tožička, Jan, Jan Jakubův, and Antonín Komenda (2014). "Generating Multi-Agent Plans by Distributed Intersection of Finite State Machines". In: *Proceedings of Twenty-First European Conference on Artificial Intelligence (ECAI) 2014*, pp. 1111–1112.

— (2015a). "On Internally Dependent Public Actions in Multiagent Planning". In: *Proceedings of Fourth Workshop on Distributed and Multi-Agent Planning (DMAP), Workshop of ICAPS 2015*.

— (2015b). "PSM-based Planners Description for CoDMAP 2015 Competition". In: *Proceedings of Competition of Distributed and Multiagent Planners (CoDMAP) 2015*.

Tožička, Jan et al. (2014). "Multiagent Planning Supported by Plan Diversity Metrics and Landmark Actions". In: *Proceedings of the Sixth International Conference on Agents and Artificial Intelligence (ICAART) 2014*, pp. 179–189.

Tožička, Jan et al. (2016). "Recursive Polynomial Reductions for Classical Planning". In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS) 2016*, pp. 317–325.

Tožička, Jan, Jan Jakubův, and Antonín Komenda (2016). "Recursive Reductions of Internal Dependencies in Multiagent Planning". In: *Proceedings of the Eight International Conference on Agents and Artificial Intelligence (ICAART) 2016*, pp. 181–191. ISBN: 978-989-758-172-4.

Tožička, Jan et al. (2015). "Extensibility Based Multiagent Planner with Plan Diversity Metrics". In: *Transactions on Computational Collective Intelligence XX*. Ed. by Ngoc Thanh Nguyen et al. Cham: Springer International Publishing, pp. 117–139. ISBN: 978-3-319-27543-7.

Tožička, Jan et al. (2016). "Privacy-Concerned Multiagent Planning". In: *Knowledge and Information Systems* 48.3, pp. 581–618. ISSN: 0219-3116.

Vickrey, William (1961). "Counterspeculation, Auctions, and Competitive Sealed Tenders". In: *Journal of Finance* 16.1, pp. 8–37.

de Weerdt, Mathijs and Brad Clement (2009). "Introduction to Planning in Multiagent Systems". In: *Multiagent Grid Systems* 5.4, pp. 345–355. ISSN: 1574-1702.