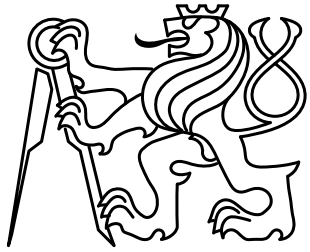
Czech Technical University in Prague
Faculty of Electrical Engineering

Doctoral Thesis

18.9.2017 *Michal Štolba*

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science



REVEAL OR HIDE: INFORMATION SHARING IN MULTI-AGENT PLANNING

Doctoral Thesis

Michal Štolba

Prague, 18.9.2017

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

Supervisor: Ing. Antonín Komenda, Ph.D.

Co-supervisor: Doc. Ing. Jiří Vokřínek, Ph.D.

Dedicated to my grandfather who showed me the beauty of science and sparked in me the desire for knowledge and the need for answers to the most intriguing questions we could think of (in the age before google).

Acknowledgments

In the first place, I would like to thank prof. Michal Pěchouček for accepting me in his research group, providing me with support since then, and encouraging me to pursue the doctorate. My warmest thanks go to my supervisor Antonín Komenda who constantly supported me and with whom we spent hours in sometimes fruitful and sometimes simply enjoyable discussions.

My thanks belong to my collaborators and coauthors, especially Daniel Fišer for all the hard implementation work and for fixing many errors, Jan Tožička for deep discussions about the intricacies of privacy, and Jiří Vokřínek for continual support. I shall not forget to mention Petr Benda whose technical support was crucial for the success of the CoDMAP competition and many experimental evaluations.

Last but not least, I would like to thank my wife Martina, son Radovan, and daughter Zora for love, support, and for making sure I do not have too much time to work on the thesis, thus greatly improving my efficiency.

Abstract

The ability to plan a sequence of action in order to achieve a given goal with respect to the initial conditions of the world is one of the crucial aspects of intelligence. It is no surprise, that this aspect has been thoroughly studied in the context of artificial intelligence since its very beginning. The same can be said about the study of multi-agent aspects of planning in the research field of multi-agent systems. Among the most important aspects of such multi-agent planning is information sharing, that is, which information should be shared by the agents and which not, and also how to share the information efficiently.

We provide several perspectives on the issue of sharing or hiding information in multi-agent planning. We mostly focus on heuristic search with domain-independent heuristics which is a well-established approach both in classical and multi-agent planning. We advance the state of the art in a number of directions.

Firstly, we focus on the distributed computation of heuristics. The main research question is how to achieve global heuristic guidance without explicitly communicating and revealing private parts of the planning problems respective to the particular agents. We approach this issue by providing a number of distributed variants of classical planning heuristics, both inadmissible and admissible (which are necessary for optimal planning). We use the acquired knowledge to design more general approaches for distributing relaxation heuristics and finally any heuristic (in an admissible way). We theoretically analyze the distributed heuristics (e.g., by showing their admissibility) and provide a thorough experimental evaluation, showing their superiority in speed or heuristic guidance compared to the same heuristics computed locally by the agents (that is, without sharing any information throughout the heuristic computation).

Secondly, we propose a heuristic search algorithm which is able to balance the use of distributed and local heuristics. The distributed heuristic approach is not always the best choice. In many problems, the heuristic guidance of the locally computed heuristic is close to the distributed variant but without the computation and communication overheads. We solve the issue by allowing the search to use the local heuristic while computing the distributed heuristic and waiting for replies from other agents. This technique is able to balance the information sharing in most domains and problems and practically dominates each approach used separately. The resulting planner also improves on the state of the art in suboptimal multi-agent planning.

Thirdly, we analyze information sharing in multi-agent planning in the context of privacy. In privacy-preserving cooperative multi-agent planning, the agents want to cooperatively plan a sequence of actions but do not want to reveal their private knowledge. In realistic scenarios, avoiding explicit communication of the private information is not enough, the agents do not want to allow any other agent even to deduce such information from the communication protocol.

The thesis builds on two major journal publications and a number of works published at the top-tier AI conferences. The designed algorithms are both theoretically analyzed and thoroughly experimentally evaluated. In order to allow for a more complete and rigorous comparison of existing multi-agent planners, we have co-organized the first Competition of Multi-Agent and Distributed Planners (CoDMAP) during the work on the above research topics. We have collaborated on the design of the formal domain and problem description language, we have designed the competition setup (in two tracks), implemented necessary software tools, and performed the evaluation. The description of the competition and the results relevant to other presented topics are provided as a part of the thesis.

Anotace

Jedním ze základních projevů inteligence je schopnost naplánovat si posloupnost akcí vedoucí k dosažení svého cíle. Není žádným překvapením, že tato schopnost byla předmětem studia umělé inteligence od samého počátku. Totéž bychom mohli říct o multi-agentních aspektech plánování studovaných v rámci multi-agentních systémů. Mezi nejzásadnější otázky takového multi-agentního plánování pak patří sdílení informací, jinými slovy, které informace je dobré mezi agenty sdílet a které ne a také jak informace sdílet efektivně.

V této práci nabízíme několik pohledů na problematiku sdílení a skrývání informací v multi-agentním plánování. Zejména se zaměřujeme na heuristické prohledávání s doménově nezávislou heuristikou, což je zaběhnutý postup jak v klasickém, tak v multi-agentním plánování. Hranice vědeckého poznání posouváme v následujících směrech.

V první řadě se zaměřujeme na distribuovaný výpočet heuristik. Hlavní výzkumnou otázkou je, jak dosáhnout globálního vedení heuristického prohledávání bez toho, aby bylo nutné explicitně komunikovat a tím odhalit privátní znalosti agentů. Jako řešení předkládáme několik distribuovaných variant heuristik známých z klasického plánování, a to jak přípustných (což je nutné pro optimální plánování), tak nepřípustných. Získané znalosti poté využíváme k vytvoření obecnějších postupů pro distribuci relaxovaných a později libovolných heuristik, a to při zachování přípustnosti. Distribuované heuristiky teoreticky analyzujeme (např. formálním důkazem jejich přípustnosti) a zároveň nabízíme experimentální evaluaci ukazující jejich výhody oproti lokálně počítaným klasickým heuristikám (tedy heuristikám které během výpočtu nesdílejí informace mezi agenty).

Druhou oblastí přínosu této práce je nový algoritmus heuristického prohledávání, který umožňuje vyvážit použití distribuované a lokální heuristiky. Distribuovaný výpočet heuristiky totiž není vždy tou nejlepší volbou. V některých problémech je kvalita lokální heuristiky srovnatelná s distribuovanou, ale bez zvýšených výpočetních a komunikačních nároků. Náš přístup k řešení tohoto problému je umožnit prohledávání pomocí lokální heuristiky, zatímco probíhá výpočet její distribuované varianty a to během čekání na odpovědi od ostatních agentů. Tato technika je schopná správně vyvážit sdílení informace ve většině plánovacích domén a problémů a prakticky dominuje oba přístupy výpočtu heuristiky použité každý zvlášť. Výsledný plánovač pak v několika metrikách překonává dosavadní multi-agentní plánovače.

Třetím tématem, na které se tato práce zaměřuje, je sdílení informací v multi-agentním plánování z pohledu zachování soukromí jednotlivých agentů. V takovém případě sice chtějí agenti společnými silami naplánovat sekvenci akcí, ale nechtějí při tom odhalit své privátní znalosti. V realistických scénářích však nestačí vyhnout se explicitnímu sdílení takových informací, ale je nutné zabránit ostatním agentům, aby tyto informace mohli dedukovat z veřejně dostupných informací a z komunikačního protokolu.

Tato práce vychází ze dvou hlavních časopiseckých publikací a z řady konferenčních článků publikovaných na prestižních zahraničních konferencích v oboru. Navržené algoritmy jsou jak teoreticky analyzovány, tak experimentálně vyhodnoceny a porovnány. Abychom mohli provést úplnější a rigoróznější experimentální evaluaci, zorganizovali jsme během práce na výše uvedených výzkumných tématech první ročník soutěže multi-agentních plánovačů (Competition of Multi-Agent and Distributed Planners, CoDMAP). Spolupracovali jsme na návrhu univerzálního jazyka pro popis multi-agentních domén a problémů, navrhli jsme hlavní principy soutěže a její rozdělení do dvou sekcí, naimplementovali jsme nezbytné softwarové nástroje a provedli samotné vyhodnocení. Popis této soutěže a jejích výsledků je rovněž obsažen v této práci.

Contents

Chapter 1

Introduction

In the last couple of decades, we have witnessed the world becoming much more interconnected. Even items of daily use are becoming connected to the Internet, intelligent and autonomous. Be it smartphones, autonomous cars, unmanned air vehicles (UAVs) or ordinary household items, contributing to the so-called Internet of Things. The interconnectedness and ubiquity are expected to increase rapidly in the near future. Such interconnected entities, which we might as well call agents, need not only to communicate but also to coordinate and cooperate with each other. In order to do so, such agents (including also humans) need to plan effectively in this increasingly complex and information-abundant environment. Moreover, the agents need to consider carefully which information to share with other agents and which information to keep for themselves, i.e., to hide. Sharing information may be beneficial if the information helps the planning process, but sharing too much information may overload the communication channels and cause the planning process to stall. Another aspect of information sharing is that some of the information may be confidential and thus cannot be shared. Indeed, it even has to be protected from leaking to the other agents through the execution of the planning algorithms.

The same phenomena are bound to arise in the industry even more. The concept of Industry 4.0 building upon automation and interconnected smart entities is quickly gaining ground. All such entities, agents, and humans act in isolation no longer and thus need to take each other into account during decision making, planning, and deliberation. Automated planning, a sub-field of classical artificial intelligence (AI), and multi-agent planning even more so, provides the tools that can be used to solve such complex problems.

Multi-agent planning covers a vast number of domains and problems, their models, and algorithms and techniques for solving them. From this wealth of related work (which is covered in Chapter 2) we carve out a concise fundamental problem closest to the classical planning literature which is *domain-independent deterministic cooperative multi-agent planning*. We first state the problem informally in the following section and provide a full formal definition in Chapter 3.

1.1 Problem Statement

This thesis is concerned with *domain-independent deterministic cooperative multi-agent planning*. Let us dissect the term and define (informally, for now) each of its parts. Let us start with *planning*. As already hinted, a planning problem is the problem of sequencing a number of actions performed in an environment so that the environment is transformed from its current state to the desired goal state, that is, any state which satisfies the goal conditions. Typically, the state of the world is described as a set of variables with finite domains (e.g., binary) and each action is defined by some precondition over the variables which must hold for the action to be applied and an effect on the variables. The precondition and effect thus describe the change in the world state caused by the application of the action. The sets

of variables and of actions are considered finite as well. The complete planning problem can be seen as a transition system, that is, a graph where the nodes are all possible states of the world and the edges are all possible transitions based on the available action. There is a transition between two states s and s' if there is an action a which is applicable in s (its preconditions are satisfied) and the application of a on s results in s' (by the application of the effects of a). Finding a solution to the planning problem is equivalent to finding a path in the transition system from the initial state to some of the goal states. In optimal planning, the path must be the shortest (or with the lowest cost) among all such paths.

As in classical planning, we are interested in off-line planning. This means that the planning problem definition is given to the solver up-front, the solver is allocated some time to solve the problem and only when the solution (the plan) is ready, its execution commences. In other words, the planning and execution phases are separated. The planning phase, which is the main focus of the thesis, comes first. The execution phase comes next and is out of the scope of this work (the execution phase is not considered by most of the work in classical planning as well). This contrasts with the on-line planning model, where the planning and execution are intertwined, often putting a short planning phase after each executed step in the plan. In that case, the complete plan is typically not prepared up-front.

Next term to be explained is *deterministic*. Deterministic in the context of planning means that the effect of the application of a particular action depends only on the state it is applied in. That is, every time we apply the same action in the same state, it always results in the same new state. This clearly is a significant assumption and simplification, as in the real world this is often not the case. Take for an example a robot performing some task such as putting an object on a table. Clearly, the robot expects the object to be on the table after the execution of the action, but other things may happen. The object may fall off the table, if it is placed too close to the edge, or if it is a ball which rolls off. Or somebody may push the table (or the robot) during the execution and thus ruin the robot's effort. We may argue that in such cases, the description of the world state was not precise enough and did not take into account all properties such as the exact position of the table. Or we may make our model more high-level in that the planning action put-object-on-table might actually represent a more complex behavior of the robot which always ends in successfully putting the object on the table, e.g., by picking it up from the ground and retrying. Of course, this approach would not work if the object was a glass of water which may get spilled or broken. Thus, although there are cases where the deterministic model is not applicable, there are many situations where it is.

Notice that in one of the examples above, the failure of the action was due to an external agent, the person pushing the table out of reach of the robot. In the case of presence of other autonomous entities (agents), we talk about multi-agent systems and *multi-agent* planning (MAP). The necessity to consider the actions of other agents brings us to a completely different level of complexity, the game theory. But again, we can simplify the problem by additional assumptions. In our case, we assume that the agents are *cooperative*, that is, the goals the agents want to achieve (or the utility of the solution the agents are aiming to maximize) are common to all agents, or are not in conflict. This means that there is no gain for any of the agents in not cooperating, or even in a stronger case, each agent cannot achieve its goals without cooperating with others. There are plenty of examples of such situations. Imagine when all the robots in a factory are owned by a single company, they are clearly cooperative. But even if the robots are owned by multiple companies, e.g., in a search and rescue mission, they may be in a situation where cooperation is needed or beneficial. Recall one of the examples presented above, where multiple factories cooperate to produce goods or multiple companies in a consortium need to coordinate their processes in order to fulfill the shared project goals.

An important feature in multi-agent systems is *privacy*. Even if we restrict our attention to cooperative multi-agent planning, where the agents have common goals or utility, still, the agents are distinct entities (such as companies) and often have concerns about privacy. In particular, in the case of planning, the agents want to coordinate their actions in order to achieve the common goal, but may not want to share all their internal data, values, and possible processes (or actions) with the other agents. In fact, the agents might want to disclose only as much information as is needed for successfully planning how to achieve the common goals, even though sharing more information might help the distributed

planning process. In our world model, this means that only the smallest necessary subset of the variables is shared among the agents (we call such variables public) and only the actions which interact with such public variables are shared (that is, public actions). Moreover, only the public parts of the public actions are necessary to be shared (a public action a restricted to the public variables is a public projection of a). Thus in order to preserve privacy, the agents want to hide the existence and values of private variables, the existence of private actions and the existence and values of private preconditions and effects of public actions. We say that private information has leaked if some of that information is either directly exposed to some other agent, or some other agent is able to deduce it from some of the public information communicated during the execution of a planning algorithm. We say that an algorithm is weak privacy-preserving if it does not openly communicate private information and strong privacy-preserving if the private information cannot be deduced even from all the public information communicated throughout the execution of the algorithm.

The last term remaining to explain is *domain-independent*, which rather describes our approach to the solution than the problem itself. As in classical planning, domain-independent means that given a general model, our approaches must be applicable to all domains and problems which can be described using the model. In classical and multi-agent planning, the model is a finite set of finite-domain variables and a finite set of deterministic actions defined on the basis of their preconditions and effects. Moreover, we have such model for each agent, describing each agent's view of the world, together composing the complete global problem. We assume the agents to be cooperative. Any problem which can be formulated using this model and its assumptions can be solved (given enough, possibly exponential but finite, time), or the non-existence of a solution reported, using the techniques described in this thesis. In other words, all the techniques can be applied on any of such problems (or domains) and thus are domain-independent.

Problem Representations

A problem in both classical and multi-agent planning can be represented as a deterministic state-transition system (or a transition system for short). Such transition system is a directed graph consisting of states representing all possible states of the world and transitions representing all possible actions applied in the states where they are applicable. Even though we restrict ourselves only to finite systems, such transition systems may be extremely large. For example, consider a small logistics problem with 10 locations, 5 trucks, and 5 packages. The state-space contains all possible combinations of the locations of the trucks and packages, that is, 10 locations for each truck and 15 locations for each package (including being in each of the trucks), which is $10^5 \cdot 15^5$ which is approximately $7 \cdot 10^{10}$ states. But the structure of the problem can be used to represent such vast numbers of states concisely.

In classical planning, there are three commonly used concise representations of the transition system.

Propositional logic representation (also known as classical representation) uses first-order literals (atomic formulas, also known as facts) and logical connectives (e.g., and, or) to describe states, action preconditions and effects, and the goal condition. In the logistics example, the literals might be for example truck1-at-loc1 and package1-at-loc1 and the precondition of a load action the conjunction truck1-at-loc1 $\wedge$ package1-at-loc1 meaning that both literals must hold true in order to apply the action in a state. Classical representation is useful in some theoretical work but is not typically used internally by the planning systems.

Set-theoretic representation (also known as STRIPS representation) reduces the possibilities of the classical representation by allowing conjunction of literals only. Such restriction leads to a concise representation of states, preconditions and effects by sets of literals which hold true, for example $\{$truck1-at-loc1, package1-at-loc1$\}$, assuming that all other literals are false (in the case of state description) or not considered (in the case of preconditions and goal conditions). Effects typically consist of two sets, one is a set of literals which become true by the application of the action (i.e.,

are added), the other is a set of literals which become false (i.e., are deleted). This representation is often used internally by the planning and heuristic algorithms.

State-variable representation (also known as multi-valued representation) represents a state as a finite set of finite-domain variables and each action as a partial function mapping a tuple of values (the precondition) to another tuple of values (the effect). This representation is used internally in most modern planners (including our work), but some concepts are better presented using the set-theoretic (STRIPS) representation. The state-variable representation can be converted to the set-theoretic representation by using the variable assignments as facts. In the logistic example, some of the variables and their domains might be $\mathsf{truck1\text{-}at} \in \{\mathsf{loc}_1, ..., \mathsf{loc}_k\}$ and $\mathsf{package1\text{-}at} \in \{\mathsf{loc}_1, ..., \mathsf{loc}_k, \mathsf{truck}_1, ..., \mathsf{truck}_l\}$. A state, precondition, or effect is then an assignment, such as $\mathsf{truck1\text{-}at} = \mathsf{loc}_1, \mathsf{package1\text{-}at} = \mathsf{loc}_1, ...$ (and each variable assignment can be considered a fact in the set-theoretic representation).

Each of the above representations concisely represents a transition system, which is exponential in the size of the representation (recall that the states of the transition system are all possible combinations of the literals or variable valuations).

Even though such representations are concise compared to the complete transition system, still, they are unwieldy to write by hand. Consider the logistics example. For each move action for each truck and between each two locations, the problem designer needs to specify the particular preconditions and effects. In order to alleviate such extensive writing, the planning research community uses a high-level representation language called Planning Domain Description Language (PDDL) [McDermott et al., 1998]. PDDL is a lifted representation with a lisp-like syntax using predicate logic instead of propositional logic to describe general domains and particular problems. Predicate logic allows for use of variables (often called parameters) in predicates and action definitions. For example in the logistics domain, the predicate $(\mathsf{at}\ ?\mathsf{v}\ ?\mathsf{l})$, where the variables $?\mathsf{v}$ and $?\mathsf{l}$ are the parameters, is a binary predicate which relates a vehicle with a location. By substituting the variables with particular objects (also defined in the PDDL description), we obtain a literal (or fact) of the propositional logic which can then be used in the propositional-logic or set-theoretic representations. This process is known as grounding and the resulting literals are sometimes referred to as ground predicates. In the logistics example, $(\mathsf{at}\ \mathsf{truck1}\ \mathsf{loc1})$ is a ground predicate equivalent to the literal truck1-at-loc1. To avoid substituting irrelevant objects (such as a truck in the place of a location), PDDL often uses typing, for example in $(\mathsf{at}\ ?\mathsf{v} - \mathsf{vehicle}\ ?\mathsf{l} - \mathsf{location})$, only the objects of correct type can be substituted for the variables. Actions are parametrized similarly to the predicates and predicates (with matching variables) are used to describe the preconditions and effects of such actions.

This way, PDDL allows for extremely concise descriptions. Moreover, the description is typically divided into two files. The more general domain file describes the possible types, predicates (that is, properties and relations of objects), and actions, thus describing the mechanics of the given domain (e.g., logistics). The more specific problem file describes an instance of a problem for such domain and contains an enumeration of the actual objects and their types, description of the initial state, and description of the goal condition (which typically is a conjunction of ground predicates and negations of ground predicates).

Note that translation from PDDL to the state-variable representation is not as straightforward as to the other two representations. An automated translation process, used by the planners presented in this thesis as well, was thoroughly described in [Helmert, 2006]. The basic principle is to first ground the predicates, then to analyze invariants (such as that some facts never appear in a state together, e.g., truck1-at-loc1 and truck1-at-loc2) and finally synthesize variables based on the invariants (in our example, truck1-at-loc1 and truck1-at-loc2 can be values of a single variable).

In classical planning, the domain-independent approach is fostered by a wide range of PDDL benchmarks which are typically used to test novel approaches and which are also the basis for the International

Planning Competition (IPC)[1]. The same holds for multi-agent planning with a number of differences described thoroughly in Appendix A.

Problem Solving by Heuristic Search

Although there are other techniques used for both classical and multi-agent planning, heuristic search is by far the most widespread. Here we provide a brief introduction to the concepts used in heuristic search. First of all, the search itself. As already described, a classical planning problem is represented by a finite set of finite-domain variables and a finite set of deterministic actions, thus forming a graph known as the transition system of the planning problem.

To find a path from the initial state to some goal state (there are typically multiple states which satisfy the goal condition), a standard graph search algorithm, such as A* [Hart et al., 1968], can be used. Note that the transition system is exponential in the size of the problem description (that is, in the number of variables describing a state), we cannot use explicit graph representations such as the adjacency matrix. Instead, an implicit search algorithm is used, built on the following principle. There is an open list initialized to contain the initial state and a closed list initially empty. In each step of the algorithm, a state s is extracted from the open list and checked for the goal condition. If s does not satisfy the goal condition, s is added to the closed list. Next, the state s is expanded, that is, all actions applicable in s are applied and the resulting states are added to the open list, except for the states which already are in the closed list. The search continues by extracting another state from the open list until either the open list is empty (in which case there is no solution), or a state which satisfies the goal conditions is found. In that case, the actual plan is reconstructed from the references to the actions used to expand the states which are stored alongside the states. The space of states and possible transitions is also referred to as the search space.

If the open-list is a queue or a stack, the above approach basically corresponds to a breadth-first search (BFS) or depth-first search (DFS) respectively. In planning, the structure used for the open list is typically a priority queue (implemented as a heap) where the states are ordered according to a state evaluation function $f(s)$. Based on how $f(s)$ is computed, we obtain different search schemes. If $f(s) = g(s)$, where $g(s)$ is the distance (or cost) from the initial state to the goal state, the search is again a breadth-first search (BFS). Moreover, we may introduce a heuristic function $h(s)$ which is an estimate of the remaining distance (or cost) from the state s to the nearest goal state. A classical example of a heuristic function is the Euclidean straight-line distance in route planning on a road graph. Then, if we set $f(s) = h(s)$, we obtain a greedy best-first search (GBFS) which orders the states purely according to the heuristic estimate. A middle ground is to set $f(s) = g(s) + h(s)$ where the states are sorted according to both the distance already traversed and the estimate of the distance to go. This search scheme is commonly known as the A* search [Hart et al., 1968]. It has been shown that if the heuristic $h(s)$ is admissible, that is, it always underestimates the true cost, the A* search is optimal (returns the shortest solution).

The search scheme can be easily adapted to the multi-agent case by the following (simplified) principle. Each agent searches its own search space where the states contain only the public variables and private variables of that particular agent and the search is restricted to the actions of the particular agent. Only when a state is expanded by a public action, the state is sent to all other agents which add the received state to their open lists. This way, the states on which any interaction between agents can happen are shared among the agents, while the states which have changed only from the perspective of a single agent are kept locally.

A question is, how the heuristic is computed in the multi-agent setting. If we aim for a distributed computation or at least some degree of privacy, the heuristic function cannot be computed in a shared memory, which would otherwise be the most efficient approach. Again, the question whether to share or hide information comes to play. The first option is to compute the heuristic only locally, using the

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

same local problem used for the search but with the addition of the projected public actions of other agents. This is important as it may not be possible to compute the heuristic value using the actions of a single agent only. This approach to computing heuristics in multi-agent planning is called *local* or *projected heuristic*. The second approach is to compute the heuristic using a distributed algorithm, that is, to compute a *distributed heuristic*. Positives of the projected heuristic approach are that it is easy and fast to compute, needs no additional communication and preserves both admissibility and privacy (with respect to the heuristic value which may be shared or not). The negative is that the projected heuristic might miss some important information and thus misguide the search, or in other words might not guide the search as well as the distributed heuristic variant. The question of a privacy-preserving distributed heuristic is much more challenging. Which information should be shared in the heuristic computation and how is one of the main topics of this thesis.

1.2 Research Objectives and Achievements

At the beginning of my Ph.D. study in 2013, the MA-STRIPS formalism, recently introduced by Ronen Brafman and Carmel Domshlak in [Brafman and Domshlak, 2008], was gradually gaining ground in multi-agent planning. After a rather inefficient and literal approach to MA-STRIPS planning, Planning First [Nissim et al., 2010], the first successful MA-STRIPS planner was MAD-A* published by Raz Nissim and Ronen Brafman in [Nissim and Brafman, 2012]. As MAD-A* is a heuristic search, the authors used some of the best classical planning heuristics at that time to guide the search and provide excellent results. Building on a classical planning system, the Fast-Downward (FD) planner [Helmert, 2006], Nissim&Brafman were able to use the newest research results by simply applying the classical off-the-shelf heuristics on the local problem views of the agents, known also as projected heuristics. The authors have observed the following:

> "Perhaps the greatest practical challenge suggested by the distributed version of MA-A* is that of computing a global heuristic by a distributed system. In some domains, the existence of private information that is not shared leads to serious deterioration in the quality of the heuristic function, greatly increasing the number of nodes expanded."

At that point, I have decided to focus my research on that challenge.

Distributed Computation of Heuristics

While the computation of projected heuristics is straightforward, the distributed variant poses a much bigger problem. There are multiple aspects that need to be considered, such as the quality of the heuristic and how does it compare to a centralized solution, the communication overheads caused by the distributed computation and the privacy-preservation of such solution. Altogether, the research question I intended to answer was:

(Objective 1) How to compute classical planning heuristics in a distributed way?

To investigate the possibilities of distributed computation of classical planning heuristics we have first focused on inadmissible heuristics, in particular on the well-known family of relaxation heuristics. In [Štolba and Komenda, 2013], we have provided a distributed version of the Fast-Forward (FF) heuristic which was provably equal to the centralized (global) version. As at that time, the only efficient planner for multi-agent planning (MAP) was MAD-A*, which is an instance of optimal planner using admissible heuristic, we have decided to develop the multi-agent version of Greedy Best-First Search and use it to evaluate the inadmissible FF heuristic. We wanted to focus more on the distributed computation and as MAD-A* was internally built on the classical FD planner we have decided to implement our own MAP codebase which would better support complex distributed computations. Later, out of this effort emerged the MADLA Planner and much later on, thanks to Daniel Fišer, the MAPlan planner

which superseded MADLA both in efficiency and in its ability to run in a fully distributed setting. But let us first focus on our work in the direction of the distributed heuristics.

In [Štolba and Komenda, 2014], we have presented a general approach to the effective computation of distributed relaxation heuristics (including the FF heuristic), but without the assurances of equality with the centralized solution. In [Štolba and Komenda, 2017], we have provided a more efficient and privacy-preserving variant of the FF heuristic, which is also an important component of the MADLA Planner [Štolba and Komenda, 2015]. Our work on inadmissible relaxation heuristics is summarized in Chapter 4.

To delve into the case of admissible heuristics (that is, for optimal planning), which is even more complex, we have first focused on the LM-Cut heuristic [Štolba et al., 2015a]. Because the LM-Cut heuristic also falls in the class of relaxation heuristics, we were able to reuse some knowledge and ideas learned in the previous work. The MA-LM-Cut heuristic has been shown to yield equal values in the distributed and centralized global version and thus provide admissible estimates as well. A very different approach was needed to distribute the family of potential heuristics, which are computed using a Linear Program (LP) formulation. In [Štolba et al., 2016a], we have shown that potential heuristics are indeed a very good fit for distributed computation as except for solving the initial distributed linear program there is no additional communication necessary compared to the projected variant. The multi-agent potential heuristic provides globally admissible estimates and provides a significant amount of privacy preservation.

Our last effort in the direction of the Objective 1 is to provide a general approach to distributed heuristic computation. We have based our work on the idea of cost partitioning and provided preliminary results in [Štolba and Komenda, 2016]. The results of the work on admissible heuristics and cost partitioning are summarized in Chapter 6. The analysis of privacy of relaxation heuristics, potential heuristics, and cost partitioning is presented in Chapter 7.

Combining Local and Distributed Heuristics

Right after our first experiments with the distributed heuristic (that was the FF heuristic at that time), it became clear that the distributed computation of heuristics is not a silver bullet. Even though computing a global estimate helped tremendously in some domains, it did not have much effect in others and it even worsened the performance in some. This clearly had something to do with the concept of coupling of the problems known from the work of Brafman&Domshlak. A low coupling means that there are not many interactions between the agents (but the interactions might be crucial for solving the problem!). In that cases, distributed heuristic computation helped by providing that crucial information, or at least did not hinder the computation if the interactions were not so crucial. In contrast, high coupling means that most (even all) of the actions of the agents interact with other agents. In that case, often even though the additional information was helpful, the complexity of the distributed computation significantly decreased the performance. The experimental results underpinning the presented conclusions are summarized in Sections 4.4.1 and 4.4.3.

In general, the conclusion was that there is a trade-off in using a local or distributed heuristic. A local heuristic is fast to compute, but provides worse search guidance, whereas the distributed heuristic might take longer to compute as additional communication is typically involved, but provides more precise estimates and thus better guidance. The trade-off manifests itself differently in different planning domains and thus in some domains, it is better to use the local (projected) heuristic and in some domains, it is better to use the distributed heuristic. We have experimented with various techniques to balance the positive and negative effects but it soon became clear that the best way will possibly be to combine the local and distributed heuristics in a single search scheme. The research question we were posed with was:

(Objective 2) How to combine local and distributed heuristics?

The clear candidate technique was multi-heuristic search, which is a well-known concept in classical

planning, but at that time it was never used in multi-agent planning. But there was another twist to the problem we were facing. In classical planning, the idea of a multi-heuristic search was used to combine multiple *different* heuristics, where different means as different as possible. In practice, the best example is the LAMA Planner [Richter and Westphal, 2010] which combined the FF heuristic with a landmark-based heuristic which works on a completely different principle, so that the heuristics nicely complement each other. As a result, for example, when the FF heuristic gives bad estimates, the landmark heuristic gives good ones and vice versa. This, combined with other clever techniques, resulted in LAMA winning two installments of the International Planning Competition (IPC).

But our case was different. In our planner, we wanted to combine two versions of essentially the same heuristic, one computed on the local (projected) view of the problem, the other computed distributedly on the global problem. One fast but less informed, the other slow, but more informed. Our efforts culminated in the Multi-Agent Distributed and Local Asynchronous (MADLA) Search, conceived and implemented in 2014 and later on published in [Štolba and Komenda, 2017]. MADLA Search is a variant of multi-agent GBFS and exploits that the distributed FF heuristic is essentially asynchronous. This means that at some times, the FF heuristic is waiting for replies from other agents. Simply put, the MADLA Search uses this spare time to perform a fast local search using only the local projected heuristic. The MADLA Search is the main component of the MADLA Planner [Štolba and Komenda, 2015]. The algorithm together with proofs of soundness and correctness is presented in Chapter 5. The analysis of privacy of the algorithms and used distributed heuristic is presented in Chapter 7.

Consolidating Comparison and Benchmarking of Multi-Agent Planners

Since my first steps in the field of multi-agent planning, the interest of the research community in the topic of multi-agent planning has grown steadily. More MAP planners and MAP approaches were published, a specialized Distributed Multi-Agent Planning (DMAP) workshop was held as part of the International Conference on Planning and Scheduling annually. Since our first algorithms, we were conceptually building on the MAD-A* planner and used the same set of benchmarks and the same language to encode them (basically extending the PDDL language with a set of objects specifying the agents). But quite naturally, the same approach was not shared by other emerging planners. Indeed, unlike in classical planning where the norm was PDDL and IPC, no such consensus was achieved in multi-agent planning. Soon, it became hard to meaningfully compare the planners and the results of our research and I knew that the next issue we needed to solve was:

> To consolidate comparison and benchmarking of multi-agent planners.

We have decided to follow the approach to standardization used by the classical planning community–we have decided to organize a planning competition. Unlike in classical planning, we were faced with a Sisyphean task to consolidate an extremely wide area of research, where the authors did not agree even on such fundamental issues as whether MAP should be centralized or distributed and whether MAP planners should care about privacy or not. It must be noted that such diversity comes naturally from the diversity of multi-agent systems and our aim was by no means to throttle it. Instead, we aimed to carve out a subset of MAP approaches which were more or less based on the MA-STRIPS model and which shared many common features and thus should be comparable.

We have decided that the planners will participate in two separate tracks. A centralized track was aiming for compatibility with classical planners and planners which do not care about distribution but are interested in other aspects of multi-agent planning. A fully distributed track was intended to set a precedent for experimental evaluation of distributed multi-agent planners, where each agent was run on a dedicated computer, communicating over TCP/IP. Moreover, the competition aimed to consolidate the MAP input language, which resulted in an updated version of MA-PDDL[2] including the definition of which parts of the problem belongs to which agent (i.e, factorization) and privacy definitions. We

[2]There was a couple of MAP languages existing prior to the competition, but the languages were used basically only by their authors.

have proposed a set of benchmarks based mostly on classical planning domains used often in the MAP literature, but also including two novel domains.

We have co-organized the competition as a part of the DMAP workshop at the ICAPS [3] 2015 conference under the title Competition of Distributed and Multi-Agent Planners (CoDMAP) [4]. We have released a number of supportive tools for further organization of follow-up competitions and published the results at a number of venues [Štolba et al., 2016b, Komenda et al., 2016, Štolba et al., 2015b]. There were 12 planners in 17 configurations from 8 teams in the centralized track and 3 planners in 6 configurations from 2 teams in the distributed track, which highly exceeded our initial expectations. Since the competition, most of MAP papers have accepted the methodology and benchmarks proposed by us and have compared against the CoDMAP results. As the competition is tangential to the main topic of the thesis, but nonetheless is a significant contribution, we have summarized the used language, setup, and results of the competition in Appendix A.

Privacy in Multi-Agent Planning

The work on the CoDMAP competition, namely on the categorization of the planners according to various parameters including the treatment of privacy, and on some of the later papers describing the MADLA Planner and distributed heuristics led us to a realization that privacy is not treated correctly in most of the MAP literature, including our earlier work. Privacy is one of the commonly cited reasons why MAP problems cannot be solved using a classical centralized planner and many published MAP planners and techniques claim to be privacy-preserving. Unfortunately, the treatment of privacy is often fuzzy, hand-waving, and not grounded in strong theory and even less properly implemented. One of the earliest exceptions is the work of Nissim&Brafman in [Nissim and Brafman, 2014] where the authors define notions of privacy-preservation based on the theory of Secure Multiparty Computation (MPC) and a number of follow-up works by Brafman together providing first steps in the proper analysis of privacy in MAP. Gradually, we have understood that the next crucial research question to be answered is:

(Objective 3) How to formalize privacy and quantify privacy leakage and how to apply secure multi-party computation techniques in multi-agent planning?

Most works in MAP considered no privacy at all or were satisfied with the weak privacy assumption which only forbids the agents to explicitly communicate private parts of their problems, disregarding the possibility of other agents deducing some private information from the execution of the planning algorithms. The whole body of work in cryptography and secure MPC which fits MAP perfectly was mostly ignored. We have decided to improve the state of the art by focusing on the wealth of literature on the theory and practical techniques in secure MPC.

We have presented our initial work on privacy leakage quantification based on counting the number of transition systems represented by the information obtained by the adversary agent in [Štolba et al., 2016c]. Apart from the privacy leakage quantification, we have introduced a novel class of secure planning algorithms, Sec-MAP, which was based on combining the techniques used in privacy preserving planners and thus obtaining the best privacy preservation out of all existing planners. In [Štolba et al., 2016d] we have summarized how to use the Sec-MAP techniques to improve privacy preservation of existing planners. In [Štolba et al., 2017] we have focused on the privacy leakage quantification and provided both an in-depth theoretical analysis and a detailed analysis on a particular example.

Moreover, we have proposed a number of privacy-preserving planners, so far only in theory. The first is the ϵ-Strong privacy preserving planner in [Tožička et al., 2017a] which can get arbitrarily close to strong privacy by diluting the information by randomly generating unsystematic solutions. In [Tožička et al., 2017b] we have provided a class of privacy-preserving planners and we have shown that for this class of planners and for the multi-agent heuristic search class of planners it is not possible to have

[3] http://www.icaps-conference.org
[4] http://agents.fel.cvut.cz/codmap

a multi-agent planner which is effective, complete and strong privacy-preserving at once. Our results in the definition of privacy, privacy leakage, privacy analysis of existing and novel algorithms, and theoretical limits of privacy-preserving MAP are summarized in Chapter 7.

Summary

Whether to share or hide information is a crucial question in multi-agent planning. I have started the work on my Ph.D. topic with the focus on information sharing in distributed heuristics for multi-agent planning, but soon, other interesting and important topics and research questions emerged. It turned out that sometimes it is better to hide the information in the heuristic computation and thus prevent the overloading of communication channels. I have investigated techniques to combine local projected and global distributed heuristics leading to a state-of-the-art distributed search scheme. As a side-effect, I have contributed to the community effort to consolidate the comparison and benchmarking of the multi-agent planners by co-organizing the highly successful Competition of Distributed and Multi-Agent Planners. In the later part of my work, I have focused on the hiding of information in multi-agent planning from the perspective of privacy. In particular, I have contributed by formulating the definition and analysis of privacy leakage and by a number of theoretical results and possible planning schemes.

Altogether, my work resulted in seven publications in top-tier AI conferences, four journal publications, and many workshop contributions. I have authored and co-authored a number of competitive planning systems such as the MADLA Planner and MAPlan. Finally, I have significantly contributed to the efforts of the multi-agent planning research community in the practical aspects of planner comparison and in the theoretical aspects of privacy in multi-agent planning.

1.3 Organization and Contributions

In this section, we present the organization and structure of the thesis, together with specifying the contributions of the author and referencing the publications on which the thesis builds. Many of the ideas presented in this thesis are the result of fruitful discussions with Antonín Komenda, the author's supervisor, and all the papers the thesis builds on were co-authored by A. Komenda. Thus we omit the supervisor's contribution as his guidance is behind all of the work.

After the brief introduction, informal problem statement, and objectives of the thesis described in this Chapter, the structure of the remaining chapters is as follows.

Chapter 2 provides a detailed overview of the related work. The related work contains outtakes from [Štolba and Komenda, 2017, Štolba et al., 2015b, 2017] but also novel content.

Chapter 3 provides a formal definition of the MAP problem including a formal definition of privacy and a brief discussion of complexity. The formalisms are based on the works by Domshlak, Brafman, and Nissim ([Brafman and Domshlak, 2008, Nissim and Brafman, 2014]) but were reformulated in our work, e.g., [Štolba and Komenda, 2017], to reflect the factored and distributed nature of multi-agent planning better.

Chapter 4 presents our answer to the **(Objective 1)** in context of the inadmissible relaxation heuristics. The distributed version of the Fast-Forward heuristic based on an excerpt from [Štolba and Komenda, 2013] is described in Section 4.1. The important property of the approach is that the distributed heuristic values are provably equal to the centralized variant. Section 4.2 focuses on a general approach to the effective computation of distributed relaxation heuristics (including the Fast-Forward heuristic again) based on [Štolba and Komenda, 2014]. Finally, Section 4.3 summarizes the most recent approach to distributed Fast-Forward heuristic computation with more focus on privacy, taken from [Štolba and Komenda, 2017]. Evaluation of all the approaches is presented in Section 4.4. The relaxation heuristics are part of the MADLA Planner, conceived and implemented by the author.

Chapter 5 details out the principles of our novel variant of multi-agent heuristic search proposed by the author as an answer to the **(Objective 2)**. The whole chapter is based on [Štolba and Komenda, 2017]. In Section 5.2 we present a high-level description of the search scheme and the ideas behind it, whereas Section 5.3 provides a full formal description. The formalism is used in Section 5.4 to show important properties of the algorithm, such as soundness and completeness. The evaluation is presented in Section 5.5. The search scheme was implemented by the author as part of the MADLA Planner.

Chapter 6 returns once more to the **(Objective 1)**, but this time in the context of optimal planning. Thus the chapter focuses mostly on distributed admissible heuristics but presents a novel search scheme as well (Section 6.3). In particular, Section 6.1 presents an admissible distributed variant of the max relaxation heuristic published in [Štolba et al., 2015a], Section 6.2 presents an admissible distributed landmark heuristic taken from [Štolba et al., 2015a] as well. Section 6.3 presents a search scheme using additive heuristics, whereas Section 6.4 presents an example of such additive heuristic, an admissible (and also privacy-preserving) distributed potential heuristic based on [Štolba et al., 2016a]. The admissible heuristics were created in collaboration with Daniel Fišer, who implemented them as part of the MAPlan planner. The MAPlan planner is a reimplementation of MAD-A* [Nissim and Brafman, 2012] by Daniel Fišer which, in contrast to the original implementation by Raz Nissim, allows for a fully distributed planning. Section 6.5 provides an insight into a general approach to distributed heuristic computation developed by the author and originally published in [Štolba and Komenda, 2016], including a preliminary evaluation by the author. Evaluation of the admissible heuristics and the general approach is presented in Section 6.6. The statistical evaluation of the potential heuristics and the evaluation of the general approach is a novel content of the thesis.

Chapter 7 summarizes the results of our work on privacy in multi-agent planning, that is, the **(Objective 3)**. Section 7.1 provides a formal framework for privacy analysis introduced by the author in [Štolba et al., 2016d, 2017]. The particular sources of privacy leakage identified as a joint work with Jan Tožička (in [Štolba et al., 2016d, 2017] again) are described in Section 7.2. The analysis of privacy leakage in Section 7.3 is the work of the author, partly novel and partly taken from [Štolba et al., 2017] and from papers about the respective algorithms. The theoretical results about the limits of privacy-preserving planning in Section 7.4 are the result of numerous discussions among the author, the author's supervisor and Jan Tožička, originally published in [Tožička et al., 2017b].

Chapter 8 concludes the thesis and presents hints on possible future work.

Appendix A wraps up our work on the Competition of Distributed and Multi-Agent Planners, which was organized by the author, the author's supervisor Antonín Komenda, and Daniel Kovacs. The appendix is composed from the texts published in [Štolba et al., 2015b, Komenda et al., 2016, Štolba et al., 2016b].

Appendix B presents a summary of the published work of the author.

Chapter 2

Related Work

Multi-agent planning lies at the intersection of Automated Planning and Multi-Agent Systems and relates to each of the fields to a variable degree. A taxonomy of MAP formalisms can be based on many properties, but the presence and relation of agents, the observability of the environment and the determinism of the effects of actions are some of the most prominent ones. Such taxonomy is shown in Table 2.1. The Agents axis determines whether there are no agents, which coincides with classical approaches without a relation to Multi-Agent Systems, whether there are cooperative agents, or whether there are adversarial agents, which coincides with approaches based on Game Theory.

Observability of the environment determines whether the agent(s) can observe the entirety of the environment (e.g., in board games such as chess or in classical planning), or just part of it (e.g., in poker). Another facet of uncertainty in the environment is the determinism of the effects of actions. Deterministic actions have always the same effects known to the agents, whereas the effects of nondeterministic actions may depend on an unobservable part of the environment or may include an element of chance. If the probability distribution of the effects of the action is known or can be presumed, we talk about stochastic actions.

Table 2.1 shows some of the best-known formalisms related to planning, based on the taxonomy introduced above. The most basic model is the classical planning (e.g., STRIPS [Fikes and Nilsson, 1971]), whereas the most general model is Partially Observable Stochastic Games (POSG). Let us start from the classical planning. By allowing nondeterministic actions in planning with no agents, the related model is either Contingent Planning (if the agent must be sure to achieve the goal) or Fault-tolerant Planning (in the case there is a limited number of possible unwanted effects). If the probability distributions of the nondeterministic effects are known (i.e., stochastic actions) the applicable model is Markov Decision Process (MDP). In the presence of multiple cooperative agents, the model needs to change to Multi-Agent MDP (MMDP) [Boutilier, 1999], Factored MDP [Guestrin et al., 2002], or Dec-MDP [Bernstein et al., 2002]. In MMDP, each agent can observe the whole environment, but communication has an associated cost or is not available. In MAP, we do not consider an explicit cost of communication, but as communication is typically orders of magnitude slower than local computation, the communication may become costly in terms of the solution time. In Factored MDP, each part (factor) of an MDP is respective to one of the agents, thus each agent can perform a set of actions. The Factored MDP can be transformed to a single monolithic MDP with an exponential number of joint actions of the agents (i.e., single joint action consists of a single action for each agent). This idea of factorization is used also in factored classical planning and in MA-STRIPS. In MA-STRIPS, the classical planning problem is factored so that each factor belongs to a single agent (see Section 3.1 for detailed formal definition). In our work (as in the majority of MA-STRIPS literature), we consider the actions of the agents to be performed sequentially and thus there are no joint actions. This significantly simplifies the model and algorithms and avoids an exponential blowup of the search space (see Section 3.5). Finally, Dec-MDP allows no communication and assumes that each agent can observe its portion of the state, together

Observ-ability	Actions	No Agents	**Cooperative Agents**	Adversarial Agents
Partial	Nondet.	POMDP	Dec-POMDP	POSG
	Det.	Conformant Planning		
Full	Nondet.	MDP, Contingent Planning, Fault-tolerant Planning	MMDP, Factored MDP, Dec-MDP	Stochastic games
	Det.	Classical/STRIPS	Factored Planning, **MA-STRIPS/MA-MPT,**	Perfect Information Games
			PP-MAP	

Table 2.1: Taxonomy of planning models depending on the environment (observability and action determinism) and the presence and relation of agents. The model used in the thesis is in bold. Used acronyms:
MDP - Markov Decision Process
POMDP - Partially Observable MDP
STRIPS - Stanford Research Institute Problem Solver (classical planning model)
Dec-POMDP - Decentralized POMDP
MMDP - Multi-agent MDP
Dec-MDP - Decentralized MDP
MA-STRIPS - Multi-Agent STRIPS
MA-MPT - Multi-Agent Multi-valued Planning Task
PP-MAP - Privacy-Preserving Multi-Agent Planning
POSG - Partially Observable Stochastic Games

resulting in a full observation of the state. This coincides with our view on the public and private parts of the states. Together, the agents have a complete view of the global state, but each agent can observe only the part accessible to it. In our case we allow communication, but the communicated information is restricted in order to maintain privacy.

The model of sequential decision making in the presence of adversarial agents is known as Stochastic Games. When the environment is partially observable, the difference between deterministic and nondeterministic actions is often diminished for reasons already mentioned (in a partially observable environment, the applicability and thus the effects of actions may be determined by the unobservable part of the environment). Also as the effect of the actions may be subject to partial observability, the agent cannot be certain that the desired effects occurred. This holds for Partially Observable MDPs (POMDPs), their decentralized cooperative variant Dec-POMDPs [Bernstein et al., 2002] and also for the adversarial variant, that is, POSGs. An in-depth overview of techniques for stochastic multi-agent planning (and Dec-POMDPs in particular) can be found in [Oliehoek and Amato, 2016] together with an implementation of various models and algorithms in the Multi-Agent Decision Process Toolbox (MDPT) [Oliehoek et al., 2015]. Somewhat different is Conformant Planning, where the assumption is the actions are deterministic, but there is no observability at all. In order to achieve the goal, a conformant plan must deal with all possible variants of the initial state, as this uncertainty encodes the unknown about the environment.

Sometimes, privacy is considered as a special case of partial observability, where the agents can observe only the public part of the environment and the part private to the particular agent, but cannot observe the private parts of other agents. The difference is, that in privacy, the agents could possibly obtain the private parts, if the respective agents were willing to provide the information. Moreover, in the privacy setting, preventing other agents to observe the private information of an agent is not considered enough. Instead, the agents want to prevent other agents to even infer the private information from the public part of the executed protocol. Privacy is reasonable mostly in the context of cooperative agents

Planning By	Planning For: Single Agent	Planning For: **Multiple Agents**
Multiple Agents	Distributed/Factored Planning	**Multi-Agent Planning**
Single Agent	Classical Planning	Classical Planning

Table 2.2: Coordination schemes of planning schemes according to [Decker, 1987].

which want to solve the planning problem together, but without providing (or leaking unintentionally) the private information. The privacy preservation in multi-agent cooperative setting is one of the topics of this thesis (see Chapter 7), in particular with deterministic actions, and the related work is discussed later in this chapter. To our best knowledge, at the point of writing of this thesis, there was no such formalism applicable to nondeterministic actions or partial observability.

2.1 Roots of Multi-Agent Planning

The first mention of Multi-Agent Planning can be traced back nearly as far as STRIPS itself – in 1980 Nillson&Konolige published a paper titled "Multiple-Agent Planning Systems" [Konolige and Nilsson, 1980], in which they presented a high-level extension of STRIPS towards multiple agents. Even a year older is the work on the distributed NOAH planner [Corkill, 1979] which is one of the first partial-order planning (POP) system that generates gradual refinements in the space of (abstract) plans using a representation similar to today's Hierarchical Task Networks (HTNs) [Nau et al., 2004]. Since then, the topic has been active mainly in the multi-agent systems community. One of the most cited works on distributed and multi-agent planning is [Durfee, 1999], which describes basics of possible coordination schemes for planning agents (Table 2.2). The point of view on multi-agent planning from the multi-agent community is extensively summarized in [De Weerdt et al., 2005, de Weerdt and Clement, 2009].

Here we mention a selection of the most prominent works in the area of MAP from the multi-agent systems point of view and relate them to the more current approaches. In the domain-specific Partial Global Planning (PGP) [Durfee and Lesser, 1991], agents build their partial global view of the planning problem, and the search algorithm finds local plans in the agents' plan-space that can be then coordinated to meet the goals of all the agents. PGP reasons not only about plans but also roles and responsibilities of the particular agents. PGP was first introduced only for a specific problem of a distributed sensor network for vehicle monitoring. The Generalized PGP (GPGP) [Decker and Lesser, 1992] is a domain-independent extension of PGP that separates the process of coordination from local scheduling, which enables agents to communicate more abstract and hierarchically organized information and has less coordination overhead.

The TALPlanner [Doherty and Kvarnström, 2001] is a temporal forward-chaining planner that searches through the space of states. For efficiency reasons, TALPlanner uses additional domain-specific information in form of temporal logic formulas. TALPlanner is also able to generate multi-agent plans with parallel actions.

The DPGM [Pellier, 2010] is a distributed agent-based planner built on the planning graph structure introduced in Graphplan [Blum and Furst, 1997] using a Constraint Satisfaction Problem (CSP) [Dechter, 2003] formulation both to extract the local plans and to coordinate the agents. The solving process is iterative where in each iteration the agents add more information to the planning graph until the solution can be reached. DPGM was implemented and experimentally evaluated by [Durkota and Komenda, 2013]. Also note that similar ideas of distributed planning graph construction were later used for the construction of their relaxed variants [Torreño et al., 2013, Štolba and Komenda, 2013].

Another well known classical planning technique is to convert the planning problem to SAT representation (SATPLAN [Kautz, 2006]). This technique was extended to multi-agent setting by [Dimopoulos et al., 2012] as μ-SATPLAN. μ-SATPLAN performs an a priori distribution of the MAP task goals

among the agents. Similarly to DPGM, agents follow an iterative response planning strategy, where each participant takes the previous agent's solution as an input and extends it to solve its assigned goals via SATPLAN. μ-SATPLAN assumes that each agent can solve its assigned goals by itself.

Similarly to the TALPlanner, TFPOP [Kvarnström, 2011] is based on temporal reasoning and durative actions where also parallel actions are commonly considered. TFPOP is a centralized approach that synthesizes a solution for multiple executors using the techniques of forward-chaining partial-order planning (POP). The approach based on POP planning was later used by the MAP-POP [Torreño et al., 2012] and FMAP [Torreño et al., 2014] planners, but without the temporal reasoning.

A bridge between classical planning and MAP is factored planning (FP) [Amir and Engelhardt, 2003, Brafman and Domshlak, 2006]. Factored planning (FP) leverages the idea of decomposing classical planning problems into a number of mostly independent sub-problems. FP provides a direct link from classical planning to MAP, where the factorization is based on the multi-agent structure. This link was first proposed in [Brafman and Domshlak, 2008] where the authors focused on a precise formulation and computational complexity guarantees of cooperative deterministic domain-independent multi-agent planning (DMAP). The formulation was based on an extension of classical planning model STRIPS to multi-agent setting coined MA-STRIPS. In this thesis, we focus on models of Multi-Agent Planning based on or closely related to MA-STRIPS.

One significant difference between FP and MAP is that in MAP, the decomposition of the problem to agent sub-problems is assumed to be given a priori, whereas in FP finding a good factorization is the main research question. A recent advancement in this direction is the fork decompositions [Gnad and Hoffmann, 2015]. A significant exception to this distinction is the ADP Planner [Crosby et al., 2013] which is a centralized planner which automatically determines a decomposition on agents and then uses a MAP algorithm to find a solution. Note that ADP neither uses a MA-STRIPS-based formalism nor considers privacy in any way.

Other interesting works on the boundaries of FP and MAP are Distoplan [Fabre et al., 2010] and A# [Jezequel and Fabre, 2012]. Distoplan is a factored planning approach that exploits independence within a planning task. In Distoplan, each factor of the global task is represented as a Deterministic Finite Automata (DFA), which recognizes the regular language formed by the local valid plan of the component. This way, all local plans are manipulated at once and a generic distributed optimization technique enables to limit the number of compatible local plans. A similar idea was later used in the PSM planner [Tožička et al., 2014]. Similarly, A# is a factored A* search that finds a path to the goal in each factor of a problem and ensures that the actions that must be performed jointly (i.e., are shared among multiple factors) are compatible. A# runs a modified version of the A* algorithm iteratively in parallel in each factor informing each other about promising and "useless" local plans.

Practically none of the MAP planners described above were experimentally compared against each other. Some were not evaluated at all (e.g., A#), some were evaluated against classical planners (e.g., ADP). The major reasons were not only the vast diversity of incompatible models used in MAP and often the inclusion of domain-specific knowledge (PGP, TALPlanner), but also the missing common MAP problem definition language. In classical planning, PDDL [McDermott et al., 1998] is a widely accepted standard. In MAP, there were attempts to formalize a multi-agent extension such as Multi-Agent Planning Language (MAPL) in [Brenner, 2003] and Multi-Agent PDDL (MA-PDDL) in [Kovacs, 2012] but neither of those was accepted by a wider community.

Since its publication, the MA-STRIPS [Brafman and Domshlak, 2008] model has been widely accepted as a common extension of the classical planning STRIPS model, thus allowing a better comparison of MA-STRIPS-based planners, but a common definition language and a set of benchmarks was still missing. As stated in Section 1.2, tackling this issue is one of the goals of this thesis. We have co-organized the Competition of Distributed and Multi-Agent Planners (CoDMAP) [Štolba et al., 2016b, Komenda et al., 2016], proposed a new variant of the MA-PDDL language, modified and improved for the CoDMAP competition by the author of the thesis in collaboration with Daniel Kovacs and Antonín Komenda, together with a set of benchmarks. The competition successfully became a standard reference for planner comparison and thus helped to improve the way MAP planners are experimentally evaluated.

Details on the competition version of MA-PDDL and the competition itself are included in the thesis in Appendix A.

Modern MAP planners not based on MA-STRIPS

The vast majority of modern MAP planners adheres to or is compatible with the MA-STRIPS model, but there are some MAP planners which participated in the CoDMAP competition (and thus can take MA-PDDL as input) but are not based on MA-STRIPS. ADP (Agent Decomposition-based Planner) by [Crosby et al., 2013, Crosby, 2015] is based on the idea of automatic decomposition of classical planning problems to multiple agents. Each agent is represented by a set of internal variables and variables not internal to any agent are considered to be environment variables (public). The problem is decomposed based on the structure of causal dependencies of the actions. The planning process itself then interleaves subgoal calculation based on relaxation and search phases, until a solution is found or its non-existence reported.

Another centralized planner which develop some aspects of MAP is the MAP-LAPKT planner by [Muise et al., 2015] which is based on the following idea: *"The task of planning for multiple agents with heterogeneous access to fact observability can be solved by classical planners using the appropriate encoding"*. Thus the multi-agent planning problem is compiled into a classical planning problem and solved by a planner from the LAPKT toolkit [Ramirez et al.]. Any action that uses a fact private to other agent cannot be executed by the agent, therefore the compilation must make sure such actions are excluded from the grounding. Clearly, this is satisfied by the MA-STRIPS factorization by definition and thus MAP-LAPKT provides no added value in that case.

The MARC planner by [Sreedharan et al., 2015] is a centralized multi-agent planner designed to use the theory of required cooperation to solve a subset of large multi-agent problems by compiling them into problems with a smaller number of agents by using the notion of transformer agents. Transformer agents are special virtual agents created by merging multiple agents from the original domain and problem definition. Transformer agents are then capable of transforming into any agent from the original domain (and thus are able to use all the capabilities of the transformed agents). Transformer agent's actions are translated into sequences of actions of particular agents and agent-independent effects are planned using a fast classical planner. Compiling multiple agents into one goes against the MA-STRIPS concept of privacy and thus makes this planner less relevant to MA-STRIPS.

2.2 MA-STRIPS-based Multi-Agent Planning

The work by [Brafman and Domshlak, 2008] ignited more interest in the topic of Multi-Agent Planning based on classical formalisms. The paper formally introduced a minimalistic extension of STRIPS known as MA-STRIPS and have shown that the complexity of MA-STRIPS-based MAP is not exponentially dependent on the number of agents, but rather on the tree-width of their interaction graph and the minimal number of interactions needed to solve the problem (see Section 3.5 for more details). Such results suggested, that at least for loosely coupled problems (where the tree-width is low), the approach may work well.

The first practical MA-STRIPS-based planner is Planning First [Nissim et al., 2010] which is based on the same principles used in the theory of [Brafman and Domshlak, 2008], that is, solving a Distributed Constraint Satisfaction Problem (DCSP [Yokoo et al., 1998]) over the possible plans of each agent. The constraints are used to ensure that the plans are coordinated, which means that they use the same public actions in the same order. An adaptation of the well known A* algorithm [Hart et al., 1968] for MA-STRIPS followed, denoted as Multi-Agent Distributed A* (MAD-A*) [Nissim and Brafman, 2012] (and a parallel version with shared memory MAP-A*). The results of MAD-A* showed that in loosely coupled problems it can outperform the classical A* (and the parallel version MAP-A* outperforms state-of-the-art multi-core approaches). The distributed search was later generalized to

Multi-Agent Forward Search (MAFS) [Nissim and Brafman, 2014]. In 2015, the vast majority of MA-STRIPS-based planners were compared in the Competition of Distributed and Multi-Agent Planners (CoDMAP [Komenda et al., 2016]). The planners are presented later in this chapter and the description of the competition itself is a part of the thesis (see Appendix A).

Taxonomy of MAP planners

From the multi-agent systems point of view, a significant property of a MA-STRIPS-based planner is whether it is centralized or distributed. A centralized planner is using the MA-STRIPS agent factorization, but the whole computation runs on a single computer, possibly using multiple threads, but also possibly with some shared memory or shared computation, e.g., for some preprocessing tasks. An example of a (partially) centralized planner is the MAD-A* planner which uses a centralized preprocessing step (translation to multi-valued variables from Fast Downward [Helmert, 2006]), but the actual planning is performed using separate computation threads and communication via TCP/IP. A distributed planner has to be able to run on completely separate machines, communicating only via TCP/IP (or any other protocol). For example, the MAPlan [Fišer et al., 2015] planner (implementing the MAD-A*/MAFS distributed search algorithm) is capable of running in such a distributed setting.

A general property of MAP planners is the coordination mechanism, which in MA-STRIPS can either be state-based or plan-based. In state-based coordination, the agents exchange states (or parts of states), an instance of such coordination mechanism is used in MAD-A* where some of the states reached by an agent are sent to the other agents. In plan-based coordination, the agents exchange plans (or partial plans), regardless of how they are found locally. An example of plan-based coordination is the Planning-First algorithm, where the local plans are computed using the classical FF planner [Hoffmann and Nebel, 2001] and the correct combination of plans is found using a DCSP.

We may also focus on a more general interplay between coordination part of the process, where the agents exchange information on their constraints and possible solutions, and the plan synthesis part of the process where the agents actually generate the global plan or local plans. Possible categories are the following. In pre-planning coordination, the agents first exchange their constraints and then synthesize the solution either cooperatively or locally, e.g., as in DPP [Maliah et al., 2016d]. In post-planning coordination, the agents first generate their plans and subsequently attempt to coordinate them, e.g., by plan merging [Luis and Borrajo, 2015]. In iterative (best) response coordination, the agents generate plans in turn, taking the results of the previous process as a part of their input, e.g., the DPGM planner [Pellier, 2010]. In interleaved synthesis and coordination, both parts of the process are interleaved, e.g., as in MAD-A*, where the states are communicated during the plan synthesis itself. Multiple approaches may be also combined as in the PSM planner [Tožička et al., 2014], which proceeds in multiple iterations and each iteration follows the post-planning coordination approach.

A very important property of a MAP algorithm or planner is the way how it treats privacy. The work related to privacy is discussed in Section 2.3 and our contribution to the topic of privacy is presented in Chapter 7.

2.2.1 MA-STRIPS-based Planners

Among the earliest MA-STRIPS planners are the Planning First and MAD-A* planners already described. Another early planners are MAP-POP [Torreño et al., 2012] which is a partial-order planner which later evolved into FMAP [Torreño et al., 2013, Torreño et al., 2014] and MH-FMAP [Torreño et al., 2015]. The mentioned planners do not use MA-STRIPS explicitly, but the used formalism is a generalization of MA-STRIPS and thus the planners can be considered MA-STRIPS-based.

State-based coordination planners The first MAP planner using state-based coordination is MAD-A*. There is a number of MA-STRIPS planners based on the MAD-A* and MAFS principle, namely MAPlan [Fišer et al., 2015], which is a fully distributed reimplementation of the distributed heuristic

forward search, and the MADLA Planner [Štolba and Komenda, 2015, 2017], which is a variant of MAFS using an intricate multi-heuristic search. The MADLA Search is a major part of this thesis, see Chapter 5. Variants of MAFS aiming at improved privacy are Secure-MAFS [Brafman, 2015], Macro-MAFS [Maliah et al., 2016c] and Forward-Backward MAFS [Maliah et al., 2016b].

Plan-based coordination planners Besides the Planning First a planner based purely on the plan-based coordination scheme is PSM and its variants, PSM-VR and PSM-VRD by [Tožička et al., 2014, 2015] which are based on the idea of Planning State Machines (PSM). A PSM is basically a finite automaton representing a set of plans. Such PSMs can be easily projected to a public part of the problem and merged. These two operations can be used to build a public PSM consisting of merged public parts of individual PSMs, which are gradually extended by new plans. Once the public PSM projection is not empty, a solution to the multi-agent planning problem has been found. PSM-based planners preserve privacy by not revealing any of the private actions and private preconditions and effects of public actions.

Somewhat related to PSM is the DPP [Maliah et al., 2016d] planner. It starts by analyzing the agent problems using a regression and enriching the public projection of a problem by private dependencies between public actions, and between public actions and initial state. The planner then uses a classical planner to solve this projected problem and classical planners to fill-in the necessary private actions for each agent. This is closely related to the process of the PSM planner, but the way how the public solution is found differs, and also it is not guaranteed that such public solution is extensible by all agents and thus DPP is not complete. Nevertheless, the incompleteness does not demonstrate on the CoDMAP benchmarks where DPP performs exceptionally well. The privacy of DPP is somewhat disputable. It clearly is weak privacy-preserving, but it would probably fail in terms of privacy leakage (e.g.,[Štolba et al., 2017]) as the private dependencies are explicitly revealed. The authors present a notion of privacy (object cardinality privacy), which is preserved by the planner and which essentially means, that the number of interchangeable private objects (e.g., packages in logistics) is not revealed.

Hybrid coordination planners Whereas the MAFS-based planners are typical cases of state-based coordination planners, the following planners lie somewhere between state-based and plan-based coordination. The GPPP [Maliah et al., 2014], and PP-LAMA [Maliah et al., 2016a] planners proceed in two phases, (i) a public plan is found using the public actions (and private actions in delete relaxation form) while the private facts are hidden (encrypted), (ii) the public plan is attempted to be extended by private actions in order to verify it as a global solution. If the extension fails, the planning continues with phase (i).

Decomposition-based planners Another direction of MA-STRIPS based planners are the MAPR and CMAP planners by [Borrajo, 2013, Borrajo and Fernández, 2015] which proceed as follows: (i) generate an obfuscated problem and domain file for each agent, (ii) assign public goals to agents based on relaxed reachability analysis, and (iii) plan using a state-of-the-art classical planner. The two planners differ in the strategy used. MAPR agents augment the solution found so far in a round-robin fashion, providing its obfuscated solution to the next agent and CMAP merges the obfuscated domains and problems into a single domain and single problem. The PMR (Plan Merging by Reuse) planner by [Luis and Borrajo, 2014, 2015] applies some of the concepts used in MAPR and CMAP. It uses the same principle of obfuscation and goal selection strategies, thus the first two steps of the algorithm are the same. PMR then continues with the following: (i) each agent plans using its obfuscated domain/problem pair including the assigned subgoals (the resulting plans from all agents are then concatenated), (ii) plan reuse (repair) if the plan found is not sound.

Comparison of MA-STRIPS-based planners

The recent Competition of Distributed and Multiagent Planners (CoDMAP 2015) [Štolba et al., 2016b, Komenda et al., 2016] compared the most of MA-STRIPS-based planners available at that time. The

Planner	complete	optimal	MA-PDDL		distributed	privacy	coordination
			factor.	privacy	track	type	
MADLA	✓	×	×	×	×	E	S
MAPR	×	×	✓	✓	×	O	O
CMAP	×	×	✓	✓	×	O	O
PMR	×	×	✓	✓	×	O	O
FMAP/MH-FMAP	✓	×	✓	✓	✓	P	S
MAPLan	✓	✓/×	✓/×	✓/×	✓	E	S
PSM	✓	×	✓	✓/×	✓	P	P
MAP-LAPKT	✓	×	✓	✓	×	T	O
MARC	✓	×	✓	✓	×	T	O
GPPP	✓	×	✓	✓	-	E	P/S
PP-LAMA	✓	×	✓	✓	-	E	P/S
MAFS/MAD-A*	✓	✓/×	×	×	-	E	S
Macro-MAFS	✓	×	✓	✓	-	S*	S
DPP	✓	×	✓	✓	-	S†	P
MAP-POP	✓	×	×	×	-	P	P
Planning First	✓	×	×	×	-	P	P

Table 2.3: Properties of MA-STRIPS-based planners with existing implementations. In the first part are the planners competing in CoDMAP 2015. The complete and optimal columns use the common meaning of the terms. The MA-PDDL column describes whether the planner used the definitions provided in the MA-PDDL input or derived them based on MA-STRIPS principles. The privacy column refers to the following treatments of privacy:
S - provably strong (the planner is provably strong privacy preserving),
P - the planner respects the weak form of privacy by not communicating private facts and actions at all,
E - encryption of states (private parts of states are encrypted, at least theoretically),
O - obfuscation of PDDL (the planner hides the private parts of the complete problem either by renaming them, or by encrypting them),
T - translation (the planner translates the problem into a different one, taking privacy into account).
The Coordination column refers to the following coordination mechanisms:
S - state-based, the agents exchange (partial) states
P - plan-based, the agents exchange (partial) plans
O - other coordination mechanism
* Only for a class of logistics problems.
† For a specialized notion of privacy (object cardinality privacy).

competition was divided into two tracks, the centralized track, where the planners were allowed to run on a single computer for all agents (indeed, it was not even enforced to separate the memory spaces of the agents) and the distributed track, where each agent had to run on a separate machine and communicate over TCP/IP. Table 2.3 summarize the properties of planners discussed in this section which have participated in the centralized and distributed track of CoDMAP respectively. The description of the competition is a part of this thesis, see Appendix A. The most relevant properties of the MA-STRIPS based planners are summarized in the Table 2.3.

2.2.2 MA-STRIPS-based Heuristics

Since MAD-A*, heuristics are a crucial part of MA-STRIPS-based planning. In MAD-A* and MAFS, the only considered approach to heuristic computation in MAP was projected heuristic. A projected heuristic is any heuristic computed on the projected problem of each of the agents separately, not taking the other agents in consideration. This practically equals to computing a heuristic on an abstract

problem, where all variables of the other agents are abstracted away. This approach has considerable positives, as it is fast to compute, independent of other agents (and their number) and maintains admissibility. On the other hand, the result of projected heuristic can arbitrarily underestimate the actual optimal heuristic of the global problem as shown in Chapter 6.

An alternative approach is to compute the heuristic in a distributed manner by all (or a subset of) the agents. the first heuristic to undergo such treatment was the FF heuristic [Hoffmann and Nebel, 2001] with several distributed variants ([Torreño et al., 2012, Štolba and Komenda, 2013]) and relaxation heuristics in general by [Štolba and Komenda, 2014]. A more effective variant was proposed by [Štolba and Komenda, 2017]. Most of the approaches are a part of this thesis, see Chapters 4 and 6.

Another classical heuristics with distributed variants are a heuristic based on Domain Transition Graphs (DTG) [Helmert, 2006] by [Torreño et al., 2014], a number of distributed heuristics based on the idea of landmarks ([Richter and Westphal, 2010, Helmert and Domshlak, 2009]) by [Štolba et al., 2015a, Maliah et al., 2016a, Torreño et al., 2015], a distributed variant of abstraction heuristic by [Maliah et al., 2015], and a distributed variant of potential heuristics ([Pommerening et al., 2015]) by [Štolba et al., 2016a]. A general approach to distributed heuristic computation based on cost-partitioning ([Katz and Domshlak, 2010]) was published in [Štolba and Komenda, 2016].

2.3 Privacy in Multi-Agent Planning

In this section, we give a closer look at the background and work done in the context of privacy in multi-agent planning. First, we present the research in secure multiparty computation, then we summarize how privacy is treated in fields related to MAP and finally we describe the work on privacy published in multi-agent planning so far.

2.3.1 Secure Multiparty Computation

Secure multiparty computation (MPC) [Yao, 1982, 1986] is a subfield of cryptography, which studies computing a function f by a set of n parties $p_1, \ldots, p_n$ such that each p_i knows part of the input of f. The goal of MPC is to compute f in a way that no party p_i learns more information about the inputs of other parties than what can be learned from the output of f. Clearly, PP-MAP is an instance of MPC, where the respective problems of the agents are the inputs and the global plan is the desired output.

In an ideal world, assuming secure communication channels, a trusted third-party could receive the inputs from the parties, perform the needed computation, and return the solutions to the respective parties. Secure MPC studies whether and how such computations can be done in the real world without the trusted third-party, and alternatively how much private information leaks in comparison to the ideal execution. In some cases, the trusted third-party can be replaced by a relatively small number (e.g., three) computation agents which can be trusted not to collude as each of them is controlled by a different party. Such approach is taken by some real-world secure MPC solutions, such as Sharemind [Bogdanov et al., 2008].

In MPC, assumptions are typically placed on the participating parties (agents in our case) and their communication and computation capabilities. Common assumptions are the following:

- There is no trusted third-party.
- The planning agents are semi-honest (or honest but curious).
- The computation power of the agents is either
  - unbounded (information-theoretic security), or
  - polynomial-time bounded (computational security).

The assumption of semi-honest agents means, as opposed to malicious agents, that every agent follows the rules of the computation protocol based on its input data, but after the computation is finished, it is allowed to use any information it has received during the protocol to compromise the privacy. The computation power of the agents (which can be used to infer additional knowledge from the executed protocol) is typically seen either as unbounded, in which case we are talking about information-theoretic security, or polynomial-time bounded, which is the case of computational security.

When applied to PP-MAP, the notion of polynomial-time bounded adversary may seem somewhat less suitable, as the planning itself is not polynomial (but PSPACE-complete [Bylander, 1994]). Nevertheless, the computation power of the agents is still polynomial, thus allowing to solve either polynomial instances or small instances. For such instances of planning problems which can be practically solved, the cryptographic assumptions (such as that the factoring of large integers is hard), for which the polynomial-time bound is typically used are still valid.

There are basically two approaches to multi-agent planning based on the MPC techniques. The first approach is to encode planning in some general MPC technique such as cryptographic circuits [Yao, 1986], oblivious RAM (ORAM) [Goldreich, 1987] or blind Turing machine (BlindTM) [Rass et al., 2015]. For example, the cryptographic circuits encode the whole computation of a function into a boolean or algebraic circuit, which can be then securely evaluated using some of the existing secure protocols. The problem related to MAP is, that the worst-case scenario has to be encoded, that is, the complete exploration of the search space, which itself is exponential in the size of the MAP input (e.g., MA-STRIPS). Therefore, it is not clear how exactly PP-MAP would be encoded in such general model, whether it is even feasible, and what the overhead of such encoding would be.

The second approach is to devise a specific PP-MAP algorithm based on MPC primitives, such as private set intersection [Li and Wu, 2007]. There is a number of solutions for a related problem, shortest path in a graph [Brickell and Shmatikov, 2005]. Such techniques solve the shortest path problem for an explicit graph, typically represented by a matrix. In classical planning and subsequently in MAP, the explicit graph (that is, the transition system) is exponential in the problem size, which, for practical problem sizes, makes it impossible to use such explicit (e.g., matrix) representation. So far, the only published (theoretical) MAP planner using MPC primitives is our [Tožička et al., 2017b]. A secure linear program computation [Dreier and Kerschbaum, 2011, Mangasarian, 2011] was used in our secure multi-agent version [Štolba et al., 2016a] of the potential heuristic [Pommerening et al., 2015].

Moreover, the MPC literature provides tools for privacy leakage quantification, such as Quantitative Information Flow (QIF) [Braun et al., 2009, Smith, 2009]. In QIF, the information leakage is quantified based on the difference in the probability of guessing the right input of a function before and after the distributed computation. A high-level formula defined in [Smith, 2009] is

$$\text{information leaked} = \text{initial uncertainty} - \text{remaining uncertainty}$$

where the initial uncertainty is related to the probability of guessing the right input without any additional knowledge gained from the execution of the algorithm, whereas the remaining uncertainty is the probability of guessing the right input given the output of the particular execution. The QIF approach is the basis of our theory of privacy leakage quantification for MAP [Štolba et al., 2017] presented in Chapter 7.

2.3.2 Privacy in Related Fields

As already mentioned, there is a number of secure algorithms for related graph problems, such as the shortest path problem. In [Brickell and Shmatikov, 2005] the authors provide a number of algorithms for solving problems on a joint graph of two semi-honest agents such that the agents do not learn additional information about the respective sub-graphs. The authors provide algorithms for set union, all pairs and single source shortest distance and a minimal spanning tree. As the authors use an explicit matrix representation of the graph, the proposed techniques are not applicable to planning, where the

graph representing the whole transition system is too large. Moreover, the proposed techniques are not applicable to single-source single-destination shortest path problem, which is the desired case in MAP.

The authors in [Aly et al., 2013] take the work on secure graph algorithms a step further by using general building blocks of [Damgård and Nielsen, 2003] and [Toft, 2007] thus providing more general algorithms, even for graphs with private structure, which corresponds to the MAP case. Nevertheless, the algorithms are again bound to an explicit adjacency matrix representation of the graphs which is not applicable for MAP problems in general. That said, the approach proposed by [Toft, 2007] may prove to be a valuable tool in designing a privacy-preserving MAP planner based on secure data structures such as priority queue [Toft, 2011].

Another related work is that on oblivious data structures and computation such as [Blanton et al., 2013] which are in general based on the idea of Oblivious RAM (ORAM) [Goldreich, 1987]. Also, the approach was originally used for different scenarios, it can be used to implement also oblivious and thus secure data structures for MPC [Keller and Scholl, 2014].

A problem closely related to planning is Constraint Satisfaction Problem (CSP) [Dechter, 2003] and Constraint Optimization Problem (COP) [Dechter, 2003]. The distributed multi-agent variant is known as DCSP [Yokoo et al., 1998]. For DCSP the privacy has been studied in a number of works. The authors in [Faltings et al., 2008] provide a formal definition of a number of privacy variants related to DisCSP (e.g, agent privacy, topology privacy, etc.) and show how an existing DisCSP algorithm can be modified to adhere to some of the privacy definitions. In [Greenstadt et al., 2006] the authors experimentally evaluate privacy leakage of a number of existing DisCPS algorithms on a number of domains. The authors use specific privacy leakage analysis technique for each of the DisCSP algorithms to exploit its weaknesses.

2.3.3 Privacy in MAP

The formal treatment of privacy in MAP was for a long time neglected, with the exception of privacy leakage quantification by [Van Der Krogt, 2009]. The author proposes a privacy measure related to ours described in Chapter 7, but in stead of possible transition systems, the authors count the plans consistent with the information gained. This approach has a weakness that it cannot be used to analyze the privacy leakage from the point of view of the adversary (and to, for example, reconstruct the search tree) as the knowledge of the possible plans is necessary to perform the analysis. Also, the approach is not based on MA-STRIPS.

A formal treatment of privacy for MA-STRIPS was proposed in [Nissim and Brafman, 2014] and later extended in [Brafman, 2015], loosely based on the concepts of secure MPC. First, the authors specify what is the private information the agents are attempting to hide, second, the authors present two degrees of privacy preservation, *weak* and *strong* privacy.

The weak privacy preservation can be rephrased as: *"Any algorithm that does not communicate the private information in a plain, understandable way is weak privacy-preserving."* This means that it is enough to encrypt the private parts or omit them from the communication altogether. The concept of weak privacy is rather artificial and unusable in any real scenario concerning privacy, as arbitrary private data can be possibly deduced. Our recent work [Štolba et al., 2016c,d, 2017] aims at quantifying the privacy leakage and thus providing a finer-grade evaluation of practical usability of planning algorithms.

The strong privacy preservation can be rephrased as: *"If by execution of an algorithm, the agents do not obtain and cannot deduce any private information additional to what can be deduced only from the public input and public output of the algorithm, the algorithm is strong privacy-preserving."* This means that by executing a planning algorithm, the agents learn no new private information of the other agents. Of course, some private information may be deduced already from the structure of the public input and public output (that is, the public actions in the plan used to coordinate the agents). It was shown that the theoretical Secure-MAFS [Brafman, 2015] (and its implementation Macro-MAFS [Maliah et al., 2016c]) algorithm is strong privacy-preserving on a restricted class of logistics problems, but not in general. In a recent work [Tožička et al., 2017b], the authors have proposed a class of planners based

on the PSM planner [Tožička et al., 2014] and secure finite automaton intersection [Guanciale et al., 2014], which can guarantee to be strong privacy-preserving, but at the cost of either being incomplete, or impractically inefficient. Indeed, the most important theoretical result of [Tožička et al., 2017b] is that no MAP planner based on the most commonly used principles, such as distributed state-space search, can be strong-privacy preserving, complete, and practically efficient at the same time. For more details on the topic, see Section 7.4. In [Tožička et al., 2017a] the authors have shown, that a PSM-based planner can get arbitrarily close to strong privacy (that is, achieve an ϵ-strong privacy), but again, at the cost of gradually losing efficiency.

Other concepts of privacy have also been investigated in relation to MAP. The idea of agent privacy, where the number and identity of agents is hidden, was first introduced in the context of DCOP by [Faltings et al., 2008]. An adaptation of this concept to MAP was proposed in [Maliah et al., 2016b] together of a variant of the MAFS algorithm [Nissim and Brafman, 2014] preserving agent privacy. Apart from hiding the existence and identity of the agents, the proposed planner Forward-Backward MAFS also sends significantly fewer messages and improves the overall planning speed. Another privacy variant is object cardinality privacy [Maliah et al., 2016d], where the types of objects may be revealed, but not the numbers of instances of each type. The authors propose a planner DPP which does preserve the object cardinality privacy and weak privacy, but not strong privacy and which outperforms a number of MAP planners.

2.3.4 Privacy-Preserving Planners and Heuristics

The planners (and heuristics) in the literature claiming to be privacy-preserving can be divided into three groups based on the approach to privacy. The first group is the planners which use obfuscation as means of hiding the private information. This means, that the private propositions and actions are renamed in order to hide their identity. Although cases, where this approach is reasonable, can be imagined, e.g., when the names describe some particular business assets which should not be disclosed, in general, the obfuscation cannot be considered to adhere even to weak privacy, as the existence of private actions and private preconditions (i.e., an isomorphic image of the problem) is disclosed. The planners using obfuscation are MAPR [Borrajo, 2013], CMAP [Borrajo and Fernández, 2015] and the PMR (Plan Merging by Reuse) planner by [Luis and Borrajo, 2014, 2015].

The largest group of planners adhere to the weak privacy definition, that is, they do not communicate the private parts of the problems in a plain, unencrypted way. There is a number of planners based on the MAFS distributed state-space search principle ([Nissim and Brafman, 2012, 2014, Štolba and Komenda, 2015, Fišer et al., 2015]) which are in theory weak privacy preserving, but only MAPlan [Fišer et al., 2015] adheres to the principles also in practice, using fully distributed PDDL translation and search. The Planning First [Nissim et al., 2010] planner also theoretically fits in the weak-privacy class, whereas the implementation violates it. The GPPP [Maliah et al., 2014], and PP-LAMA [Maliah et al., 2016a] are weak privacy-preserving planners as in the public solution phase, the private actions are used only locally and the private facts are hidden by encryption. The FMAP planner [Torreño et al., 2014, 2015] is based on the forward-chaining plan space search principle, where partial plans are gradually extended towards the final plan. Private actions, private preconditions, and private effects are hidden when the partial plans are communicated and thus FMAP is weak privacy preserving both in theory and practice. The PSM planner [Tožička et al., 2014, 2015] is a planner based on similar ideas as Planning First. In PSM, the agents generate sets of plans (represented concisely as Planning State Machines), restrict them to only the public actions and variables and performs intersection in order to find a solution acceptable by all agents (that is, private actions can be added to the public actions so that the result is a valid global plan, somewhat similarly to GPPP).

Planners adhering to the strong privacy definition were studies mostly theoretically. The first example is the Secure-MAFS planner [Brafman, 2015] which is based on the MAFS principle and which was shown to be strong privacy-preserving on a special class of logistics problems. In [Štolba et al., 2017] it was shown, that Secure-MAFS improves the privacy leakage over MAFS in general, but is not strong

privacy preserving in general, indeed. In [Štolba et al., 2016c,d] the authors have introduced a novel class of privacy-preserving algorithms Sec-MAP, which albeit being not strong privacy-preserving, preserve more privacy than all other planners at that time. The idea is based on the combination of the approaches used in Secure-MAFS and PSM to prevent private information leakage, in particular, not sending states which differ only in the private part of a single agent more than once, and not sending states which are not a part of the local solution. A planner based on PSM which can trade-off privacy-preservation for efficiency and thus gets arbitrarily close to strong privacy was introduced in [Tožička et al., 2017a]. Moreover, a strong privacy-preserving planner, based on PSM again, was presented in [Tožička et al., 2017b] together with a proof, that such planner cannot be strong privacy-preserving, complete and efficient at the same time. The result also carries on to the MAFS-based planners.

Apart from privacy-preserving planners, there is a number of distributed heuristics which adhere to the weak privacy definition ([Torreño et al., 2014, Maliah et al., 2015, Štolba and Komenda, 2013, 2014, Štolba et al., 2015a]) but without any deeper analysis or theoretical guarantees. The only distributed heuristic with privacy guarantees is [Štolba et al., 2016a] which, nevertheless, falls short to prove full strong privacy.

Chapter 3

Multi-Agent Planning

In this chapter, we present three formalisms used throughout the rest of the thesis. The first formalism, MA-STRIPS by [Brafman and Domshlak, 2008], is the most basic multi-agent formalism building on STRIPS, that is, set representation of the propositional description of the world. The MA-STRIPS is used in the most of the thesis, in Chapter 4, Chapter 5, and Sections 6.1 and 6.2. The second formalism, MA-MPT, was introduced by multiple authors including us and is a direct application of the MA-STRIPS approach to the SAS+ or multi-valued planning task (MPT) formalism. This formalism generalizes MA-STRIPS and is used internally by both our planners, MADLA and MAPlan. It is used in Chapter 6 as some of the described concepts are more naturally described using a multi-valued formalism. The third formalism describes the transition system underlying both MA-STRIPS and MA-MPT formalism and is important to understand many concepts throughout the thesis, most prominently in Chapter 7.

In this chapter, in addition to the formalism definitions, we give a brief introduction to the basic algorithms and techniques used to solve multi-agent planning problems and conclude with a brief discussion of the complexity of classical and multi-agent planning and a description of benchmarks used in the evaluations throughout the thesis.

Before delving into the multi-agent world, we briefly define the basic classical planning formalism, STRIPS [Fikes and Nilsson, 1971].

Definition 1. (STRIPS) A STRIPS planning problem Π is defined as a tuple $\Pi = \langle P, A, s_I, s_\star \rangle$ where

P is a finite set of propositions describing the world,

A is a finite set of actions,

$s_I \subseteq P$ is an initial state, and

$G \subseteq P$ is a goal condition.

Definition 2. (STRIPS state) In STRIPS, a state of the world is described as a set $s \subseteq P$ of propositions which hold true in the state of the world. All other propositions ($P \backslash s$) are assumed to be false.

Definition 3. (STRIPS action) A STRIPS action $a \in A$ is denoted by

$\mathsf{pre}(a) \subseteq P$ the set of preconditions,

$\mathsf{add}(a) \subseteq P$ the set of positive (add) effects, and

$\mathsf{del}(a) \subseteq P$ the set of negative (delete) effects.

A STRIPS state s is modified by the application of an action $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$ as $s \circ a = (s \backslash \mathsf{del}(a)) \cup \mathsf{add}(a)$, but only if the preconditions hold, that is, $\mathsf{pre}(a) \subseteq s$. Otherwise, a is not applicable in s.

The MA-STRIPS formalism enriches STRIPS by the concepts of agents and privacy. The set $\mathcal{A} = \{\alpha_i\}_{i=1}^{n}$ of agents corresponds with a factorization of the set of actions $A = \bigcup_{i=1}^{n} A_i$, where each A_i is a set of actions the agent α_i can perform and plan with. It is assumed that the sets A_i of actions of agents are pairwise disjoint, that is, $A_i \cap A_j = \emptyset$ for each $i \neq j$. The MA-STRIPS formalism can be extended to consider so called joint actions, which are actions performed by multiple agents at once, but we do not consider such actions in this thesis. Considering joint-action space brings an exponential blow-up in the number of actions in general. In Section 3.1 we present a detailed formal definition of our variant of the MA-STRIPS formalism, which differs in that the multi-agent planning problem is explicitly split into separate STRIPS agent problems. This allows us to better formalize and reason about distributed algorithms and privacy. In Section 3.2, we extend the formalism for multi-valued variables.

Another concept introduced to STRIPS by MA-STRIPS is privacy. In MA-STRIPS, privacy is treated as a means of factorization, that is, the propositions (and consequently actions) needed only by one agent are private, those needed (shared) by at least two agents are public. In the original work by Brafman&Domshlak, no other assumptions on privacy were made, but most MA-STRIPS planners treated the definition so that private propositions and actions should not be openly shared with other agents, whereas public propositions and actions can. Our treatment of privacy is described in more depth in Chapter 7.

Examples

Throughout the thesis, we use several examples, two represent simplified real world problems and three are abstract. The first example **(Logistics)** is a small logistics example depicted in Figure 3.0.1a. In this example, there are two agents, a truck t and a plane a (naturally, the agents could be the drivers, the logistic companies, etc. here we adhere to the benchmark domains commonly used in multi-agent planning literature). They are transporting a package p from city A to city C, while the truck can move between A and B and the plane between B and C. We use this example to illustrate the formalisms described in the following sections. In Chapter 4, we use a slightly larger logistics example **(Logistics-3)** with an additional depot location connected by road with location C and an additional truck driving from C to the new location D.

Figure 3.0.1b shows the **(UAV)** example. In this example there are two agents, one is a UAV which task is to survey locations 1 and 2, the other is the base agent which can refuel the UAV between the survey missions. This example is used in Chapter 7 to illustrate privacy as the UAV and the base belong each to a different coalition partner and thus do not want to disclose their private information (e.g., the state of supplies at the base).

The three abstract examples are **(LM-Cut)** example, **(pot)** example and **(CP)** example, which are all used in Chapter 6 to illustrate the computation of the respective heuristics and are introduced in detail when due.

In addition to the small examples used to demonstrate the principles and algorithms, in Section 3.6, we briefly introduce the benchmark domains used in the evaluation sections.

3.1 The MA-STRIPS Formalism

The global MA-STRIPS definition from [Brafman and Domshlak, 2008] becomes confusing when describing more complex distributed algorithms. To state explicitly the distributed nature of our approach to MA-STRIPS we use a more suitable definition. The multi-agent planning problem, where each agent is planning for itself, is defined as a set of classical STRIPS problems, one for each agent. Otherwise, we keep the MA-STRIPS factorization and treatment of privacy described in the original work. We also

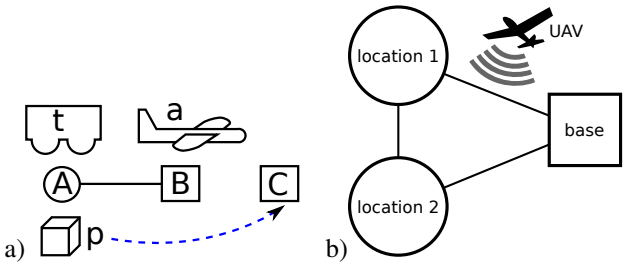


Figure 3.0.1: a) A logistics example with two agents. b) An UAV example with two agents.

explicitly define that if a fact is a goal fact, it is public, resulting in one common public goal condition definition. This is a simplification, but without loss of generality. A problem with a set of private goals can be converted to an equivalent problem with a single public goal by adding a new public goal fact and adding an action which has private preconditions on all the private goal facts and a single effect which is the new public goal fact. This simplification is assumed by most MA-STRIPS planners, but it is not explicitly stated in the MA-STRIPS definition by Brafman&Domshlak. The formal definition of distributed MA-STRIPS is as follows.

Definition 4. (Multi-agent planning problem) Let $\mathcal{A} = \{\alpha_i\}_{i=1}^n$ be a set of n agents and P be a set of propositions describing the world, where $s \subseteq P$ is a state of the world. Then $\mathcal{M} = \{\Pi_i\}_{i=1}^n$ is a *multi-agent planning problem* where for each agent $\alpha_i \in \mathcal{A}$, Π_i is a STRIPS *agent planning problem* for the agent α_i.

Definition 5. (Agent planning problem) A STRIPS *agent planning problem* Π_i is a quadruple

$$\Pi_i = \langle P_i \subseteq P, A_i, s_I \cap P_i, G \rangle$$

where:

$P_i \subseteq P$ is a finite set of propositions describing facts about the world relevant to agent α_i.

A_i is a finite set of actions agent α_i can perform.

- The sets of actions of agents are pairwise disjoint, that is $\forall i \neq j : A_i \cap A_j = \emptyset$.
- Let $A = \bigcup_{i=1}^n A_i$ be the set of all actions in $\mathcal{M}$.

$s_I \subseteq P$ is the initial state of the world, each agent observes only its part $s_I \cap P_i$,

$G \subseteq P^{\mathsf{pub}}$ is the common goal condition defining the goal (final) states of the problem; a state $s_\star$ is a goal state iff $G \subseteq s$.

Definition 6. (MA-STRIPS action) A MA-STRIPS action $a \in A_i$ is a tuple

$$a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a), \mathsf{lbl}(a), \alpha_i \rangle$$

where $\mathsf{pre}(a) \subseteq P_i$ represent preconditions, $\mathsf{add}(a) \subseteq P_i$ represent add effects, and $\mathsf{del}(a) \subseteq P_i$ represent delete effects according to the standard STRIPS syntax and semantics (Definition 3). Additionally, we define

$\mathsf{lbl}(a)$ is a label of the action a unique across all agents, and

α_i is the owner of the action a.

We denote the application of a in a state s as $s' = s \circ a$. Each action is owned exactly by one agent exclusively, that is, there are no joint actions (owned by multiple agents).

Definition 7. (MA-STRIPS privacy rule) MA-STRIPS defines the following rules on public facts and actions:

- A fact $p \in P_i$ is public if $p \in P_j$ for some $j \neq i$.
- Let $P_i^{\mathsf{pub}} \subseteq P_i$ be the set of public facts and $P_i^{\mathsf{priv}} \subseteq P_i$ be the set of private facts, then $P_i^{\mathsf{pub}} \cup P_i^{\mathsf{priv}} = P_i$ and $P_i^{\mathsf{pub}} \cap P_i^{\mathsf{priv}} = \emptyset$.
- An action a is public if either $\mathsf{pre}(a) \cap P^{\mathsf{pub}} \neq \emptyset$, $\mathsf{add}(a) \cap P^{\mathsf{pub}} \neq \emptyset$ or $\mathsf{del}(a) \cap P^{\mathsf{pub}} \neq \emptyset$. Otherwise a is private to α_i.

Let $A_i^{\mathsf{pub}} \subseteq A_i$ be the set of public actions and $A_i^{\mathsf{priv}} = A_i \backslash A_i^{\mathsf{pub}}$ the set of private actions of agent α_i. Each public fact p is known to all agents, that is, if $p \in P_i^{\mathsf{pub}}$, then also $p \in P_j^{\mathsf{pub}}$ for all $j \in 1..n$. Thus we can ignore the agent index and state $P^{\mathsf{pub}} = P_i^{\mathsf{pub}}$ for all $i \in 1..n$.

Now let us demonstrate the above formal definitions on a concrete example.

Example. (Logistics) In the logistics example, the truck problem Π_{t} consists of the following (public facts and actions are bold):

- $P_{\mathsf{t}} = \{\mathsf{truck\text{-}at\text{-}A}, \mathsf{truck\text{-}at\text{-}B}, \mathsf{package\text{-}at\text{-}A}, \mathbf{package\text{-}at\text{-}B}, \mathsf{package\text{-}in\text{-}t}, \mathbf{package\text{-}at\text{-}C}\}$
- $A_{\mathsf{t}} = \{\mathsf{move\text{-}t\text{-}A\text{-}B}, \mathsf{move\text{-}t\text{-}B\text{-}A}, \mathsf{load\text{-}t\text{-}A}, \mathbf{load\text{-}t\text{-}B}, \mathsf{unload\text{-}t\text{-}A}, \mathbf{unload\text{-}t\text{-}B}\}$
  $\mathsf{move\text{-}t\text{-}A\text{-}B} = \langle\{\mathsf{truck\text{-}at\text{-}A}\}, \{\mathsf{truck\text{-}at\text{-}B}\}, \{\mathsf{truck\text{-}at\text{-}A}\}, \mathsf{move\text{-}t\text{-}A\text{-}B}, \mathsf{t}\rangle$
  $\mathsf{move\text{-}t\text{-}B\text{-}A} = \langle\{\mathsf{truck\text{-}at\text{-}B}\}, \{\mathsf{truck\text{-}at\text{-}A}\}, \{\mathsf{truck\text{-}at\text{-}B}\}, \mathsf{move\text{-}t\text{-}B\text{-}A}, \mathsf{t}\rangle$
  $\mathsf{load\text{-}t\text{-}A} = \langle\{\mathsf{truck\text{-}at\text{-}A}, \mathsf{package\text{-}at\text{-}A}\}, \{\mathsf{package\text{-}in\text{-}t}\}, \{\mathsf{package\text{-}at\text{-}A}\}, \mathsf{load\text{-}t\text{-}A}, \mathsf{t}\rangle$
  $\mathbf{load\text{-}t\text{-}B} = \langle\{\mathsf{truck\text{-}at\text{-}B}, \mathbf{package\text{-}at\text{-}B}\}, \{\mathsf{package\text{-}in\text{-}t}\}, \{\mathbf{package\text{-}at\text{-}B}\}, \mathbf{load\text{-}t\text{-}B}, \mathsf{t}\rangle$
  $\mathsf{unload\text{-}t\text{-}A} = \langle\{\mathsf{truck\text{-}at\text{-}A}, \mathsf{package\text{-}in\text{-}t}\}, \{\mathsf{package\text{-}at\text{-}A}\}, \{\mathsf{package\text{-}in\text{-}t}\}, \mathsf{unload\text{-}t\text{-}A}, \mathsf{t}\rangle$
  $\mathbf{unload\text{-}t\text{-}B} = \langle\{\mathsf{truck\text{-}at\text{-}B}, \mathsf{package\text{-}in\text{-}t}\}, \{\mathbf{package\text{-}at\text{-}B}\}, \{\mathsf{package\text{-}in\text{-}t}\}, \mathbf{unload\text{-}t\text{-}B}, \mathsf{t}\rangle$
- $s_I \cap P_{\mathsf{t}} = \{\mathsf{truck\text{-}at\text{-}A}, \mathsf{package\text{-}at\text{-}A}\}$
- $G = \{\mathbf{package\text{-}at\text{-}C}\}$

and the plane problem Π_{a}:

- $P_{\mathsf{a}} = \{\mathsf{plane\text{-}at\text{-}B}, \mathsf{plane\text{-}at\text{-}C}, \mathbf{package\text{-}at\text{-}B}, \mathsf{package\text{-}in\text{-}a}, \mathbf{package\text{-}at\text{-}C}\}$
- $A_{\mathsf{a}} = \{\mathsf{move\text{-}a\text{-}B\text{-}C}, \mathsf{move\text{-}a\text{-}C\text{-}B}, \mathbf{load\text{-}a\text{-}C}, \mathbf{load\text{-}a\text{-}B}, \mathbf{unload\text{-}a\text{-}C}, \mathbf{unload\text{-}a\text{-}B}\}$
  $\mathsf{move\text{-}a\text{-}B\text{-}C} = \langle\{\mathsf{plane\text{-}at\text{-}B}\}, \{\mathsf{plane\text{-}at\text{-}C}\}, \{\mathsf{plane\text{-}at\text{-}B}\}, \mathsf{move\text{-}a\text{-}B\text{-}C}, \mathsf{a}\rangle$
  $\mathsf{move\text{-}a\text{-}C\text{-}B} = \langle\{\mathsf{plane\text{-}at\text{-}C}\}, \{\mathsf{plane\text{-}at\text{-}B}\}, \{\mathsf{plane\text{-}at\text{-}C}\}, \mathsf{move\text{-}a\text{-}C\text{-}B}, \mathsf{a}\rangle$
  $\mathbf{load\text{-}a\text{-}C} = \langle\{\mathsf{plane\text{-}at\text{-}C}, \mathbf{package\text{-}at\text{-}C}\}, \{\mathsf{package\text{-}in\text{-}a}\}, \{\mathbf{package\text{-}at\text{-}C}\}, \mathbf{load\text{-}a\text{-}C}, \mathsf{a}\rangle$
  $\mathbf{load\text{-}a\text{-}B} = \langle\{\mathsf{plane\text{-}at\text{-}B}, \mathbf{package\text{-}at\text{-}B}\}, \{\mathsf{package\text{-}in\text{-}a}\}, \{\mathbf{package\text{-}at\text{-}B}\}, \mathbf{load\text{-}a\text{-}B}, \mathsf{a}\rangle$
  $\mathbf{unload\text{-}a\text{-}C} = \langle\{\mathsf{plane\text{-}at\text{-}C}, \mathsf{package\text{-}in\text{-}a}\}, \{\mathbf{package\text{-}at\text{-}C}\}, \{\mathsf{package\text{-}in\text{-}a}\}, \mathbf{unload\text{-}a\text{-}C}, \mathsf{a}\rangle$
  $\mathbf{unload\text{-}a\text{-}B} = \langle\{\mathsf{plane\text{-}at\text{-}B}, \mathsf{package\text{-}in\text{-}a}\}, \{\mathbf{package\text{-}at\text{-}B}\}, \{\mathsf{package\text{-}in\text{-}a}\}, \mathbf{unload\text{-}a\text{-}B}, \mathsf{a}\rangle$
- $s_I \cap P_{\mathsf{a}} = \{\mathsf{plane\text{-}at\text{-}B}\}$
- $G = \{\mathbf{package\text{-}at\text{-}C}\}$

The locations of the truck and plane are private to the respective agents as well as the location of the package unless it is in the cities B or C. The location of the package in B is public because it is shared by the two agents, whereas the package in location C is public because **package-at-C** is a public goal. Because of that, also the **unload-t-B**, **unload-a-C** and the respective load actions are public.

3.1.1 Views of the MA-STRIPS Problem

The MA-STRIPS problem $\mathcal{M} = \{\Pi_i\}_{i=1}^n$ can be viewed from different perspectives, called *projections*. We can consider the *global problem*, ignoring the factorization into agents, which corresponds to a classical centralized STRIPS version of $\mathcal{M}$. Alternatively, we can be interested in a *public projection* of $\mathcal{M}$, which is the global problem restricted only to its public part. The view of a single agent α_i on the global problem is not restricted only to the agents problem Π_i, but the public projection is available to the agent as well. By combining the two views, we obtain an *i-projected problem* $\Pi_i^{\triangleright}$ which consists of the problem of the agent α_i and public parts of the problems of other agents. In some cases we need to consider only the private part of the agent's problem, that is, an *i-private projection* $\Pi_i^{\triangledown}$ which consists of the agent's problem without the public part. In the following, we formalize the above concepts. First, we focus on how states differ in each of the problem views.

Definition 8. (Projections of a state) Let $s \subseteq P$ be a state of the MA-STRIPS problem $\mathcal{M}$, then

$s^{\triangleright} = s \cap P^{\mathsf{pub}}$ is a public projection of s,

$s^{\triangleright i} = s \cap P_i$ is an i-projection of s, and

$s^{\triangledown i} = s \cap P_i^{\mathsf{priv}}$ is an i-private projection of s.

Next, we define the projected variants of actions.

Definition 9. (Projections of an action) Let $a \in A_j^{\mathsf{pub}}$ be a public action of agent α_j, let α_i be an agent such that $i \neq j$. We define

$$
\begin{aligned}
a^{\triangleright} &= \left\langle \mathsf{pre}(a) \cap P^{\mathsf{pub}}, \mathsf{add}(a) \cap P^{\mathsf{pub}}, \mathsf{del}(a) \cap P^{\mathsf{pub}}, \mathsf{lbl}(a)^{\triangleright}, \alpha_j \right\rangle \\
a^{\triangleright i} &= \left\langle \mathsf{pre}(a) \cap P_i, \mathsf{add}(a) \cap P_i, \mathsf{del}(a) \cap P_i, \mathsf{lbl}(a)^{\triangleright i}, \alpha_j \right\rangle \\
a^{\triangledown j} &= \left\langle \mathsf{pre}(a) \cap P_j^{\mathsf{priv}}, \mathsf{add}(a) \cap P_j^{\mathsf{priv}}, \mathsf{del}(a) \cap P_j^{\mathsf{priv}}, \mathsf{lbl}(a)^{\triangledown j}, \alpha_j \right\rangle
\end{aligned}
$$

where $a^{\triangleright}$ is the public projection of a, $a^{\triangleright i}$ is the i-projection of a, and $a^{\triangledown j}$ is the j-private projection of a.

Let $a \in A_j^{\mathsf{priv}}$ be a private action of agent α_j, then the projections are defined as follows:

$$a^{\triangleright} = a^{\triangleright i} = \epsilon = \langle \emptyset, \emptyset, \emptyset, \mathsf{lbl}(\epsilon), \alpha_j \rangle$$

which is a no-op action, and $a^{\triangledown j} = a$.

As in MA-STRIPS we do not consider joint actions, the definition of i-projected action is slightly more general than necessary, consider

Lemma 10. *Let $a \in A_i$ for some agent α_i, then $a^{\triangleright} = a^{\triangleright j}$ for all $i \neq j$.*

Proof. Let $a \in A_i$ for some agent α_i. From Definition 6 the sets $\mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)$ are subsets of P_i. From the MA-STRIPS rule (Definition 7) follows that $P_i^{\mathsf{priv}} \cap P_j^{\mathsf{priv}} = \emptyset$ for all $i \neq j$ and thus for $\mathsf{pre}(a)$ holds $\mathsf{pre}(a) \cap P_j = \mathsf{pre}(a) \cap P^{\mathsf{pub}}$ and the same holds for $\mathsf{add}(a)$ and $\mathsf{del}(a)$ as well. □

Now we define the projected problems themselves, that is, the views on the MA-STRIPS problem $\mathcal{M}$.

Definition 11. (Projections of the problem) Let $\mathcal{M} = \{\Pi_i\}_{i=1}^n$ be a MA-STRIPS problem. Then we define

$$\Pi^{\mathsf{G}} = \left\langle P, A = \bigcup_{i \in 1..n} A_i, s_I, G \right\rangle$$

$$\mathcal{M}^{\triangleright} = \left\langle P, \{a^{\triangleright} | a \in \bigcup_{i \in 1..n} A_i^{\mathsf{pub}}, s_I^{\triangleright}, G\} \right\rangle$$

$$\Pi_i^{\triangleright} = \left\langle P_i, A_i^{\triangleright} = A_i \cup \{a^{\triangleright} | a \in \bigcup_{j \in 1...n \wedge j \neq i} A_j^{\mathsf{pub}}\}, s_I^{\triangleright i}, G \right\rangle$$

$$\Pi_i^{\triangledown} = \left\langle P_i, A_i^{\triangledown} = \{a^{\triangledown i} | a \in A_i\}, s_I^{\triangledown i}, \emptyset \right\rangle$$

where the STRIPS problem Π^{G} is the global problem respective to $\mathcal{M}$, the STRIPS problem $\mathcal{M}^{\triangleright}$ is the public projection of $\mathcal{M}$, the STRIPS problem $\Pi_i^{\triangleright}$ is the i-projection of $\mathcal{M}$, and the STRIPS problem $\Pi_i^{\triangledown}$ is the i-private projection of $\mathcal{M}$.

Although the agent α_i cannot plan using $\Pi_i^{\triangleright}$, as the resulting plan might not be executable in the multi-agent problem $\mathcal{M}$, it can be used for heuristic computation (the projected problem is, in fact, an abstraction of $\mathcal{M}$, see Theorem 36). On the contrary, the i-private projection cannot be used on its own even to compute a heuristic estimate as the goals are public and thus are not present in the projection.

Example. (Logistics) In the example, the problem $\Pi_{\mathsf{t}}^{\triangleright}$ projected to agent t differs from Π_{t} in that it contains the additional projected actions, that is

- **load-a-C**$^{\triangleright} = \langle\{$**package-at-C**$\}, \emptyset, \{$**package-at-C**$\},$ **load-a-C**$, \mathsf{a}\rangle$
- **load-a-B**$^{\triangleright} = \langle\{$**package-at-B**$\}, \emptyset, \{$**package-at-B**$\},$ **load-a-B**$, \mathsf{a}\rangle$
- **unload-a-C**$^{\triangleright} = \langle\emptyset, \{$**package-at-C**$\}, \emptyset,$ **unload-a-C**$, \mathsf{a}\rangle$
- **unload-a-B**$^{\triangleright} = \langle\emptyset, \{$**package-at-B**$\}, \emptyset,$ **unload-a-B**$, \mathsf{a}\rangle$

Notice the empty preconditions of the actions **unload-a-B**$^{\triangleright}$ and **unload-a-C**$^{\triangleright}$ as all the facts plane-at-B, plane-at-C and package-in-a are private to the agent a. The projected problem for agent a, that is, $\Pi_{\mathsf{a}}^{\triangleright}$ is defined analogously.

3.1.2 Solution Concepts

Let us now formally define the notion of a multi-agent plan.

Definition 12. (s_0–s_m-plan) Let $\pi = (a_1, ..., a_m)$ be a sequence of actions from A (the actions may be owned by different agents). The sequence π is applicable in state s_0 if there are states $s_1, ..., s_m$ s.t. for $1 \leq k \leq m$, action a_k is applicable in s_{k-1} (that is, $\mathsf{pre}(a) \subseteq s_{k-1}$) and $s_k = s_{k-1} \circ a_k$. The sequence of actions π is referred to as a s_0–s_m*-plan* and $s_0 \circ \pi$ denotes the resulting state s_m.

Definition 13. (s-plan) An s_0–s_m-plan is a s-plan iff s_m is a goal state, that is, $G \subseteq s_m$.

Definition 14. (Plan) An s_I-plan is a (multi-agent) plan (or a global plan) solving the multi-agent problem $\mathcal{M}$.

Clearly, the multi-agent plan may contain public and private actions of different agents. An $s^{\triangleright i}$-plan solving $\Pi_i^{\triangleright}$ is defined analogously, but using only the actions from $A_i^{\triangleright}$. Any multi-agent s-plan can be expressed in a *distributed multi-agent plan* form $\{\pi_i\}_{i=1}^n$, where π_i retains all actions in $\pi \cap A_i$ (here we overload the notation of a set intersection to an intersection of a set and a sequence) and all actions from $A \backslash A_i$ are replaced with an empty (no-op) action $\epsilon = \langle\emptyset, \emptyset, \emptyset, -, -\rangle$. In other words, a_k in π_i is a_k from π if $a_k \in A_i$. Otherwise, a_k in π_i is ϵ, that is a no-op action with no preconditions and effects.

Example. (Logistics) A multi-agent plan solving the example problem consists of the following 6 actions:

- $\langle$load-t-A, move-t-A-B, **unload-t-B**, **load-a-B**, move-a-B-C, **unload—a—C**$\rangle$

and can be reformulated as a distributed multi-agent plan

- $\{\langle$load-t-A, move-t-A-B, **unload-t-B**, $\epsilon, \epsilon, \epsilon\rangle$,
- $\langle \epsilon, \epsilon, \epsilon,$ **load-a-B**, move-a-B-C, **unload-a-C**$\rangle\}$

Notice, that an optimal solution to $\Pi^{\triangleright}_{\mathsf{t}}$ is a plan consisting of a single action $\langle$**unload-a-C**$^{\triangleright}\rangle$. The projected problem $\Pi^{\triangleright}_{\mathsf{a}}$ is analogous and can be solved using four actions, in particular $\langle$**unload-t-B**, **load-a-B**, move-a-B-C, **unloa**

In the rest of the thesis, we use the term *plan* for multi-agent plans. In the context of the views on the MAP problem, we define also views on the multi-agent plans.

Definition 15. (Local plan) A plan π_i solving the i-projected problem $\Pi^{\triangleright}_i$ is a *local plan*.

Definition 16. (Public plan) A plan $\pi^{\triangleright}$ solving the public projected problem $\mathcal{M}^{\triangleright}$ is a *public plan*.

Let us have a look at the relations of a multi-agent plan, a public plan, and a local plan.

Proposition 17. *A public projection $\pi^{\triangleright}_i$ of a local plan π_i is a public plan.*

Proof. Projection only removes constraints (private preconditions), the applicability of public actions is retained. □

Clearly, a local plan might not be part of a multi-agent plan, as it may ignore unsatisfiable private preconditions. Also, a public plan might not be a projection of a multi-agent plan (for the same reasons). Finally,

Definition 18. (i-extensibility) A public plan $\pi^{\triangleright}$ is *i-extensible*, if by adding private actions from A^{priv}_i to $\pi^{\triangleright}$and replacing all projections $a^{\triangleright}$ s.t. $a \in A^{\mathsf{pub}}_i$ by a, we obtain a local solution to Π_i.

According to Tožička et al. [2016]:

Theorem 19. *Let $\mathcal{M} = \{\Pi_i\}^n_{i=1}$ be a MA-STRIPS problem. A public plan $\pi^{\triangleright}$ which is i-extensible by all agents $i \in 1, ..., n$, the i-extensions $\{\pi_i\}^n_{i=1}$ form a multi-agent plan solving $\mathcal{M}^{\triangleright}$.*

Finally, we define a number of related concepts. A state from which there is no solution as follows.

Definition 20. (Dead-end state) A state s is a *dead end state* (or a dead-end) if no s-plan exists.

In the MA-STRIPS formalism, we assume the quality criterion for a plan π to be its length $|\pi|$, that is a plan π is optimal iff it contains the minimal number of actions among all plans solving $\mathcal{M}$. Nevertheless, all techniques presented in this article are easily modified for a quality criterion based on the (non-negative) cost of actions.

After defining the solution concepts we can show the equality of the multi-agent problem and the global problem.

Theorem 21. *(Global problem equality) Let $\mathcal{M} = \{\Pi_i\}^n_{i=1}$ be a MA-STRIPS problem. The global problem Π^{G} from Definition 32 is equal to $\mathcal{M}$, that is, every plan π solving $\mathcal{M}$ solves Π^{G} and every plan π' solving $\mathcal{M}$ solves Π^{G}.*

Proof. According to the Definition 14, a plan (solving $\mathcal{M}$) is a sequence of actions $\pi = (a_1, ..., a_m)$ such that a_1 is applicable in the initial state s_I and the application of π results in a goal state $s_\star$, that is, $G \subseteq s_\star = s_I \circ \pi$. Each state s_k visited by π is a subset of P, each action $a_k \in A = \bigcup_{i \in 1..n} A_i$ is applicable in $s_{k-1} \subseteq P$ and its application results in a state $s_k \subseteq P$. As Π^{G} is defined over the set of facts P and the set of actions A, the same holds also for π as a plan solving Π^{G}. The other direction is analogous. □

The global problem Π^{G} is a useful concept, e.g., for showing the admissibility of a distributed heuristic by showing that it returns the same values as in the global problem (see Chapter 6).

3.2 The MA-MPT Formalism

Similarly as MA-STRIPS [Brafman and Domshlak, 2008] presented in the previous section is an extension of STRIPS Fikes and Nilsson [1971] towards privacy and multi-agent planning, we now present MA-MPT as a multi-agent extension of the Multi-Valued Planning Task Helmert [2006] or SAS+ [Bäckström, 1992]. All formal definitions are analogous to the MA-STRIPS definition in the previous section. Indeed, if all the variables in a MA-MPT problem are binary, we obtain a MA-STRIPS problem (with the addition of the cost function), thus, MA-MPT is a generalization of MA-STRIPS. We present the MA-MPT formalism analogously to MA-STRIPS in Section 3.1.

In MPT, the world is described by a finite set of variables $\mathcal{V}$ instead of a set of binary predicates in STRIPS. Each such variable $V \in \mathcal{V}$ has a finite domain of possible values, denoted as $\mathsf{dom}(V)$. The state of the world in MPT is defined as follows.

Definition 22. (State) Let $\mathcal{V}$ be a finite set of variables with finite domains $\mathsf{dom}(V)$ and let $s : \mathcal{V} \mapsto \prod_{V \in \mathcal{V}} \mathsf{dom}(V)$ be a mapping (assignment) from the set of variables to their respective values. Then s is a state, more specifically a state over $\mathcal{V}$. The value of V in s is denoted as $s[V]$.

In MPT we define also a partial state which is used to describe the goal condition, and preconditions and effects of the operators.

Definition 23. (Partial state) Let $\mathcal{V}$ be a finite set of variables with finite domains $\mathsf{dom}(V)$ and let $\mathcal{V}' \subset \mathcal{V}$. A mapping p from $\mathcal{V}'$ to the values $p[V]$ of the respective variables is a partial state (assignment). We use $\mathsf{vars}(p) \subseteq \mathcal{V}$ to denote a subset of $\mathcal{V}$ on which p is defined and $p[V]$ to denote the value of V assigned by p.

Definition 24. (Partial state consistency) A partial state p is *consistent* with a state s iff $p[V] = s[V]$ for all $V \in \mathsf{vars}(p)$.

Now we can proceed to the definition of the syntax and semantics of an MPT operator.

Definition 25. (MPT Operator) Let $\mathcal{V}$ be a finite set of variables with finite domains $\mathsf{dom}(V)$. an MPT operator is a tuple

$$o = \langle \mathsf{pre}(o), \mathsf{eff}(o), \mathsf{lbl}(o) \rangle$$

where

$\mathsf{pre}(o)$ is a partial assignment over $\mathcal{V}$,

$\mathsf{eff}(o)$ is a partial assignment over $\mathcal{V}$, and

$\mathsf{lbl}(o)$ is a unique label.

The operator is applicable in a state s (over $\mathcal{V}$) iff $\mathsf{pre}(o)$ is consistent with s. The application of operator o in a state s results in a state s' such that all variables in $\mathsf{vars}(\mathsf{eff}(o))$ are assigned to the values in $\mathsf{eff}(o)$ and all other variables retain the values from s. We denote the application of o in s as $s' = s \circ o$.

In the thesis, we use the term action and operator for an MPT operator interchangeably (and so do we use the symbols $a \in \mathcal{O}$ and $o \in \mathcal{O}$). Before defining the multi-agent version of the MPT formalism, let us first show an alternative view of the MPT formalism.

Definition 26. (Facts in MPT) A *fact* in MPT is a pair $\langle V, v \rangle$ of a variable V and one of the values $v \in \mathsf{dom}(V)$ (i.e., an assignment of v to V). A (partial) variable assignment p can be represented as a set of facts $\{\langle V, p[V] \rangle \, | V \in \mathsf{vars}(p)\}$.

Now, we can proceed to the definition of a MA-MPT problem (or task). The multi-agent problem definition is the same as in Definition 4, that is, $\mathcal{M} = \{\Pi_i\}_{i=1}^{n}$. The difference is, that the agent planning problem is an MPT problem instead of a STRIPS problem. Formally

Definition 27. (Agent planning problem) Let the world be described by a finite set $\mathcal{V}$ of variables with finite domains. an MPT agent planning problem for an agent $\alpha_i \in \mathcal{A}$ is a tuple

$$\Pi^i = \left\langle \mathcal{V}^i, \mathcal{O}^i, s_I^{\triangleright i}, s_\star^{\triangleright i}, \mathsf{cost}^i \right\rangle$$

where

$\mathcal{V}^i \subseteq \mathcal{V}$ s.t. $\mathcal{V}^i = \mathcal{V}^{\mathsf{pub}} \cup \mathcal{V}^{\mathsf{priv}_i}$ is a finite set of variables consisting of a set $\mathcal{V}^{\mathsf{pub}}$ of public variables and a set $\mathcal{V}^{\mathsf{priv}_i}$ of private variables. Each variable $V \in \mathcal{V}^i$ has a finite domain of values $\mathsf{dom}(V)$.

$\mathcal{O}^i = \mathcal{O}^{\mathsf{pub}_i} \cup \mathcal{O}^{\mathsf{priv}_i}$ is a finite set of MPT operators consisting of a set $\mathcal{O}^{\mathsf{pub}_i}$ of public operators and a set $\mathcal{O}^{\mathsf{priv}_i}$ of private variables.

$s_I^{\triangleright i}$ is the initial state over $\mathcal{V}$ restricted to $\mathcal{V}^i$.

$s_\star^{\triangleright i}$ is the goal partial state over $\mathcal{V}$ restricted to $\mathcal{V}^i$.

$\mathsf{cost}^i : \mathcal{O}^i \mapsto \mathbb{R}_0^+$ is a cost function assigning a non-negative cost to the operators.

The set $\mathcal{V}^{\mathsf{pub}}$ of public variables is shared among all agents, for each i $\mathcal{V}^{\mathsf{pub}} \cap \mathcal{V}^{\mathsf{priv}_i} = \emptyset$, and for each $i \neq j$, $\mathcal{V}^{\mathsf{priv}_i} \cap \mathcal{V}^{\mathsf{priv}_j} = \emptyset$ and $\mathcal{O}^i \cap \mathcal{O}^j = \emptyset$.

An MPT operator in MA-MPT also contains a reference to the owner agent as in Definition 6. The privacy of operators in MA-MPT is defined by a MA-MPT rule analogous to the MA-STRIPS rule in Definition 7.

Definition 28. (MA-MPT rule) an MPT operator $o \in \mathcal{O}^i$ is private ($o \in \mathcal{O}^{\mathsf{priv}_i}$) iff $\mathsf{vars}(\mathsf{pre}(o)) \subseteq \mathcal{V}^{\mathsf{priv}_i}$ and $\mathsf{vars}(\mathsf{eff}(o)) \subseteq \mathcal{V}^{\mathsf{priv}_i}$. an MPT operator $o \in \mathcal{O}^i$ is public ($o \in \mathcal{O}^{\mathsf{pub}_i}$) iff $\mathsf{vars}(\mathsf{pre}(o)) \cap \mathcal{V}^{\mathsf{pub}} \neq \emptyset$ or $\mathsf{vars}(\mathsf{eff}(o)) \cap \mathcal{V}^{\mathsf{pub}} \neq \emptyset$.

We conclude the description of the MA-MPT planning problem by a MA-MPT formulation of the logistics example.

Example. (Logistics) In the case of the example in Section 3, the problem Π_t of the truck agent t can be defined using the following variables

- $\mathcal{V}^{\mathsf{pub}} = \{\textbf{package-at} \in \{\mathsf{B}, \mathsf{C}, \bot\}\}$
- $\mathcal{V}^{\mathsf{priv}_\mathsf{t}} = \{\mathsf{truck{-}at} \in \{\mathsf{A}, \mathsf{B}\}, \mathsf{package{-}at} \in \{\mathsf{A}, \bot\}\}$

The symbol $\bot$ means *none-of-those*, e.g., the variable has a value not known to the agent (this technique is used in classical planning as well). The operators in $\mathcal{O}^{\mathsf{pub}_\mathsf{t}}$ and in $\mathcal{O}^{\mathsf{priv}_\mathsf{t}}$ are the same as the actions in A_i^{pub} and A_i^{priv} respectively with the only difference, that they are defined based on the MPT formalism. For example, the operator $\mathsf{move\text{-}t\text{-}A\text{-}B} \in \mathcal{O}^{\mathsf{pub}_\mathsf{t}}$ is defined as

- $\mathsf{move\text{-}t\text{-}A\text{-}B} = \langle \{\mathsf{truck\text{-}at} = \mathsf{A}\}, \{\mathsf{truck\text{-}at} = \mathsf{B}\}, \mathsf{move\text{-}t\text{-}A\text{-}B}, \mathsf{t} \rangle$

which means that the precondition of move-t-A-B is that the variable truck-at has the value of A and the effect is that the variable is set to B. Of course, the precondition and effect of an action might be defined over different variables. The label of the operator is the same as the label in the MA-STRIPS formalism, in particular move-t-A-B. . In MA-MPT we also consider the costs of actions, which can be defined for example as $\mathsf{cost}^\mathsf{t}(\mathsf{move\text{-}t\text{-}A\text{-}B}) = 10$ where 10 is the distance between the cities A and B.

The initial state of the truck agent is defined as an assignment to all variables in $\mathcal{V}^\mathsf{t}$, in particular

- $s_I^{\triangleright \mathsf{t}} = \{\textbf{package-at} = \bot, \mathsf{package{-}at} = \mathsf{A}, \mathsf{truck{-}at} = \mathsf{A}\}$

Notice, that the public variable **package-at** and private variable $\mathsf{package{-}at}$ are actually two different variables, interconnected only by the definition of the actions so that only one of the variables can be set to a particular value, whereas the other variable is set to an "undefined" value $\bot$. Nevertheless, the $\bot$ value is treated in the model as any ordinary value (that is, any algorithm using the formalism is not explicitly aware of its special semantics). The goal state $s_\star^{\triangleright \mathsf{t}}$ is a partial variable assignment analogous to $s_I^{\triangleright \mathsf{t}}$ and the problem of the plane agent a is defined analogously to the problem of the truck agent.

3.2.1 Views of the MA-MPT Problem

Similarly to the MA-STRIPS problem, the MA-MPT problem can be viewed from multiple perspectives, that is, the global problem, the public projection, the i-projection and the i-private projection. We define the views analogously to Section 3.1.1, but first, we need to formally define the following.

Definition 29. (Restriction of an assignment) Let p be a (partial) assignment (state) over a set $\mathcal{V}$ of variables and let $\mathcal{V}' \subseteq \mathcal{V}$. The assignment p' such that $\mathsf{vars}(p') = \mathcal{V}'$ and $p'[V] = p[V]$ for each $V \in \mathcal{V}'$ is a restriction of p on the set $\mathcal{V}'$ of variables.

Now we can proceed to the definition of state projections.

Definition 30. (Projections of a state) Let s be a (partial) state over the finite set $\mathcal{V}$ of finite-domain variables, then

$s^{\triangleright}$ A public projection of s is a restriction of s to the set $\mathcal{V}^{\mathsf{pub}}$ of variables.

$s^{\triangleright i}$ An i-projection of s is a restriction of s to the set $\mathcal{V}^{i}$ of variables.

$s^{\triangledown i}$ An i-private projection of s is a restriction of s to the set $\mathcal{V}^{\mathsf{priv}_i}$ of variables.

Next, we define the operator projections.

Definition 31. (Projections of an operator) Let $o \in \mathcal{O}^{\mathsf{pub}_j}$ be a public operator of agent α_j, let α_i be an agent such that $i \neq j$. We define

$$
\begin{aligned}
o^{\triangleright} &= \left\langle \mathsf{pre}(a)^{\triangleright}, \mathsf{eff}(o)^{\triangleright}, \mathsf{lbl}(o)^{\triangleright}, \alpha_j \right\rangle \\
o^{\triangleright i} &= \left\langle \mathsf{pre}(a)^{\triangleright i}, \mathsf{eff}(o)^{\triangleright i}, \mathsf{lbl}(o)^{\triangleright i}, \alpha_j \right\rangle \\
o^{\triangledown j} &= \left\langle \mathsf{pre}(a)^{\triangledown j}, \mathsf{eff}(o)^{\triangledown j}, \mathsf{lbl}(o)^{\triangledown j}, \alpha_j \right\rangle
\end{aligned}
$$

where $o^{\triangleright}$ is the public projection of o, $o^{\triangleright i}$ is the i-projection of o, and $o^{\triangledown j}$ is the j-private projection of o. Let $o \in \mathcal{O}^{\mathsf{priv}_j}$ be a private operator of agent α_j, then the projections are defined as follows: $o^{\triangleright} = o^{\triangleright i} = \epsilon = \langle \emptyset, \emptyset, \emptyset, \mathsf{lbl}(\epsilon), \alpha_j \rangle$, which is a no-op action, and $o^{\triangledown j} = o$.

Trivially, Lemma 10 holds also for the operator projections in MA-MPT. We proceed with the definitions of the problem views.

Definition 32. (Projections of the problem) Let $\mathcal{M} = \{\Pi_i\}_{i=1}^{n}$ be a MA-MPT problem. Then we define

$$
\begin{aligned}
\Pi^{\mathsf{G}} &= \left\langle \mathcal{V} = \bigcup_{i \in 1..n} \mathcal{V}^i, \bigcup_{i \in 1..n} \mathcal{O}^i, s_I, s_\star, \mathsf{cost}^{\mathsf{G}} \right\rangle \\
\Pi^{\triangleright} &= \left\langle \mathcal{V}^{\mathsf{pub}}, \mathcal{O}^{\triangleright} = \{o^{\triangleright} | o \in \bigcup_{j \in 1...n} \mathcal{O}^{\mathsf{pub}_i}\}, s_I^{\triangleright}, s_\star^{\triangleright}, \mathsf{cost}^{\triangleright} \right\rangle \\
\Pi_i^{\triangleright} &= \left\langle \mathcal{V}^i, \mathcal{O}^{\triangleright i} = \mathcal{O}^i \cup \{o^{\triangleright i} | o \in \bigcup_{j \in 1...n \wedge j \neq i} \mathcal{O}^{\mathsf{pub}_j}\}, s_I^{\triangleright i}, s_\star^{\triangleright i}, \mathsf{cost}^{\triangleright i} \right\rangle \\
\Pi_i^{\triangledown} &= \left\langle \mathcal{V}^i, \mathcal{O}^{\triangledown i} = \{o^{\triangledown i} | o \in \mathcal{O}^i\}, s_I^{\triangledown i}, s_\star^{\triangledown i}, \mathsf{cost}^{i} \right\rangle
\end{aligned}
$$

where $\mathsf{cost}^{\mathsf{G}}$ is a union of the cost functions cost^i, $\mathsf{cost}^{\triangleright}$ is $\mathsf{cost}^{\mathsf{G}}$ restricted to operators in $\mathcal{O}^{\triangleright}$, and $\mathsf{cost}^{\triangleright i}$ is $\mathsf{cost}^{\mathsf{G}}$ restricted to operators in $\mathcal{O}^{\triangleright i}$. The MPT problem Π^{G} is the global problem respective to $\mathcal{M}$, the MPT problem $\mathcal{M}^{\triangleright}$ is the public projection of $\mathcal{M}$, the MPT problem $\Pi_i^{\triangleright}$ is the i-projection of $\mathcal{M}$, and the MPT problem $\Pi_i^{\triangledown}$ is the i-private projection of $\mathcal{M}$.

3.2.2 Solution Concepts

The solution concepts of a MA-STRIPS problem, that is, an s_0-s_m-plan (Definition 12), an s-plan (Definition 13), and a plan (Definition 14), are defined only using the concept of applicability of actions. This means, that by using the applicability of MPT operators instead (Definition 25), we can directly reuse the already defined solution concepts also for MA-MPT planning problems, the Theorem 19 and Theorem 21 hold for MA-MPT as well.

3.3 Multi-Agent Planning Problem as a Transition System

In Section 1.1 we have described three levels of formalisms used to describe the (multi-agent) planning problems, the actual transition system, ground representations, and lifted representation. Both formalisms described so far, MA-STRIPS and MA-MPT falls into the category of ground representations which are very useful to describe most of the algorithms and techniques presented in the thesis and are close to the actual implementation in the planners. The lifted representation, such as PDDL [McDermott et al., 1998], are very useful to describe the planning domains and problems but are not very relevant to the topics discussed in the thesis. Thus, we describe our contribution in this area, the MA-PDDL language, in the Appendix A. In this section, we take a closer look at the actual transition system of a planning problem and its multi-agent variant. This approach is most useful in Chapter 7, but may be used throughout the thesis for a better understanding of the underlying concepts.

We start by defining the transition system of a classical MPT planning problem (or task). An analogous definition can be used to define the transition system of a STRIPS problem by using binary variables.

Definition 33. (Transition system) A transition system of an MPT planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_I, s_\star, \mathsf{cost} \rangle$ is a tuple

$$\mathcal{T}(\Pi) = \langle S, L, T, s_I, S_\star, \mathsf{cost} \rangle$$

where $S = \prod_{V \in \mathcal{V}} \mathsf{dom}(V)$ is a set of states, $L = \{\mathsf{lbl}(o) | o \in \mathcal{O}\}$ is a set of transition labels corresponding to the operators in $\mathcal{O}$, and $T \subseteq S \times L \times S$ is a transition relation of Π s.t. $\langle s, o, s' \rangle \in T$ if $o \in \mathcal{O}$ s.t. o is applicable in s and $s' = o \circ s$. A state-changing transition is $\langle s, o, s' \rangle \in T$ such that $s \neq s'$. The state $s_I \in S$ is the initial state and $S_\star$ is the set of all goal states (that is all states s s.t. $s_\star$ is consistent with s). The cost (or weight) of a transition $\langle s, o, s' \rangle$ is defined as $\mathsf{cost}(o)$.

Definition 34. (Solution) A solution in the transition system $\mathcal{T}(\Pi) = \langle S, L, T, s_I, S_\star, \mathsf{cost} \rangle$ is a path $\pi = \{t_1, ..., t_k\}$ in $\mathcal{T}(\Pi)$ where $t_{i \in 1..k} \in T$ such that π starts in the initial state s_I (that is, $t_1 = \langle s_I, o_1, s_1 \rangle$) and ends in any state $s \in S_\star$ (that is, $t_k = \langle s_{k-1}, o_k, s \rangle$). The cost of π is defined as $\sum_{i \in 1..k} \mathsf{cost}(o_i)$ where $t_i = \langle s_{i-1}, o_i, s_i \rangle$. An optimal solution is such π^* which minimizes the cost.

From Definition 27 follows that each agent problem is an MPT problem and thus can be represented as a transition system. Based on Definition 32, the same can be said about the global problem, the i-projected problem, and the i-private projected problem. Unfortunately, this does not tell us much about the relation of the transition systems, for example, whether a solution in one transition system implies a solution in any of the other transition systems. To provide such statements, we first define a concept well known in the classical planning literature, an abstraction of a transition system.

Definition 35. (Abstraction) Let $\mathcal{T} = \langle S, L, T, s_I, S_\star \rangle$ and $\mathcal{T}' = \langle S', L', T', s'_I, S'_\star \rangle$ be transition systems with the same label sets $L = L'$ and let $\sigma : S \mapsto S'$. We say that T' is an abstraction of T with the abstraction function (mapping) σ if

- $s'_I = \sigma(s_I)$,
- for all $s \in S_\star$ also $\sigma(s) \in S'_\star$, and

- for all $\langle s, o, s' \rangle \in T$, $\langle \sigma(s), o, \sigma(s') \rangle \in T'$.

It is easy to see, that a solution π in the original transition system is also a solution in the abstract transition system. This does not hold in the other direction, but a cost of the optimal solution in the abstract transition system is an admissible (i.e., lower or equal) estimate of the cost of the optimal solution in the original transition system.

We proceed by showing that the transition system of an i-projection of a MAP problem $\mathcal{M}$ is an abstraction of the respective global problem Π^{G}.

Theorem 36. *(Projection is an abstraction) Let* $\mathcal{T}(\Pi^{\mathsf{G}}) = \left\langle S^{\mathsf{G}}, \bigcup_{i \in 1..n} \mathcal{O}^i, T^{\mathsf{G}}, s_I, S_\star \right\rangle$ *be the transition system of the global problem* Π^{G} *and* $\mathcal{T}(\Pi_i^{\triangleright}) = \left\langle S^{\triangleright i}, \mathcal{O}^{\triangleright i}, T^{\triangleright i}, s_I^i, S_\star^i \right\rangle$ *the transition system of the* i*-projected problem* $\Pi_i^{\triangleright}$*. Then* $\mathcal{T}(\Pi_i^{\triangleright})$ *is an abstraction of* $\mathcal{T}(\Pi^{\mathsf{G}})$ *with respect to the state-changing transitions.*

Proof. We define an abstraction mapping $\sigma^{\triangleright i} : S^{\mathsf{G}} \mapsto S^{\triangleright i}$ such that for a state $s \in S^{\mathsf{G}}$ we define $\sigma^{\triangleright i}(s)$ as a restriction of s to the variables in $\mathcal{V}^i$. Then from definition, $\sigma^{\triangleright i}(s) = s^{\triangleright i}$. From definition also $s_I^{\triangleright i} = \sigma^{\triangleright i}(s_I)$. If $s \in S_\star$ then $s_\star$ is compatible with s, if both are restricted to $\mathcal{V}^i$, the compatibility is not violated and thus $\sigma^{\triangleright i}(s) \in S_\star^i$.

For each action $o \in \mathcal{O}^i$ and each transition $\langle s, o, s' \rangle \in T^{\mathsf{G}}$ there is a transition $\left\langle s^{\triangleright i}, o^{\triangleright i}, s'^{\triangleright i} \right\rangle \in T^{\triangleright i}$ as $o^{\triangleright i} = o$. For $j \neq i$ and for each action $o' \in \mathcal{O}^{\mathsf{pub}_j}$ and each transition $\langle t, o', t' \rangle \in T^{\mathsf{G}}$, there is a transition $\left\langle t^{\triangleright i}, o'^{\triangleright i}, t'^{\triangleright i} \right\rangle \in T^{\triangleright i}$ as $\mathsf{pre}(o'^{\triangleright i})$ is $\mathsf{pre}(o')$ restricted to $\mathcal{V}^i$ and $t^{\triangleright i}$ is t restricted to $\mathcal{V}^i$ (the same goes for $\mathsf{eff}(o^{\triangleright i})$. For each action $o'' \in \mathcal{O}^{\mathsf{priv}_j}$ and each transition $\langle u, o', u' \rangle \in T^{\mathsf{G}}$, there is no transition $\left\langle u^{\triangleright i}, o''^{\triangleright i}, u'^{\triangleright i} \right\rangle \in T^{\triangleright i}$, but as both $\mathsf{pre}(o'')$ and $\mathsf{eff}(o'')$ are defined only over $\mathcal{V}^{\mathsf{priv}_j}$, $u^{\triangleright i} = u'^{\triangleright i}$ and thus the missing transition $\left\langle u^{\triangleright i}, o''^{\triangleright i}, u'^{\triangleright i} \right\rangle \in T^{\triangleright i}$ is a loop. □

The missing loops never influence the shortest path and thus can be ignored (or added at will). Analogously, the same holds also for the public projection and for the i-private projection, but as in an i-private projection, all states are goal states (as goals are public and thus the i-private projection has an empty goal condition), the solutions to such abstraction are trivial.

Importantly, the same does not hold for the agent planning problems.

Theorem 37. *(Agent planning problem is not an abstraction) Let* $\mathcal{T}(\Pi^{\mathsf{G}}) = \left\langle S^{\mathsf{G}}, \bigcup_{i \in 1..n} \mathcal{O}^i, T^{\mathsf{G}}, s_I, S_\star \right\rangle$ *be the transition system of the global problem* Π^{G} *and* $\mathcal{T}(\Pi_i) = \left\langle S^i, \mathcal{O}^i, T^i, s_I^i, S_\star^i \right\rangle$ *the transition system of the agent planning problem* Π_i*. Then* $\mathcal{T}(\Pi_i)$ *is not an abstraction of* $\mathcal{T}(\Pi^{\mathsf{G}})$*.*

Proof. We can simply construct a counter-example. Let $\mathcal{M} = \{\Pi_1, \Pi_2\}$ with the respective global transition system being $\mathcal{T}(\Pi^{\mathsf{G}}) = \langle \{s_1, s_2, s_3\}, \{o_1, o_2\}, \{\langle s_1, o_1, s_2 \rangle, \langle s_2, o_2, s_3 \rangle\}, s_1, \{s_3\} \rangle$ such that $o_1 \in \mathcal{O}^{\mathsf{pub}_1}$ and $o_2 \in \mathcal{O}^{\mathsf{pub}_2}$. In the transition system $\mathcal{T}(\Pi_1)$ there is no (abstract) transition respective to $\langle s_2, o_2, s_3 \rangle$ as $o_2 \notin \mathcal{O}^1$ and thus $\mathcal{T}(\Pi_1)$ is not an abstraction of $\mathcal{T}(\Pi^{\mathsf{G}})$. □

Notice that in the above proof, the transition system $\mathcal{T}(\Pi_i)$ has no solution. This has the consequence, that in general, an agent α_i may not be able to solve its agent panning problem Π_i on its own.

3.4 Introduction to Multi-Agent Planners and Heuristics

Here, we formally define the notion of a multi-agent planner and a heuristic function based on the MA-STRIPS formalism. The definition carries over to MA-MPT analogously. Moreover, we define a number of properties of a multi-agent planner and of a heuristic relevant for further chapters.

Definition 38. A (distributed) algorithm is a *multi-agent planner* iff it accepts a multi-agent planning problem $\mathcal{M} = \{\Pi_i\}_{i=1}^n$ as an input and produces a set of sequences $\{\pi_i\}_{i=1}^n$ of actions from $A \cup \{\epsilon\}$ s.t. each π_i contains only actions from $A_i \cup \{\epsilon\}$. Such a planner is

(i) sound iff every set of sequences $\{\pi_i\}_{i=1}^n$ produced is a distributed multi-agent plan for $\mathcal{M}$,

(ii) complete iff the multi-agent planner produces a distributed multi-agent plan for any multi-agent problem $\mathcal{M}$ for which a plan exists.

Notice that according to Definition 38, any classical planner accepting the defined input (e.g., by converting it to the global STRIPS problem) is a multi-agent planner. In the Chapter 7 we focus on a more interesting property a multi-agent planner should satisfy to set it apart from the classical planners, namely the property of being *privacy-preserving*.

Heuristic search uses heuristics to guide the search towards a goal state. Formally, *heuristic function* is a function estimating the length of the shortest s-plan for a given state s. A *heuristic estimator* is the actual algorithm computing the heuristic function. We formalize two variants:

Definition 39. A function $h^{\mathcal{M}} : 2^P \to \mathbb{Z}_0^+$ is a *global heuristic function* for a multi-agent planning problem $\mathcal{M}$. If the function $h^{\mathcal{M}}$ is computed by a distributed heuristic estimator, we say it is a *distributed (global) heuristic*.

Definition 40. For an agent α_i, a function $h_i^{\triangleright} : 2^{P_i} \to \mathbb{Z}_0^+$ is an i-projected heuristic function.

The term *local heuristic* denotes the i-projected heuristic function for an unspecified agent. For brevity, we refer to a heuristic function simply as a heuristic.

Typically, a heuristic function is based on a simplification of the problem. We can divide existing planning heuristics into a number of families based on the approach to the simplification of the problem:

Relaxation heuristics are based on the idea of relaxing some of the constraints of the problem. In STRIPS the most common relaxation is delete relaxation where the delete effects of actions are ignored. Such planning problem is monotonic, that is, any achieved fact is never deleted again. Solving (sub-optimally) a monotonic planning problem is polynomial and thus ideal for a heuristic estimate. Example of such heuristics is the FF heuristic [Hoffmann and Nebel, 2001] which uses the length (or cost) of a sub-optimal relaxed plan as the heuristic value.

Abstraction heuristics are based on the idea of abstracting away some detail in the planning problem. Imagine an MPT planning problem restricted to a single variable. The length or cost of a solution of such restricted problem can be used as a heuristic estimate of the original problem. Example of such heuristics is the Merge&Shrink heuristic [Helmert et al., 2007] which is much more involved but is based on similar ideas.

Structural heuristics are computed from some structural information, an example of which are landmarks. Landmark is a state, an action, or a set of action which has to be present in every plan solving the problem (imagine for example a hub in star-like topology network). The LAMA Planner [Richter and Westphal, 2010] uses landmarks in combination with the FF heuristic and the LM-Cut heuristic [Helmert and Domshlak, 2009] is based on the idea of landmarks, but computed on the relaxed problem.

LP-based heuristics. Many of the already mentioned heuristics can be reformulated as a Linear Program (LP). Moreover, novel heuristics can be derived by formulating required properties of the heuristic as an LP, such as are the potential heuristics [Pommerening et al., 2015] where the LP is used to compute potentials of individual facts present in a state.

There is a number of properties of heuristics which influence their usability in heuristic search. The properties are the following.

Definition 41. A heuristic function h is

(i) optimal if $h(s)$ returns the actual shortest distance (or cost) to a goal state for each state $s \subseteq P$. An optimal heuristic is denoted as h^*,

(ii) admissible if for each $s \subseteq P$ holds $h(s) \leq h^*(s)$,

(iii) goal-aware if $h(s) = 0$ for each s such that $s_\star \subseteq s$, that is, s is a goal state,

(iv) safe if $h(s) = \infty$ implies that s is a dead-end, that is, there is no path to a goal state from s,

(v) consistent if for each state s and each successor s' of s such that $s' = s \circ a$ holds $h(s) \leq h(s') + \mathsf{cost}(a)$. Typically, goal awareness is assumed as part of consistency. Consistent heuristics are also often referred to as monotonic.

Note, that a well-known result is that consistency together with goal-awareness imply admissibility. Also, note, that an admissible heuristic does not have to be consistent. When comparing two heuristics we can use the following notion of dominance

Definition 42. Let h, h' be two heuristic functions. We say that h dominates h' if $h(s) \geq h'(s)$ for all states s.

The dominance of heuristics is relevant especially in the case of admissible heuristics where the dominating heuristic typically provides better heuristic guidance as it is closer to the optimal heuristic h^*. The quality of heuristic guidance can be assessed by comparing the number of states expanded in the A* search.

3.5 Discussion on the Complexity of Planning

The complexity results of classical planning were discussed in [Bylander, 1994] for STRIPS and in [Bäckström, 1992] for SAS+ (that is, MPT), a concise overview can be found in [Nau et al., 2004]. In this section, we provide a brief overview and discussion of the existing results. The result for both cases (as they are equivalent) is that classical planning (that is, its decision variant of plan existence) in the ground set-theoretic or state-variable representation is PSPACE-complete in the worst case. To show that planning is in PSPACE is easy. The maximum number of possible states in a STRIPS problem is exponential in the number of propositions, in particular, $2^{|P|}$. In the worst case, the plan may visit all such states and thus is polynomial in the size of P. In such case, no more than polynomial nondeterministic decisions is needed to prove the existence of such plan. To show that planning is PSPACE-complete, a polynomial encoding of polynomially bounded Turing machine to STRIPS planning can be used. As polynomially bounded Turing machine is PSPACE-complete, the same holds for STRIPS planning. The details of the compilation are out of the scope of the thesis and can be found in [Bylander, 1994]. An example of planning problem which indeed is PSPACE-complete is the binary counter or the sokoban puzzle. It is much more common for particular planning domains to fall in NP or even P. The complexity of used benchmarks is discussed in Section 3.6.

It is known, that some subsets of the STRIPS planning problems are easier to compute (e.g., [Helmert, 2003]). This knowledge can be used to construct P time heuristics. An example of such is the delete relaxation. According to [Bylander, 1994], to find a solution to the additive STRIPS problem, that is, a STRIPS problem without delete effects, falls in the class of polynomial problems (P). Nevertheless, to find an optimal plan in such a problem is still NP. By ignoring the delete effects, the relaxation heuristics transform a general planning problem into an additive one.

The complexity analysis of multi-agent planning was one of the main contributions of the Brafman&Domshlak's MA-STRIPS paper [Brafman and Domshlak, 2008]. Obviously, the STRIPS complexity carries over from classical planning, but the crucial question was how the complexity depends on the number of agents. The authors have based their analysis on the structure of agent interaction graph defined as $IG = \langle \mathcal{A}, E \rangle$, where the nodes of the graph are the agents and the edges are defined as follows

$$E = \{\langle \alpha_i, \alpha_j \rangle \mid \exists a_i \in A_i \wedge \exists a_j \in A_j \; s.t. \; \mathsf{pre}(a_j) \cap \mathsf{add}(a_i) \neq \emptyset \vee \mathsf{pre}(a_j) \cap \mathsf{del}(a_i) \neq \emptyset\}$$

in other words, there is an edge between the agents α_i and α_j if an action of agent α_i affects an action of agent α_j. An important property of the graph IG is its tree width ω. informally, tree width shows the likeness of the graph to a tree (or a complete graph in the other extreme), for a tree, $\omega = 1$, for a complete graph K_n, $\omega = n - 1$, and for a $n \times n$ planar graph $\omega = n$. More formally, tree width of a graph G can be defined as the minimum width of a tree decomposition of G among all such tree decomposition, where the width of a tree decomposition of graph G is the size of the largest set of vertices of G in the tree decomposition minus one.

Another term used in the complexity analysis of MA-STRIPS is the minimal number δ of public actions in a plan solving $\mathcal{M}$ (also known as the minimal length of a coordination sequence or a public plan). The overall complexity of the MA-STRIPS planning problem $\mathcal{M}$ is

$$f(\Pi_i) \cdot \mathsf{exp}(\delta) + \mathsf{exp}(\delta\omega)$$

where $f(\Pi_i)$ is a bound on the individual agents' planning problem complexity. This shows, that the MA-STRIPS planning complexity is either dominated by the individual planning problems, or is exponential in the tree width of the interaction graph and the number of coordination points, but not in the number of agents or the length of the global plan. This result was achieved largely thanks to the exclusion of joint actions which bring an exponential dependency on the number of agents as the number of joint actions is exponentially dependent on the number of agents (i.e., all combinations of all actions of all agents need to be considered).

The above analysis builds on the DisCSP-based approach of Brafman&Domshlak and Planning-First [Nissim et al., 2010] and thus its relevance to search-based approaches such as MAD-A* is not completely clear, but as the result depends on the exclusion of the joint actions, similar results can be expected also in the approach based on state-space search.

3.6 Benchmark Domains

To conclude the introduction to Multi-Agent Planning, we provide a description of the MAP domains used to evaluate the presented algorithms and techniques and also to illustrate the range of problems which can be formulated using the described formalisms and solved by the MAP planners. The set of benchmarks used in the thesis is drawn from the literature on the state-of-the-art MA-STRIPS based planners [Nissim and Brafman, 2014, 2012, Štolba and Komenda, 2014, Torreño et al., 2014, Borrajo, 2013, Maliah et al., 2014]. Later on, we have evaluated some of the algorithms on the domains used in the Competition of Deterministic and Multi-Agent Planners (CoDMAP) [Komenda et al., 2016] which are detailed at the end of this section.

The multi-agent domains are based on the classical IPC domains converted to multi-agent domains by choosing some objects to be treated as agents. This choice is arbitrary and we adhere to the conventions used in the cited papers. All agents are chosen so that each action is assigned to exactly one agent (this is required by the MA-STRIPS formalism), sometimes necessitating minor changes in the domain descriptions. Privacy is determined by the MA-STRIPS privacy definition—a fact is public if used by actions of multiple agents, an action is public if it uses some public fact. Here, we describe the domains and their complexities in more detail:

blocksworld In classical blocksworld domain, the task is to reassembly block towers on a table to different combinations. The multi-agent domain is the same as the classical blocksworld domain except for having multiple hands as agents, the holding and free facts being private. Each agent can solve the problem on its own, which makes it hard for MAP as for the projected heuristic it seems that the solution by other agents is cheaper (ignoring the private preconditions). All actions in the domain are public but have some private precondition, which means there are some dependencies among the actions not known to all agents. As an example, agents do not know that other agents must use pick-up before put-down, it seems to them that other agents can simply

move blocks by the put-down action without picking them up first, that is ignoring the necessary preconditions of the pick-up action such as that the block is free (on top). In [Helmert, 2003], the complexity of plan existence for blocksworld has been shown to be in P, whereas the optimal planning to be in NP.

depot In depot, the task is to move crates between depots and distributors, where the crates are located on pallets, thus combining a blocksworld-like domain with transportation. The trucks, depots, and distributors are the agents. Most of the actions are public, nearly a quarter of them have private preconditions. Only the drive-truck actions are completely private. Truck locations and truck loads (crates) are always private. The position of a crate is specified by a fact on. A crate can be either on a hoist at a depot or on a truck. When it is on a truck, the fact is private.

driverlog In driverlog, the drivers are driving trucks between locations. The drivers are agents (the problem is modified so that each action has the driver as a parameter). Their locations, walk action, and the fact that a driver is driving a truck are private, everything else is public. Most of the problems can be solved by a single agent, but unlike the blocksworld domain, this does not cause agents to significantly underestimate costs of other agents (the ignored costs are basically only those of walking actions, which are not that significant).

elevators08 The task in the elevators domain is to move people between floors using a number of slow and fast elevators, such that not all elevators stop at each floor. The elevators are the agents. The locations of passengers are public only if shared among multiple elevators (i.e., changing floors). Public actions are only those board and leave actions involving a floor accessible to multiple elevators. Most of the problems can be solved by a subset of agents (typically the slow elevators). All public actions have private preconditions on the state of the lift, its capacity, etc. which makes the agents significantly underestimate the costs of other agents.

logistics00 The logistics domain contains two types of agents: trucks and planes, transporting packages between cities. The goal specifies only the locations of packages. The transporting task often requires cooperation of several agents (that is so in the classical benchmark domains as well). Locations of packages accessible to only one truck are private to that truck. The loading and the unloading actions at these locations are private as well. The multi-agent logistics domain, similarly to the classical variant, is suitable for the relaxation heuristics. In [Helmert, 2003], the complexity of plan existence for logistics has been shown to be in P, whereas the optimal planning to be in NP.

openstacks The openstacks problems contain two agents: a manager and a manufacturer, which are added atop of the classical IPC domain. The goal of the problems is to produce and ship several orders. The manufacturer has a number of orders. The orders are started and shipped by the manager agents. Each order is for a combination of different products, and the manufacturer can only make one product at a time. The stacks are temporary storage spaces for open orders. The information about made products is private. The rest of the information is public and the two agents have to coordinate finishing the orders. Shipment is public because it is in the goal.

rovers The domain models Mars exploration rovers, each represented by one agent, moving between locations and performing experiments using various tools. The goal is to collect samples and communicate acquired data. Rovers problems can be well decomposed as each agent has its own private set of targets and reachable locations (even if a location is shared, the rovers do not interfere), but the communication channel is public, shared and imposes coordination constraints. If a sample can be analyzed only by a single rover, the location of this sample is the agent's private fact. Rovers is another domain suitable for relaxation heuristics, in contrast to logistics, the number of required interactions is lower, however, the private plans are longer.

satellites The problems of the domain contain agents representing satellites independently taking images in space by various instruments, which have to be powered from a limited on-board power source. The state of the instruments is private to the particular satellites. Pointing directions of each satellite are private unless they appear in the goal. The domain is almost completely decomposed to agents as each satellite is practically independent, sharing only the global goal.

sokoban Sokoban is a classical puzzle game, where the player is pushing blocks from their initial position to the goal position. Any block can end up in any goal position, but neither block nor the player can move through a block. The blocks are placed in a restricted labyrinth-like room and thus if a block is pushed too near to a wall, it can never be moved away from that wall, as the blocks can only be pushed, not pulled. This results in a presence of a large number of dead-ends, which is the major source of complexity in Sokoban. The multi-agent version introduces more players pushing the blocks which need to coordinate in order to avoid ruining each other's effort. In some configurations, a single agent is able to solve the complete problem, but it is easy to see that in many configurations this is not the case. Sokoban is PSPACE-complete [Culberson, 1997], placing it among the hardest planning problems.

woodworking08 In the woodworking domain, a certain amount of raw wood has to be processed by various tools to acquire certain shape, color, etc. In the multi-agent version, each tool is an agent. All facts and actions in this domain are public, except for the fact stating that a high-speed saw is empty or loaded. Subsequently, loading and unloading the high-speed saw are the only public actions with private preconditions.

zenotravel The zenotravel problems contain agents representing planes with limited fuel. The goal is to transport passengers between cities and park some of the planes at designated airports. Only the planes are represented by agents. Positions of planes are private and positions of passengers are public. Fly and zoom (fast fly) actions are private. The fuel levels and the positions of passengers in cities reachable by only one plane are also private.

In addition to the domains described above, the CoDMAP competition (see Appendix A) introduced two new domains[1] which we have used in the evaluation of optimal planning algorithms (Section 6). The new domains are not based on any classical planning domain and are the following:

Taxi Problems model on-demand transport in a city (see Figure 3.6.1-left). There are two types of agents: taxis and passengers. Each taxi and passenger are always at a particular location. A location can be free of taxis and two locations can be directly connected. Connected locations form a topology of the city. Each taxi can transport only one passenger from the location it stays at and only to a free drop off location (a location containing no other taxis at that time). A taxi can drive only between connected locations. All facts and action are public. The problem instances in the competition ranged from 2 taxis and 2 passengers to 3 taxis and 7 passengers (each representing an agent).

Wireless Problems model distributed gathering, transmission, and aggregation of data by sensor nodes in a wireless sensor network. The goal is to relay all data to a base station (see Figure 3.6.1-right). The base and sensors are represented by agents, where some of them are neighbors (they are in the range of their radios). The neighbor relation defines the topology of a virtual ad-hoc radio network among the sensors and the base. Sensors have a private battery charge of four possible levels: zero, low, normal and high. A sensor with more than zero energy can do a measurement and generate sensory data. Data generation decreases the battery capacity of a sensor node by one level. A sensor can add measurement data (possibly of other sensors) to a message (if it currently has the data and the message). This operation is called data aggregation and may be useful to reduce the number of transmissions in the network. A

[1] The domains were conceived and developed by Daniel Kovacs.

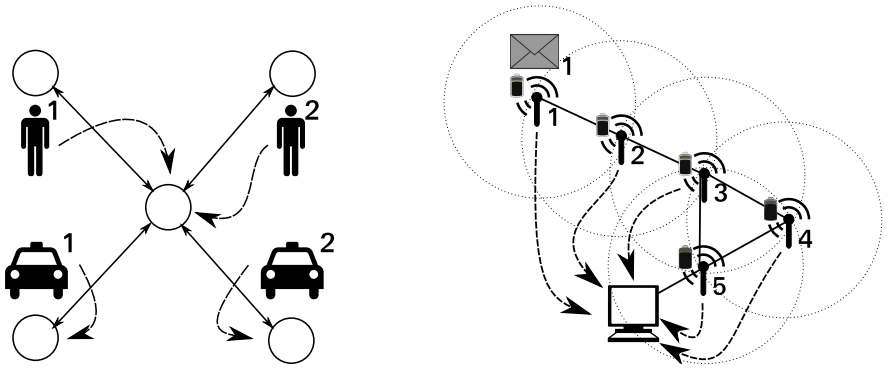


Figure 3.6.1: Example problem instances of the two novel CoDMAP domains: **taxi** (left) and **wireless** (right). The figures represent initial states of the easiest instances of the domains. Dashed arrows show the goals of individual agents. In the taxi problem, both passengers want to be transported to the central location and the taxi drivers want to end at the same locations they started from (the garage). In the wireless problem, all five sensor nodes are initially at normal battery level, and there is only one allowed message in the system represented by an envelope initially at sensor 1. Data from all sensors has to be gathered by the base station represented by a computer.

sensor with more than zero battery charge can send a message to a neighboring sensor or to the base station, which decreases its energy by one level. Receiving a message as well as extracting measurement data from a message does not change the energy level. The number of messages usable in parallel is limited, however, they can be reused sequentially. The data can be gathered either in an aggregated fashion, or without aggregation, depending on the number of allowed messages in the problem. The number of available messages can be set above the number of nodes, to provide complete freedom to a planner, when deciding about aggregation, thus efficiency and quality of the virtual communication. The energy levels of the sensors are private. The problem instances had always 1 base station, and 5 to 9 sensor nodes (6–10 agents).

Chapter 4

Distributed Computation of Relaxation Heuristics

The first research objective of the thesis is to answer the following question:

(Objective 1) How to compute classical planning heuristics in a distributed way?

We first focus on the case of satisficing planning, that is, the task to find any valid solution to the planning problem, regardless of its length or cost. For some planning problems, this is easier than finding the optimal plan (e.g., logistics as shown in [Helmert, 2003]), but for some problems, even finding any solution is NP-hard or PSPACE-complete (e.g., sokoban).

The heuristics for satisficing planning are not required to be admissible (Definition 41(ii)), as often they are used in the Greedy Best-First Search scheme (see Section 5.1) or even greedier search schemes such as Enforced Hill-Climbing [Hoffmann and Nebel, 2001]. Thus, the main aim of the heuristic design is to obtain informed heuristics, that is, heuristics which give close estimates of the actual solution length (or cost) regardless whether they are underestimating or overestimating the optimal solution.

In later IPC competitions, the quality of plans is assessed even in satisficing planning, which leads to the use of less greedy search schemes such as weighted-A* in LAMA [Richter and Westphal, 2010] but still using inadmissible heuristics.

This chapter describes our efforts in distributing the inadmissible relaxation heuristics. First, we describe the common principles of relaxation heuristics (Section 4). Next, in Section 4.1, we proceed with our first approach to the distribution of the FF heuristic [Hoffmann and Nebel, 2001] published in [Štolba and Komenda, 2013]. This approach aimed at returning provably the same results (i.e., heuristic values) to the centralized version of the FF heuristic, but showed to be not very efficient. Our next approach published in [Štolba and Komenda, 2014] and described in Section 4.2 is more general as it allows to distribute multiple relaxation heuristics, in particular, $h_{\max}$ and h_{add} (both [Bonet and Geffner, 1999]), and also h_{FF}. This general approach discarded the equality of the results to the centralized counterparts for the sake of efficiency. Finally, Section 4.3 describe a specific approach to the distribution of the FF heuristic aiming mostly for efficiency again (but using completely different approach than in the previous case), but also for privacy preservation.

Relaxation Heuristics

Probably the most successful and most studied family of heuristics for satisficing planning are the delete relaxation heuristics. The idea behind delete relaxation heuristics is to simplify the problem by ignoring negative effects of actions. In the STRIPS formalism, this means that an action $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$ is transformed to a relaxed form $a^+ = \langle \mathsf{pre}(a), \mathsf{add}(a), \emptyset \rangle$. A set of relaxed actions $A^+ = \{a^+ | a \in A)$ is used in the definition of a classical relaxed planning problem

$\Pi^+ = \langle P, A^+, s_I, G\rangle$ respective to the original planning problem $\Pi = \langle P, A, s_I, G\rangle$. By relaxation, the whole problem becomes additive, meaning that whenever a fact is added it is never deleted again and as both preconditions and goal are positive in STRIPS, any action that becomes applicable remains applicable in all subsequent relaxed states. The solution of a relaxed problem Π^+ is a relaxed plan π^+. Notice that thanks to the additivity of the relaxed problem, the relaxed plan can be represented as an unordered set of relaxed actions. From such set, the relaxed plan can be reconstructed by iteratively applying all actions as soon as they are applicable. This way, all facts achievable by any permutation of the actions in π^+ are achieved. This also means that in an optimal relaxed plan, no action is used more than once.

An optimal relaxed heuristic h^+ is defined as the length of an optimal relaxed plan π^+. In contrast to STRIPS planning, which is PSPACE-complete, finding an optimal relaxed plan π^+ is NP-Complete [Bylander, 1994], which is still impractical as a heuristic. In order to lower the complexity even more, approximations of h^+ are used in classical planning. The most commonly used approximation in satisficing planning is to use the length of a sub-optimal relaxed plan (RP) instead of an optimal one (the FF heuristic [Hoffmann and Nebel, 2001]). Finding a sub-optimal RP can have as low as polynomial complexity and therefore can be fast enough in practice.

The most commonly known relaxation heuristics are the $h_{\max}$, h_{add}, and h_{FF} heuristics. The $h_{\max}$ heuristic provides an admissible lower-bound estimate of h^+ by assuming that each fact can be achieved by the cheapest action and in order to achieve an action, only the cheapest precondition must be achieved (that is, all other preconditions are achieved as side-effects). The h_{add} works on a similar principle except for its pessimistic assumption that in order to achieve an action, all preconditions must be achieved separately and thus their heuristic costs can be added together. Both mentioned heuristics can be defined by a simple set of recursive equations (as in Equation 4.2.1-4.2.3) and computed by a relaxed reachability analysis. The h_{FF} proceeds by actually computing a relaxed plan, albeit a suboptimal one.

The main idea behind the FF heuristic is to find a sub-optimal relaxed plan by analyzing which facts are successively reachable by applied relaxed actions (reachability analysis). From this analysis, the relaxed plan is determined in a backward fashion. The principle can be understood as based on a notion of *supporter action a of fact p* which is an action a s.t. $p \in \mathsf{add}(a)$. Let $\Pi^+ = \langle P, A^+, s_I, G\rangle$ be a relaxed planning problem, then the principle of relaxed plan extraction is the following:

1. Initialize a set of unsupported facts U to contain all goal facts and a set of supported facts S to contain all initial state facts: $U \leftarrow G, S \leftarrow s_I$.
2. Move an unsupported fact p from U to a set of supported facts S and determine its supporter a.
3. Mark all preconditions of a as unsupported if not supported already: $U \leftarrow U \cup (\mathsf{pre}(a) \setminus S)$.
4. Loop 1–3 until all facts in U are supported: until $U \setminus S = \emptyset$.

There are many ways of implementing this high-level scheme (which differ mainly in the way the supporters are chosen) and many methods to perform the reachability analysis.

In general, the relaxed reachability analysis can be performed using Algorithm 1. The reachability analysis can be used to find all reachable facts in the relaxed planning problem and to determine whether the relaxed planning problem has a solution, if it does not, also the original planning problem is unsolvable (the other direction does not hold). Apart from being the base for relaxation heuristics (using a more complex algorithm), relaxed reachability analysis is often used in the grounding process from PDDL to STRIPS or MPT (e.g., [Helmert, 2006]). Note that because facts are only added and in each iteration at least one fact is added, the number of iterations of Algorithm 1 is linear in the size of P.

Relaxation Heuristics in Multi-Agent Planning

Before describing the particular approaches to the distributed computation of the particular relaxation heuristics, we introduce a number of common definitions. A baseline in distributed heuristic search is an

Algoritmus 1: Relaxed Reachability Analysis

Procedure `Relaxed-Reachability` (A^+, s)
  $R_0 \leftarrow s$; //The set of reachable facts (relaxed state)
  $k \leftarrow 0$;
  while $R_k \neq R_{k-1}$ **do**
    $A_k \leftarrow \{a \in A^+ | \mathsf{pre}(a) \subseteq R_k\}$; //Find all applicable actions
    $R_{k+1} \leftarrow R_k \cup \{p | p \in \mathsf{add}(a) : \forall a \in A_k\}$; //All reachable facts
    $k \leftarrow k + 1$;
  return R_k;

i-projected (Definition 40) relaxation heuristic is computed on a relaxed i-projected problem, formally

$$\Pi_i^{\triangleright +} = \left\langle P_i, A_i^{\triangleright +} = \{a^+ | a \in A_i^{\triangleright}\}, s_I^{\triangleright i} = s_I \cap P_i, G \right\rangle . \tag{4.0.1}$$

In the next sections we describe a number of approaches to the distribution of the relaxed heuristics, namely $h_{\max}$, h_{add}, and h_{FF}. The distributed relaxation heuristics are computed on a relaxed multi-agent problem, formally for a set $\mathcal{A} = \{\alpha_i\}_{i=1}^n$ of agents

$$\mathcal{M}^+ = \{\Pi_i^+\}_{i=1}^n$$

where $\Pi_i^+ = \left\langle P_i, A_i^+, s_I \cap P_i, G \right\rangle$ is the relaxed problem of agent α_i.

The distributed reachability based on Algorithm 1 is shown in Algorithm 2. The principle is straightforward. The agent α_i initiating the analysis starts by performing the reachability analysis using his set of actions A_i^+ (by calling Algorithm 1). If any facts were added, the newly reachable public facts are broadcasted to all other agents. Each agent α_j, upon receiving the set of reachable facts, extends the set using its respective relaxed actions A_j^+ using the same procedure and thus broadcasts the result. When α_i receives such result, cannot extend it anymore, and there are no possible replies from other agents, the distributed procedure terminates and returns the resulting set of globally reachable facts.

Algoritmus 2: Distributed Relaxed Reachability Analysis

Procedure `Distributed-Relaxed-Reachability` $(\alpha_i, A_i^+, s^{\triangleright i})$
  $R^{\mathsf{G}} \leftarrow s^{\triangleright i}$; //The set of globally reachable facts (relaxed state)
  `extendReachability` $(\alpha_i, A_i^+, R^{\mathsf{G}})$;
Procedure `extendReachability` $(\alpha_i, A_i^+, R^{\mathsf{G}})$
  $R^{\mathsf{L}} \leftarrow$`Relaxed-Reachability` $(A_i^+, R^{\mathsf{G}}) \setminus R^{\mathsf{G}}$;
  if $R^{\mathsf{L}} \neq \emptyset$ **then**
    send $M_{\mathsf{REACH}} = \left\langle \alpha_i, R^{\mathsf{L}} \cap P_i^{\mathsf{pub}} \right\rangle$ **to** all $\alpha_{j \neq i}$;
    $R^{\mathsf{G}} \leftarrow R^{\mathsf{G}} \cup R^{\mathsf{L}}$
  else if *no messages pending* **then**
    return R^{G}
Procedure `receiveMessage` $(\alpha_i, M_{\mathsf{REACH}} = \left\langle \alpha_k, R^{\mathsf{pub}} \right\rangle)$
  $R^{\mathsf{G}} \leftarrow R^{\mathsf{G}} \cup R^{\mathsf{pub}}$
  `extendReachability` $(\alpha_i, A_i^+, R^{\mathsf{G}})$;

The multi-agent distribution of relaxation heuristics can be based either on the distribution of the reachability analysis as outlined in Algorithm 2 and described in much more detail in in Section 4.1 and

Section 4.2, or on the distribution of the relaxed plan extraction as described in Section 4.3. The privacy properties of distributed relaxation heuristics are analyzed in Section 7.3.4.

An alternative approach to distributed reachability analysis was published in [Torreño et al., 2014], where the reachability is performed using Domain Transition Graphs (DTGs) [Helmert, 2006]. The DTG-based FF heuristic follows a similar high-level scheme described above. The difference is, that the reachability is not assessed using a supporter relation based on an RPG, but instead by an existence of a path in the respective DTG. The relaxation here is not achieved by ignoring delete effects, but by accumulating the reachable variable-value pairs. Note that this is in accordance with the most common interpretation of delete relaxation in FDR, which is an accumulating semantics (variables are accumulating values instead of switching). The difference is, that the accumulating semantics is exhibited only in the set of supported facts, but the DTGs are unaffected and keep their switching semantics. The benefit of the DTG heuristic for MAP is assumed by Torreño et al. [2014] to be that the underlying structure (the DTGs) can be built only once (and the transitions allowed by other agents cached), whereas the distributed RPG has to be built for each state from scratch. We compare the approach of Torreno with our approach in Section 4.4.4.

4.1 Multi-Agent Fast-Forward Heuristic

In this section, we present a formal and algorithmic adaptation of the Fast-Forward h_{FF} [Hoffmann and Nebel, 2001] heuristic for multi-agent planning, originally published in [Štolba and Komenda, 2013]. We argue that such treatment is important as it demonstrates algorithmic challenges in the decentralization of computation of h_{FF} and other related heuristics. Additionally, since the h_{FF} heuristic is based on relaxed planning, we propose a multi-agent (MA) approach for building factored relaxed planning graphs among the agents.

4.1.1 Agent Relaxed Planning Graph

A classical technique for finding the relaxed plan is to build a Relaxed Planning Graph (RPG). RPG is a graph representing the reachability of facts and applicability of actions in the relaxed problem.

To obtain a more informed global heuristic estimate in a MA planning problem using the estimation based on an RPG, the RPG has to be decentralized. In this work, we propose a distributed global RPG in form of a set of distinct Agent RPGs. Such Agent RPG (ARPG) contains only facts of its owner agent. The initial state is the projection for that agent and since the goals are treated as public, all agents have complete goals in their ARPGs. The usage of actions is straightforward in the case of owner agent's internal and public actions which are used equally as in a classical RPG. Additionally, the Agent RPGs are extended by projections of other agents' public actions which were reachable by their particular owners. This extension enables the agents to take other agents' capabilities into account, but only at the time points, where their owners are able to reach them.

Definition 43. An *agent relaxed planning graph (ARPG)* is a directed, labeled and layered graph $\mathcal{R}_i = (P_i \cup A_i^{\triangleright +}, E)$ of one particular agent α_i for a relaxed multi-agent planning task $\mathcal{M}^+$. As in RPG, the nodes of the graph represent propositions P_i and actions A_i^+. The arcs E represent links between the propositions and the actions.

A k-th proposition layer and action layer will be denoted as P_i^k and A_i^k respectively. The layers alternate, so that $(P_i^0, A_i^0, P_i^1, A_i^1, \ldots, A_i^{n-1}, P_i^n)$ and all layers $P_i^k \subseteq P_i$ and all layers $A_i^k \subseteq A_i$. The first proposition layer P_i^0 contains nodes labeled by propositions of the agent's projection of the initial state, formally

$$P_i^0 = \{p | p \in s_I \cap P_i\}.$$

Each action layer contains action nodes for all applicable relaxed actions of the agent α_i in a state represented by the previous fact layer and public projections of other agents' public actions reachable

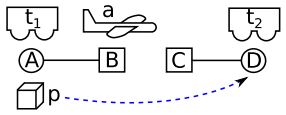


Figure 4.1.1: Logistics example with three agents.

in the same layer

$$A_i^k = \{a | a \in A_i^{\triangleright +}, \mathsf{pre}(a) \subseteq P_i^k\} \cup \bigcup_{j \in 1..n, j \neq i} \{b^{\triangleright i} | b \in P_j^k\}.$$

In all successive fact layers, each fact p is copied to the next fact layer by using a special no-op action $\epsilon_p = \langle \{p\}, \{p\}, \emptyset \rangle$ and transforms the facts by actions in the previous action layer, since for all relaxed actions $\mathsf{del}(a) = \emptyset$, we can write

$$P_i^k = P_i^{k-1} \cup \{p | p \in \mathsf{add}(a), a \in A_i^{k-1}\}.$$

Finally, one of the following terminating conditions has to hold for the last fact layer P_i^n:

- the last fact layer fulfills the goal condition $G \subseteq P_i^n$,
- or $P_i^n = P_i^{n-1}$, meaning there are no additional actions which can extend further fact layers (i.e., a fixed-point was reached).

The arcs in ARPG represent applicability and application of actions in the relaxed states (fact layers). We can split the arcs between two fact layers P_i^k and P_i^{k+1} into three groups. The first one contains arcs among facts of layer P_i^k and preconditions of actions in a layer A_i. The second one contains relation between effects of actions and next induced fact layer P_i^{k+1}. Additionally, there are arcs for the no-op actions respective to all facts from the previous layer . Formally,

$$\begin{aligned} E_{i,k}^{\mathrm{pre}} &= \{(p_k, a_k) | p_k \in \mathsf{pre}(a_k), a_k \in A_i^k\}, \\ E_{i,k}^{\mathrm{add}} &= \{(a_k, p_{k+1}) | a_k \in A_i^k, p_{k+1} \in \mathsf{add}(a_k)\}, \\ E_{i,k}^{\epsilon} &= \{(p_k, p_{k+1}) | p_k \in P_i^k, p_{k+1} \in P_{k+1}, p_k = p_{k+1}\} \end{aligned}$$

and $E_i^k = E_{i,k}^{\mathrm{pre}} \cup E_{i,k}^{\mathrm{add}} \cup E_{i,k}^{\epsilon}$ where ϵ represents the no-op action. Now we will provide an algorithm for distributed building of ARPGs.

Distributed RPG Algorithm

The algorithm starts with each agent building an ARPG using only its own internal and public actions. An iterative process is then initiated, in which the agents exchange information about their *public* actions and extends their ARPGs with projected public actions of other agents. The algorithm terminates when the goal (or a fixed-point) is globally reached and there are no more messages to process. The full details and pseudocode of the ARPG building algorithm are provided in [Štolba and Komenda, 2013], here, we briefly rephrase the main principles of the algorithm and show how it works on an example.

In the **init** phase, a Relaxed Planning Graph $\mathcal{R}_i$ is built using only agent's own actions A_i from the initial state projection $s_I^{\triangleright i}$. A function $e : A_i^{\mathsf{pub}} \rightarrow \mathbb{N}$ is used to map the agent's public actions to their *earliest layer of appearance* (the earliest layer of appearance of an action a in a RPG or an ARPG is the first action layer, where a is applicable). After the initialization phase, reaching the goal (or a fixed-point) is checked, and if positive, all agents are informed that the agent is idle now. Next, a **check** procedure is responsible for checking whether $\mathcal{R}_k$ contains any public actions. If so, each public action is sent to all other agents $\alpha_j \in \mathcal{A} \setminus \alpha_i$ as a public projection $a^{\triangleright}$ together with its earliest layer of

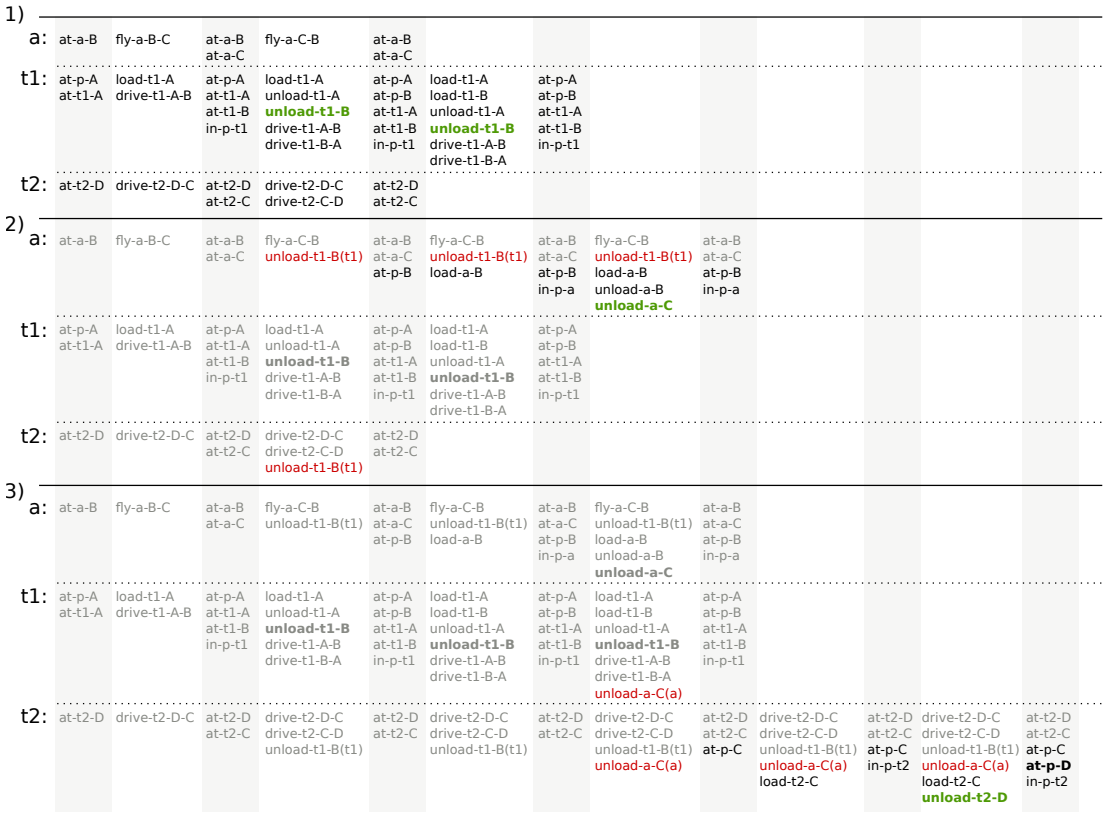


Figure 4.1.2: Distributed building of Agent Relaxed Planning Graphs decomposed into iterations.

appearance $e(a)$, unless it was already sent with equal or lower $e(a)$ before (this can happen in future **check** calls).

In the algorithm, there are four asynchronous message types possibly received by α_i from some other agent α_j. The first one contains a projection of other agent's public action $a^{\triangleright}$ together with its earlier layer of appearance $e(a)$. If received, the action $a^{\triangleright}$ is integrated into $\mathcal{R}_i$ on the $e(a)$-th layer and the change is propagated into further layers, so that all actions newly applicable in the following layers are applied accordingly. Then the built ARPG is checked, whether new public actions (and public actions newly applicable on earlier layers) are reachable and whether the goal or the fixed-point was reached. The last three messages maintain the control information needed for distributed termination detection [Chandy and Lamport, 1985]. The *acks* counter keeps track of the number of sent external actions and postpones termination until all sent actions are processed. If an *idle* message is received, there are no pending *acks* and the number of idle agents is equal to $|\mathcal{A}|$, the algorithm terminates and the resulting ARPG $\mathcal{R}_i$ is returned. Since *not-idle* and *ack* messages are sent in this particular order and the messages from one agent are presumed to keep ordering, the algorithm terminates synchronously when all external actions are processed and no messages are pending.

Example. (Logistics) In Figure 4.1.2, the ARPG building algorithm is applied on a slightly larger logistics example depicted in Figure 4.1.1. Although the algorithm is running asynchronously, we can decompose it for clarity into several iterations. In the first iteration, the ARPGs are built using only the actions of the respective agents a, t_1 and t_2 (airplane and two trucks). Notice the bold green action unload-t1-B, which is a public action of the truck t_1, can be applied thanks to the initial position of the package. In the next iteration, a projection of the public action is broadcasted and received by

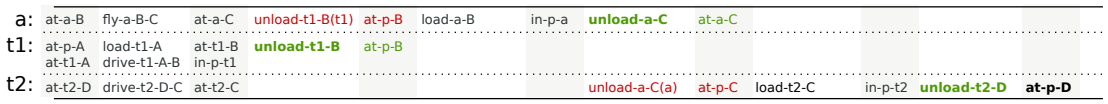


Figure 4.1.3: Multi-agent Relaxed Plan

other agents. Upon receiving, their ARPGs are updated, which for the airplane means that the ARPG is expanded with further layers. Another public action unload-a-C is applied and therefore broadcasted. In the third iteration, the projection of the airplane's unload action is added to the ARPGs of the trucks. For truck t_1 it has no effect, but it allows truck t_2 to expand the ARPG and reach goal at-p-D. Notice that when the projected unload-a-C(a) was received by truck t_2, its ARPG was first extended to have enough layers for the action to be added to the correct layer.

Although not shown in Figure 4.1.2, the algorithm would continue with one more iteration after broadcasting the public action reached by truck t_2, resulting in all agents having ARPGs with the same number of layers and all having reached the goal. Additionally, the algorithm does not have to terminate when the goal is reached but can continue until the fixed-point, which can be desirable in some situations and which is also the case when the goal is not reachable.

All ARPGs together form a single factored RPG $\mathcal{R} = \{\mathcal{R}_i\}_{i=1}^{n}$ which is in [Štolba and Komenda, 2013] shown to be equal to the RPG of a centralized problem Π^{G} equal to $\mathcal{M}$. The centralized RPG can be obtained by performing the union of each fact and action layer and by removing the projected actions from the action layers. Formally for each k, $P^k = \bigcup_{i=1}^{n} P_i^k$ and $A^k = \bigcup_{i=1}^{n} (A_i^k \cap A_i)$. We omit the proof here for brevity as the technical details of the proof are not important for further understanding of how the heuristic works.

4.1.2 Distributed Relaxed Plan Extraction

With the help of the ARPGs, the Fast-Forward heuristic estimate can be straightforwardly adapted to the multi-agent setting. We will denote such heuristic as h_{MAFF}. The multi-agent (MA) relaxed plan backing the h_{MAFF} estimate can be in general spread over all ARPGs of the agents in the team as illustrated in Figure 4.1.3. The most left achieving actions have to be considered from all agents. In the case of projected public actions, the owner agent has to define part of the the relaxed plan, possibly using its internal actions, to achieve the internal facts of the provided public action. Additionally, the relaxed plan has to share public actions which are required by more agents at the same layers. The private parts of the relaxed plan do not have to be communicated. The final heuristic estimate is the count of actions of the MA relaxed plan.

Let $\pi^+ = \{\pi_i^+\}_{i=1}^{n}$ be the relaxed plan for $\mathcal{M}_s$ and let $m = \max_{i \in 1..n} |\pi_i^+|$. Each π_i^+ is defined using the ARPG $\mathcal{R}_i$ of agent α_i built from the state s. The MA relaxed plan π^+ is defined recursively from the last action layer m for each agent α_i. The action layer $A_i^{m*} \subseteq A_i^m$ is a minimal set of actions from A_i^m, achieving the goal facts. For each two successive action layers A_i^{k-1*}, A_i^{k*} of π_i^+, the layer A_i^{k-1*} is a minimal set of actions from A_i^{k-1} achieving all preconditions of A_i^{k*}. If there is a no-op arc $(p_{k-1}, p_k) \in E_{i,k}^{\epsilon}$, the fact p_k does not need an explicit achieving action from this particular layer as it will be achieved by an action from an earlier (more left) layer.

Moreover, if a public projection of some other agent's action is selected as an achiever, formally $a^{\triangleright} \in A_i^k$ for some k such that $a \in A_j$, then also $a \in A_j^k$. Algorithmically this is done by sending a message from α_i to α_j informing the latter that its action has been selected in the layer k.

This principle effectively selects the globally most-left achievers of a fact as proposed by FF heuristic. Thus, $h_{\mathrm{MAFF}}(s)$ can be computed by first building the set of ARPGs $\mathcal{R} = \{\mathcal{R}_i\}_{i=1}^{n}$ for the relaxed MA planning problem $\mathcal{M}^+$, then simultaneously extracting relaxed plans π_i for each agent while sharing the information on use of the public projected actions and finally summing the lengths of the resulting relaxed plans, excluding projections of other agent's public actions.

Theorem 44. *Let a MA relaxed plan $\{\pi_i^+\}_{i=1}^n$ be a solution of the MA relaxed problem $\mathcal{M}_s^+ = \{\Pi_i^+\}_{i=1}^n$, where $\Pi_i^+ = \langle P_i, A_i^+, s \cap P_i, G\rangle$ and s is the state, we are estimating the cost for. Let*

$$h_{\mathrm{MAFF}}(s) = \sum_{i=1}^{n} |\pi_i \cap A_i|$$

then $h_{\mathrm{MAFF}}(s) = h_{\mathrm{FF}}(s)$ where $h_{\mathrm{FF}}(s)$ is computed on the global problem Π^{G} respective to $\mathcal{M}$.

The full proof can be found in [Štolba and Komenda, 2013]. Intuitively, as the underlying structure of ARPGs is equivalent to an RPG of the global problem and the construction of the relaxed plan follows the same rules as in the centralized version, also the resulting h_{MAFF} relaxed plan is equal with the exception of the publicly projected actions, which are removed in the computation.

Example. (Logistics) Continuing with the logistics example in Figure 4.1.1, based on the set of ARPGs in Figure 4.1.2 we proceed to produce the distributed relaxed plan as in Figure 4.1.3 as follows. First the agent t_2 needs to achieve the goal at-p-D. This can be done using his private action unload-t2-D, which in turn requires the preconditions in-p-t2 and at-t2-D to be satisfied. The latter is satisfied by a no-op action (propagating right to the initial state), the former is satisfied by a load action of t_2. When the unload-a-C(a) action is reached, as it is a projected action of agent a, a is informed about the necessity of unload-a-C(a) in layer $A^3_{\mathsf{t}_2}$ and thus also of unload-a-C in A^3_{a}. Now the agent a needs to satisfy preconditions of unload-a-C(a). This process continues until all preconditions of all agents are satisfied by the initial state. The resulting heuristic is obtained by the sum of the lengths of the relaxed plans of particular agents minus the number of projected actions used which is $(4-1)+3+(4-1) = 9$.

4.2 Recursive Distributed Relaxation Heuristics

In this section, we present the distribution of a general principle of delete relaxation heuristics in MA-STRIPS planning with state-of-the-art implementation approaches, originally published in [Štolba and Komenda, 2014]. We focus on the following classical delete relaxation heuristics: (i) inadmissible h_{add}, (ii) admissible $h_{\max}$ both [Bonet and Geffner, 1999] and (iii) inadmissible h_{FF}, which was published in [Hoffmann and Nebel, 2001]. In the following sub-sections, we present efficient multi-agent distributions of those three heuristics. The distribution principle aims for efficiency but does not guarantee admissibility even for the distributed variant of the $h_{\max}$ heuristic. For an admissible distributed $h_{\max}$ heuristic, see Section 6.1.

4.2.1 Distribution of the Additive and Max Heuristics

Let $\Pi^+ = \langle P, A^+, s_I, G\rangle$ be a classical STRIPS relaxed problem with a cost function $\mathsf{cost} : A^+ \to \mathbb{R}_0^+$. Both additive and max heuristics follow a very similar principle and are typically formalized as a set of recursive equations, such the following for h_{add}:

$$h_{\mathrm{add}}(P, s) = \sum\nolimits_{p \in P} h_{\mathrm{add}}(p, s) \tag{4.2.1}$$

$$h_{\mathrm{add}}(p, s) = \begin{cases} 0 & if\ p \in s \\ h_{\mathrm{add}}(\arg\min_{a \in O(p)} [h_{\mathrm{add}}(a, s)], s) & \\ & \text{otherwise} \end{cases} \tag{4.2.2}$$

$$h_{\mathrm{add}}(a, s) = \mathsf{cost}(a) + h_{\mathrm{add}}(\mathsf{pre}(a), s), \tag{4.2.3}$$

where P is a set of propositions (i.e., goal or action preconditions), s is a state, a is an action and $O(p)$ is a set of actions which achieve p, formally $O(p) = \{a \in \alpha | p \in \mathsf{add}(a)\}$. The equations defining $h_{\max}$

are the same except for Equation 4.2.1 where is a max function instead of sum, therefore everything we state about $h_{\max}$ applies analogously to $h_{\max}$.

In the multi-agent setting, where $\mathcal{M}^+ = \{\Pi_i^+\}_{i=1}^n$ and each $\Pi_i^+ = \langle P_i, A_i^+, s_I \cap P_i, G\rangle$, some of the actions in the $\arg\min$ clause in Equation 4.2.2, where we are choosing the minimal cost action among actions achieving the proposition p, may be projections of other agent's public actions. In such a case, there are two options how to handle the situation.

One option is to ignore the fact that the action is a projection and continue as if it was an ordinary action. This way, we may leave out some preconditions of the action (private to the owning agent), but we still get lower or equal estimate of the action cost (by including the private preconditions we can only increase the cost), overall obtaining a *projected heuristic*. Obviously, projected heuristics require no communication at all.

The other option is to always compute the true estimate. Let α_i be the computing agent and α_j the owner of such action a and s the state for which the heuristic is computed. In order to do so, the agent α_i sends a request $r = \langle a^{\triangleright}, s^{\triangleright i}\rangle$ to the agent α_j to obtain the true estimate of the cost of the action $a^{\triangleright}$. Upon receiving the request, agent α_j calls $h_{\max}(\mathsf{pre}(a), s^{\triangleright j})$ and returns the result in a reply. It is obvious that in order to compute the heuristic estimate, agent α_j may need to send similar requests to other agents, or even back to agent α_i. This way, we end up with a distributed recursive function, which returns exactly the same results as a centralized $h_{\max}$ on a global problem Π^{G} respective to $\mathcal{M}$, since for every projection $a^{\triangleright}$ of action $a \in A_j$, the true cost of a is obtained from the agent α_j.

A middle ground between the presented two extremes is to limit the recursion depth δ. If the *maximum recursion depth* δ_{max} is reached, all projected actions are evaluated without sending any further requests. This limit introduces another relaxation of the original problem where the interaction between agents is limited—the *agent coupling relaxation*. Such heuristic estimation is always lower or equal to the heuristic estimation in the global problem Π^{G} using a centralized heuristic estimator, because ignoring preconditions of an action in its projection can never increase the cost of the action. By limiting the recursion depth to $\delta_{max} = 0$, we return back to the *projected heuristic*, where all interactions between agents are relaxed away. This approach is also one possibility of tackling the **(Objective 2)** we have investigated in our work.

Relaxed exploration.

Although the definition of $h_{\max}$ by a set of recursive equations is intuitively clear and provides good theoretical background, in practice, the recursive functions are typically not used. Recursive calls are limited by the call stack. Converting such recursion, where the recursive call is within a complex function such as $\arg\min$ into iteration is possible, but rather cumbersome. Instead, the idea of *relaxed exploration* is typically utilized.

The *relaxed exploration* is, in fact, a reachability analysis of the relaxed planning problem, which can be conveniently seen as building a *relaxed planing graph* (RPG). A relaxed planning graph is a layered (alternating fact and action layers) directed graph. In its first layer it contains all facts which hold in the initial state, the next layer contains all actions of which preconditions are satisfied in the previous layer (and no-op actions), the next layer contains all (add) effects of the actions from previous layer and so forth, see Section 4.1 for details. In practice, an RPG is often not built explicitly, but the exploration is achieved via an effective representation we will refer to as an *exploration queue* (based on the Fast-Downward planning system Helmert [2006]).

The *exploration queue* considers only *unary actions*—actions which have a single proposition as an add effect (any relaxed problem can be converted so it contains only unary actions). The *exploration queue* is supported by a data structure representing the *precondition-of* and *achieved-by* relations. The queue is initialized with the propositions which are true in the state s. Until the queue is empty, a proposition p is polled, it is checked whether p is a goal proposition and if so, whether all goals are satisfied. If not, for each action that depends on p ($p \in \mathsf{pre}(a)$ where $a \in A_i^{\triangleright}$), the action cost is incremented by the cost of proposition p (that is either added for $h_{\max}$, or maxed for $h_{\max}$) and if there

are no more unsatisfied preconditions of the action, the action is applied. The process is detailed in Algorithm 3, lines 1–13. Thanks to the sole use of *unary operators*, the application of an action a can be interpreted as adding the (only one) proposition $p = \mathsf{eff}(a)$ to the *exploration queue*, thus the procedure is named enqueueProposition (line 10).

The effectiveness of this approach is achieved because, during the relaxed exploration, cost estimates of facts and actions can be conveniently computed and once all goal facts are reached, the heuristic can be computed by simple sum or max of costs of all goal facts. In general, the approach can be seen as a bottom-up computation of recursive function using dynamic programming.

Distributed relaxed exploration.

An algorithm capable of building RPGs in a distributed manner published in Štolba and Komenda [2013] was presented in the previous section (Section 4.1). The major drawback of the approach was the necessity to build the RPG for each state by all agents at once, thus preventing the search to run independently in parallel. It was shown that the resulting heuristic estimate is equal to the centralized estimate. In this section, we do not place the requirement of obtaining the same value as in the centralized variant, which allows us to build a much more efficient algorithm. The algorithm is based on building the *exploration queue* and requesting other agents when projections of their actions are encountered. Moreover, the presented algorithm allows for precise control of the recursion depth and thus enables us to trade-off the estimation precision with the computation and communication complexity.

The basic process of building the *exploration queue* $\mathcal{Q}$ is similar to the centralized version as described in the previous section. The main principle of the distributed process is that whenever a projection of some other agent's action should be applied (and its effect added to the queue), a request is sent to the owner of the action to obtain its true cost. The effect of the action is added to the queue only after the reply is received. Note that when computing the reply, the agent may need to send requests as well, thus ending up with a distributed recursion. In order to effectively handle the recursion, it is flattened so that all requests are sent by the initiator agent and the replies are augmented with the parameters of the next recursive call. The context of the proposition p is kept throughout the computation for all unfinished propositions.

The exploration part of the algorithm is shown in Algorithm 3, whereas Algorithms 4 and 5 details the inter-agent communication. The entry point of the algorithm is the **relaxedExploration** procedure. First, it is invoked with the r parameter set to true, indicating that whenever a projected action is encountered, a request is sent to its owner.

The main difference between the centralized and distributed approaches lays in the **enqueueProposition** procedure. If the cost of the action improves the current cost of the proposition, the cost of the proposition is set equal to the cost of the action, as usual, but if the action is a projection $a^{\triangleright}$ such that $a \in A_j$ for some agent α_j and sending of requests is enabled, i.e., $r = \mathsf{true}$, a request message $M_{\mathsf{req}} = \langle s, a, \delta \rangle$ is sent to α_j. The request message is a tuple where s is the current state, a is the action, and initial recursion depth $\delta = 0$. Otherwise, the proposition is added to the *exploration queue* as usual.

Processing the messages.

When the request message is received by the agent α_j (see Algorithm 4, **processRequest**), the *relaxed exploration* is run with the goal being the preconditions of the requested action a and without sending any requests, i.e., $r = \mathsf{false}$. After finishing the exploration, the set $\mathcal{P}$ of public actions which have contributed to the resulting heuristic estimate is determined (line 4). In principle, the procedure is similar to extracting a relaxed plan in the FF heuristic. A reply $M_{\mathsf{re}} = \langle h, \mathcal{P}, \delta \rangle$ is sent, where h is the computed heuristic value, $\mathcal{P}$ is the set of the contributing public actions and δ is the current recursion depth.

Receiving the reply from agent α_i is managed by procedure **processReply** in Algorithm 5. If the recursion depth has already reached the limit $\delta > \delta_{max}$, the original $\mathsf{receiveReplyEnqueueCallback}(p, h)$

Algoritmus 3: Distributed Relaxed Exploration

Require: Boolean flag r (true when first called), global exploration queue $\mathcal{Q}$

relaxedExploration(r)**:**

1: **while** $\mathcal{Q} \neq \emptyset$ **do**
2: $p \leftarrow poll(\mathcal{Q})$
3: **if** $p \in G$ **and** $\mathsf{achieved}(p') : \forall p' \in G$ **then**
4: **return**
5: **end if**
6: $O_p \leftarrow \{a \in A_i^{\triangleright} | p \in \mathsf{pre}(a)\}$
7: **for all** $a \in O_p$ **do**
8: **increment** $\mathsf{cost}(p)$ **by** $\mathsf{cost}(a)$
9: **if** $\mathsf{achieved}(p') : \forall p' \in \mathsf{pre}(a)$ **then**
10: $\mathsf{enqueueProposition}(a, \mathsf{eff}(a), r)$
11: **end if**
12: **end for**
13: **end while**

Require: Action a, proposition p, Boolean flag r

enqueueProposition(a, p, r):

14: **if** $\mathsf{cost}(p) = \perp$ **or** $\mathsf{cost}(p) > \mathsf{cost}(a)$ **then**
15: $\mathsf{cost}(p) \leftarrow \mathsf{cost}(a)$
16: **if** r **and** $a \in A_i^{\triangleright} \setminus A_i$ **then**
17: **send** request message $M_{req} = \langle s, a, 0 \rangle$
to $\mathsf{owner}(a)$,
process the reply by
$\mathsf{receiveReplyEnqueueCallback}(p, \_)$
18: **else**
19: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{p\}$
20: **end if**
21: **end if**

Require: Heuristic estimate h, proposition p (set from enqueueProposition)

receiveReplyEnqueueCallback(p, h):

22: **if** $\mathsf{cost}(p) > h$ **then**
23: $\mathsf{cost}(p) \leftarrow h$
24: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{p\}$
25: **end if**
26: $\mathsf{relaxedExploration}(\mathsf{false})$
27: **if** *no unresolved requests* **then**
28: **return** compute the total cost
29: **end if**

from Algorithm 4 is called for the action a, the cost estimate of the context proposition p is finalized and p is added to the *exploration queue*. Since the messaging process is asynchronous, the original *relaxed exploration* has already terminated, therefore it is started again (line 7), with the original data structures and with the newly evaluated proposition added to the queue. When the exploration is finished and there

Algoritmus 4: Request Processing

Require: Request message $M_{\mathsf{req}} = \langle s, a, \delta \rangle$, where s is a state, a an action, δ the recursion depth, α_i the sender

processRequest($M_{req} = \langle s, a, \delta \rangle$, α_i):

1: $\mathcal{Q} \leftarrow \{s\}$
2: relaxedExploration(false)
3: $h \leftarrow$compute the total cost
4: $\mathcal{P} \leftarrow$mark public actions
5: **send** reply message $M_{\mathsf{re}} = \langle h, \mathcal{P}, \delta \rangle$ to α_i

Algoritmus 5: Reply Processing

Require: Reply message $M_{re} = \langle h, P, \delta \rangle$, where h is the heuristic estimate, $\mathcal{P}$ a set of actions, δ the recursion depth, p the proposition from context

processReply($M_{re} = \langle h, \mathcal{P}, \delta \rangle$):

1: **if** $\delta < \delta_{max}$ **then**
2: $h_{sum} \leftarrow h$
3: **for all** $a \in \mathcal{P}$ s.t. $a \in A_j$ for some $j \neq i$ **do**
4: **send** request message $M_{req} = \langle s, a, \delta + 1 \rangle$ to α_j,
asynchronously process the reply by receiveReplyCallback(h)
5: **end for**
6: **end if**
7: receiveReplyEnqueueCallback(p, h)

Require: Heuristic estimate h, p the proposition from context

receiveReplyCallback(h):

8: $h_{sum} \leftarrow h_{sum} + h$
9: **if** all replies received **then**
10: receiveReplyEnqueueCallback(p, h_{sum})
11: **end if**

are no pending requests, the final heuristic estimate is computed depending on the actual heuristic (sum or max) and is returned via a callback to the search, so that the evaluated state can be expanded.

Otherwise, if $\delta \leq \delta_{max}$, Algorithm 4 iterates through all actions $a' \in \mathcal{P}$ and sends requests to their respective owners. The heuristic estimate received in each reply is added to the shared h_{sum}. When all replies are received (the replies undergo the same procedure if there are any other public actions involved) and all costs are added together, again the receiveReplyEnqueueCallback(p, h) from Algorithm 3 is called with $h = h_{sum}$.

The **processReply** procedure stands for the distributed recursion, but the deeper recursive call is not called by the agent managing the request, but the parameters of the recursion (the set of actions $\mathcal{P}$ which should be resolved next) are sent back to the initiator agent. This is rather an optimization to avoid having multiple heuristic evaluation contexts needed to handle multiple interwoven request/reply traces. Each context would need to have a separate instance of the *exploration queue* data structure, which would present major inefficiency. Instead, the initiator agent is responsible for tracking the recursion and the replying agent only processes one reply at a time, locally, without sending any requests. Therefore, each agent needs to have only two instances of the *exploration queue*, one used to compute their own

heuristic estimates (and possibly send requests and await replies), and one used to compute the local estimates for the replies.

4.2.2 Recursive Distribution of the Fast-Forward Heuristic

The Fast-Forward h_{FF} heuristic is not directly based on the estimation of the cost of actions in the relaxed problem, but on finding a plan solving the relaxed problem (a *relaxed plan* or RP). The heuristic is not typically described using recursive equations, but the implementation based on *relaxed exploration* can be easily reused. The difference is that the evaluation does not end when the exploration is finished (all goal propositions have been reached), but continues with the relaxed plan extraction. The extraction of RP starts with the goal propositions and traverses the data structure towards the initial state while marking the relaxed plan.

Since the algorithm is implementation-wise very similar to the h_{add} and $h_{\max}$ heuristics, one of the possible approaches to distribution of h_{FF} is to perform the distributed *relaxed exploration* exactly as in h_{add} and simply add RP extraction routine at the end of the heuristic evaluation (as part of the total cost computation). Another approach was conceptually introduced in Štolba and Komenda [2013] as lazy multi-agent FF heuristic h_{lazyFF}, which we have adopted and compared with the previously described approach and both additive and max heuristics.

The lazy FF algorithm published in [Štolba and Komenda, 2014] starts by building a local *exploration queue*. When all goal propositions are reached, a relaxed plan π^+ is extracted. For all actions $a \in \pi^+$, which are projections $a^{\triangleright} \in A_i^{\triangleright+}$ s.t. $a \in A_j$, a request message $M_{req} = \langle s, a, \delta \rangle$ is sent to the agent α_j . When the agent α_j receives the request, it constructs a local relaxed plan from the state s (by local *relaxed exploration* and local RP extraction without sending any requests), satisfying the preconditions (both public and private) of the action a. Then, the agent α_j sends a reply $M_{re} = \langle h, \mathcal{P}, \delta \rangle$, where h is the length of the relaxed plan and $\mathcal{P}$ is a set of projected actions contained in the plan. When the reply is received by agent α_j, the algorithm iterates through all actions $a' \in \mathcal{P}$ and sends requests to their respective owners. Each of the requests undergoes the same procedure as the original request, adding the returned heuristic estimates to the resulting h_{sum}. When all requests are processed, h_{sum} is added to the length of the local relaxed plan of agent α_j and returned via callback as the heuristic estimate of state s. This approach has several drawbacks which are discussed and improved in the next Section 4.3.

The recursion depth of the h_{lazyFF} heuristic can be limited in a similar manner as in the h_{add} and $h_{\max}$ heuristics. Whenever a request should be sent and the maximum recursion limit δ_{max} has been reached, the request is not sent and the possible relaxed sub-plan is ignored.

4.3 Privacy-Preserving Set-Additive Fast-Forward Heuristic

The general technique of computing the (classical) FF heuristic was already described in the introduction of this chapter. In this section, we focus on the distribution of the relaxed plan extraction and thus the particular technique used for the reachability analysis (performed on the projected problem) is not important. The principle based on an explicit construction of Relaxed planning Graph (RPG) is described in Section 4.1, more efficient principle based on Exploration Queue was described in Section 4.2. In this section, we assume the latter, but, as already said, the details of the reachability analysis algorithm are not relevant for the work presented in this section.

The two leading ideas of the Privacy-Preserving Set-Additive version of the distributed FF heuristic (h_{ppsaFF}) published in [Štolba and Komenda, 2017] and described in this section are the use of the lazy approach, already mentioned in Section 4.2, and the idea of the set-additive heuristic [Keyder and Geffner, 2008]. The lazy principle basically means that the parts of the heuristic respective to other agents are computed only when needed. The distributed heuristic first starts as a projected FF, computing

the reachability analysis and relaxed plan as shown above. Such relaxed plan π_i^+ computed on an i-projected relaxed problem $\Pi_i^{\triangleright+}$ may contain projected actions, which may have private preconditions. The private preconditions are not satisfied in π_i^+ as α_i is not aware of them. Let $a^{\triangleright+}$ be such a projected action and let $a^+ \in A_j^+$ for some $j \neq i$. In that case, α_i requests α_j to provide a relaxed plan that satisfies the private part of the precondition of a^+ (the public part is already satisfied in the projected RP). Here the set-additive principle comes into play. In the original lazy FF variant described in Section 4.2.2, agent α_j would report just the cost of achieving private preconditions of a, which leads to significant over-counting. In the new variant, the agent α_j provides the actual relaxed plan π_a^+, which satisfies the private precondition of a^+, that is, $\mathsf{pre}(a^+) \cap P_j^{\mathsf{priv}}$, which can be merged with the original relaxed plan $\pi_i^+ \leftarrow \pi_i^+ \cup \pi_a^+$ as both relaxed plans are represented as sets of actions. This request-reply protocol is performed for all projected actions in π_i^+, even those newly received from α_j and even for public actions of α_i itself received in π_a^+ (in which case the request can be handled by an internal call).

At this moment we have possibly violated privacy by sharing a relaxed plan π_a^+ which may contain private actions of agent α_j, even though we share only a unique identifier of that private action and no preconditions or effects. In order to treat the privacy correctly, the algorithm has to be further modified.

Instead of sending back the relaxed plan π_a^+, agent α_j builds a local relaxed reply plan $\pi_{j,i}^{\mathsf{RE}+}$ for the requesting agent α_i (we always write the index of the agent owning the variable/data structure first and the other agent it relates to second), which is updated for every requested action a^+ as $\pi_{j,i}^{\mathsf{RE}+} = \pi_{j,i}^{\mathsf{RE}+} \cup \pi_a^+$. To maintain privacy, α_j keeps the private part of the plan locally, that is $\pi_{j,i}^{\mathsf{priv}+} = \pi_{j,i}^{\mathsf{RE}+} \cap A_j^{\mathsf{priv}+}$ and sends only its public part $\pi_{j,i}^{\mathsf{pub}+} = \pi_{j,i}^{\mathsf{RE}+} \cap A_j^{\mathsf{pub}+}$ together with the length of the private part $l_{j,i} = |\pi_{j,i}^{\mathsf{priv}+}|$. The relaxed plan $\pi_{j,i}^{\mathsf{RE}+}$ is maintained by α_j throughout the whole computation of the heuristic estimate for the agent α_i and a single state s, so that each action of α_j is counted at most once. This is made easier by the fact that each agent is computing the distributed heuristic for at most one state, thus the agent has to keep track of at most $|\mathcal{A}| - 1$ relaxed plans of other agents. Meanwhile, agent α_i builds a single relaxed plan π_i^+ containing actions from $A_i^{\triangleright+}$ and a value $l_{i,j}^{\mathsf{priv}}$ for each agent $\alpha_{j \neq i}$ representing the length of the private part of the relaxed plan of agent α_j. After all projected actions in π_i^+ are processed (that is all replies received), the resulting heuristic value is computed as:

$$h_{\mathrm{ppsaFF}}(s^{\triangleright i}) = |\pi_i^+| + \sum_{j \in 1..n \wedge j \neq i} l_{i,j}^{\mathsf{priv}} \tag{4.3.1}$$

For clarity, the algorithm is transcribed into pseudo-code in Algorithm 6. The algorithm is split into three procedures, each has the agent which is performing the procedure (i.e., is written from its perspective) as its first parameter. The procedures work as follows.

The main procedure `computeDistributedFF` $(\alpha_i, s^{\triangleright i}, \langle \delta_1, ..., \delta_n \rangle, \Pi_i^{\triangleright+})$ is called by the search to evaluate a state $s^{\triangleright i}$ by agent α_i (shown as a call to h_{G}^i at line 5 in Algorithm 10). After the initialization steps, the relaxed plan π_i^+ is computed such that π_i^+ achieves the goal G in the projected relaxed problem (Equation 4.0.1), using actions from $A_i^{\triangleright+}$. Then, while there is some projected action $a^{\triangleright+}$ in π_i^+ which has not been processed yet (i.e., is not in A_{DONE}), process it by sending a request $M_{\mathsf{REQUEST}} = \langle \alpha_i, s^{\triangleright}, \delta_j, a^{\triangleright+} \rangle$ to the agent α_j, the owner of a^+ (the owner of an action is known by definition). The loop also does not terminate while there are some actions in A_{WAITING}, which means a reply has not been received for them (a request is not sent if there is no unprocessed action in π_i^+, this condition has been omitted for simplicity). When all projected actions and replies are processed, the heuristic value (Equation 4.3.1) is returned. Note that the actual implementation differs from the pseudocode in that the loop is implemented as an asynchronous event-based message processing (i.e., waiting for replies from other agents is non-blocking).

When the agent α_j receives a request from the agent α_i to evaluate private preconditions of some action $a^+ \in A_j^{\mathsf{pub}+}$, the procedure `processRequest` $(\alpha_j, M_{\mathsf{REQUEST}} = \langle \alpha_i, s^{\triangleright}, \delta_j, a^{\triangleright+} \rangle)$ is called. Agent α_j first reconstructs the state $s^{\triangleright j}$ and then computes a relaxed plan π_a^+, which solves the j-projected relaxed problem of α_j starting in $s^{\triangleright j}$ with goal being the private preconditions of a^+, formally

Algoritmus 6: Procedures for computing the Privacy-Preserving Set-Additive Lazy FF

Procedure `computeDistributedFF` $(\alpha_i, s^{\triangleright i}, \langle \delta_1, ..., \delta_n \rangle, \Pi_i^{\triangleright +} = \langle P_i, A_i^{\triangleright +}, s_I^{\triangleright i}, G \rangle)$
  $l_{i,j}^{\mathsf{priv}} \leftarrow 0$ for each $j \neq i$;
  $\pi_i^+ \leftarrow$`computeRelaxedPlan` $(s^{\triangleright i}, G, A_i^{\triangleright +})$;
  if $\pi_i^+ = \mathsf{fail}$ **then**
    return ∞;
  $A_{\mathsf{DONE}} \leftarrow \pi_i^+ \cap A_i^{\mathsf{pub}+}$; $A_{\mathsf{WAITING}} \leftarrow \emptyset$;
  while $\exists a^{\triangleright +} \in \pi_i^+ \setminus A_{\mathsf{DONE}}$ *s.t.* $a^+ \in A_j^+ \wedge j \neq i$ **or** $A_{\mathsf{WAITING}} \neq \emptyset$ **do**
    $A_{\mathsf{WAITING}} \leftarrow A_{\mathsf{WAITING}} \cup \{a^{\triangleright +}\}$;
    $A_{\mathsf{DONE}} \leftarrow A_{\mathsf{DONE}} \cup \{a^{\triangleright +}\}$;
    send $M_{\mathsf{REQUEST}} = \langle \alpha_i, s^{\triangleright}, \delta_j, a^{\triangleright +} \rangle$ **to** α_j;
  return $|\pi_i^+| + \sum_{j \in 1..n \wedge j \neq i} l_{i,j}^{\mathsf{priv}}$;
Procedure `processRequest` $(\alpha_j, M_{\mathsf{REQUEST}} = \langle \alpha_i, s^{\triangleright}, \delta_j, a^{\triangleright +} \rangle)$
  // α_j can use $a^+ \in A_j^{\mathsf{pub}+}$ instead of $a^{\triangleright +}$ for $\mathsf{lbl}(a^+) = \mathsf{lbl}(a^{\triangleright +})$
  $s^{\triangleright j} \leftarrow \mu^j(s^{\triangleright}, \delta_j).\mathsf{state}$; // reconstruct the state
  $\pi_a^+ \leftarrow$`computeRelaxedPlan` $(s^{\triangleright j}, \mathsf{pre}(a^+) \cap P_j^{\mathsf{priv}}, A_j^{\triangleright +})$;
  if M_{REQUEST} *is a first request for* $s^{\triangleright j}$ **then**
    $\pi_{j,i}^{\mathsf{RE}+} \leftarrow \emptyset$;
  $\pi_{j,i}^{\mathsf{RE}+} \leftarrow \pi_{j,i}^{\mathsf{RE}+} \cup \pi_a^+$;
  $l_{j,i} \leftarrow |\pi_{j,i}^{\mathsf{RE}+} \cap A_j^{\mathsf{priv}+}|$;
  // all except for the private part of π_a^+ is sent
  $\pi_{j,i}^{\mathsf{pub}+} \leftarrow \pi_a^+ \cap (A_j^{\triangleright +} \setminus A_j^{\mathsf{priv}+})$;
  send $M_{\mathsf{REPLY}} = \left\langle \pi_{j,i}^{\mathsf{pub}+}, l_{j,i}, a^{\triangleright +} \right\rangle$ **to** α_i;
Procedure `processReply` $(\alpha_i, M_{\mathsf{REPLY}} = \left\langle \pi_{j,i}^{\mathsf{pub}+}, l_{j,i}, a^{\triangleright +} \right\rangle)$
  $\pi_i^+ \leftarrow \pi_i^+ \cup \pi_{j,i}^{\mathsf{pub}+}$;
  $l_{i,j}^{\mathsf{priv}} \leftarrow l_{j,i}$;
  $A_{\mathsf{WAITING}} \leftarrow A_{\mathsf{WAITING}} \setminus \{a^{\triangleright +}\}$;

$\mathsf{pre}(a^+) \cap P_j^{\mathsf{priv}}$. The computed relaxed plan (RP) is then used to update the reply RP $\pi_{j,i}^{\mathsf{RE}+}$, whose private length $l_{j,i} \leftarrow |\pi_{j,i}^{\mathsf{RE}+} \cap A_j^{\mathsf{priv}+}|$ is sent back to α_i together with the public part of $\pi_{j,i}^{\mathsf{RE}+}$. The public part of $\pi_{j,i}^{\mathsf{RE}+}$ consists of public actions of α_j and all projected actions of $\alpha_{k \neq j}$ including projected actions of α_i, which are in π_a^+. Note that the reply RP $\pi_{j,i}^{\mathsf{RE}+}$ is kept for each agent α_i over all requests regarding one particular state $s^{\triangleright j}$. When the agent α_j receives a request for another state s' from the agent α_i, the reply RP $\pi_{j,i}^{\mathsf{RE}+}$ is initialized to $\pi_{j,i}^{\mathsf{RE}+} = \emptyset$ first. This works thanks to the fact that each agent α_k computes the heuristic estimate for at most one state at any moment.

When a reply message $M_{\mathsf{REPLY}} = \left\langle \pi_{j,i}^{\mathsf{pub}+}, l_{j,i}, a^{\triangleright +} \right\rangle$ is received from the agent α_j for the request $M_{\mathsf{REQUEST}} = \langle \alpha_i, s^{\triangleright}, \delta_j, a^{\triangleright +} \rangle$ by agent α_i, the procedure `processReply(`$\alpha_i, M_{\mathsf{REPLY}} = \left\langle \pi_{j,i}^{\mathsf{pub}+}, l_{j,i}, a^{\triangleright +} \right\rangle$`)` is called. The relaxed plan π_i^+ is updated with the received public part and the estimate of the private part for agent α_j is replaced with the new received value. Action $a^{\triangleright +}$ is removed from A_{WAITING} as its processing has been finished.

Note that the search and heuristic estimation is all running in a single thread and all messages are managed through a message queue and the calls on the line 6 and line 21 in Algorithm 9. This means that the procedure `computeDistributedFF()` is called once (line 5 of Algorithm 10) and the loop on line 7 is in fact managed through callbacks and thus the call to the heuristic computation is asynchronous. Meanwhile, the `processRequest()` and `processReply()` procedures are called in response to the messages received on line 25 of Algorithm 11, sequentially, one at a time.

Example. (Logistics) We illustrate the process on the running example. Let us start with a situation, where the truck is computing the heuristic estimate for the initial state. First, the projected relaxed plan π_{t}^+ is computed

$$\pi_{\mathsf{t}}^+ = \{\textbf{unload-a-C}\}.$$

In the t-projected problem, the unload action of plane has no preconditions and it fulfills the goal. Next, the truck sends a request to the plane, which computes a relaxed plan from the initial state to the private precondition of **unload-a-C**, which is $\{\mathsf{package\text{-}in\text{-}a}, \mathsf{plane\text{-}at\text{-}C}\}$. The plane comes up with the following relaxed plan

$$\pi_{\mathsf{a,t}}^{\mathsf{RE}+} = \{\textbf{unload-t-B}, \textbf{load-a-B}, \mathsf{move\text{-}a\text{-}B\text{-}C}\}$$

and sends back a reply containing only the public actions and the number of private actions which is 1. The truck accordingly updates its relaxed plan to

$$\pi_{\mathsf{t}}^+ = \{\textbf{unload-t-B}, \textbf{load-a-B}, \textbf{unload-a-C}\}$$

and proceeds by sending requests for the newly added actions. The request for **load-a-B** does not have to be sent as it was received from the plane (this optimization is ignored in the algorithm for simplicity). The request for **unload-t-B** has to be sent, but as the receiver is the truck itself, it can be forwarded via a direct call. Also the private actions of the truck are directly included in the relaxed plan π_{t}^+ by the local computation. The resulting relaxed plan is

$$\pi_{\mathsf{t}}^+ = \{\mathsf{load\text{-}t\text{-}A}, \mathsf{move\text{-}t\text{-}A\text{-}B}, \textbf{unload-t-B}, \textbf{load-a-B}, \textbf{unload-a-C}\}$$

with the additional number of private actions of the plane being 1, thus the complete heuristic estimate is $h_{\mathrm{ppsaFF}}(s_I) = 5 + 1 = 6$.

This wraps up the description of the Privacy-Preserving Set-Additive distributed variant of the FF heuristic. We did not pay attention to the actual process of finding the relaxed plan, as the distribution is general so that any approach can be used and the reachability analysis is kept local (computed on the projected relaxed problem). Now, we formally show that the Privacy-Preserving Set-Additive FF heuristic always terminates.

Theorem 45. *Assuming that every sent message is eventually received, the heuristic* ppsaFF *shown in Algorithm 6 always terminates.*

Proof. Let s be the state the ppsaFF heuristic is computed for by agent α_i. If the goal is not reachable from s in $\Pi_i^{\triangleright+}$ then the computation of `computeRelaxedPlan(`s, G, $A_i^{\triangleright+}$`)` will fail and ∞ is returned. Otherwise, π_i^+ contains a finite number of (relaxed) actions. For each action $a^{\triangleright+} \in \pi_i^+ \setminus A_{\mathsf{DONE}}$ such that $a^+ \in A_j^+ \wedge j \neq i$ a request is sent to the action owner α_j. The computation of the reply, that is `computeRelaxedPlan(`$s, \mathsf{pre}(a^+) \cap P_j^{\mathsf{priv}}, A_j^{\triangleright+}$`)` always finishes, with either a finite non-empty or an empty plan π_a^+. When the reply is received, the action a is added to A_{DONE} and the public actions in π_a^+ are added to π_i^+. In this step, a finite (but possibly zero) number of actions is added to π_i^+ and the number of actions in A_{DONE} increases by 1 as a is added. As the number of actions in A is finite (and so is the number of public actions), and in each iteration, the number of actions in A_{DONE} increases, eventually, the set of actions $a^{\triangleright+} \in \pi_i^+ \setminus A_{\mathsf{DONE}}$ such that $a^+ \in A_j^+ \wedge j \neq i$ becomes empty and the computation terminates. □

4.4 Evaluation

Evaluation of the inadmissible heuristics is done using the MADLA Planner multi-agent single heuristic search (equivalent to MAFS [Nissim and Brafman, 2014]). The search algorithm is described in Section 5.1 and the implementation of the planner in Section 5.5. All experiments were performed on FX-8150 8-core processor at 3.6GHz, each run limited to 8GB of RAM and 10 minutes. Each measurement is a mean from 5 runs as the order of messages received introduces nondeterminism of the algorithm runs.

4.4.1 Comparison of Relaxation Heuristics

The first batch of experiments focused on two classical planning metrics used in the comparison of heuristic efficiency: planning time t and number of explored states e. Those metrics were supplied by a multi-agent metric of communicated bytes b among the agents during the planning process. Used planning problems stem from IPC domains modified for multi-agent planning as presented, e.g., in [Nissim and Brafman, 2012]. The problems with * in their names were either based on IPC domains, but simplified, or other state-of-the-art multi-agent benchmarks, e.g., from [Komenda and Novak, 2011]. Most of the used benchmarks are described in Section 3.6. The recursion depth was limited to three values $\delta_{max} = \{0, 1, \infty\}$ as other settings of δ_{max} showed similar results. Missing rows were not successfully planned with any of the tested heuristics.

Selected results are shown in Table 4.1. No single heuristic and δ_{max} dominates the other ones. In Rovers, the most successful in terms of time seems to be h_{max}. In Satellites, h_{lazyFF} performs well, but in other domains, it does not solve some problems at all. The Logistics is dominated by h_{FF} and in Cooperative Path-Finding[1] and Sokoban, the best are h_{FF} and h_{add}.

Figure 4.4.1 shows graphically all problems for two selected domains, loosely coupled rovers and tightly coupled cooperative path-finding. The shown results are for the h_{add} heuristic, but other heuristics demonstrate similar behavior. A number of interesting observations can be made. The execution time indicates the trade-off between heuristic computation time and quality of heuristic guidance (which is shown a bit more clearly by the number of expanded states). The projected heuristic ($\delta_{max} = 0$) is significantly faster to compute as it does not have to communicate with other agents but in the case of the rovers domain, even such fast computation does not compensate for the worse heuristic guidance. The cooperative path-finding shows slightly different results as the heuristic guidance of the projected heuristic is not much worse than other variants. The worst case seems to be setting $\delta_{max} = 1$ which do

[1] Cooperative path-finding (also known as deconfliction) is a simple domain where robots need to navigate through a grid-world while avoiding each other. The position of the robots is private to them, but the fact that a grid position is occupied is public.

		time (seconds)				expanded states (thousands)				communicated data (MB)			
prob. ($\|\mathcal{A}\|$)	δ_{max}	h_{FF}	h_{add}	h_{max}	h_{lazyFF}	h_{FF}	h_{add}	h_{max}	h_{lazyFF}	h_{FF}	h_{add}	h_{max}	h_{lazyFF}
Rov12 (4)	0	70.9	40.7	–	57	4617.3	3939.5	–	4583.5	35.7	31.2	–	35.6
Rov12 (4)	1	1.2	1.3	**1.1**	–	0.8	**0.3**	**0.3**	–	0.7	**0.2**	0.3	–
Rov12 (4)	∞	**1.1**	1.2	1.2	–	0.4	0.7	**0.3**	–	0.4	0.6	0.3	–
Sat* (14)	0	–	–	–	–	–	–	–	–	–	–	–	–
Sat* (14)	1	69.2	68	69	**60.6**	9.3	**8.7**	9	9	36.9	33.4	34.8	**17.5**
Sat* (14)	∞	69	68.5	68.5	61.3	9.3	9.3	9	9.7	36.7	37	35.7	18.4
Log* (6)	0	**0.7**	0.7	0.7	**0.7**	6.8	5.7	7	7.2	0	0	0	0
Log* (6)	1	1.2	1.6	1.2	1.6	0.5	1.3	0.7	5.4	0.3	0.6	0.3	0.6
Log* (6)	∞	1.1	1.3	1.2	1.4	**0.4**	0.8	0.5	0.6	**0.2**	0.5	0.3	0.8
CP* (7)	0	2.2	2.2	6.3	2.3	183.2	223.1	1252.5	205	2.4	3.2	15.7	2.7
CP* (7)	1	**1.9**	18.3	35.5	50.4	**162.1**	248.1	451.4	371.2	**2.2**	150.8	274.5	738.6
CP* (7)	∞	2	2.1	6.3	160.9	188.9	225.2	1255.6	249.5	2.5	3.2	15.5	249.5
Sok* (2)	0	1.6	1.5	1.7	1.6	8.5	7.7	11.7	8.8	**0.5**	**0.5**	0.7	0.6
Sok* (2)	1	1.5	17.1	3.9	4.3	7.6	22	**4.3**	12.9	**0.5**	66.8	12.1	24.1
Sok* (2)	∞	1.5	**1.4**	1.6	–	8.3	7.8	11.7	–	**0.5**	**0.5**	0.7	–

Table 4.1: Comparison of relaxation heuristics for selected problems and metrics.
Abbreviations: Rov = rovers, Sat = satellites, Log = logistics, CP = cooperative path-finding, Sok = sokoban.

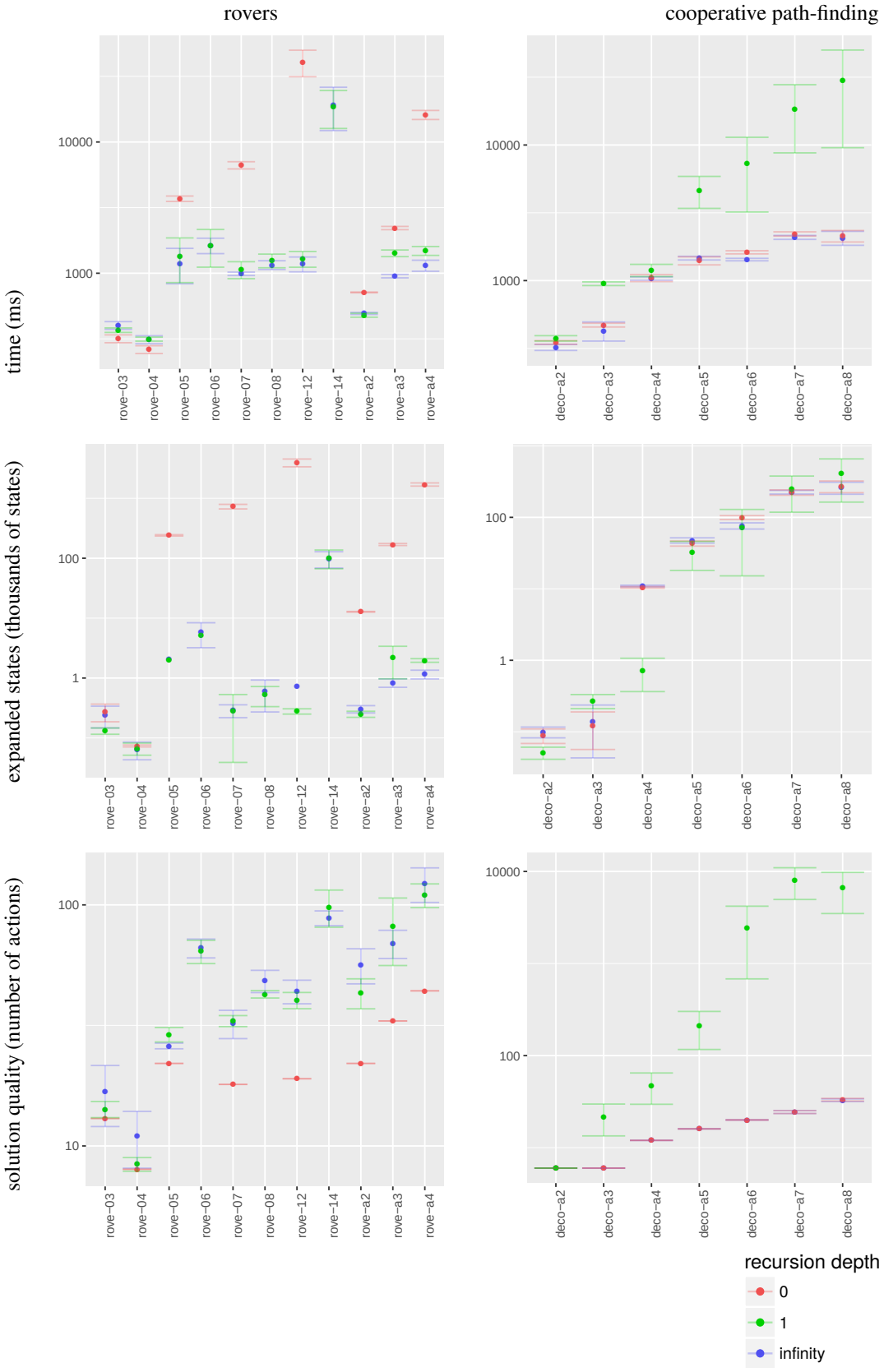


Figure 4.4.1: Comparison of relaxation heuristics for the h_{add} heuristic in three recursion levels ($\delta_{max} \in \{0, 1, -\infty\}$). The figures show all problems of the two selected domains for selected metrics. The planning time metrics is in seconds, the explored states in thousands of states and the communicated information in bytes. Abbreviations of domains:
rove = rovers
deco = cooperative path-finding (deconfliction)

δ_{max}	0	1	2	4	∞
h_{FF}	35 /7	38/15.4	38 /15	38 /14.4	38 /15
h_{add}	35 /7	38 /14	38 /14	38 /14	38 /14
h_{max}	35 /3.2	38 /14	38 /14	38 /14	38 /14
h_{lazyFF}	35.2 /6.8	38 /8	36.2 /8	36.5 /8	36.8 /8

Table 4.2: Coverage for various heuristics and recursions depth δ_{max}. The results are in the form of multi-agent domains / IPC domains.

not significantly improve on the heuristic guidance but incurs a substantial amount of communication as in cooperative path-finding, all actions are public and thus requests need to be sent.

The quality of the solutions shows a slightly different trend. As in greedy best-first search, overestimating the true cost typically leads to worse solutions than underestimating (which is closer to breadth-first search), the search guided by the projected heuristic provides shorter and thus better plans.

In the next experiment, we have evaluated the coverage of all the described heuristics (h_{add}, h_{max}, h_{FF}, and h_{lazyFF}) with the maximum recursion depth δ_{max} set to 0, 1, 2, 4 and ∞. The coverage has been evaluated over two sets of benchmarks. The first set consists of 40 specifically multi-agent problems, which are typically not that combinatorially hard, but contain more agents (taken from Štolba and Komenda [2013]). The second set consists of 21 problems converted directly from IPC benchmarks (as in [Nissim and Brafman, 2012]), which are typically much combinatorially harder, but with fewer agents. The results are summarized in Table 4.2.

The results show a clear dominance of h_{FF}, but interestingly the other distribution approach of the Fast-Forward heuristic, h_{lazyFF}, is on the other side of the spectrum. This is most probably because one of the biggest strengths of the FF heuristic, compared to other delete relaxation heuristics used here, is that it does not suffer from *over-counting* (one action is included in the estimate several times) thanks to the explicit relaxed plan extraction. In the h_{lazyFF}, we partially lose this advantage, because when sending a reply, only the length of the plan is sent. Therefore, a single action can be included several times in multiple replies from a single agent, or even multiple agents.

Another observation is that the setting of $\delta_{max} = 0$ is dominated by other values. This may be due to the choice of the domains, the effect of various δ_{max} settings is thoroughly analyzed in the next set of experiments. Also, various settings of δ_{max} for $\delta_{max} > 0$ affect the coverage only marginally.

4.4.2 Effect of the Recursion Depth

In the following set of experiments, we have evaluated the effect of changing the *maximal recursion depth* δ_{max} on the speed and communication requirements of the planning process. The data set was measured on four selected domains with varied couplings (rovers, satellites, cooperative path-finding, and logistics), each represented by a single problem. The *maximal recursion depth* ranged from 0 (a *projected heuristic*) to 9, for comparison, the results were normalized against the result of run with $\delta_{max} = \infty$.

By coupling, we understand the concept formalized in [Brafman and Domshlak, 2008], which can be rephrased as "the more interactions must take place among the agents in order to solve the problem, the more coupled the problem is"—at one extreme there are problems, where all actions interact with other agents (containing only public actions) meaning *full coupling*. In problems of the other extreme, the agents can solve their individual problems without any interaction. Because of our decision to treat all goals as public, we cannot achieve *full decoupling*—at least goal-achieving actions are public and thus causing some level of coupling. The experimental domains were chosen such that rovers and satellites are loosely coupled. In satellites, only the assumption that all goal-achieving actions are public introduces some coupling, in rovers, there are also interacting preconditions among the goal-achieving actions. Logistics is moderately coupled (private movement of agents and public handling of packages)

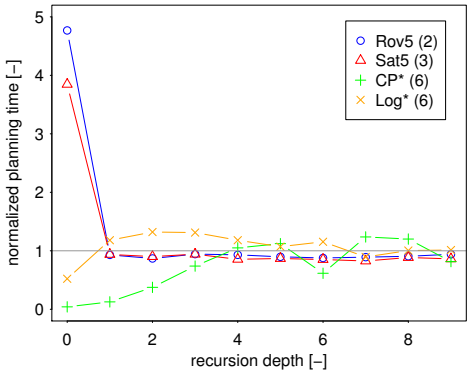


Figure 4.4.2: Planning time normalized to result for of $\delta_{max} = \infty$ for h_{lazyFF} heuristics.

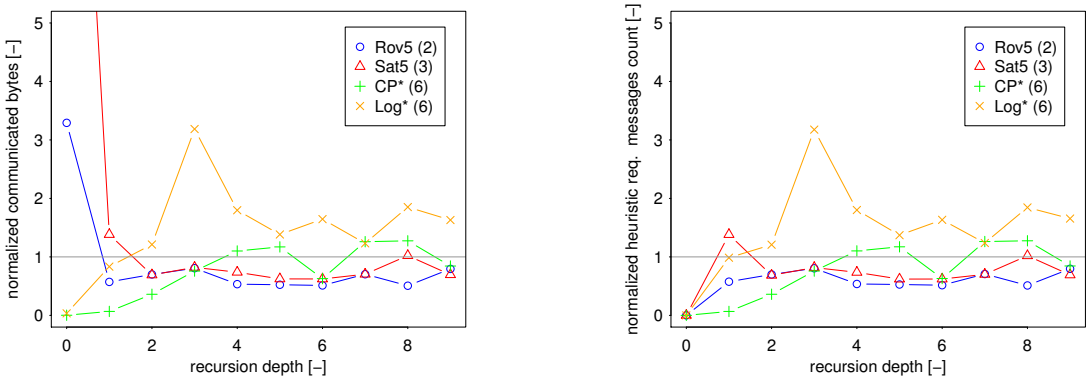


Figure 4.4.3: Communicated bytes and heuristic message requests normalized to $\delta_{max} = \infty$ for h_{lazyFF} heuristics.

and cooperative path-finding is fully coupled.

The experimental results for the h_{lazyFF} heuristic are plotted in Figures 4.4.2 and 4.4.3. In the fully coupled cooperative path-finding, the results are best for $\delta_{max} = 0$ and are converging to the results for $\delta_{max} = \infty$ as δ_{max} grows. This is because in a fully coupled problem, all actions are public and in cooperative path-finding, all their preconditions and effects are also public (which does not have to be always the case). Therefore each agent has complete information about the problem in form of the action projections ($a^{\triangleright} = a$ for all actions and agents) and the *projected heuristic* gives a perfect estimate (the same as would global heuristic give). For $\delta_{max} > 0$, requests are sent for every projected action, causing more communication and computation without bringing any improvement to the heuristic estimate.

Completely different picture give the results for the loosely coupled problems. The results are significantly worse for $\delta_{max} = 0$, from $\delta_{max} = 1$ they are practically equal to $\delta_{max} = \infty$. The solution of those problems typically consists of long private parts finished by a single public action (the goal achieving action). When estimated by a *projected heuristic*, the private parts of other agents get ignored and the estimates are thus much less informative. Even the fact, that when a state is expanded by a public action, it is sent with the original agent's heuristic estimate, does not help, because estimation of states expanded further from such state ignore the information again. But even $\delta_{max} = 1$ is enough to resolve this issue.

Lastly, in the logistics problem, the $\delta_{max} = 0$ estimates are rather good (but not as good as in the

cooperative path-finding) and with growing δ_{max}, the results converge towards $\delta_{max} = \infty$, but for $0 < \delta_{max} < \infty$ the results are slightly worse. This may suggest that as the coupling is moderate, it is best either to fully exploit the coupled part of the problem and use *projected heuristics* or to rely on the decoupled part of the problem and employ the full recursion approach, depending on the exact balance.

The results for communication are in Figure 4.4.3. The left chart compares the total bytes communicated and shows the same tendencies as the planning time in Figure 4.4.2. In fact, limiting the interactions may lead to increased communication. The right table shows the data for heuristic requests, there we see the expected result for $\delta_{max} = 0$, where no requests are sent, otherwise, the tendencies are surprisingly similar. This indicates that the communication complexity is dominated by the search communication complexity (the longer the search takes, the more messages are passed).

Presented results suggest, that for tightly coupled problems, sharing of the information is not only less important, because the agents have most of the information in their problem projections, but may even lower the effectiveness by redundant communication. For loosely coupled problems, the communication is vital, even if the communication is very limited. For moderately coupled problems, both extremes are equally good. In general, it is hard to determine, which approach will yield the best results, but it is sensible to choose from either no communication $\delta_{max} = 0$, full communication $\delta_{max} = \infty$, or even communication limited to very low recursion depth limits, i.e., $\delta_{max} = 1$. If we can expect some properties of the problems at hand, we can suggest preferred approach much easier—if we are *not* expecting loosely coupled problems, $\delta_{max} = 0$ is the best choice, for *no* tightly coupled problems $\delta_{max} = \infty$ and for *no* moderately coupled problems, $\delta_{max} = 1$ seems to be the best choices.

The results in the presented figures are for h_{lazyFF} mainly because they are the most illustrative, other heuristics follow the same patterns as described here.

4.4.3 Comparison of the Projected and Privacy-Preserving Set-Additive FF

A comparison of the multi-agent single-heuristic searches with projected FF and distributed Privacy-Preserving Set-Additive FF is presented in Figure 4.4.4. The top graph shows the heuristic values for the initial state of all problems for which the value was computed. It is clear that for most of the domains, as the complexity of the problem grows, also the difference between the distributed and projected heuristic grows (note this does not say anything about the heuristic quality). Bottom is the number of expanded states.

Together the two plots show some interesting properties. First, the elevators08 domain is an example of a domain where the distributed heuristic gives much larger heuristic estimates, which also seems to be significantly more informed, as suggested by the number of expanded states. As the heuristic difference grows, also the difference of the number of expanded states grows in favor of the distributed heuristic. Similar behavior, only not as prominent, can be observed in the blocksworld domain. The depot domain paints a completely different picture, where the distributed heuristic also gives significantly larger estimates, but as shown in the plot of the expanded states, the heuristic guidance degrades and for larger problems, the projected heuristic is better informed for the search. The driverlog domain also fits into this category, where the larger distributed heuristic estimates do not necessarily lead the search better. On the other hand, in the woodworking08 domain, we can observe that, although the heuristic estimates are pretty much the same for both heuristics, the number of states expanded by the projected heuristic grows in comparison with the distributed privacy-preserving set-additive FF, which suggests that even slight differences in the heuristic may have a significant impact on the heuristic quality and its ability to lead the search.

Table 4.3 shows the coverage of the MAFS search (implemented in the MADLA planner) using either the projected heuristic, the privacy-preserving set-additive FF heuristic described in Section 4.3 and a version sending directly IDs of the private actions. The numbers are averages over 5 runs as the distributed search brings non-determinism to the planning process. The results show that the treatment of privacy does not deteriorate the effectivity with an exception of the openstacks domain, quite the contrary, with the privacy-preserving heuristic, the MAFS search solves overall 5 more problems.

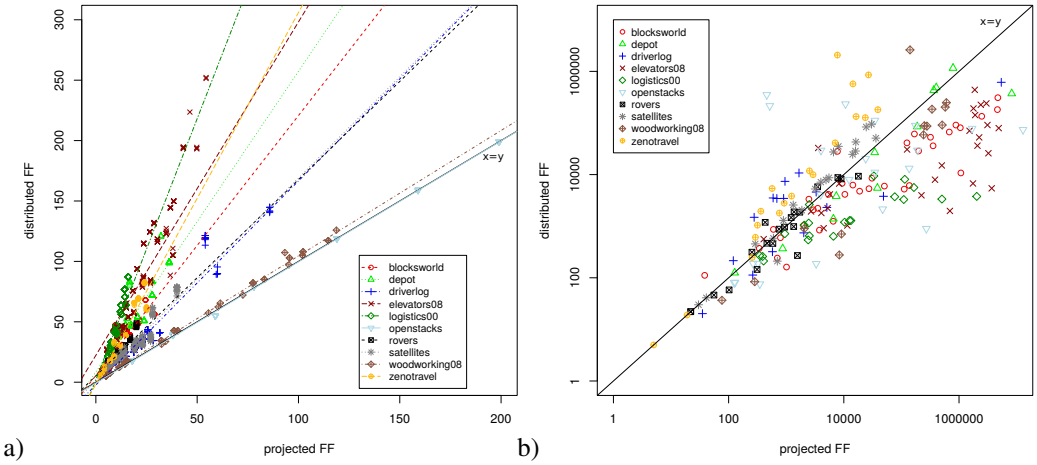


Figure 4.4.4: Heuristic values for initial states (a) and the number of expanded states (b).

domain	$\|\mathcal{A}\|$	projFF	salFF	ppsaFF
blocksworld (35)	4	32.9	32.8	**35**
depot (20)	5-12	10.3	9.2	**10.5**
driverlog (20)	2-8	**17.2**	14	14
elevators08 (30)	4-5	17.5	28.1	**29**
logistics00 (20)	3-7	**20**	20	**20**
openstacks (30)	2	13.6	**17.1**	14.9
rovers (20)	1-8	**20**	19.9	**20**
satellites (20)	1-5	**20**	**20**	**20**
woodworking08 (30)	7	**8.8**	5	4.8
zenotravel (20)	1-5	**19**	14.1	16.9
total (245)		179.3	180.2	**185.1**

Table 4.3: Average coverage of the projected (projFF), distributed using the set-additive principle (salFF), and privacy-preserving distributed (ppsaFF) heuristics. The number of problems in a domain are in the brackets, $|\mathcal{A}|$ denotes the number (interval) of agents in the problems. The best results are emphasized.

Domain	problems	DTG-based	RPG-based
blocksworld	35	35	35
depot	20	11	**16**
driverlog	20	**20**	16
elevators08	30	29	**30**
logistics00	20	19	**20**
ma-blocks	24	13	**15**
openstacks	30	21	**30**
rovers	18	18	18
rovers-large	20	20	20
satellites	18	18	18
satellites-hc	15	**13**	8
sokoban	10	9	**10**
woodworking08	30	8	**22**
zenotravel	17	**17**	15
Total	307	251	**273**

Table 4.4: Coverage of the DTG-based and RPG-based FF heuristics in the distributed variant, both ignoring costs of actions. We use an extended set of benchmarks, where ma-blocks is a version of blocksworld where not all agents can reach all positions at the table, and the rovers-large and satellites-hc domains consist of larger problem instances.

4.4.4 Comparison of RPG-based and DTG-based Distributed FF

In this section, we compare two approaches to the distribution of the FF heuristic. The first approach is based on the original method of FF computation based on Relaxed Planning Graphs (RPGs) and their effective implementation. This method was evaluated in the literature using multi-agent greedy best-first state-space search in Štolba and Komenda [2013, 2014]. The second approach replaces the Relaxed Planning Graphs with Domain Transition Graphs (DTGs) Helmert [2006] in order to reduce the communication among agents. The second method was evaluated in the literature using a multi-agent forward-chaining plan-space search in the FMAP planner Torreño et al. [2014]. Due to the very different planning paradigms, the two methods of distributed FF computation were never directly compared. To bridge this gap, we have re-implemented the DTG-based heuristic in a multi-agent greedy best-first state-space search in order to evaluate it and compare it with the RPG-based heuristic.

Both approaches were implemented in the MAPlan planner and the evaluation was performed using a standard MAFS search. The experimental comparison of the described heuristics was performed on a set of benchmarks commonly used in the MA planning literature, derived from the classical IPC benchmarks and described in Section 3.6. The rovers-large and satellites-hc are larger instances of the described domains. Each run (per problem) of the planner was limited to 30 min. and 8GB of memory (total for all agents) on a 16 core machine.

We measured the performance of the distributed versions of the heuristics using a distributed greedy best-first search. The results for coverage are shown in Table 4.4. The results confirms the hypothesis that the bad results of the RPG-based heuristic in the elevators08 and openstacks domains were caused by the use of action costs instead of simple plan length. In the case of unit costs, the RPG-based heuristic performs significantly better in terms of coverage.

We compare not only the coverage but also the search speed using the distributed variants of the heuristics (Figure 4.4.5). Multiple patterns can be observed in various domains. In the blocksworld domain, neither of the heuristics dominate, although, in one of the problems, the RPG-based heuristic takes significantly longer time to find the solution. In the logistics00 domain, the RPG-based heuristic finds the solution faster on all problems and scales better, but the worse performance of the DTG-based

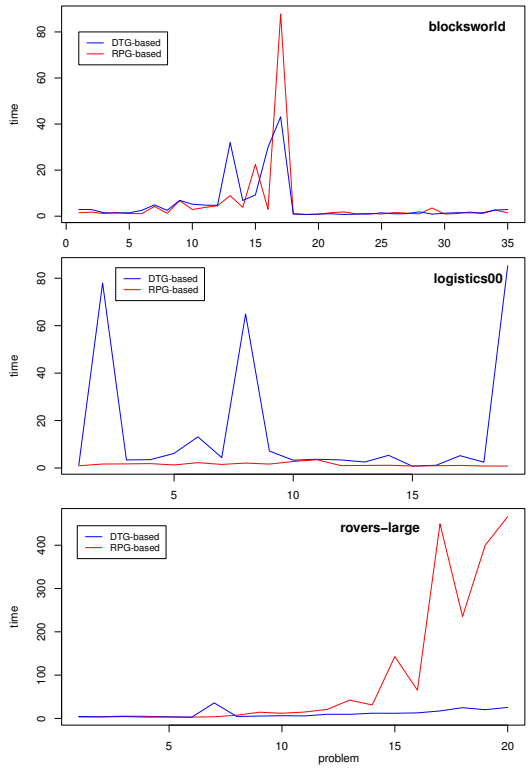


Figure 4.4.5: Comparison of solution time (s) on each problem solved by DTG-based and RPG-based FF heuristics.

heuristic does not have an effect on the coverage. In the contrary, the RPG-based heuristic scales significantly worse in the rovers-large domain, where the effect of scaling is not apparent in the coverage, but the trend suggests that unlike the DTG-based heuristic, the RPG-based heuristic would not be able to solve even larger problems.

The results of this evaluation led to the configuration of the MAPlan planner used in the CoDMAP competition, see Appendix A for details.

4.5 Summary

In this Chapter we have addressed the **(Objective 1)** of computing heuristics distributedly for the case of inadmissible heuristics. In particular, we have focused on the class of relaxation heuristics with the most work dedicated to the FF heuristic. We have provided a distributed FF heuristic provably equivalent to the centralized solution (Section 4.1), a more efficient approach to the distribution of relaxation heuristics in general (Section 4.2), and an efficient privacy-preserving variant of the FF heuristic (Section 4.3). We have thoroughly evaluated the proposed solutions. We have compared the distributed relaxation heuristic against each other and with their projected counterparts. The results of the comparison can be roughly summed up by stating that the performance of the search guided by the distributed heuristic performs better for some benchmark domains (e.g. elevators) and worse for other domains (e.g. woodworking). This leads to the formulation of **(Objective 2)**: How can we combine the projected and distributed heuristics in order to combine their benefits?

In this chapter, we have also presented our first approach to tackling the **(Objective 2)**. We have introduced a general scheme of defining relaxation heuristic variants which are in between the distributed and projected heuristics. The scheme is based on limiting the depth of recursion of the distributed computation, thus making the heuristic values less precise but also less communication intensive to compute. We have evaluated the approach and shown that in most cases any recursion depth higher than zero (which is equivalent to the projected heuristic) is practically equivalent to the distributed heuristic and thus this approach does not provide the deserved balance of projected and distributed heuristics.

Chapter 5

Combining Distributed and Local Heuristics in a Heuristic Search

In this Chapter we present our results regarding **(Objective 2)** of this thesis, which is answering the following question:

> "How to combine local and distributed heuristics?"

We have investigated multiple approaches, some with partial success such as the recursion depth limitation described in Section 4.4.2. In this chapter, we present our most successful approach, which is a variant of distributed multi-heuristic search tailored to handle local and distributed heuristics. To introduce the topic, we first describe the preliminaries, which are heuristic search, multi-agent heuristic search and multi-heuristic search (Section 5.1). Next, we describe our contribution, which is the Multi-Agent Distributed Lazily Asynchronous (MADLA) Search (Section 5.2). finally, we prove soundness and completeness of the MADLA Search (Section 5.4). This chapter is based on our work in [Štolba and Komenda, 2017].

5.1 Heuristic Search and its Variants

Forward-chaining state-space heuristic search is a well-established technique in classical planning and was already briefly outlined in the introduction. Here we provide a more formal description. We base the description on the STRIPS and MA-STRIPS formalisms (see Section 3.1), but the algorithms can be equally well formulated in the MPT and MA-MPT variant.

State-space search is a search in the space of states, that is, $s \subseteq P$ using the actions from A as the search operators (this contrasts for example with a plan-space search approach where the search nodes are partial plans and the search operators are modifications of the partial plans). Forward-chaining in this context means, that the search starts in the initial state s_I and progresses forward by applying the actions, an alternative is backward-search which starts from the goal condition and uses regression of the actions to reach new search states. We use heuristic search to refer to forward-chaining state-space heuristic search unless noted otherwise. In the next sections, we provide descriptions of three established variants of heuristic search relevant to our approach described in Section 5.2.

5.1.1 Heuristic Search

The Algorithm 7 shows a generic outline of a Best-First Heuristic Search (BestFS) for a STRIPS planning problem Π. The algorithm starts by initializing the open list to contain the initial search node. Each search node u consists of the state $u.\texttt{state} \subseteq P$, its associated distance from the initial state $u.g$ and

Algoritmus 7: Best-First Search on a STRIPS problem.

```
1  Algorithm BestFirstSearch(Π)
2      O ← {u_I}; C ← ∅; u_I.g ← 0; u_I.h ← h_0;
3      while true do
4          u ← arg min_{u∈O} f(u.h, u.g) // extract the best state from the open list O according to f
5          if u ∈ C then
6              continue with next iteration
7          C ← C ∪ {u} // close search node
8          // if solution found:
9          if s_⋆ ⊆ u.state then
10             reconstructPlan(u) // reconstruct and return plan
11         // expand successors of state:
12         for all a ∈ A_i s.t. pre(a) ⊆ u.state do
13             u' ← u ∘ a
14             u'.g ← u.g + cost(a)
15             u.h ← h(u.state) // compute the heuristic
16             if u' ∉ C then
17                 O ← O ∪ {u'} // add expanded states to the open list O^i
```

heuristic value $u.h$ and also a reference to its parent state (later used to reconstruct the plan) and the action $a \in A$ which was used to generate the search node.

The search loop starts by extracting the best node u from the open list O, which is ordered according to a function $f(u.h, u.g)$. The exact computation of f determines the type of search resulting from the general scheme:

$f(u.h, u.g) = u.h + u.g$ results in the A* search [Hart et al., 1968]. Typically, the requirement of admissibility is placed on the heuristic function h.

$f(u.h, u.g) = u.g$ results in the Dijkstra's algorithm [Dijkstra, 1959] or breadth-first search for unit-cost actions.

$f(u.h, u.g) = u.h$ results in a greedy best-first search (GBFS) algorithm, often used for sub-optimal planning.

Next, the extracted node is checked, whether it was already visited and closed. If yes, the node is skipped (more complex handling might be necessary in A* search if the heuristic is not consistent in order to obtain an optimal solution), if not, the node is added in the closed list C and thus is closed. Another check is performed in order to determine, whether the node represents a solution state. If yes, the solution plan is reconstructed by following the parent states in a backward manner.

The last step of the BestFS algorithm is to expand the search node, create its successors and add them to the open list O. in the typical variant, the heuristic value is computed for each new node. In planning, the heuristic computation is often very time consuming, thus a variant called deferred heuristic evaluation is often used, e.g., in the LAMA Planner [Richter and Westphal, 2010]. In BestFS with deferred heuristic evaluation, the newly expanded nodes are assigned the heuristic value of their predecessor and the actual heuristic value is computed only when a node is extracted from the open list. This saves a lot of heuristic computations typically without significantly deteriorating the search guidance.

5.1.2 Multi-Heuristic Search

In classical planning, multi-heuristic search was pioneered by the Fast Downward planning system [Helmert, 2006] as a way to combine different heuristic estimators without the need to combine the heuristic values, and was also one of the main mechanisms behind the success of the LAMA planner [Richter and Westphal, 2010].

The principle of multi-heuristic search is simple. Instead of a single open list O, multi-heuristic search uses a set of open lists $\{O_1, ..., O_m\}$, one for each used heuristic $h_1, ..., h_m$. In each search step, a state is extracted from one open list according to an open list selection function. Then the state is evaluated by each heuristic and its successors placed in the respective open list (if using the deferred evaluation scheme). This means that if a state s is evaluated by a heuristic h_k, its successors are placed in O_k.

The choice of an appropriate open list selection function for classical planning was thoroughly examined in [Röger and Helmert, 2010] with a conclusion that the simple alternation mechanism, where the open lists are chosen in turns, appears to be the best one (this mechanism was used in both FD and LAMA planners).

The idea of multi-heuristic search has been applied to MAP in [Maliah et al., 2016a] by extending the MAFS scheme with multiple heuristics (and thus multiple open lists) for each agent exactly the same way as was done in LAMA, and in [Torreño et al., 2015, Torreno et al., 2015] a very similar principle was applied to the plan-space search of FMAP. If we do not use such search scheme with significantly different heuristics, but with a projected and distributed variant of the same heuristic, we encounter several limitations of this simple approach. Nevertheless, we propose this simple approach as a baseline and present a significantly better one in the following section.

5.1.3 Multi-Agent Heuristic Search

The main principle of multi-agent heuristic search (Algorithm 8) is that all agents explore their portions of the search space asynchronously and in parallel, each agent using only its actions from A_i. In order to manage the coordination, states expanded using a public action are sent to all other agents (line 20). Thus the other agents are informed about the new reached state and can expand it. A simplified principle is illustrated in Figure 5.1.1. The figure is simplified in that the states are expanded in synchronous steps, whereas in reality, the agents proceed asynchronously.

In more detail, in *multi-agent single-heuristic search* with deferred heuristic evaluation, each agent α_i has its own separate open list O^i, closed list C^i and a heuristic function h^i. The search begins with $O^i = \{s_I^{\triangleright i}\}$ and $C^i = \emptyset$ for each agent. In parallel, each agent α_i extracts a state $s_{min}^{\triangleright i} = \arg\min_{u \in O^i} f(u.h, u.g)$ from O^i, adds $s_{min}^{\triangleright i}$ into C^i, computes a new heuristic value $h^i(s_{min}^{\triangleright i})$ and expands $s_{min}^{\triangleright i}$. All $s'^{\triangleright i} \in S$, where S is the set of new expanded states

$$S = \{s'^{\triangleright i} | s'^{\triangleright i} = s^{\triangleright i} \circ a, a \in A_i \text{ s.t. } \mathsf{pre}(a) \subseteq s^{\triangleright i}\} \setminus C^i$$

are added into the open list O^i (and communicated to other agents if the expanding action was public, as described later). The search terminates if any agent α_i finds $s^{\triangleright i}$ s.t. $G \subseteq s^{\triangleright i}$, or if all open lists are empty and no communication is waiting to be processed. Recall that the goal G is public and thus it is the same for all agents.

As each agent uses only its own actions for the state expansions, it is necessary to communicate reached states to other agents in order to ensure completeness. In MAFS, the states are communicated via message broadcasts, as exemplified later. If a state $s^{\triangleright i}$ is expanded by an agent α_i using a public action $a \in A_i^{\mathsf{pub}}$, the state is sent to all other agents $\alpha_{j \neq i}$ and added to their open lists O^j. In order to hide the private facts $P_i^s = P_i^{\mathsf{priv}} \cap s$ of a sent state s, the agent α_i can obfuscate the facts in P_i^s (as proposed in [Borrajo, 2013]) or replace the facts in set P_i^s with a private unique identifier (or a hash value) $\delta_i(s)$ known only to α_i (this private part of the state cannot be modified by other agents). If a modified state amended by such identifier returns by one of the later broadcasts back to the agent α_i, it

Algoritmus 8: Multi-Agent Best-First Search on a MA-STRIPS problem (somewhat simplified).

```
1  Algorithm Multi-AgentForwardSearch(α_i, Π_i)
2      O^i ← {u_I}; C^i ← ∅; u_I.g ← 0; u_I.h ← h_0;
3      while true do
4          processComm(α_i) // receive states and add them to the open list O^i
5          u ← arg min_{u∈O^i} f(u.h, u.g) // extract the best state from the open list O^i according to f
6          if u ∉ C^i then
7              C^i ← C^i ∪ {u} // close search node
8              // if solution found:
9              if s_⋆ ⊆ u.state then
10                 inform agents
11                 reconstructPlan(u) // distributedly reconstruct and return plan
12         // expand successors of state:
13         for all a ∈ A_i s.t. pre(a) ⊆ u.state do
14             u' ← u ∘ a
15             u'.g ← u.g + 1
16             u.h ← h^i(u.state)// compute heuristic
17             if u' ∉ C then
18                 O ← O ∪ {u'} // add expanded states to the open list O^i
19             if a ∈ A_i^pub then
20                 send state u.state // send state if expanded by a public action
```

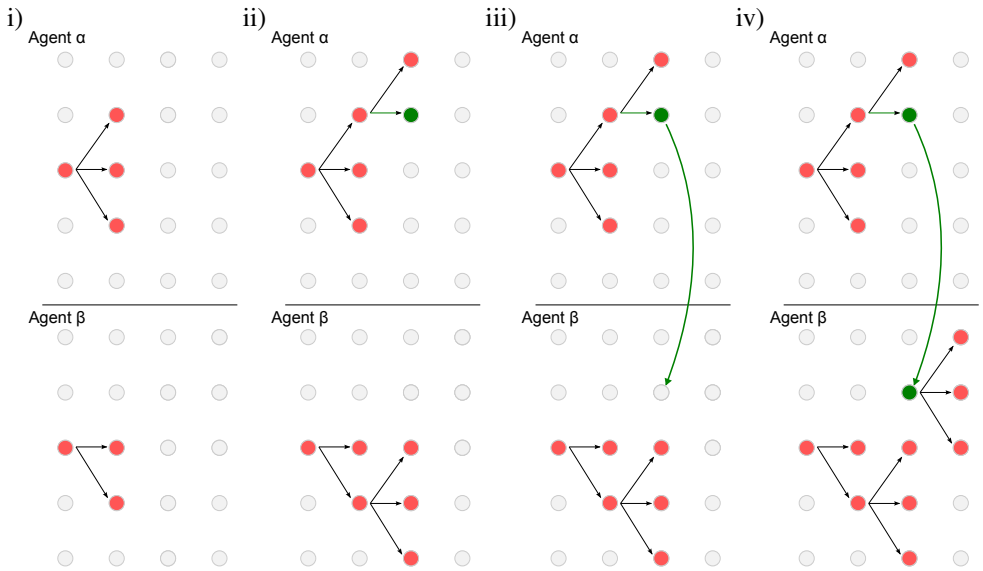


Figure 5.1.1: Simplified example of four steps in a multi-agent heuristic search for two agents α and β. In i both agents expand the initial state by private actions (black arrows), in ii agent α expands one state by a public action (green arrow and circle) and in iii it is sent to agent β. In iv) agent β expands the received state.

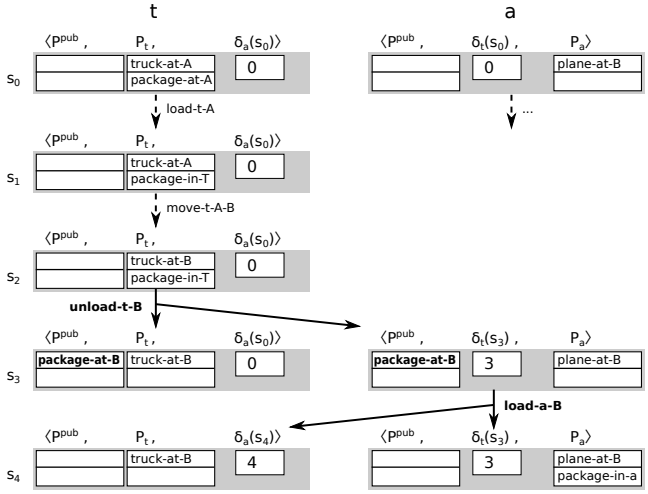


Figure 5.1.2: State communication with encrypting and restoring private values.

can use the identifier $\delta_i(s)$ and restore the private facts P_i^s back as if they were always part of the state. The initial state is treated as a special case. As already said, each agent α_i starts only with its projection of the initial state, that is $s_I^{\triangleright i}$. As all agents start in fact from the same initial state s_I (although not completely observable by any of them), the unique identifier can use some specific value (the same for all agents), such as $\delta_i(s_I) = 0$ for all i. When an agent α_j receives a state from α_i with its private part P_j^s replaced by the identifier $\delta_j(s_I) = 0$, agent α_j knows that it must restore the values in P_j^s from the initial state, that is, $s_I^{\triangleright j}$.

When a goal state s_g s.t. $G \subseteq s_g$ is found, the solution plan needs to be also reconstructed in a distributed way. This can be done by modifying the parent of a state to contain either the previous state (and the respective action) as in classical search or a reference to the agent from which the state was received. The backward reconstruction of the plan then proceeds as usual, except for when instead of a state, the parent is an agent α_j, in which case a message is sent to α_j to continue the plan extraction process.

Example. (Logistics) We illustrate the process on the running example in Figure 5.1.2. Each agent starts with its projection of the initial state $s_I = s_0 = \{\mathsf{truck\text{-}at\text{-}A}, \mathsf{plane\text{-}at\text{-}B}, \mathsf{package\text{-}at\text{-}A}\}$. The truck starts expanding $s_0^{\triangleright \mathsf{t}} = \{\mathsf{truck\text{-}at\text{-}A}, \mathsf{package\text{-}at\text{-}A}\}, \delta_{\mathsf{a}}(s_0)$ using the load-t-A and move-t-A-B actions sequentially. Further on in the process, when the truck expands the state $s_2^{\triangleright \mathsf{t}} = \{\mathsf{truck\text{-}at\text{-}B}, \mathbf{package\text{-}at\text{-}B}\}, \delta_{\mathsf{a}}(s_0)$ using the action **unload-t-B**, the resulting state $s_3 = \{\mathsf{truck\text{-}at\text{-}B}, \mathsf{plane\text{-}at\text{-}B}, \mathbf{package\text{-}at\text{-}B}\}$, seen by the truck as $s_3^{\triangleright \mathsf{t}} = \{\mathsf{truck\text{-}at\text{-}B}, \mathbf{package\text{-}at\text{-}B}\}, \delta_{\mathsf{a}}(s_0)$, is sent to the plane as $\{\mathbf{package\text{-}at\text{-}B}\}, \delta_{\mathsf{t}}(s_3), \delta_{\mathsf{a}}(s_0)$. Here $\delta_{\mathsf{t}}(s_3)$ encodes the private part of the truck in the state s_3. When received by plane, it reconstructs the state s_3 as $s_3^{\triangleright \mathsf{a}} = \{\mathsf{plane\text{-}at\text{-}B}, \mathbf{package\text{-}at\text{-}B}\}, \delta_{\mathsf{t}}(s_3)$ by using the private part of the state with the identifier $\delta_{\mathsf{a}}(s_0) = 0$ which is s_0.

5.2 The MADLA Search

The main contribution of this chapter is the MADLA Search, which is a modification of the multi-heuristic search mentioned in Section 5.1.2 towards MAFS (Section 5.1.3) but with the combination of local and distributed heuristics on mind. The search is performed in parallel by all agents, each searching using its own set of actions and communicating states expanded by public actions, in order to reach a common public goal. The plan is then extracted in a distributed manner so that each agent knows only

its respective part of the plan.

The main distinctive feature of the MADLA search is its use of two open lists per agent, where the first one is associated with a local projected heuristic h^i_{L} and the second one with a distributed global heuristic h^i_{D}. The open list selection function is tailored to handle this special case. The main high-level principles of the search are the following:

a) Evaluate a state only using a single heuristic.

b) Prefer the distributed heuristic, and only if the distributed heuristic is waiting for replies from other agents (i.e. is busy), use the projected heuristic instead.

An overview of the principle is shown in Figure 5.2.1 and it is elaborated on in the next paragraphs. In the main search loop, after processing the communication (i.e. receiving and sending queued messages), a state is extracted from an open list. Which open list is used for the extraction is determined based on whether the distributed estimator of h^i_{D} is busy and the open lists are empty or not (as shown in Figure 5.2.1). As we use deferred heuristic evaluation, the extracted state is evaluated by a heuristic (after it was checked for being in the closed list or being a solution), again depending on the state of the distributed heuristic estimator. If the extracted state was created using a public action, it is sent to all other agents. Then, the state is expanded using all applicable actions of the agent and its successors are added to the respective open list(s) as shown in Figure 5.2.1 with the heuristic estimate of the parent state (again because of the deferred heuristic evaluation used).

The implementation of the distribution in the proposed search follows the principles of MAFS, i.e., broadcasts are used to inform other agents about states reached by public actions. Additionally, information as to which open list the state should be added to is included (whether it is the local or the distributed one). The overall principles of the MADLA Search will now be presented and the algorithm itself will be described in detail later on.

The MADLA Search uses two heuristics for each agent α_i, a local projected h^i_{L} and a distributed h^i_{D}. The search uses an open list for each of the heuristics $O^i_{\mathsf{L}}, O^i_{\mathsf{D}}$ for each agent α_i. The open list selection function prioritizes expansion of states in the open list O^i_{D} respective to the distributed heuristic h^i_{D}, if the heuristic estimator of h^i_{D} is not in the process of computing a heuristic estimate (i.e. waiting for some replies from other agents).

Unlike the classical multi-heuristic search, in the MADLA Search, the extracted states are not evaluated using both heuristics. The heuristic used depends on the state of the distributed heuristic h^i_{D} estimator, represented in the algorithms as a boolean variable $\mathsf{busy}^i_{\mathsf{D}}$. The variable is set by the heuristic estimator to true if it is currently evaluating a state and false if not. If $\mathsf{busy}^i_{\mathsf{D}} = \mathsf{false}$, the state is evaluated by h^i_{D}. If $\mathsf{busy}^i_{\mathsf{D}} = \mathsf{true}$, the state is evaluated by h^i_{L}, that is the distributed heuristic is preferred if possible. This approach is most reasonable if h^i_{D} dominates h^i_{L} for most states, which is typically the case for a projected and a distributed variant of the same heuristic such as FF.

As the MADLA Search is running in a single process (except for the communication) for each agent, the local heuristic search is performed only when the distributed heuristic search is waiting for the distributed heuristic estimation to finish. This principle makes sense only if finishing an estimation of h^i_{D} takes longer than that of h^i_{L} and if computation of the h^i_{D} estimator does not block the search process (incl. h^i_{L} estimations). These two requirements often hold for distributed and projected versions of one heuristic and hold for the two variants of the FF heuristic we use as well (described in detail in Section 4.3).

Using two separate open lists has the benefit of using two heuristics independently, but if some information between the two searches could be shared[1], most importantly the heuristically best state found so far, it could boost the efficiency of the planner. The direction $O^i_{\mathsf{D}} \rightarrow O^i_{\mathsf{L}}$ is straightforward as most of the time, h^i_{D} dominates h^i_{L}. Thus, we can add all states evaluated by h^i_{D} also to O^i_{L} without ever skipping a better state evaluated by h^i_{L} with a worse state evaluated by h^i_{D}.

[1]The two searches are both run by a single agent, therefore the question of privacy is irrelevant here.

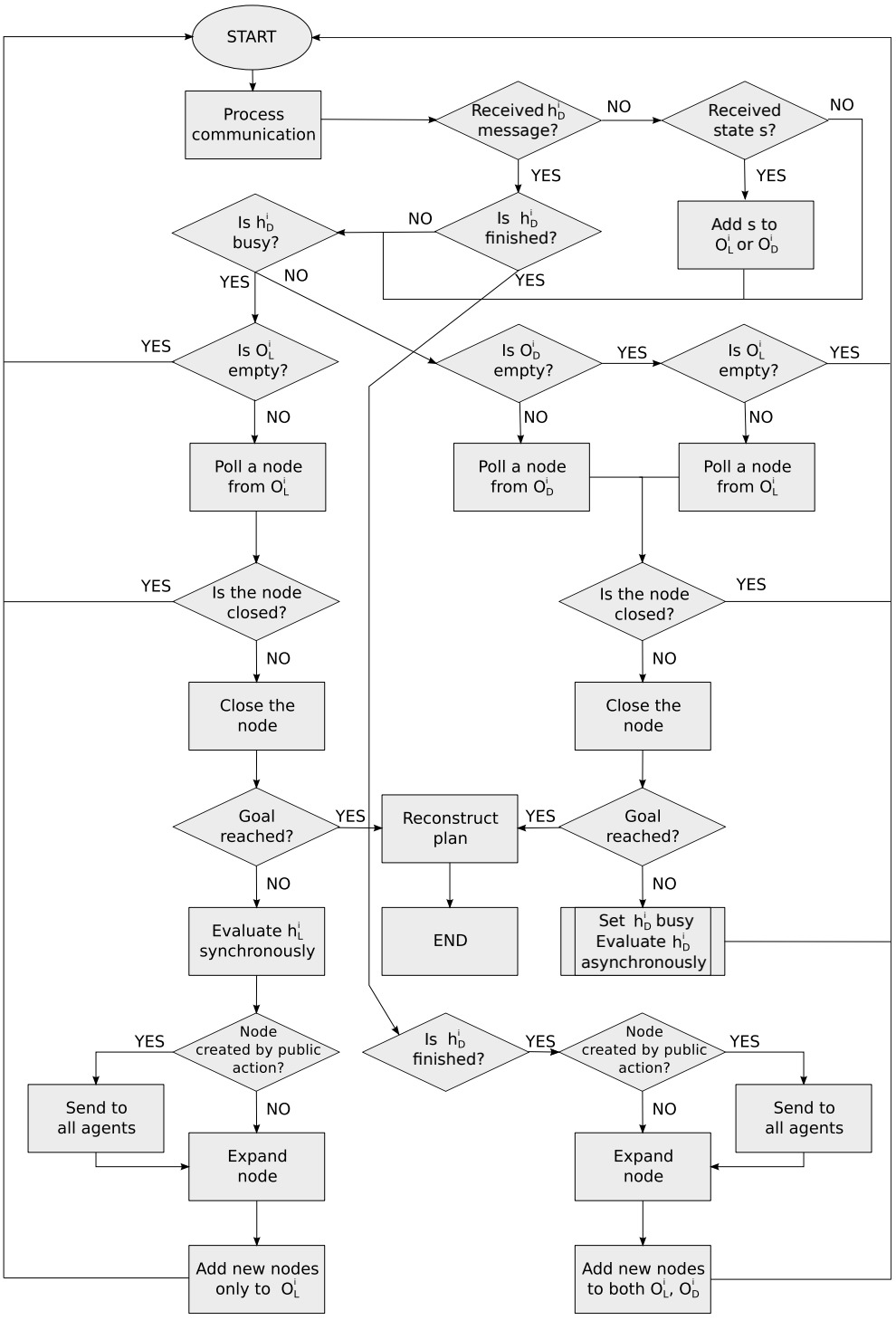


Figure 5.2.1: Flow chart showing one iteration of the main search loop for a single agent.

If a state s is taken from the local open list, its successor is inserted only in the local open list. If the state s is taken from the distributed open list, its successor is inserted into both local and distributed open lists. If state s' was obtained by application of a private action, it is inserted in the respective open lists of agent α_i. If the action is public, it is also inserted in open lists of all agents $\alpha_{j\neq i}$ via a message sent to them by α_i. The messages include additional information whether the state should be added to O^j_{D} (if it was evaluated by h^i_{D}) or to O^j_{L} (if it was evaluated by h^i_{L}). In the latter case, the heuristic estimate is recomputed using h^j_{L}, because the local heuristic estimate from the agent α_i may significantly differ from that of agent α_j.

The other direction $O^i_{\mathsf{L}} \rightarrow O^i_{\mathsf{D}}$ is trickier. If we added a state s evaluated by h^i_{L} to O^i_{D} the search would skip many states which are actually closer to the goal only because the local, less informative heuristic, gives a lower estimate. The way at least some information can be shared in this direction is that whenever the open list O^i_{D} is empty and the heuristic estimator h^i_{D} is not computing any heuristic, the best state s is extracted from the local open list O^i_{L} and evaluated by the distributed heuristic h^i_{D} and its successors are added to both open lists. This way, the states placed in O^i_{D} are evaluated only by h^i_{D}, but sometimes, the best state from O^i_{L} is taken to be evaluated by h^i_{D}.

The search is terminated when a goal state is reached by one of the agents, followed by a distributed plan extraction, or if there is no solution. Detection of solution nonexistence in the multi-agent setting is more complicated than in classical planning. Even if both open lists are empty, the agent cannot be sure that some other agent is not going to find a solution or broadcast some new state in the future. Therefore, the agents need to check that all open lists are empty and also that there are no pending messages to be delivered (that is no state is "in the air"). By that, we close the high-level description of the MADLA Search and the MADLA Planner and continue with a detailed formal description.

5.3 Formal Description of the MADLA Search

To talk about the actual algorithm formally, we first need to distinguish the state $s \subseteq P$ and its representation in the search algorithm. Note, that due to privacy concerns (as explained later), each agent must have its own set of search nodes with its own representation of the actual state. When a state is to be sent to another agent, only its public projection is sent together with a tuple of ids representing the private parts of each agent. Thus, the search node of agent α_i is represented as follows.

Definition 46. (Search node) A *search node* representing a state $s \in P$ is a tuple

$$u = \left\langle s^{\triangleright i}, p, a_{\mathsf{par}}, h, g, \alpha_i, \langle \delta_1, ..., \delta_n \rangle \right\rangle$$

where $s^{\triangleright i} \subseteq P_i$ is the i-projected state, p is the parent of the search node u. Action $a_{\mathsf{par}} \in A_i \cup \{\epsilon\}$ is the action used to create s, h is the heuristic value, g the distance of s from s_I, α_i is the agent u belongs to and $\langle \delta_1, ..., \delta_n \rangle$ is the n-tuple of private unique identifiers representing the private parts $s \cap P^{\mathsf{priv}}_j$ of s for all agents $\alpha_j \in \mathcal{A}$ including α_i.

The parent p is determined as follows. If the search node u is created from a predecessor search node u' by the application of an action a, then $p = u'$ and $a_{\mathsf{par}} = a$. In the case the search node is created from a state received from another agent $\alpha_{j\neq i}$, then $p = \alpha_j$ and $a_{\mathsf{par}} = \epsilon$. Each search node is agent-dependent, that is, it belongs to the agent α_i, which is part of the search node definition, and thus indexing a node u as u_i would be superfluous and is omitted.

We extend the action application to search nodes so that for a search node

$$u_k = \left\langle s^{\triangleright i}_k, p, a_k, h_k, g_k, \alpha_i, \langle \delta_1, ..., \delta_i, ..., \delta_n \rangle \right\rangle$$

and an action $a_{k+1} \in A_i$ we define

$$u_k \circ a_{k+1} = u_{k+1} = \left\langle s^{\triangleright i}_k \circ a_{k+1}, u_k, a_{k+1}, h_k, g_{k+1} = g_k + 1, \alpha_i, \langle \delta_1, ..., \delta'_i, ..., \delta_n \rangle \right\rangle$$

where p can either be a search node or an agent. As we use deferred heuristic evaluation, the heuristic estimate h_k is in fact the heuristic estimate of the state represented by the parent node. Thus, the heuristic estimate is not changed by the action application, but is updated later after the new search node is extracted from the open list and evaluated.

We define $\mathcal{U}^i$ as the set of all possible search nodes of agent α_i.

Definition 47. (Public search node) A *public search node* is a tuple $u^{\triangleright} = \langle s^{\triangleright}, p, a_{\mathsf{par}}, h, g, \langle \delta_1, ..., \delta_n \rangle \rangle$ where $s^{\triangleright} \subseteq P^{\mathsf{pub}}$ is the public projection of state s, i.e., $s^{\triangleright} = s \cap P^{\mathsf{pub}}$.

For the use in the algorithm (and consequent proofs) we define $u.\mathsf{state} = s^{\triangleright i}$, $u.\mathsf{action} = a_{\mathsf{par}}$, $u.\mathsf{parent} = p$ (that is $u.\mathsf{parent} = u'$ or $u.\mathsf{parent} = \alpha_j$), $u.h = h$, $u.g = g$, $u.\mathsf{agent} = \alpha_i$, $u.\mathsf{uids} = \langle \delta_1, ..., \delta_n \rangle$ and $\mathsf{enc}_u(i) = \delta_i$. A search node is agent-dependent, it is always created by an agent and never sent to another agent. When sending a state represented by a search node u, it is first converted to a public search node $u^{\triangleright}$ and then only the public projection $s^{\triangleright}$ of the state is sent together with the private unique identifiers $\langle \delta_1, ..., \delta_n \rangle$.

Each agent α_i starts with an initial search node u_I created from the initial state s_I as the following

$$u_I = \left\langle s_I^{\triangleright i}, \mathsf{null}, \mathsf{null}, h_0, 0, \alpha_i, \langle 0, ..., 0 \rangle \right\rangle$$

By assigning 0 to each δ_j we represent that the private part of other agents represent the initial state (of course any value can be used as long as it is agreed upon by all agents).

Definition 48. (MADLA Agent) The agent data structure is defined as

Object `Agent(`α_i`)`
- O^i_{D}; // distributed heuristic open list
- O^i_{L}; // local heuristic open list
- C^i; // closed list
- $\mathsf{busy}^i_{\mathsf{D}}$; // determines if the distributed heuristic is being computed
- search^i; // determines if the search should continue
- h^i_{D}; // distributed heuristic estimator
- h^i_{L}; // local heuristic estimator
- μ^i; // mapping of the sent states to the search nodes
- $\mathsf{plans}^i = \{\langle \alpha_j, \pi^i_j \rangle, ...\}$ // currently reconstructed plans

The data structures related to each agent α_i are grouped in an object-like description in Definition 48. In each algorithm or procedure, the agent object is given as the first parameter (similarly to "this" in object-oriented languages) and the data structures are accessed directly. The structures O^i_{D}, O^i_{L} and C^i denote the open lists and the closed list of agent α_i and $\mathsf{busy}^i_{\mathsf{D}}$ is a boolean variable denoting whether the distributed heuristic estimator of agent α_i is busy or not, similarly search^i denotes whether the search loop should continue or not. Moreover, h^i_{D} and h^i_{L} denote the distributed and local heuristic evaluators of agent α_i. Finally, the definition includes the state reconstruction function μ^i, and a set plans^i of all currently reconstructed plans in the form of a tuple $\langle \alpha_j, \pi^i_j \rangle$ consisting of an agent (the plan reconstruction originator) and the (partially) reconstructed plan π^i_j, initially empty, prospectively a part of the multi-agent plan $\{\pi^i\}_{i=1}^n$. The set plans^i is initially empty.

Now, let us have a detailed look at the pseudo-code and some implementation details. First, we start with Algorithm 9 which outlines the main loop of the search together with initialization. The algorithm (and all subsequent algorithms) is seen from the perspective of agent α_i, that is each agent runs a copy of Algorithm 9 in parallel with all other agents. All data structures are local to agent α_i and all search nodes contain i-projected states.

The initialization starts with the open lists containing the initial search node consisting of the i-projected initial state $s_I^{\triangleright i}$, no parent, no action and the agent α_i. The closed list C^i is initialized as empty and the distance of the initial search node is set to $u_I.g \leftarrow 0$. Although the open lists are ordered only by the heuristic value, we need the g value for the later plan reconstruction to know the length of

Algoritmus 9: MADLA Search for agent α_i and multi-agent planning problem $\mathcal{M}$. The plan is not returned directly by the algorithm as the reconstruction is asynchronous and thus the plan is returned by the Procedure 11.

```
1  Algorithm MADLA-Search(α_i, Π_i)
2      O^i_D ← {u_I}; O^i_L ← {u_I};
3      C^i ← ∅; u_I.g ← 0; u_I.h ← h_0;
4      busy^i_D ← false;
5      search^i ← true; // search begins
6      while search^i do
7          processComm(α_i) ; // see Algorithm 11
8          if busy^i_D = false then
9              if O^i_D ≠ ∅ then
10                 u ← arg min_{u∈O^i_D} u.h;
11             if O^i_L ≠ ∅ then
12                 u ← arg min_{u∈O^i_L} u.h;
13             else
14                 goto Line 6;
15             processNode(α_i, u, true) ;
16         // local search loop
17         while (O^i_D = ∅ or busy^i_D = true) and O^i_L ≠ ∅ do
18             u ← arg min_{u∈O^i_L} u.h;
19             processNode(α_i, u, false) ;
20             //process communication also in the local search loop
21             processComm(α_i) ;
22 Procedure processNode(α_i, u, d)
23     if u ∉ C^i then
24         C^i ← C^i ∪ {u}; // close search node
25         if G ⊆ u.state then
26             search^i ← false; //end search
27             //inform agents
28             send M_PLANFOUND to all α_j ∈ A;
29             //(asynchronously) reconstruct and return the plan
30             reconstructPlan(α_i, u, u.g, α_i) ; // see Algorithm 12
31         expand(α_i, u, d) ; // see Algorithm 10
```

the plan being reconstructed upfront in order to be able to insert the appropriate number of no-op actions in the place of actions of other agents. The heuristic value of the initial node is set to some initial value $u_I.h = h_0$ which is only a mock value for later state selection.

The main loop terminates when a solution is found, or both open lists are empty (for all agents) and there are no pending messages, which means that there is no solution. Such synchronization can be straightforwardly achieved using the textbook distributed snapshot algorithm [Chandy and Lamport, 1985], but we leave it out in the pseudo-code for simplicity.

The distributed snapshot algorithm is a general technique to determine the state of a distributed system and roughly works as follows. The agent performing the snapshot saves its own local state and sends a snapshot request message with a snapshot token to all other agents. When any agent receives this particular snapshot token for the first time, it sends the initiator agent its own saved state (e.g., all open lists are empty) and attaches the snapshot token to all subsequent messages. When an agent that has already received the snapshot token receives a message that does not have the snapshot token, the agent needs to inform the snapshot initiator about the message (e.g., if it was a state message, its open lists are no longer empty). This message was sent before the snapshot started and needs to be included in the snapshot. This means that all messages need to have the possibility to include the snapshot token, which we have also not included in the formal description in order to keep it simpler.

The main loop follows the classical heuristic search with deferred evaluation scheme with some modifications. From line 8 to line 15, an open list is determined based on the state of the distributed heuristic and the node u with minimal heuristic value $u.h$ is extracted from the selected open list (if both open lists are empty, the loop continues with checking the messages). Node u is processed as in classical search (added to open list, checked for the goal), but if u contains a goal state, distributed plan reconstruction is initiated (see Algorithm 12). If u does not contain a goal state, u is expanded (see Algorithm 10). Since we use the deferred heuristic evaluation, the heuristic is evaluated before the actual expansion.

Algoritmus 10: Procedure expand(α_i, u, d)

```
1  Procedure expand(α_i, u, d)
2      if d = false then
3          u.h ← h^i_L(u.state) // local
4      else
5          u.h ← h^i_D(u.state); // distributed, asynchronous
6          set busy^i_D ← true when h^i_D starts evaluation
7          set busy^i_D ← false when h^i_D finishes evaluation (asynchronously)
8      if u.action ∈ A^pub_i then
9          send M_STATE = ⟨u^▷.state, u^▷.uids, u.h, d⟩;
10         μ(u^▷.state, enc_{u^▷}(i)) ← u;
11     E ← ∅
12     for all a ∈ A_i s.t. pre(a) ⊆ u.state do
13         u' ← u ∘ a;
14         u'.g ← u.g + 1;
15         u'.h ← u.h; // deferred heuristic
16         E ← E ∪ {u'};
17     O^i_L ← O^i_L ∪ {u' | u' ∈ E ∧ u' ∉ C^i};
18     if d = true then
19         O^i_D ← O^i_D ∪ {u' | u' ∈ E ∧ u' ∉ C^i};
```

The expansion procedure is detailed in Algorithm 10. The node u is evaluated using either the

local projected heuristic h^i_{L}, which proceeds as usual or using the distributed global heuristic h^i_{D}. The distributed heuristic evaluation is represented as a function call for simplicity. In reality, the procedure evaluating the heuristic function is asynchronous. This means that a callback is passed to the procedure and the `expand(`α_i, u, d`)` procedure exits. Only when the heuristic evaluation is finished, the rest of the procedure continues. This means the following steps are performed either directly after the local heuristic evaluation or in the callback of the distributed heuristic evaluation.

If node u was expanded by a public action, a message M_{STATE} is sent to all other agents, containing the public projection $s^{\triangleright}$ of state s together with the private unique identifiers representing the private parts of all agents, its $u.g$, its heuristic value $u.h$ and a parameter d determining whether it was evaluated using the local or distributed heuristic in order to determine to which open list it should be added.

The states sent to other agents are stored by a mapping function $\mu^i : 2^{P^{\mathsf{pub}}} \times \mathbb{N} \to \mathcal{U}^i$, which assigns to a public projection of a state $s^{\triangleright}$ and a private unique identifier δ_i the respective search node. The function μ^i is used both to be able to reconstruct the path later and to reconstruct the private part of a received state. The storing does not cause significant memory overheads as the states are stored in the closed list anyway.

Next, the node u is expanded using all applicable actions from A_i, the new nodes u' are created according to the Definition 46 (so that $u'.\mathsf{state} = s^{\triangleright i} \circ a$, $u'.\mathsf{parent} \leftarrow u$, $u'.\mathsf{action} \leftarrow a$, $u.h \leftarrow u'.h$, $u.g \leftarrow u'.g+1$, $u'.\mathsf{agent} \leftarrow \alpha_i$, $\mathsf{enc}_{u'}(j) \leftarrow \mathsf{enc}_u(j)$ for all $j \neq i$ and $\mathsf{enc}_{u'}(i)$ is assigned a new private unique identifier) and the new search nodes are added to the open list(s) based on which heuristic was used for evaluation.

In addition to the main loop in Algorithm 9, there is an inner loop on lines 17–21, which is performed when the distributed open list is empty or busy with a heuristic evaluation (this happens when the distributed heuristic is being asynchronously evaluated). This inner loop is again the classical heuristic search loop with deferred evaluation, this time taking into account only the local open list and local heuristic. This inner local search loop also needs to process the communication, that is to send and to receive messages (otherwise the distributed heuristic computation could not be finished).

The part of the communication taking care of message receiving is shown in Algorithm 11 (message sending is trivial). There are four types of messages sent among the agents for the distributed search (more messages are sent for the distributed heuristic computation). A state message $M_{\mathsf{STATE}} = \langle s^{\triangleright}, \langle \delta_1, ..., \delta_n \rangle, h, g, d \rangle$ contains a public projection $s^{\triangleright}$ of a state s, the private unique identifiers, the state's heuristic value h, g and the parameter d. When received, the i-projection $s^{\triangleright i}$ is reconstructed from $s^{\triangleright}$ and the private unique identifier δ_i using the function μ^i, which encodes the node and respective i-projected state from which the private part should be copied. Thus the previously anonymized private part of agent i is restored, and a new search node u is created, its parent is set to the sending agent α_j. The heuristic is re-computed to reflect the local problem of the receiving agent, although not recomputing the heuristic does not have a significant effect in most domains. As the heuristics are not admissible, taking the maximum as in MAD-A* is not reasonable. Finally, the node u is added to the open list(s) selected based on the received parameter d.

The "plan found" message $M_{\mathsf{PLANFOUND}}$ (with no parameters, except for the implicit sender α_j) is sent by an agent α_j when it reaches a goal state to inform other agents that it has found a plan and it is starting the plan reconstruction process. When received by the agent α_i, it knows that it can exit the search loop (the communication still has to be processed). A new tuple $\langle \alpha_j, \pi^i_j \rangle$ is added to plans^i as the reconstruction of plan initiated by α_j will take place. Note that before receiving the $M_{\mathsf{PLANFOUND}}$ message, the agent α_i could have already started the plan reconstruction itself, thus multiple plans can be reconstructed in parallel.

The plan reconstruction message $M_{\mathsf{RECONSTRUCT}} = \langle s^{\triangleright}, \delta_i, t, \alpha_k \rangle$ contains the state s (represented as $s^{\triangleright}$ and the private unique identifier δ_i of the receiving agent), from which the reconstruction should continue, a discrete time-point t, initially set to $u.g$ of the last search node u (that is, the search node which satisfied the goal condition) and the agent α_k which started the plan reconstruction. The identifier of the agent α_k is used to determine which partial plan $\langle \alpha_k, \pi^i_k \rangle \in \mathsf{plans}^i$ the reconstruction message belongs to. To reconstruct the plan, first the respective search node needs to be retrieved, which is the

Algoritmus 11: processComm(α_i)

```
1  Procedure processComm(α_i)
2      for each message M in message queue, received from α_j do
3          switch M do
4              case M_STATE = ⟨s^▷, ⟨δ_1, ..., δ_n⟩, h, g, d⟩
5                  u_sent ← μ^i(s^▷, δ_i);
6                  s^▷i ← u_sent.state;
7                  u ← ⟨s^▷i, α_j, ϵ, h, g, α_i, ⟨δ_1, ..., δ_n⟩⟩;
8                  if d = true then
9                      O^i_D ← O^i_D ∪ {u};
10                 else
11                     u.h ← h^i_L(u.state) // local heuristic recomputed
12                 O^i_L ← O^i_L ∪ {u};
13             case M_PLANFOUND
14                 search^i ← false;
15                 plans^i ← plans^i ∪ {⟨α_j, ∅⟩};
16             case M_RECONSTRUCT = ⟨s^▷, δ_i, t, α_k⟩
17                 //continue reconstruction of the best plan so far
18                 u ← μ^i(s^▷, δ_i); //reconstruct search node for the sent state s^▷
19                 reconstructPlan(α_i, u, t, α_k);
20             case M_TERMINATE = ⟨α_k⟩
21                 if received M_TERMINATE = ⟨α_j⟩ for all α_j s.t. ⟨α_j, π^i_j⟩ ∈ plans^i then
22                     select π^i_min of minimal length from plans^i;
23                     report solution π^i_min and terminate;
24             case else
25                 forward M to h^i_D;
```

search node returned by $\mu^i(s^{\triangleright}, \delta_i)$ based on the public part $s^{\triangleright}$ and private part δ_i of the state $s^{\triangleright i}$, which was sent to α_j on line 9 in Algorithm 10. Next, the plan reconstruction procedure is invoked (see Algorithm 12).

Finally, the termination message $M_{\mathsf{TERMINATE}} = \langle \alpha_k \rangle$ announces that the reconstruction of the plan initiated by the agent α_k has been finished by the sender agent α_j. Only when a message $M_{\mathsf{TERMINATE}} = \langle \alpha_k \rangle$ is received for all α_k such that $M_{\mathsf{PLANFOUND}}$ was received from α_k before and thus there is $\langle \alpha_k, \pi_k^i \rangle \in \mathsf{plans}^i$, the plan reconstruction of all plans is finished and the shortest plan can be selected (ties are broken based on the ordering of the agent indexes). Of course, it may happen that the agent α_i was not involved in the plan reconstruction at all and thus its plan π_i^i consists of l no-op actions ϵ, where l is the length of the shortest plan.

Note that messages for the distributed heuristic h^i_{D} are also handled in Algorithm 11 and are forwarded to the heuristic estimator.

Algoritmus 12: reconstructPlan(α_i, u, t, α_k)

```
1  Procedure reconstructPlan(α_i, u, t, α_k)
2      π^i_k ← ⟨α_k, π^i_k⟩ ∈ plans^i
3      π^i_k[t] ← u.action;
4      π^i_k[t'] ← ε for all t' > t s.t. π^i_k[t'] is empty;
5      t ← t − 1;
6      if u = u_I then
7          send M_TERMINATE = ⟨α_k⟩ to all α_j ∈ A;
8      if u.action = ε then
9          //node received from u.parent
10         send M_RECONSTRUCT = ⟨u^▷.state, enc_u(j), t, α_k⟩ to α_j ← u.parent
11     else
12         reconstructPlan(α_i, u.parent, t, α_k);
```

The distributed plan reconstruction procedure is shown in Algorithm 12. Recall that the plan is stored in a list $\langle \alpha_k, \pi_k^i \rangle$ of actions for each agent, together forming a multi-agent plan $\{\pi_k^i\}_{i=1}^n$ for each such k. By $\pi_k^i[t]$, we denote the action on the t-th position in the plan π_k^i. The reconstruction process is started by an agent α_k in a search node u s.t. $G \subseteq u.\mathsf{state}$ (see Algorithm 9 line 25) and with $t = u.g$ which is the distance from u_I to u. Every time an action is added to the position t of π_k^i (line 3), t is decreased by 1, which represents backward reconstruction of the plan. All positions between the last added action and t are padded with the no-op action ϵ (line 4).

If the initial node is reached (line 6), the terminate message is sent to all other agents. Otherwise, the parent $u.\mathsf{parent}$ of node u is retrieved, which is either the predecessor node or an agent $\alpha_{j\neq i}$. If $u.\mathsf{parent} = \alpha_{j\neq i}$ a reconstruct message is sent to α_j prompting it to continue with the reconstruction from a state s represented by the search node u.

As multiple agents may start plan reconstruction independently, any agent that found a solution reports to all other agents by sending a $M_{\mathsf{PLANFOUND}}$ message. This way, each agent α_i receiving the message terminates the search and adds $\langle \alpha_j, \pi_j^i \rangle$ to the plans^i set (see Algorithm 11 line 15). This means that a plan reconstruction was started by agent α_j. When the plan reconstruction started by the agent α_k is finished by an agent α_j by reaching the initial search node u_I (see Algorithm 12 line 6) the $M_{\mathsf{TERMINATE}} = \langle \alpha_k \rangle$ message is sent to all agents. When $M_{\mathsf{TERMINATE}} = \langle \alpha_k \rangle$ sent by α_j is received by α_i, the agent α_i knows that the reconstruction of $\langle \alpha_k, \pi_k^i \rangle$ has been finished. When received for all such $\langle \alpha_k, \pi_k^i \rangle \in \mathsf{plans}^i$, the shortest plan from plans^i is chosen (see Algorithm 11 line 22) and reported as the solution for agent α_i (see Algorithm 11 line 23). To ensure that each agent α_i chooses the corresponding plan π_k^i together forming the distributed multi-agent plan $\{\pi_k^i\}_{i=1}^n$, the ties are broken based on the ordering of the indexes k of the agents which started the reconstruction of the respective

plan. The description of details of the distributed plan reconstruction concludes the proposed MADLA Search.

5.4 Properties of the MADLA Search

Here we present the proofs of soundness and completeness of the MADLA Search algorithm. The proofs are based on proofs for classical single-agent single-heuristic search.

Before delving into the proof, we state the assumptions on the used heuristics. First, the heuristics always terminate. Second, the heuristics are *safe* (see Definition 41iv). In other words, if the heuristic reports a dead-end, it truly is a dead-end (note that the other direction of the implication does not have to hold, a heuristic (e.g., FF) can report a finite heuristic value for a state which actually is a dead-end).

Throughout the section, we use the notion of a search node from Definition 46. Note that the search node is agent-specific, that is a search node $u = \langle s^{\triangleright i}, p, a_p, h, g, \alpha_i, \langle \delta_1, ..., \delta_n \rangle \rangle$ is known to and manipulated by agent α_i only. Subsequently, a search node represents a global state s as an i-projection $s^{\triangleright i}$ and a tuple $\Delta = \langle \delta_1, ..., \delta_n \rangle$ encoding the private parts of all agents in $\mathcal{A}$. We use the dot notation $u.\mathsf{state} = s^{\triangleright i}$, $u.\mathsf{action} = a$, $u.\mathsf{parent} = p$ (that is $u.\mathsf{parent} = u'$ or $u.\mathsf{parent} = \alpha_j$), $u.h = h$, $u.g = g$, $u.\mathsf{agent} = \alpha_i$, $u.\mathsf{uids} = \Delta = \langle \delta_1, ..., \delta_n \rangle$ and $\mathsf{enc}_u(i) = \delta_i$.

We define a global equality relation for search nodes as follows.

Definition 49. Let u, u' be search nodes such that $u.\mathsf{agent} = \alpha_i$ and $u'.\mathsf{agent} = \alpha_j$ and let s, s' be global states reconstructed from $u.\mathsf{state}$ and $u'.\mathsf{state}$ respectively using all private parts defined in $u.\mathsf{uids}$ and $u'.\mathsf{uids}$ respectively. Then the search nodes u and u' are globally equal iff the states s and s' are equal, formally $u \overset{\mathsf{G}}{=} u' \Leftrightarrow s = s'$.

In other words, two search nodes are globally equal if the global states represented by them are equal.

5.4.1 Proof of Soundness

First, we define the sequence of search nodes corresponding to a particular run of the algorithm.

Definition 50. (Path) A *path* with a *resulting search node* u_l is a sequence of search nodes $\bar{u_l} = (u_0, ..., u_l)$, possibly of different agents (different k s.t. $1 \leq k < l$ can exist for which $u_k.\mathsf{agent} \neq u_{k+1}.\mathsf{agent}$). The search nodes do not repeat, although the respective states may repeat.

Definition 51. (Valid path) A path is a *valid path* iff $u_0 = u_I$ is the initial node for some agent α_I and for each k s.t. $1 \leq k \leq l$: $u_{k-1} = u_k.\mathsf{parent}$ and $a_k = u_k.\mathsf{action}$ is applicable in $u_{k-1}.\mathsf{state}$, or $u_k.\mathsf{agent} = \alpha_i$ and $u_k.\mathsf{parent} = \alpha_j$ s.t. $i \neq j$, in which case $u_{k-1}.\mathsf{agent} = \alpha_j$ and $u_k \overset{\mathsf{G}}{=} u_{k-1}$.

Informally, a path represents one particular trace of the exploration of the search space. In the case a state was reached through expansions of multiple agents, agent α_j sends a state s_{k-1} to agent α_i which creates a new search node u_k from it. This results in a sequence of search nodes $(u_0, ..., u_{k-1}, u_k, ..., u_l)$ where the search nodes u_{k-1}, u_k represent the same state s_{k-1}, but u_{k-1} was created by α_j (using a public action) and u_k was created subsequently (after receiving the message) by α_i, setting $u_k.\mathsf{parent}$ to α_j (see Algorithm 11, line 7). Next, we define the corresponding sequence of actions.

Definition 52. For a path $\bar{u_l} = (u_0, u_1, ..., u_l)$, we say *$\bar{u_l}$-plan* is a sequence of actions $(a_1, ..., a_l)$ s.t. $a_k = u_k.\mathsf{action}$ and $a_k \in A \cup \{\epsilon\}$ for all $1 \leq k \leq l$. For all a_k it holds that either $a_k \in A_i$ where $\alpha_i = u_k.\mathsf{agent}$ or $a_k = \epsilon$ if u_k was received form another agent (that is $u_{k-1}.\mathsf{agent} \neq u_k.\mathsf{agent}$). The $\bar{u_l}$-plan is a *valid $\bar{u_l}$-plan* iff $\bar{u_l}$ is a valid path.

A trivial consequence of the above definitions and Definition 4 is that a valid $\bar{u_l}$-plan $(a_1, ..., a_l)$ is a multi-agent plan solving $\mathcal{M}$ iff $u_l.\mathsf{state}$ is a goal state, that is $G \subseteq u_l.\mathsf{state}$ (remember that the goal is public, i.e. $G \subseteq P^{\mathsf{pub}}$).

To prove the soundness, the following lemma will be shown first:

Lemma 53. *(Invariant) At any given step of the MADLA Search, for any search node $u_{\mathsf{L}} \in O^i_{\mathsf{L}}$ and any search node $u_{\mathsf{D}} \in O^i_{\mathsf{D}}$ for any agent α_i, $\bar{u_{\mathsf{L}}}$ and $\bar{u_{\mathsf{D}}}$ are valid paths.*

Proof. In the initial step for every agent, $O^i_{\mathsf{L}} = O^i_{\mathsf{D}} = \{u_I\}$, where $\bar{u_I} = (u_I)$, which is a valid path. There are two possibilities where new nodes are added to any of the open lists. In `expand(`u, d`)`, regardless of the d parameter, for each action applicable in $u.\mathsf{state}$, a new search node $u' = \left\langle s'^{\triangleright i}, u, a, h', g+1, \alpha_i, \Delta \right\rangle$ is created such that $s'^{\triangleright i} = u.\mathsf{state} \circ a$. If we assume that $\bar{u} = (u_I, ..., u)$ is a valid path, then after expansion, $\bar{u'} = (u_I, ..., u, u')$ is also valid for each new u'.

A node may be received from another agent in `processComm()`. Assume that for some node $u_k = \left\langle s_k^{\triangleright j}, u_{k-1}, a_k, h_k, g_k, \alpha_j, \Delta_k \right\rangle$, $\bar{u_k}$ is a valid path and u_k is a first node in $\bar{u_k}$ expanded using a public action $a_k \in A^{\mathsf{pub}}_j$. According to Algorithm 10, line 9, u_k is sent to α_i as a message $M_{\mathsf{STATE}} = \langle s^{\triangleright}, \Delta_k, h, g, d \rangle$. The message is received by α_i and a new search node $u_{k+1} \leftarrow \left\langle s_k^{\triangleright i}, \alpha_j, \epsilon, h_k, g_k, \alpha_i, \Delta_k \right\rangle$ is created (based on Δ_k and the state reconstruction function μ^i) and added to either open list based on the parameter d. According to Definition 50, $\bar{u}_{k+1}$ is a valid path. There is no other possible way a node could be added to any of the open lists. □

Theorem 54. *(Soundness of the MADLA Search) When the MADLA Search terminates and returns a solution, it is a distributed multi-agent plan solving $\mathcal{M}$.*

Proof. The algorithm terminates on line 23 of Algorithm 11. In each step, either an action $a_k = u_k.\mathsf{action}$ is added to the solution π^i_m for the solution initiated by agent α_m and continues the recursion on $u_{k-1} = u_k.\mathsf{parent}$ (if $u_k.\mathsf{action} \neq \epsilon$), or a $M_{\mathsf{RECONSTRUCT}} = \langle u_k^{\triangleright}.\mathsf{state}, \mathsf{enc}_{u_k}(j), t, \alpha_m \rangle$ message is sent to $\alpha_j = u_k.\mathsf{agent}$. When received, the `reconstructPlan()` procedure is called on a node u'_k s.t. $u'_k \overset{\mathsf{G}}{=} u_k$ and $u'_k.\mathsf{action}$ is public (u'_k is obtained from μ^i). Thanks to the condition on line 6 of Algorithm 12, upon termination, the last action added to the returned solution π^i_m is the action that was applied on the initial node u_I. This ensures that the recursion proceeds along the path $\bar{u_t}$, where u_t is the node on which `reconstructPlan()` was first called.

Apart from the recursive call, the procedure `reconstructPlan()` is called only from line 30 of Algorithm 9, where u_t was extracted from O^i_{L} or O^i_{D}. From Lemma 53 it follows that $\bar{u_t}$ is a valid path. Because on line 30 of Algorithm 9, $G \subseteq u_t.\mathsf{state}$ always holds, the $\bar{u_t}$-plan corresponding to $\bar{u_t}$ is a multi-agent plan π solving $\mathcal{M}$. As each agent α_i adds to π^i_m only actions from A_i (and ϵ actions as padding, see Algorithm 12 line 4), the resulting set $\{\pi^i_m\}^n_{i=0}$ is a distributed multi-agent plan solving $\mathcal{M}$ for each m as the reconstructions are independent if started by different agents and each agent can start the reconstruction of at most one plan. The final plan is chosen consistently and uniquely as it is the shortest plan, ties broken based on unique agent indices (ordering). □

5.4.2 Proof of Completeness

To show completeness, we first consider a modification of the algorithm such that any reachable state is expanded eventually. The modified algorithm is named MADLA$^+$ Search in which the condition on lines 25–30 in Algorithm 9 are ignored and the algorithm terminates only when both O^i_{L} and O^i_{D} are empty for all agents α_i and no messages are pending, which can be detected using the distributed snapshot algorithm [Chandy and Lamport, 1985].

Lemma 55. *In the MADLA$^+$ Search, each state s is added to O^i_{D} and to O^i_{L} of any agent α_i at most finite times, each time represented by a different search node.*

Proof. Because the number of possible search nodes (with respect to the state $s^{\triangleright i}$ and the set of private unique identifiers Δ) is finite as each search node represents one state of the finite sets of states ($2^{|P|}$ since $s \subseteq P$) and the number of actions (of each agent) is also finite, each expansion produces a finite number of search nodes consequently added to O^i_{D}, O^i_{L} or both (line 17 and 19 of Algorithm 10 respectively). If a search node u is extracted from O^i_{L} or O^i_{D}, it is added to the closed list C^i and no search node u' s.t. $u \overset{\mathsf{G}}{=} u'$ is ever added to any of the open lists of agent α_i again.

Another possibility of adding a search node to an open list is when it is received from another agent. A state s is sent by the agent α_i, only if a search node u' (representing a different state s') was extracted from either O^i_{L} or O^i_{D} and a public action $a \in A^{\mathsf{pub}}_i$ was applied. Since there is a finite number of public actions and a finite number of agents and each action is applicable only in a finite number of states (which are then placed into C^i and never expanded again), state s can be sent and received only a finite number of times. □

Lemma 56. *In the MADLA$^+$ Search, each search node in O^i_{L} and in O^i_{D} of all agents α_i is eventually extracted.*

Proof. In each step of the outer search cycle (lines 6–21 of Algorithm 9), a node is extracted from O^i_{D}, if h^i_{D} is not busy. Since h^i_{D} always terminates, any finite number of nodes can be extracted in finite time. From Lemma 55 follows that only a finite number of nodes may be added to O^i_{D}, therefore O^i_{D} eventually becomes empty. When O^i_{D} is empty, in each step of the inner search cycle a node is extracted from O^i_{L}. Following the same reasoning as before, O^i_{L} eventually becomes empty as well. □

Theorem 57. *(Termination of the MADLA$^+$ Search) The MADLA$^+$ Search terminates.*

Proof. Follows directly from Lemma 55 and Lemma 56. □

Recall that for states s_0 and s_m a s_0–s_m-plan is a sequence of actions $\pi = (a_1, ..., a_m)$ from A (the actions may be from different agents and may repeat) if there are states $s_1, ..., s_m$ s.t. for all k in $1 \leq k \leq m$, action a_k is applicable in s_{k-1} and $s_k = s_{k-1} \circ a_k$. For short, $s_0\,[\pi]$ denotes the resulting state s_m. We extend this notion to the search nodes the same way action application was extended (i.e., the actions are applied on the respective nodes).

We define the notion of reachability for the multi-agent planning problem $\mathcal{M}$ as follows.

Definition 58. (Reachability) A state $s \subseteq P$ is reachable in $\mathcal{M}$ iff a s_I–s-plan $\pi = (a_1, ..., a_m)$ exists such that s_I is the initial state. We say that s is reachable by agent α_i iff $a_m \in A_i$.

Now we show equality of reachability and the existence of a valid path.

Lemma 59. *A state s is reachable in $\mathcal{M}$ by agent α_i iff a valid path $\bar{u} = (u_I, ..., u)$ exists such that $u_I.\mathsf{state} = s_I^{\triangleright j}$ for some agent α_j, $u.\mathsf{state} = s^{\triangleright i}$, $u.\mathsf{uids}$ represent private parts of s and $u.\mathsf{agent} = \alpha_i$.*

Proof. If a valid path exists, the corresponding $\bar{u}$-plan proves the reachability. If a state s is reachable in $\mathcal{M}$ by α_i, there exists the sequence $\pi = (a_0, ..., a_l)$ of actions from Definition 58 (and $a_l \in A_i$). Let $a_0 \in A_j$. Since a_0 is applicable in $s_I^{\triangleright j} = u_I.\mathsf{state}$, it will be applied by the agent α_j and the resulting search node u_1 will be added to its open lists. Additionally, if $a_0 \in A^{\mathsf{pub}}_j$, $s_1^{\triangleright} = u_1.\mathsf{state}$ together with $u_1.\mathsf{uids}$ will be sent to other agents. According to the Definition 58, a_1 is applicable in $s_1^{\triangleright j}$. If $a_1 \in A_j$ the process is repeated. If $a_1 \in A_{k \neq j}$, a_1 is applicable in the received state $s_1^{\triangleright k}$ and thus will be applied by agent α_k. By induction we conclude that $\bar{u} = (u_I, ..., u)$ is a valid path. □

Now, we provide an alternative definition of reachability in $\mathcal{M}$, with the focus shifted on the agents. We use the bracketed index (k) to annotate the k-th agent in the sequence which is not the same as the agent $\alpha_k \in \mathcal{A}$.

Definition 60. (Reachability by a sequence of agents) A state s is reachable in $\mathcal{M}$ by a sequence of agents $\varpi = (\alpha_{(1)}, ..., \alpha_{(m)})$ of length $m+1$ (agents in ϖ can repeat and an agent can perform zero actions or a no-op action ϵ) if a sequence of $u_1^{(i)}$–$u_{k_i}^{(i)}$-plans $(\bar{\pi}_1, ..., \bar{\pi}_m)$ exists such that each $\bar{\pi}_i$ contains only actions from $A_{(i)}$, $\bar{\pi}_1$ is applicable in u_I and for each i s.t. $1 \leq i \leq m$, $u_{k_{i-1}}^{(i-1)} \stackrel{\mathsf{G}}{=} u_1^{(i)}$.

Informally, in the sequence of agents, each agent performs a sequence of actions, such that the final resulting state is s.

The following lemma uses either the assumption that the used heuristics are safe, or requires the states evaluated as dead ends to be placed in the open-list nonetheless, with the heuristic value of ∞. This is necessary to make sure that a reachable state is never unreached only because of the heuristic evaluation.

Lemma 61. *If a state s is reachable in $\mathcal{M}$ by the sequence of agents $\varpi = (\alpha_{(1)}, ..., \alpha_{(m)})$ of length m, it is placed into the closed list $C^{(m)}$ after a finite number of steps.*

Proof. We will show the proof by induction in the number of agents m.

If a state s is reachable by a sequence containing single agent $\varpi = (\alpha_i)$, then assume that no search node u such that $s = u.\mathtt{state}$ is ever placed in C^i. Since s is reachable, based on Lemma 59 there exists a search node u s.t. $u.\mathtt{state} = s$ and $\bar{u} = (u_I, ..., u)$. Let u_m be the first search node in $\bar{u}$, s.t. u_m is not added to C^i. Note that there exists an action $a \in A_i$, such that $u_m = u_{m-1} \circ a$. Since at some point $u_{m-1} \in C^i$, u_{m-1} must have been taken from O^i_{L} or O^i_{D}. At that point, u_{m-1} was also expanded and because a is applicable in $u_{m-1}.\mathtt{state}$ it must have been applied. The resulting node $u_m = u_{m-1} \circ a$ was added to either O^i_{L} or O^i_{D} and because of Lemma 56, u_m was eventually extracted and added to C^i. This contradicts the assumption that u_m is not added to C^i.

Let us now assume that for all k s.t. $k \leq m$ if a node is reachable by a sequence of agents $\varpi = (\alpha_{(1)}, ..., \alpha_{(k)})$ of length k, it is added to $C^{(k)}$ after finite many steps. We will show that the same holds if a node is reachable by a sequence of agents $\varpi' = (\alpha_{(1)}, ..., \alpha_{(m)}, \alpha_{(m+1)})$ of length $m+1$. We will show the induction step by a contradiction. For the contradiction let us assume that s is a state reachable by a sequence of agents ϖ' of length $m+1$, but no search node u such that $s = u.\mathtt{state}$ is ever added to $C^{(m+1)}$. Let $\bar{u} = (u_1, ..., u)$ and let u_l be the first node that is never added to $C^{(m+1)}$. One of the following holds:

i) u_l is reachable by agent α_{m+1}. If so, the same reasoning used in the first part of the proof can be used to obtain a contradiction.

ii) u_l is reachable by some $k' < m+1$ agents $\varpi'' = (\alpha_{(1)}, ..., \alpha_{(k')})$. In that case, we have a contradiction with the assumption of the induction.

□

Now we can conclude the proof by the following theorem.

Theorem 62. *(Completeness of the MADLA Search) The MADLA Search is complete.*

Proof. In the MADLA$^+$ Search, for every state s reachable in $\mathcal{M}$ by an agent α_i such that $s \subseteq G$, a search node u such that $s = u.\mathtt{state}$ is eventually placed into C^i (Lemma 61). The first such search node is, after adding it to the closed list C^i, given to the `reconstructPlan()` procedure which reports a valid multi-agent plan (as MADLA Search is sound, Theorem 54). □

5.4.3 Projected and Privacy-Preserving Set-Additive FF in the MADLA Search

The MADLA Search places a number of assumptions on the properties of the pair of heuristics it operates with. Here, we examine, how these assumptions hold for projected FF used as the local heuristic h^i_{L} and Privacy-Preserving Set-Additive FF (Section 4.3) used as the global heuristic h^i_{D}. Let us recall and expand upon the assumptions, where (i) is a required property and (ii) and (iii) are desirable properties:

i) The distributed heuristic h^i_{D} is non-blocking. This is a required property, meaning that the distributed heuristic allows the agent computing it to run other computations meanwhile (all in a single computational process). The motivation behind this is the expectation that a distributed heuristic will communicate with other agents and while waiting for replies, the search will continue using the local heuristic.

ii) The distributed heuristic h^i_{D} is better informed than the local heuristic h^i_{L}, i.e., a single-heuristic search using h^i_{D} expands fewer states than the same search using h^i_{L}. This assumption is not strict, but using a less informed distributed heuristic gives no advantage. Although it cannot be guaranteed in general, it can be expected that for most states the dominance holds.

iii) The local heuristic h^i_{L} is easier to compute than the distributed heuristic h^i_{D}. Again, this assumption is not strict. Let us assume unit-time atomic computational steps used by both heuristic estimators (i.e., procedures computing the heuristic functions), in that sense, h^i_{L} is assumed to take fewer steps than h^i_{D} for a given state s. As the assumption is not strict, it suffices to hold for a significant number of states. The reasoning behind this assumption is that in conjunction with the assumption (ii) it does not make sense to use a less informed local heuristic which takes longer time to compute.

Let us have a look how the proposed projected FF and distributed FF algorithms adhere to assumptions (i)–(iii). As addressed in the description of Algorithm 6, the loop on lines 7–10 is actually implemented as an asynchronous event-based message processing, which means that after all messages are sent, another computation can proceed until some reply message is received. This can be utilized by the MADLA Search as shown in Algorithm 9, where on line 15 the `processNode` (α_i, u, d) is called with the parameter $d =$ `true` and thus Algorithm 10 proceeds with the asynchronous call to the distributed heuristic on line 5 and the search can continue with the local search loop on lines 17–21 in Algorithm 9. Notice that the communication is processed also inside the local search loop, thus if a reply message is received, the local search loop is exited as the $\mathsf{busy}_{\mathsf{D}}$ flag is set to `true`. This behavior assures that the ppsaFF heuristic adheres to the Assumption (i).

To assess the assumptions (ii) and (iii) we first observe that the initial phase of the distributed heuristic computation as shown in Algorithm 6 is to compute the projected relaxed plan (line 3), which is exactly how the projected FF is computed. After that, some additional steps are performed, in order to improve the quality of the relaxed plan estimation. This means that the number of computational steps performed by the distributed FF is always at least as high as the number of steps performed by the projected FF or higher, thus confirming the Assumption (iii).

For similar reasons as above, it can be expected that the informativeness of the distributed heuristic would be higher as information is only added, nevertheless, the informativeness of a heuristic is best assessed by an experimental evaluation. Here we refer to the Section 4.4.3 where the number of expanded states is compared.

5.5 Evaluation

We evaluate the MADLA Search implemented in a planning system called the MADLA Planner[2] [Štolba and Komenda, 2015, 2017] using a combination of local projected FF heuristic and the Privacy-Preserving Set-Additive FF heuristic described in-detail in Section 4.3. The MADLA Planner is a distributed multi-agent system, where the agents communicate over a TCP/IP connection, even if running on a single physical machine.

As the core comparison metrics, we use coverage, that is the number of solved problems under 20 minutes with 8GB total memory limit (the memory is shared among all agents, assigned on an as needed basis until exhausted). In the detailed analysis, we use the number of expanded states (and their ratios for

[2] http://github.com/stolba/ MADLAPlanner.

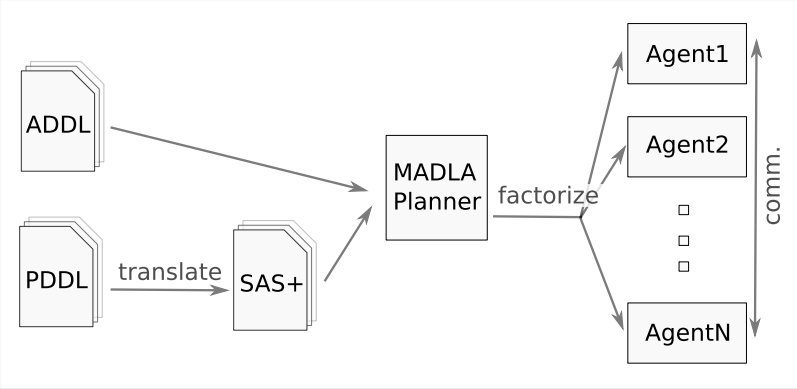


Figure 5.5.1: Stages of the MADLA Planner.

the used heuristics), the computation time of each particular heuristic and the number of public actions requiring a (private) action supporter for a private precondition fact. The 20 minute time limit used for coverage timeout represents a make-span of the distributed process (that is, the time between the earliest time any agent starts computing and the latest time any agent finished) and thus we used wall-clock time, as it is not viable to deduce make-span, including communication, based on CPU time of the individual processes. For the same reasons, we have used wall-clock time also for all other time-related measurements. We are aware, that wall-clock time can be influenced by external sources, such as other processes running on the system. We attempted to mitigate such effects by running the experiments on dedicated machines. Moreover, the non-determinism of the planning process is inherent and impossible to be synchronized under the assumption of unknown ordering in message delivery from two different agents to one recipient. Considering the non-determinism and the unlikely, but possible, interference of other system processes, every measurement was repeated 10 times[3] and the results were averaged. Each machine was equipped with 8 hyper-threading[4] i7 cores (i.e., 16 threads) at 2.6GHz. Each agent was running on two threads. One thread is receiving messages and filling in content data structures in appropriate collections (e.g., states into the open lists $O^i_{\mathsf{L}}, O^i_{\mathsf{D}}$) and the other thread is searching and evaluating the heuristics.

5.5.1 Implementation of the MADLA Planner

The MADLA Planner proceeds in a number of stages (similarly to many classical planners), shown in Figure 5.5.1. The input of the planner is a description of the domain and problem in classical PDDL, accompanied with a file in a novel format Agent Domain Description Language (ADDL). The ADDL file lists which objects in the PDDL represent agents.

The first stage is the translation of the PDDL input into SAS+ format, which is used internally by the planner. The translation is performed by the standard tool in the Fast-Downward Planning System [Helmert, 2006] and thus is centralized (we leave the distribution of the translation process for future work). Next, the bootstrapping part of the MADLA Planner takes the SAS+ and ADDL inputs and partitions the problem for the specified agents.

The partitioning process starts with partitioning of (grounded) actions. A grounded action a is assigned to an agent α_i if α_i (its respective PDDL object) is the first parameter of a mentioned in

[3] Although 10 samples is not enough for a reasonable statistical confidence, it helps to identify cases with extreme variance. This phenomenon did not manifest in any of our experiments, therefore we concluded the planner is deterministic enough for the used metrics.

[4] Although hyper-threading may affect the measurement of CPU-time, as we use solely wall-clock time, the experiments were not affected by hyper-threading.

the ADDL input file representing an agent. Next step is partitioning of the variables and values and determining which of them are public (i.e. those shared among multiple agents).

Alternatively, the MADLA Planner accepts the unfactored MA-PDDL (see Appendix A) input format introduced for the Competition of Distributed and Multi-Agent Planners [Komenda et al., 2016]. The described process is used also for MA-PDDL with the exception that the first step is to extract the agents from MA-PDDL and translate MA-PDDL into PDDL and ADDL. Subsequently, the factorization process continues as described regardless of the factorization and privacy definition of MA-PDDL based on the MA-STRIPS definition. This means, that the problem solved may differ from the problem defined in MA-PDDL, but the MA-PDDL problems can be defined so that they adhere to the MA-STRIPS rules as was the case in the CoDMAP competition. In such situation the MADLA Planner solves the same problem as defined.

After the partitioning step, each agent is initialized and run in a separate process. From now on, the planning is distributed, each agent is running on its own thread and communicating with other agents via message passing over the TCP/IP protocol. Detecting the nonexistence of a solution in the distributed setting is nontrivial (using the distributed snapshot algorithm) and introduces communication overheads and thus we currently resort to a time limit instead, which is not a problem in practice, as the planner is typically run within a time limit anyway. When a solution is found, we use a variant of the plan reconstruction process slightly more efficient than the described one. The main difference is that if a plan of length l is being reconstructed, the reconstruction process of any plan with length l' such that $l' > l$ is terminated as soon as it is detected (e.g., when the reconstruction message is received).

Also, in the implementation, we use separate closed lists C^i_{D} and C^i_{L} for the states evaluated using the distributed and the local heuristic respectively, but we are using only a single closed list C^i for the simplicity of the presentation. The use of the separate closed lists improves the coverage of MADLA Search by 2.9% and the use of local heuristic re-computation when a state is received by 3.2% (see the next sections). All the theoretical results hold also when using two closed lists as eventually, all states will be in both closed lists.

The last implementation detail we present here is related to the non-blocking property of the distributed heuristic (Section 4.3). The heuristic computation is not ideally non-blocking, as parts of the heuristic are computed locally in the agent's thread and thus do not allow other computation to run at the same time. Such situation is namely when the heuristic is initiated and computes local relaxed plan. Another such situation occurs when the initiator agent needs to satisfy private preconditions of its own public action. The experimental evaluation shows, that in some domains, this violation of the non-blocking property degrades the efficiency of the MADLA Planner.

5.5.2 Comparison of the Building Blocks

The building blocks of the MADLA Planner are the projected and distributed FF heuristics and the scheme that combines these in a search. A baseline approach (as we presented in Section 5.1.2) is to adapt a classical multi-heuristic (MH) search for multi-agent planning without the non-blocking MADLA principle with a simple alternation mechanism of the open lists respective to the projected and distributed FF heuristics. However, this approach is not viable as the heuristics are not "orthogonal". The proposed MADLA Search utilizes the requirement for a non-blocking distributed heuristic estimator (particularly implemented in the form of the Privacy-Preserving Set-Additive FF heuristic, see Section 4.3) by running projected FF in the spare time, therefore in contrast to the MH search utilizing the waiting times for computation of the projected heuristic. Table 5.1 summarizes the coverage of the projected FF (projFF) and Privacy-Preserving Set-Additive FF (ppsaFF) heuristics in separate multi-agent single-heuristic search (see Section 5.1.3), in the classical multi-heuristic search (Section 5.1.2) naively modified for multi-agent planning and in the MADLA Search (Section 5.2) with both heuristics.

The results clearly indicate (as expected) that the baseline multi-heuristic approach is not suitable for the pair of projected and distributed FF heuristics. The summed up coverage results are similar for the single-heuristic searches as the price for better estimates of the distributed variant of FF than projected

domain	$\|\mathcal{A}\|$	projFF	ppsaFF	MH search	MADLA Search
blocksworld (35)	4	32.9	**35**	15	34.1
depot (20)	5-12	10.3	10.5	7	**10.8**
driverlog (20)	2-8	**17.2**	14	13	**17.2**
elevators08 (30)	4-5	17.5	**29**	2	27.7
logistics00 (20)	3-7	**20**	**20**	3	**20**
openstacks (30)	2	13.6	14.9	15.5	**20**
rovers (20)	1-8	**20**	**20**	6.6	**20**
satellites (20)	1-5	**20**	**20**	6	19.8
woodworking08 (30)	7	**8.8**	4.8	5.6	7.5
zenotravel (20)	1-5	**19**	16.9	8	**19**
total (245)		179.3	185.1	81.7	**196.1**

Table 5.1: Average coverage of the building blocks. Number of problems in a domain are in the brackets, $|\mathcal{A}|$ denotes the number (interval) of agents in the problems. The best results are emphasized.

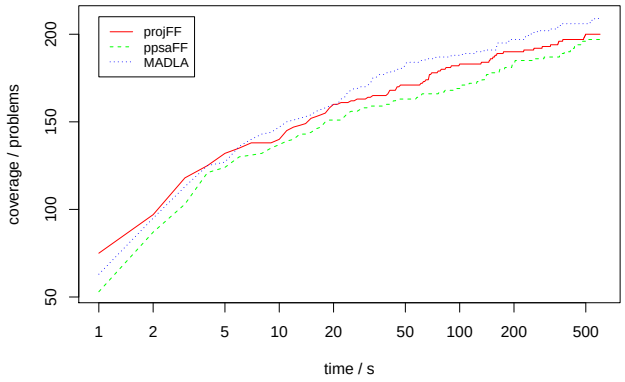


Figure 5.5.2: Variation of coverage depending on time.

FF is its slower computation (see Section 4.4 for more detailed comparison of different variants of inadmissible heuristic computation). Driverlog, woodworking08, and zenotravel are domains where projected FF is performing substantially better than the distributed variant. In such cases, where the FF heuristic is not appropriate or the overhead of its distributed computation overweights the fact the heuristic is more informative, the search using only the projected FF performs better.

The openstacks domain is the only case where even the classical multi-heuristic search outperforms both single-heuristic searches. The efficiency of the FF heuristic in the openstacks problems differs even between the projected and distributed heuristics, therefore their combination outperforms each one separately (see Section 5.5.3 for more details) even with the alternating scheme. Elevators08 and woodworking08 are domains where one of the single-heuristic searches outperformed the MADLA Search (with a reasonable difference). In the former case, as the implementation of the MADLA scheme is not completely non-blocking (see Section 5.5.1 for details), the computation of the distributed heuristic can be slowed down by the computation of the local heuristic. In the latter case, as the FF heuristic itself is not guiding the search well, a large number of states needs to be evaluated. The distributed heuristic does not achieve better guidance, only slows down the whole search process and thus the projected heuristic performs slightly better on its own.

	projFF	ppsaFF	MADLA
Expanded states	109	151	96
Plan length	171	154	184
Bytes communicated	161	49	84
Total time	169	136	160

Table 5.2: Total sums of IPC Scores over all domains and problems for additional metrics. IPC Score is calculated per problem as $^{V^*}/_{V}$, where V^* is the best and V the particular value for a configuration. Higher numbers mean better performance with respect to the given metric.

Figure 5.5.2 shows the coverage of MAFS using a single heuristic compared with the MADLA Search using both. The results were obtained by first averaging the runtime over all 10 runs per problem, which results in slightly different final coverage than in Table 5.1 as each problem is counted as 1 even if not solved in some runs. The plot shows very interesting properties. First, it shows that the projected heuristic (projFF) solves more problems in the same time than the distributed heuristic (ppsaFF), this holds for all time limits. Even more interesting result is that the MADLA Search solves fewer problems than pure projFF in very short time limits (approx. less than 10s), because the MADLA Search is slowed down by the distributed heuristic. For longer time limits, the MADLA Search solves more problems than projFF as it is able to harness the benefits of the distributed heuristic.

Table 5.2 lists results of the two heuristics run separately in MAFS and their MADLA combination evaluated using additional metrics of expanded states, plan length, bytes communicated among the agents and total planning time, all in form of the IPC score. For a particular problem, the IPC score is computed as a ratio of the optimal value (or the best value of all configurations if the optimum is not known) V^* and the particular value V, formally $^{V^*}/_{V}$. This means that the best configuration for a given problem gets 1, worse configurations get a value < 1. The number in the table is the sum of the IPC scores over all problems in all domains for each configuration. The table shows that the distributed heuristic ppsaFF expands the least states (highest score) indicating the heuristic is most informed in comparison to projFF and MADLA which both compensate for the informativeness by the speed of computation of projFF, as the total planning time shows. MADLA slightly improves over projFF in the metric of plan length as the distributed more informed heuristic can shorten some parts of the plans. Notice that on its own, the distributed heuristic leads to longer plans, as it overestimates the true cost more often. This illustrates one of the benefits of the heuristic combination in the MADLA Search. The results of communicated bytes demonstrate the fact that the projected heuristic does not communicate at all (only the search messages are counted). On the contrary, the distributed heuristic is communication intensive and MADLA is balancing both.

The proposed MADLA Search scheme improves the overall coverage over both single-heuristic searches and doubles the coverage of the classical multi-heuristic scheme with the same heuristics. It also provides the best quality plans.

5.5.3 Detailed Analysis

In this section, we analyze the performance of the presented building blocks and the MADLA Search[5] in detail. Table 5.3 shows a comparison of various metrics measured using the multi-agent single-heuristic search with projected FF, distributed Privacy-Preserving Set-Additive FF and MADLA Search using both. The first two columns are ratios of coverage and the number of expanded states of the single-heuristic search with projected FF and distributed FF respectively. The next two columns show the percentage of states in MADLA Search expanded using projected FF and the ratio of states expanded using projected FF and distributed FF. The next three columns show the time per state (in mil-

[5] In this section, the MADLA Search does not re-compute the local heuristic on received states. The results slightly differ from the coverage results in Section 5.5.2, but the total difference in coverage is 0.6 problems and thus negligible.

	projFF / ppsaFF		MADLA exp	MADLA t_h [ms] per state			Action ratio	
domain	cvg	exp	$\frac{\text{projFF}}{\text{all}}$	projFF	ppsaFF	$\frac{\text{ppsaFF}}{\text{projFF}}$	$\frac{\text{public}}{\text{all}}$ %	$\frac{\text{PD}}{\text{all}}$ %
blocksworld	0.94	6.4	0.94	0.4	0.7	1.7	100	100
depot	0.98	11.1	0.96	0.7	9.6	8.0	95.7	23
driverlog	1.23	2.4	0.80	0.9	1.1	1.3	91.9	26.7
elevators08	0.60	50.7	0.94	0.3	0.4	1.5	66	66
logistics00	1.00	47.4	0.95	0.2	0.8	6.3	67.4	33.7
openstacks	0.91	12.9	0.67	2.3	1.6	0.9	100	0
rovers	1.00	1.5	0.75	0.4	0.5	1.4	26.1	11.5
satellites	1.00	0.8	0.77	0.3	2.4	9.8	7.2	2.7
woodwork.	1.83	7.2	0.90	2.2	12.2	7.4	99.9	13
zenotravel	1.12	0.3	0.77	0.5	0.9	1.6	20.7	14.1

Table 5.3: Comparison of various metrics. Ratios of coverage and expanded states of MAFS running projFF and ppsaFF, ratio of states expanded in MADLA using projFF to all expanded states, time per state in MADLA spent on evaluation of projFF and ppsaFF and their ratios and ratios of public action to all actions and state discerning (PD) actions to all actions in each domain. All values are averages per domain.

liseconds) the MADLA Search spends on computing the projected and distributed heuristics and the distributed/projected ratio. The last two columns show the average percentage of public and privately-dependent (PD) actions (see Definition 95) in the domain. A PD action is public and has some private preconditions, which are hidden for agents other than the owner of the PD actions and may cause dependencies between PD actions. Ignoring such dependencies in the projected heuristic may significantly influence the quality of estimates.

Now, we analyze the results shown in the Table 5.3 for each of the domains in detail.

blocksworld All actions in the domain are PD actions as they depend on the private state of the hand. This results in a better heuristic guidance of the distributed heuristic, best-first search expanding over $6\times$ more states with the projected heuristic than with the distributed one (column exp in Table 5.3). In the MADLA Search, only 6% of states are expanded using the distributed heuristic (column *MADLA exp*), which is, nevertheless, enough to slightly reduce the coverage of MADLA compared to MAFS with only the distributed heuristic.

depot The distributed heuristic takes approx. $8\times$ longer to evaluate on average (column *MADLA* t_h right in Table 5.3), but its better heuristic guidance ($11\times$more expanded states using the projected heuristic) results in almost equal coverage (Table 5.1). In MADLA, due to the time demanding distributed heuristic computation, about 96% of states are expanded using the projected heuristic (column *MADLA exp* left). But the small number of states expanded using the distributed heuristic improves the coverage of MADLA in comparison to the single-heuristic search using either of the heuristics.

driverlog The distributed heuristic seems to lead the search slightly better (approx. $2\times$ less expanded states, column exp in Table 5.3) and takes approx. $1.3\times$ more time per state. Nonetheless, the coverage of MAFS using the distributed heuristic is worse than using the projected one, which is likely caused by the higher chance of finding the goal with more expanded states (although less informed), especially in the harder problems (column *MADLA* t_h right, column *cvg* and Table 5.1). In MADLA, this is improved by approx 80% of states expanded using the projection (column *MADLA exp* left), which is enough to reach the coverage score of the projected heuristic on its own.

elevators08 The single-heuristic search with a projected heuristic expands over $50\times$ more states than with the distributed one (column exp in Table 5.3) and the distributed heuristic takes on average only approx. $1.5\times$ longer to compute (column *MADLA* t_h right). This results in a difference in problem coverage of over 10 problems in favor of the distributed heuristic (Table 5.1). In MADLA, even though only 6% of states are expanded using the distributed heuristic (column *MADLA exp* left), it is enough to reduce the performance of the single-heuristic search with the distributed heuristic.

openstacks The openstacks domain is the only case, where even the classical multi-heuristic search outperforms both single-heuristic searches. The projected heuristic is in several instances not informed enough, as the two agents need to strongly coordinate the orders. On the other hand, the distributed estimation is in the other instances computationally hard and not leading the search well (see Figure 4.4.4-bottom). There are no state discerning actions. The favorable combination of distributed and projected heuristic (over 30% of states evaluated with the distributed one) improves significantly the coverage of MADLA.

woodworking08 In the single-heuristic search, the distributed heuristic expands approx. $7\times$ fewer states and takes over $7\times$ longer to evaluate per state (columns exp and *MADLA* t_h right in Table 5.3). The MAFS using projected heuristic solves nearly twice as many problems as MAFS using the distributed heuristic. This is probably caused by the dominant effect of the number of expanded states, where a higher number of expanded (albeit worse guidance) leads to a solution more often. The MADLA Search, however, is able to take advantage of projected heuristic by expanding only 10% of states using the distributed one. Combined, MADLA performs nearly as well as MAFS using only the projected heuristic.

zenotravel The zenotravel domain is one of a few domains where the projected heuristic actually offers better guidance resulting also in better coverage of MAFS using only the projected heuristic. Again, MADLA is able to take advantage of that and match the coverage.

Based on the measured values and also on the understanding of the domain, the logistics00 domain is similar to the elevators08 domain, although much easier to solve. The rovers and satellites domains are very loosely coupled domains with only a small portion of public actions and are also easy to solve.

In summary, the distributed heuristic is useful in domains with a high number of public actions with private preconditions, which is a sign of necessary interaction among the agents, not visible to the projected heuristic, or with some crucial information being private (as in woodworking). On the contrary, the projected heuristic performs better on domains where the agents are interchangeable (driverlog, zenotravel). In most cases, the MADLA Search is able to let the better heuristic dominate the search and thus on most domains, MADLA closely matches the better one of the single heuristic approaches. Notice that on some domains, MADLA even improves the coverage over each heuristic used separately (namely depot and openstacks). It is also important to note that in woodworking, both projected and distributed FF heuristics ignore a significant number of dead-ends, thus slowing the search and solving fewer problems.

5.5.4 Comparison with a Centralized Planner

To report on the effects of the multi-agent partitioning (sometimes referred to as factorization for its resemblance with factored planning), we run the benchmark problems on a centralized Greedy Best-First Search with the FF heuristic. In Table 5.4 the results are compared with the MADLA Search with the projected and distributed FF heuristics. The centralized planner is a configuration of the MADLA Planner using the same codebase but no agent factorization in order to have a fair comparison of the techniques. The original FF planner or Fast Downward (FD) would most probably perform better as they are more optimized and use additional techniques such as preferred operators. In order to account

domain	$\|\mathcal{A}\|$	Centralized	MADLA Planner
blocksworld (35)	4	34	**34.1**
depot (20)	5-12	6	**10.8**
driverlog (20)	2-8	**19**	17.2
elevators08 (30)	4-5	17	**27.7**
logistics00 (20)	3-7	**20**	**20**
openstacks (30)	2	9	**20**
rovers (20)	1-8	**20**	**20**
satellites (20)	1-5	**20**	19.8
woodworking08 (30)	7	**8**	7.5
zenotravel (20)	1-5	**19**	**19**
total (245)		172	**196.1**

Table 5.4: Comparison of the MADLA Search with projected and distributed FF heuristics and a centralized search without multi-agent partitioning with centralized FF heuristic. Average coverage is used for MADLA and one coverage value is used for the (deterministic) centralized search.

domain	$\|\mathcal{A}\|$	rdFF	GPPP	PSMM	FMAP	MADLA
blocksworld (35)	4	6.8	3	25	19	**34.1**
depot (20)	5-12	6.2	8	0	6	**10.8**
driverlog (20)	2-8	14	9	13	15	**17.2**
elevators08 (30)	4-5	2.9	16†	4	**30**	27.7
logistics00 (20)	3-7	5.8	**20**	9	10	**20**
openstacks (30)	2	11.7	0‡	**30**	23	20
rovers (20)	1-8	14.7	10	14	19	**20**
satellites (20)	1-5	10.8	16	8	16	**19.8**
woodworking08 (30)	7	5.6	0‡	**25**	22	7.5
zenotravel (20)	1-5	6.1	**20**	17	18	19
total (245)		84.6	102	145	178	**196.1**

Table 5.5: Comparison MADLA and state-of-the-art planners. †In GPPP experiments a version of the domain without action costs was used, consisting of 16 problems. ‡GPPP does not support action costs.

for the MADLA Planner running on 8 cores, the centralized planner was allowed an $8\times$ longer time limit.

The results show substantial differences in the depot, driverlog, elevator08, and openstacks domains. The multi-agent search doubles the efficiency in depot and $1.5\times$ in elevators08, as the agent subproblems are loosely coupled (the same holds for logistics00, rovers, satellites, and zenotravel). Loose coupling results in beneficial decomposition, making the agents' problems significantly smaller but with not much overhead in coordination. The strong efficiency improvement in openstacks is caused by a better informed combination of heuristics and beneficial partitioning of the problem. In driverlog, the most plausible explanation of the coverage drop is the relaxation principle of the FF heuristic provides better estimates without the partitioning. In general, the results follow the results of similar experiments performed with MAD-A* [Nissim and Brafman, 2014].

5.5.5 Comparison with the State of the Art

In Table 5.5, we show a comparison of problems solved by MADLA and four complete and distributed multi-agent planners. The results show that MADLA loses considerably in woodworking08 and openstacks against all planners supporting action costs. These domains contain a substantial number of

dead-ends of which the FF heuristic (especially in the projected form) is oblivious.

We argue that MADLA is not fairly comparable to (a) planners which do not consider multi-agent privacy (ADP [Crosby et al., 2013]), (b) planners incompatible with MA-STRIPS (μ-SATPLAN [Dimopoulos et al., 2012], BRP [Jonsson and Rovatsos, 2011], , and others, see Chapter 2 for details), or (c) optimal planners (MAD-A* [Nissim and Brafman, 2012]). Additionally, we present comparison only with the most efficient planners using a particular paradigm. We do not present detailed comparisons with ADP planner, however on the benchmark set present, ADP outperforms MADLA by more than 28% solved problems. By definition, ADP cannot preserve privacy in the same sense as MADLA in general, as it does not obey the definition of the agents by which MA-STRIPS defines the privacy. Moreover, MADLA has to use one partitioning of the planning problem defined in the input PDDL and ADDL, but ADP targets classical planning benchmarks and is free to partition the problem as it sees fit.

Although the result table does not contain the PMR [Luis and Borrajo, 2014] planner, MADLA outperforms it on the presented benchmark set as well, just because PMR is an incomplete planner as stated in [Borrajo, 2013, Luis and Borrajo, 2014]. PMR solves only problems where each goal fact is solvable by a single agent. Thus it does not solve problems of depot, logistics00, openstacks, and woodworking08 domains. Even if PMR solved all problems of all other domains, MADLA would outperform it by 38%.

Against MAFS running recursive distributed FF [Štolba and Komenda, 2014] (see Section 4.2 for description), MADLA shows more than $2\times$ improvement over all domains with an exception of driverlog and woodworking08, where the improvement is about 20%. Similarly, MADLA outperforms GPPP [Maliah et al., 2014] nearly $2\times$ over many domains and PSMM [Tožička et al., 2014] by 36%. Finally, MADLA solves 16 more problems of the benchmark set in contrast to the top performing multi-agent planner FMAP [Torreño et al., 2014], which correspond to nearly 11% improvement. In comparison to FMAP, MADLA is considerably better on four domains (blocksworld, depot, logistics, satellites) and significantly worse only on two (elevators08, woodworking08).

The MADLA Planner also took part in the Competition of Distributed and Multi-Agent Planners (CoDMAP)[Komenda et al., 2016][6]. Note, that although the MADLA Planner itself was not significantly modified for the competition, one of the best performing planners, MAPlan [Fišer et al., 2015] is built on the same principles. In particular, it uses distributed heuristic search with the privacy-preserving set-additive FF heuristic (Section 4.3), but without the MADLA Search (Section 5.2). See Appendix A for the competition details and results of the MADLA Planner in comparison with other state-of-the-art planners.

5.6 Summary

In this Chapter, we have provided an answer to the question of how to combine local and distributed heuristics **(Objective 2)**. The proposed approach is to use a multi-agent multi-heuristic scheme, the Multi-Agent Distributed and Local Asynchronous (MADLA) Search, where common properties of distributed heuristics are used to improve the efficiency. The most important property is that the distributed heuristic is evaluated asynchronously as it sometimes must wait for replies from other agents. In such situation, the local heuristic can be used for a fast local search. By cleverly sharing the states reached by the distributed and the local search we can improve the overall performance of the search in comparison to a search using each of the heuristics separately, or a standard multi-heuristic search. We have shown such improvement in a detailed experimental evaluation.

[6] http://agents.fel.cvut.cz/codmap

Chapter 6

Distributed Optimal Planning

There has been significantly less work done in optimal MAP than in satisficing MAP which we have considered so far in Chapter 4 and Chapter 5. The only optimal multi-agent planning algorithm for the MA-STRIPS and MA-MPT is MAD-A* [Nissim and Brafman, 2012] which is also implemented in a fully distributed way as one of the search algorithms in MAPlan [Fišer et al., 2015]. In the thesis, we have mostly focused on the distributed computation of admissible heuristics in fulfillment of the **(Objective 1)**.

In MAD-A*, the search is guided by the well known LM-Cut [Helmert and Domshlak, 2009] or Merge&Shrink Helmert et al. [2007] heuristics, computed on the projected problem (Definitions 11 and 32). Projected heuristic computation is a reasonable choice as it preserves admissibility in general. Recall the definition of an i-projected heuristic (Definition 40). An **i-projected heuristic** is a heuristic computed on the i-projected problem $\Pi_i^{\triangleright}$. Now we can state the following theorem:

Theorem 63. *Any admissible heuristic computed on an i-projected problem $\Pi_i^{\triangleright}$ is admissible also for the global problem Π^G.*

Proof. Directly follows from $\Pi_i^{\triangleright}$ being an abstraction of Π^G (Theorem 36). □

A fundamental flaw of computing an admissible heuristic on a projected problem is that the shortest path in the projected problem may be arbitrarily shorter than is the global one, which may result in an arbitrarily bad heuristic estimate. Consider an example, where each agent can achieve the (global) goal g by a sequence of n private actions followed by a single public action (Figure 6.0.1a). In Figure 6.0.1a, $a_1, ..., a_n$ and $a'_1, ..., a'_n$ are private to respective agents and a_g, a'_g are visible to all agents (public). If we consider unit cost actions, the cost of an optimal solution is $n + 1$.

The projected problem looks quite different (Figure 6.0.1b). The agent knows only about the action a'_g of the other agent. Moreover, it does not know about its precondition and considers a'_g applicable in the initial state i. In the projected problem, the cost of an optimal solution is 1. Therefore no admissible heuristic computed on the projected problem can give estimate higher than 1. This example can be

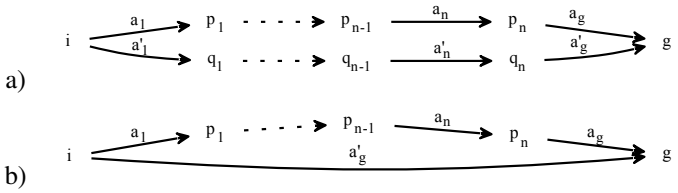


Figure 6.0.1: The full (a) and projected (b) problem.

scaled to any number of agents and enlarged to bound the estimate of the projected heuristic arbitrarily far from the real optimal solution cost.

In this chapter, we solve this problem by providing a number of distributed versions of classical planning heuristics. First, we provide an admissible distributed version of the $h_{\max}$ [Bonet and Geffner, 1999] relaxation heuristic (Section 6.1). A distributed variant of $h_{\max}$ was already described in Section 4.2.1, but without the guarantees on admissibility. Building on $h_{\max}$ we continue with the landmark-based relaxation LM-Cut heuristic (Section 6.2). In Section 6.3 we define desirable additive properties of a heuristic and propose a modification of the MAD-A* search taking advantage of such heuristics. An example of such additive heuristic is presented in Section 6.4, where we venture away from the relaxation heuristics and provide a distributed version of the family of potential heuristics [Pommerening et al., 2015]. We continue the investigation of the idea of additive heuristics for general heuristic computation in Section 6.5 where we focus on multi-agent cost-partitioning.

6.1 Distributed Admissible Max Heuristic

The $h_{\max}$ heuristic belongs to the family of relaxation heuristics. Relaxation heuristics base the estimate on a solution of a relaxed problem Π^+, in which all delete effects are ignored (for all actions, $a^+ = \langle \mathsf{pre}(a), \mathsf{add}(a), \emptyset \rangle$). Although the cost of an optimal solution to Π^+ is an admissible estimate, it is NP-hard to compute, thus various approximations are used instead. In the following, we rephrase one of the first such estimates, the $h_{\max}$ heuristic, and provide a method for its distributed computation. A distributed $h_{\max}$ published in [Štolba and Komenda, 2014] was already described in Section 4.2.1, but without provable equality with the centralized version and also with no guarantees of admissibility, as is required here.

Let $O(p)$ be a set of actions achieving p in Π^{G}, formally

$$
\begin{aligned}
O(p) &= \{a \in A | p \in \mathsf{add}(a)\} \\
O^{\triangleright i}(p) &= \{a \in A_i^{\triangleright} | p \in \mathsf{add}(a)\} \\
O^{\triangledown i}(p) &= \{a \in A_i^{\triangledown} | p \in \mathsf{add}(a)\}
\end{aligned}
$$

where $O^{\triangleright i}(p)$ is a set of actions achieving p in $\Pi^{\triangleright i}$ and $O^{\triangledown i}(p)$ in $\Pi^{\triangledown i}$.

The $h_{\max}$ heuristic is defined by a set of recursive equations (similarly to h_{add} in Section 4.2.1):

$$
h_{\max}(P, s) = \mathsf{max}_{p \in P}(h_{\max}(p, s)) \tag{6.1.1}
$$

$$
h_{\max}(p, s) = \begin{cases} 0 & p \in s \\ h_{\max}(\arg\min_{a \in O(p)}(h_{\max}(a, s)), s) & \\ & p \notin s \end{cases} \tag{6.1.2}
$$

$$
h_{\max}(a, s) = \mathsf{cost}_a(+) h_{\max}(\mathsf{pre}(a), s), \tag{6.1.3}
$$

where P is a set of facts (goal or action preconditions). Throughout the text, projected $h_{\max}^{\triangleright i}$ is $h_{\max}$ computed on $\Pi^{\triangleright i}$, local $h_{\max}^{\triangledown i}$ is $h_{\max}$ computed on $\Pi^{\triangledown i}$ and distributed $h_{\max}^{\mathsf{G}i}$ is $h_{\max}$ computed on $\mathcal{M}$, but estimating Π^{G}.

Initiator agent is the agent α_i which starts the computation of distributed heuristic (e.g., $h_{\max}$) for a given state s. All other agents $\alpha_{j \neq i}$ participating on the computation will be called participant agents.

6.1.1 Distributed Max Heuristic Algorithm

The distributed $h^{\mathsf{G}i}_{\max}$ algorithm is shown in Algorithm 68. The computation is initiated by agent α_i. The values of $h^{\mathsf{G}i}_{\max}$ are first initialized to the values of the i-projected $h^{\triangleright i}_{\max}$(line 2). The algorithm then steps into a loop. In each iteration, the initiator sends requests to all other agents $\alpha_{j\neq i}$ containing current heuristic values for i-projections of all $a \in A^{\mathsf{pub}}_j$ (and for the facts in s_I).

The participant α_j receives the request and computes heuristic values for all $a \in A^{\triangledown i}_j$ from the received values. Afterwards, α_j sends reply to α_i containing $h^{\triangledown j}_{\max}(a^{\triangledown j}, s)$ for all $a \in A^{\mathsf{pub}}_j$ (in fact only the values which have changed must be sent).

When received (line 6), the $h^{\mathsf{G}i}_{\max}$ values are updated based on the received reply (line 7). If no actions are updated, the loop exits, the algorithm terminates and returns heuristic value for the goal.

Algoritmus 13: Distributed $h^{\mathsf{G}i}_{\max}$ heuristic

```
1 Algorithm computeDistHmax(α_i, s, G)
2     initialize h^{Gi}_max to h^{▷i}_max for all p ∈ P_i
3     while h^{Gi}_max(a, s) changed for some a ∈ A^▷_i do
4         for each agent α_j ∈ A\{α_i} do
5             send h^{Gi}_max(a^{▷i}, s) for all a ∈ A^{pub}_j to α_j
6             receive h^{▽j}_max(a^{▷i}, s) for all a ∈ A^{pub}_j
7             update h^{Gi}_max
8     return h^{Gi}_max(G, s)
```

6.1.2 Equality of Centralized and Global Max Heuristic

In this section we show that the distributed $h^{\mathsf{G}i}_{\max}$ algorithm returns the same value as the centralized $h_{\max}$ for any fact or action in any state. In summary, the proof proceeds as follows. First, a relation between the i-projected and distributed $h^{\mathsf{G}i}_{\max}$ is established, stating that $h^{\triangleright i}_{\max}(p, s) \leq h_{\max}(p, s)$ for all $p \in P_i$ (see Lemma 64). This is necessary because in the algorithm, all facts are initialized to the i-projected $h^{\triangleright i}_{\max}$ values and iteratively refined until the global $h_{\max}$ values are reached.

Computing the $h^{\triangledown j}_{\max}$ updates based on the public action and the initial state is shown to be sufficient (see Lemma 66). As a consequence, each $p \in P_i$ such that $h^{\mathsf{G}i}_{\max}(p, s) < h_{\max}(p, s)$ is updated to $h^{\mathsf{G}i}_{\max}(p, s) = h_{\max}(p, s)$ after finitely many steps (see Lemma 67).

From this fact and from the relation of i-projected and distributed $h^{\mathsf{G}i}_{\max}$ follows that eventually all facts are updated to the desired value of centralized $h_{\max}$ (see Theorems 68 and 69).

Lemma 64. *For each state s and fact $p \in P_i$, $h^{\triangleright i}_{\max}(p, s) \leq h_{\max}(p, s)$.*

Proof. Let $p \in P = \bigcup_{i=1}^{n} P_i$. For induction assume that for all preconditions of actions achieving p the lemma holds. Because $p \in P_i$ for some i, the set of achievers in the projected problem contains actions $a \in A_i$, or projections. For the former $h^{\triangleright i}_{\max}(a, s) = h_{\max}(a, s)$ holds trivially, for the latter, a projected action has less or equal number of preconditions and because the assumption holds for all the preconditions, $h^{\triangleright i}_{\max}(a^{\triangleright i}, s) \leq h_{\max}(a, s)$. Because in the set of projected achievers are the same actions as in the global set or their projections (with lower or equal heuristic value), the $\arg\min$ function in the $h_{\max}$ equation (Eq. 6.1.2) gives a lower or equal number and because the cost of an action and its projection is the same, $h^{\triangleright i}_{\max}(p, s) \leq h_{\max}(p, s)$. □

To correctly update a fact or action value in the j-local $h^{\triangledown j}_{\max}$ it is enough to provide correct $h_{\max}$ values for all preceding public actions of agent α_j.

Definition 65. An action a is **preceded** by action a' iff $\mathsf{add}(a') \cap \mathsf{pre}(a) \neq 0$ or a'' exists such that a is preceded by a'' and a'' is preceded by a'. We say that a is **succeeding** a'.

Lemma 66. *If $h^{\nabla i}_{\max}(a^{\nabla i}, s) = h_{\max}(a, s)$ for all $a \in A^{\mathsf{pub}}_i$ preceding a', then $h^{\nabla i}_{\max}(a'^{\nabla i}, s) = h_{\max}(a', s)$.*

Proof. If $a^{\nabla i}$ is preceded only by actions in A^{priv}_i the lemma holds trivially. Similarly if the precondition p maximizing $h_{\max}$ is an effect of such $a^{\nabla i}$. If the maximizing precondition p is an effect of some $a' \in A^{\mathsf{pub}}_i$ the lemma holds because of its assumption and because a' is preceding a.

□

To conclude the proof of equality of $h^{\mathsf{G}i}_{\max}$ and $h_{\max}$ we show that each fact with incorrect value is eventually updated and that the algorithm terminates with the desired values for all facts (and all actions).

Lemma 67. *Each $p \in P_i$ such that $h^{\mathsf{G}i}_{\max}(p, s) < h(p, s)$ is updated to $h^{\mathsf{G}i}_{\max}(p, s) = h(p, s)$ after finitely many steps.*

Proof. Distributed $h^{\mathsf{G}i}_{\max}$ is initialized to $h^{\triangleright i}_{\max}$ for all facts and actions. As already shown in the proof of Lemma 64, if $h^{\triangleright i}_{\max}(a, s) \leq h_{\max}(a, s)$ holds for some action $a \in A_i$, it is caused by some projected action preceding a and missing the (private) precondition maximizing $h_{\max}$. Let $a^{\triangleright i}_0$ (where $a_0 \in A_{j\neq i}$) be such a projected action preceding a for which $h^{\mathsf{G}i}_{\max}(a^{\triangleright i}_0, s) \leq h_{\max}(a_0, s)$. Let for all $p \in \mathsf{pre}(a^{\triangleright i}_0)$ hold $h^{\mathsf{G}i}_{\max}(p, s) = h_{\max}(p, s)$, which means that all actions preceding $a^{\triangleright i}_0$ already have $h^{\mathsf{G}i}_{\max}$ equal to $h_{\max}$. Such action a_0 always exists, because as $h^{\mathsf{G}i}_{\max}$ is initialized to $h^{\triangleright i}_{\max}$, all actions applicable in s_I or preceded only by actions in A_i have already $h^{\mathsf{G}i}_{\max}$ equal to $h_{\max}$.

The inequality $h^{\mathsf{G}i}_{\max}(a^{\triangleright i}_0, s) \leq h_{\max}(a_0, s)$ is caused by a fact $p_m \in \mathsf{pre}(a_0) \backslash P_i$ maximizing $h_{\max}(a_0, s)$. The action a_0 is sent alongside all other actions in A^{pub}_j to agent α_j in order to obtain an update. Agent α_j computes the updated heuristic for all actions from the local problem $\Pi^{\nabla i}$ and sends the information back.

From Lemma 66 and because we assumed that, for all actions $a' \in A^{\mathsf{pub}}_j$ preceding a_0, the equality $h^{\mathsf{G}i}_{\max}(a'^{\triangleright i}, s) = h_{\max}(a', s)$ holds, it holds also for the returned value of a_0. Subsequently, $h^{\mathsf{G}i}_{\max}$ is updated so that $h^{\mathsf{G}i}_{\max}(a^{\triangleright i}_0, s) = h_{\max}(a_0, s)$. In the next iteration, for some other action a_1 preceding a holds $h^{\mathsf{G}i}_{\max}(a^{\triangleright 1}_1, s) \leq h_{\max}(a_1, s)$ while for all actions preceding a_1 the equality holds. The same reasoning can be applied to a_1. Because there is only a finite number of actions and in each iteration one of the actions is updated to the correct value, action a is also updated after finitely many steps. □

Theorem 68. *(Equality of distributed and centralized $h_{\max}$) Algorithm 13 terminates with $h^{\mathsf{G}i}_{\max}(p, s) = h_{\max}(p, s)$ for any given state s and all $p \in P_i$.*

Proof. For each $a \in A^{\triangleright}_i$, when $h^{\mathsf{G}i}_{\max}(p, s) = h_{\max}(p, s)$, the heuristic value for fact p is never changed again. Due to the finite number of facts in a problem and Lemma 67, all facts are updated to $h^{\mathsf{G}i}_{\max}(p, s) = h_{\max}(p, s)$ after a finite number of iterations. After that, no fact and therefore no action is updated and the algorithm terminates. □

In the next sections, the heuristic values computed by participant agents $\alpha_{j\neq i}$, that is the $h^{\nabla j}_{\max}(p, s)$ for the given state s and all facts $p \in P^{\mathsf{priv}}_j$, will be preserved throughout the computation. For $h^{\nabla j}_{\max}$, the equality holds as well.

Theorem 69. *(Equality of i-private and centralized $h_{\max}$) Algorithm 13 terminates with $h^{\nabla j}_{\max}(p, s) = h_{\max}(p, s)$ for all $j \neq i$, $p \in P_j$ and any given state s.*

Proof. From Theorem 68 the equality of $h^{\mathsf{G}i}_{\max}$ and $h_{\max}$ holds for all fact and thus for all actions. It holds also for all projections of public actions $a \in A^{\mathsf{pub}}_j$. From Lemma 66, $h^{\nabla j}_{\max}(a^{\nabla j}, s) = h_{\max}(a, s)$ for all $a \in A_j$ (and for all $p \in P_j$).

□

6.2 Distributed Admissible Landmark Heuristic

One of the best approximations of the optimal relaxation heuristic h^+ known up to date is the LM-Cut heuristic [Helmert and Domshlak, 2009] which computes the estimate by iteratively searching for landmark actions and updating the cost function. To do so the $h_{\max}$ heuristic is computed in each iteration.

The LM-Cut heuristic (denoted as $h_{\mathsf{LM\text{-}Cut}}$, shown in Algorithm 14) provides an admissible estimate of the optimal plan for a relaxed problem Π^+ by utilizing the idea of *disjunctive action landmarks*. Here, we present a distributed version $h^{\mathsf{G}}_{\mathsf{LM\text{-}Cut}}$ for which we show $h^{\mathsf{G}}_{\mathsf{LM\text{-}Cut}}(s) = h_{\mathsf{LM\text{-}Cut}}(s)$ for any state s.

6.2.1 The LM-Cut Heuristic

Algoritmus 14: LM-Cut Heuristic

1. Compute $h^k_{\max}$ based on cost_k for every $p \in P$. If $h^k_{\max}(\mathsf{g}) = 0$ terminate and return $h_{\mathsf{LM\text{-}Cut}}$.
2. Construct a justification graph J_k
3. Construct a disjunctive landmark L_k
   (a) Find all facts p s.t. g is 0-reachable from p, add p to V^*_k
   (b) Find all facts reachable from i without visiting a fact in V^*_k
      i. If an edge cross to V^*_k, add its label to L_k
4. Let $c_{k+1}(a) = \begin{cases} \mathsf{cost}_k(a) & a \notin L_k \\ \mathsf{cost}_k(a) - \mathsf{cost}^{\mathsf{lm}}_k(L_k) & a \in L_k \end{cases}$
5. Continue with Step 1. for $k = k + 1$

Definition 70. Disjunctive action landmark (or landmark) is a set of actions $L \subseteq A$ s.t. each plan must contain at least one $a \in L$. Cost of landmark L is $c^{\mathsf{lm}}(L) = \mathsf{min}_{a \in L} c(a)$, where $c(a)$ is the cost of action a.

The $h_{\mathsf{LM\text{-}Cut}}$ heuristic is obtained from a sequence $\{(L_k, \mathsf{cost}_k)\}^m_{k=0}$ of landmarks and cost functions, $h_{\mathsf{LM\text{-}Cut}} = c^{\mathsf{lm}}_0(L_0) + c^{\mathsf{lm}}_1(L_1) + ... + c^{\mathsf{lm}}_m(L_m)$. Initially, $c_0 = c$ and in each iteration k, landmark L_k is computed using cost_k and a new cost function c_{k+1} is determined (Algorithm 14, Step 4.).

We assume that there is a single fact i representing the initial state and a single fact g representing the goal. If it is not the case, the problem can be transformed by adding zero-cost action a_p for each $p \in s_I$ s.t. $\mathsf{pre}(a_p) = \{\mathsf{i}\}$ and $\mathsf{add}(a_p) = \{p\}$. The goal G can be treated analogously. Moreover, we assume that each action has at least one precondition and one effect, again, general problem can be transformed by setting i as precondition of actions for which $\mathsf{pre}(a) = \emptyset$ and adding a dummy effect $\perp$ for actions for which $\mathsf{add}(a) = \emptyset$.

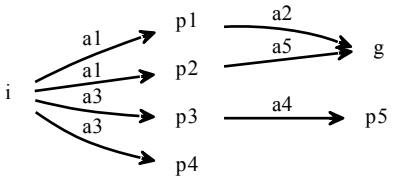
Example. (LM-Cut) The algorithm will be illustrated on a STRIPS example with a set of 5 actions $\{a_1, ..., a_5\}$ (later, in a MA-STRIPS formulation, the actions will be divided among two agents α_1, α_2), the actions are denoted as $a : \mathsf{pre}(a) \to \mathsf{add}(a)$ and have no delete effects:

A_1 : $a_1 : \mathsf{i} \to p_1, p_2$ $a_2 : p_1, p_4 \to \mathsf{g}$
A_2 : $a_3 : \mathsf{i} \to p_3, p_4$ $a_4 : p_3 \to p_5$ $a_5 : p_2, p_5 \to \mathsf{g}$

The facts are $\{\mathsf{i}, \mathsf{g}, p_1, ..., p_5\}$, where i is the initial fact and g is the goal fact. The initial cost function $\mathsf{cost}_0 = \mathsf{cost}$ is defined as $\mathsf{cost}(a_1) = 3$, $\mathsf{cost}(a_2) = 1$, $\mathsf{cost}(a_3) = 1$, $\mathsf{cost}(a_4) = 1$ and $\mathsf{cost}(a_5) = 1$.

In **Step 1** the $h_{\max}$ heuristic is computed for all facts based on cost_0, that is $h_{\max} = \{p_1 : 3, p_2 : 3, p_3 : 1, p_4 : 1, p_5 : 2, \mathsf{g} : 3\}$.

In **Step 2** a justification graph J_0 is constructed. A **Justification graph** J is a directed graph with a vertex for each $p \in P$ and an edge (p, q) labeled a if there exists an action a s.t. $p = pcf(a)$ and $q \in \mathsf{add}(a)$. Function pcf (a precondition choice function) assigns to a given action a one of its preconditions. In $h_{\mathsf{LM\text{-}Cut}}$, the pcf assigns a precondition maximizing $h_{\max}$, ties broken arbitrarily. In the example, $pcf = \{a_1 \mapsto \mathsf{i}, a_2 \mapsto p_1, a_3 \mapsto \mathsf{i}, a_4 \mapsto p_3, a_5 \mapsto p_2\}$, resulting in J_0:



In **Step 3** the landmark L_k is constructed. **(a)** All facts p from which the goal g is reachable through a path on which each edge has a label a s.t. $\mathsf{cost}_k(a) = 0$ (g is **0-reachable** from p) are added to V_k^*. In the iteration $k = 0$ of the example it is $V_0^* = \{g\}$. **(b)** Find all fact reachable from i without visiting any fact from V_k^*. In the example it is all facts except for g. If an edge crossing to V_k^* (that is $e = (p, q)$ and $q \in V_k^*$) is reached, label of the edge is added to L_k. In the example this includes all edges leading to g, resulting in $L_0 = \{a_2, a_5\}$.

In **Step 4** new cost function cost_{k+1} is defined. The costs of all actions in L_k is reduced by the cost of L_k, that is the cost of the least-cost action in L_k. In the example, $\mathsf{cost}_1(a_2) = 0$ and $\mathsf{cost}_2(a_5) = 0$, for all other actions it is the same as cost_0.

The computation continues by Step 1. of iteration $k + 1$, until $h_{\max}(\mathsf{g}) = 0$.

6.2.2 Distributed LM-Cut Heuristic

In the following, we assume that the participant agents $\alpha_{j \neq i}$ keep the result of computation (context) of $h^{\mathsf{G}i}_{\max}$, that is the heuristic values for all $p \in P_j^{\mathsf{priv}}$ and $a \in A_j^{\triangledown}$. Moreover, we will modify the tie-breaking behavior of the pcf function so that if the tie is between a public and private fact, the public fact will be preferred. We assume that the pcf always chooses the same precondition in both $h_{\mathsf{LM\text{-}Cut}}$ and $h^{\mathsf{G}}_{\mathsf{LM\text{-}Cut}}$.

To compute a distributed version of the heuristic we introduce a projected version of landmarks:

Definition 71. An **i-projected disjunctive action landmark** (or i-projected landmark) $L^{\triangleright i}$ corresponding to the disjunctive landmark L is $L^{\triangleright i} = (L \cap A_i^{\triangleright}) \cup \{\bar{a}\}$, where $\bar{a}$ is a place-holder action.

The placeholder action represents the cost of private actions of other agents in L, so when the landmark is completed, $\mathsf{cost}(\bar{a}) = \mathsf{cost}^{\mathsf{lm}}(L)$.

Using the $h^{\mathsf{G}i}_{\max}$ values computed by the distributed algorithm for each fact, the justification graph for the global problem can be reconstructed. The resulting **distributed justification graph** $J^{\mathsf{G}i} = (J^{\triangleright i}, \{J^{\triangledown j}\}_{j \neq i})$ consists of an i-projected justification graph $J^{\triangleright i}$ and a set of j-local justification graphs $\{J^{\triangledown j}\}_{j \neq i}$.

An **i-projected justification graph** $J^{\triangleright i}$ is a justification graph over $P_i \cup \{\bot\}$ with labels from $A_i^{\triangleright}$. The pcf is modified so that for each projected action $a^{\triangleright i}$, $pcf(a^{\triangleright i}) = p$ if $p \in P_i$ maximizes $h^{\mathsf{G}i}_{\max}$ and $h^{\mathsf{G}i}_{\max}(a^{\triangleright i}, s) = h^{\mathsf{G}i}_{\max}(p, s)$, otherwise $pcf(a^{\triangleright i}) = \bot$ if $h^{\mathsf{G}i}_{\max}(a^{\triangleright i}, s) > h^{\mathsf{G}i}_{\max}(p, s)$. This means that the maximizing fact is private to some other agent. If $\mathsf{add}(a^{\triangleright i}) \cap P_i^{\mathsf{pub}} = \emptyset$, we treat the action as if $\mathsf{add}(a^{\triangleright i}) = \{\bot\}$. Edges are not connected via $\bot$. An **i-local justification graph** $J^{\triangledown i}$ is similarly defined over $P_i^{\mathsf{priv}} \cup \{\bot\}$ with labels from $A_i^{\triangledown}$.

The distributed justification graph is a distributed graph where the partitions have pairwise disjunctive sets of vertices. Each edge in $J^{\triangleright i}$ with label containing a projected action $a^{\triangleright i}$ s.t. $a \in A_{j \neq i}$ can

be seen as an edge shared with $J^{\nabla j}$, where the corresponding edge has a label containing the respective $a^{\nabla j}$.

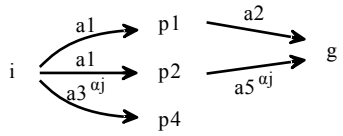
The distributed version of $h_{\mathsf{LM\text{-}Cut}}$, denoted as $h^{\mathsf{G}}_{\mathsf{LM\text{-}Cut}}$, follows the same major steps as the centralized version - it differs in that the computation is distributed in some of the steps.

Example. (LM-Cut) To illustrate we will use the previous example in a MA-STRIPS formulation in which $\{p_2, p_4, \mathsf{g}\}$ are public facts and $\{a_1, a_2, a_3, a_5\}$ public actions.

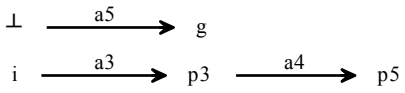
In **Step 1** of the k-th iteration, distributed version of $h^{\mathsf{G}i}_{\max}$ is computed based on the cost function cost_k. The initiator agent α_i computes $h^{\mathsf{G}i}_{\max}$ for all facts in P_i while all other agents $\alpha_{j\neq i}$ compute $h^{\nabla j}_{\max}$ for all facts in P^{priv}_j. The computed values are identical to the values of centralized $h_{\max}$ (Theorems 68 and 69).

In **Step 2** the initiator agent α_i builds an i-projected justification graph $J^{\triangleright i}_k$ based on the values of $h^{\mathsf{G}i,k}_{\max}$ whereas all other agents build j-local justification graph $J^{\nabla j}_k$ based on the values of $h^{\nabla j,k}_{\max}$, together forming a distributed justification graph $J^{\mathsf{G}i}_k$. In our example, the justification graphs are the following:

J^1_0 :



$J^{\nabla 2}_0$:



Notice, that a_5 has $\bot$ as its precondition. This is because p_5 maximizes $h^{\nabla 2,0}_{\max}$ and $h^{\nabla 2,0}_{\max}(a_5, s) = 3 > h^{\nabla 2,0}_{\max}(p_5, s)$=2 (the globally maximizing fact is $p_2 \notin P^{\mathsf{priv}}_2$).

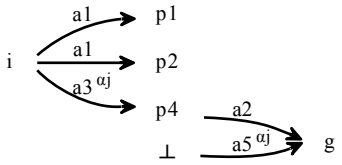
In **Step 3** the i-projected landmark $L^{\triangleright i}_k$ must be determined in a distributed manner. To obtain the same heuristic estimate as in the centralized version, the cost of $L^{\triangleright i}_k$ must be equal to the centralized landmark L_k. This will be achieved by the place-holder action $\bar{a}$. But first, all facts from which is g 0-reachable must be found.

Step 3.1 Find all facts p such that g is 0-reachable from p.

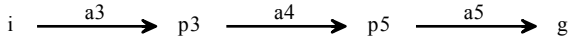
The algorithm starts as in $h_{\mathsf{LM\text{-}Cut}}$ and puts all facts from which g is 0-reachable into $V^*_{k,i}$. As g is public, all actions achieving g are also public and the initiator α_i knows about them (they or their projections are in $A^{\triangleright}_i$), therefore the algorithm can be initiated by α_i. When the algorithm reaches some projected $a^{\triangleright i}$ a request is sent to α_j to determine all facts from which a is 0-reachable. Agent α_j finds all such facts, places them in $V^*_{k,j}$ and sends all such public facts $V^a_k = V^*_{k,j} \cap P^{\mathsf{pub}}_j$ back to α_i. When received, α_i finds all facts p' s.t some $q \in V^a_k$ is 0-reachable from p' and adds p' to $V^*_{k,i}$. This ensures that all facts p from which g is 0-reachable are found and added either to $V^*_{k,i}$ if $p \in P_i$ or to $V^*_{k,j}$ if $p \in P^{\mathsf{priv}}_j$.

Example. (LM-Cut) We illustrate this step on the iteration $k = 2$ of the example, where the cost of actions has already been modified so that $\mathsf{cost}_2(a_1) = 0, \mathsf{cost}_2(a_2) = 0, \mathsf{cost}_2(a_5) = 0$ and $h^{\mathsf{G}1,2}_{\max} = \{p_1 : 0, p_2 : 0, p_3 : 1, p_4 : 1, g : 1\}$ and $h^{\nabla 2,2}_{\max} = \{p_3 : 1, p_5 : 2, \mathsf{g} : 1\}$. In this situation the justification graphs are:

$J^{\triangleright 1}_2$:



$J_2^{\triangledown 2}$:



Since the cost of a_2 and $a_5^{\triangleright 1}$ is 0, p_4 is added to $V_{2,1}^*$ and a request for a_5 is sent to α_2. Agent α_2 starts the reachability analysis from a_5 and finds that a_5 is 0-reachable only from p_5. Because p_5 is internal, the reply $V_2^{a_5}$ is empty, but p_5 is added to $V_{2,2}^*$ and will be used in the next step.

The distributed 0-reachability algorithm ensures, that g is 0-reachable from a fact p in J if and only if it is 0-reachable in $J_2^{\mathsf{G}i}$. When all such facts are stored in respective $V_{k,i}^*$ or $V_{k,j}^*$, the next step of the $h_{\mathsf{LM\text{-}Cut}}^{\mathsf{G}i}$ algorithm can be performed.

Step 3.2 Find all facts reachable from i without visiting a fact in $V_{k,i}^*$ or any $V_{k,j}^*$.

Again, the algorithm starts as in $h_{\mathsf{LM\text{-}Cut}}$. Similarly to the previous case, a fact p may be reachable from i via some agent $\alpha_{j\neq i}$. To find all such facts, it is enough, to find all edges which contain a projected actions a^{α_j} in the label reachable from i and for each such action send a request to α_j. Agent α_j then finds all facts reachable from all $q \in \mathsf{add}(a)$ without visiting any fact in $V_{k,j}^*$. All public actions in labels of edges visited in the process are added to A_k^0 and sent back in reply. When received, agent α_i finds all facts p' reachable from all $i' \in \mathsf{add}(a')$ for all $a' \in A_k^0$ without visiting any fact in $V_{k,i}^*$.

Unlike the previous case, i is not public and therefore additional request has to be sent for the initial fact i. The request and respective reply are handled the same way as in the case of a projected action.

Example. (LM-Cut) Recall, that in the example, iteration $k = 2$, $V_{2,1}^* = \{p_4\}$ and $V_{2,2}^* = \{p_5\}$. The facts reachable in $J_2^{\triangleright 1}$ without visiting p_4 are $\{\mathsf{i}, p_1, p_2\}$. Requests are sent for $a_3^{\triangleright 1}$ and for i. In $J_2^{\triangledown 2}$, the facts reachable without visiting p_5 are $\{\mathsf{i}, p_3\}$.

The distributed reachability algorithm ensures, that a fact p is reachable from i in J_k if and only if it is reachable in $J_k^{\mathsf{G}i}$.

Step 3.3 Find landmarks.

In $h_{\mathsf{LM\text{-}Cut}}$, the purpose of the reachability analysis is to find actions forming the disjunctive landmark L_k. Those are all actions in labels of edges in J_k, starting from a fact reachable from i and ending in fact in V_k^*. The distributed algorithm aims for the same.

In Step 3.2 performed by α_i on $J_k^{\triangleright i}$, action a is added to $L_k^{\triangleright i}$ if a is in a label of edge reachable from i ending in some $p \in V_{k,i}^*$, as in $h_{\mathsf{LM\text{-}Cut}}$. When the reply in Step 3.2 is computed by agent α_j for some requested projected action, landmark actions are added to $L_k^{\triangleright j}$ private to α_j, again as in $h_{\mathsf{LM\text{-}Cut}}$. To capture the cost of private actions (which may possibly be the lowest cost actions), a place-holder action $\bar{a}$ is created and its cost set to $\mathsf{cost}_k(\bar{a}) = \mathsf{cost}_k^{\mathsf{lm}}(L_k^{\triangleright j})$. The public part of the landmark $L_k^{\mathsf{pub}_j} \leftarrow (L_k^{\triangleright j} \cap A_j^{\mathsf{pub}}) \cup \{\bar{a}\}$ is sent in reply alongside the set of reached public actions A_k^0. When received, it is merged with $L_k^{\triangleright i}$ while keeping only the lowest-cost place-holder action $\bar{a}$.

Example. (LM-Cut) In the $k = 0$ iteration of the example, the found landmarks are the following $L_0^{\triangleright 1} = \{a_2, a_5^{\triangleright 1}, \bar{a}\}$ and $L_0^{\triangleright 2} = \{a_5\}$ where $\mathsf{cost}_0(\bar{a}) = \mathsf{cost}_0(a_5) = 1$. In this case $\bar{a}$ has no influence in the cost of $L_0^{\triangleright 1}$ which is 1. In the $k = 2$ iteration, $L_2^{\triangleright 1} = \{a_3^{\triangleright 1}, \bar{a}\}$ and $L_2^{\triangleright 2} = \{a_4\}$ where $\mathsf{cost}_2(\bar{a}) = \mathsf{cost}_2(a_4) = 1$, whereas $\mathsf{cost}(a_3^{\triangleright 1}) = 2$ and the information encoded in $\bar{a}$ is crucial.

In Step 4

of the distributed algorithm the cost function for the next iteration $k + 1$ is constructed. Thanks to the use of place-holder action $\bar{a}$ which stores the cost of the lowest-cost action over all j-local landmarks, the same update formula as in $h_{\mathsf{LM\text{-}Cut}}$ can be used also in the distributed version. The only difference is that when the $\mathsf{cost}_k^{\mathsf{lm}}(L_k^{\triangleright i})$ value is computed it is sent to all participating agents α_j so that the cost of actions in $L_k^{\triangleright j}$ can be locally updated as well.

Example. (LM-Cut) Notice, that in the example iteration $k = 2$ the 2-projected landmark $L_2^{\triangleright 2} = \{a_4\}$ is missing the action a_3. It is not a problem for the computation of the cost of $L_2^{\triangleright 1}$, because it contains $a_3^{\triangleright 1}$, but the cost of a_3 will not be updated. This issue can be handled in various ways, in $h^{\mathsf{G}}_{\mathsf{LM\text{-}Cut}}$ it is handled in the computation of $h^{\mathsf{G}1,3}_{\max}$ where the updated cost of projected actions is sent from the initiator to the participants.

6.2.3 Equality of Centralized and Distributed LM-Cut Heuristic

To show the equality of the centralized $h_{\mathsf{LM\text{-}Cut}}$ and distributed $h^{\mathsf{G}}_{\mathsf{LM\text{-}Cut}}$ heuristic, it is crucial to have the distributed $h^{\mathsf{G}i}_{\max}$ equal to the centralized $h_{\max}$. This has been shown in Theorems 68 and 69. Then, a distributed justification graph has to be constructed, such that a reachability relation is preserved. From the definition of $J^{\mathsf{G}i}$ and the presented algorithms directly follows:

Lemma 72. *Fact q is reachable (0-reachable) from fact p in a justification graph J iff q is reachable (0-reachable) from p in a distributed justification graph $J^{\mathsf{G}i}$.*

Next, we proceed by showing that in each iteration, the union of the set of projected landmarks constructed by the distributed algorithm is equal to the landmark constructed by the centralized algorithm (see Lemma 73) and its cost is equal to the cost of the projected landmark constructed by the initiator agent (see Lemma 74). We conclude the proof by showing that the heuristic estimate obtained by the distributed version is equal to the centralized estimate (see Theorem 75).

Lemma 73. *For each step k, landmark L_k constructed by the centralized algorithm on J_k and landmarks $L_k^{\triangleright 0}, ..., L_k^{\triangleright n}$ constructed by the distributed algorithm on $J_k^{\mathsf{G}i}$ holds $L_k = \bigcup_{j=1}^{n} L_k^{\triangleright j} \backslash \{\bar{a}\}$.*

Proof. In each step k, $V_k^* = \bigcup_{j=1}^{n} V_{k,j}^*$ holds (from Lemma 72). In the centralized search for landmarks, an action a is added to L_k if and only if $p \in \mathsf{add}(a)$ exists s.t. $p \in V_k^*$ and p is reachable from i. From the previously stated, for such p must hold $p \in V_{k,j}^*$ for some j and from Lemma 72, p is reachable from i in $J^{\mathsf{G}i}$. If $p \in P_i$, a is in $A_i^{\triangleright}$ and is added to $L_k^{\triangleright i}$, otherwise, p is in some P_j^{priv} and $a \in A_j^{\triangledown}$ and a is added to $L_k^{\triangleright i}$. Therefore the lemma holds (the place-holder action $\bar{a}$, introduced by the distributed algorithm, is ignored). □

The constructed i-projected landmark represents the cost of the centralized landmark, formally:

Lemma 74. *For each step k, landmark L_k constructed by the centralized algorithm on J and i-projected landmark $L_k^{\triangleright i}$ constructed by the distributed algorithm initiated by agent α_i on $J^{\mathsf{G}i}$ holds $\mathsf{cost}_{\mathsf{lm}}(L_k^{\triangleright i}) = \mathsf{cost}_{\mathsf{lm}}(L_k)$.*

Proof. From proof of Lemma 73, for each $\alpha_{j \neq i}$, $L_k^{\triangleright j} = L_k \cap A_j$. When $L_k^{\triangleright j}$ is finished, the public part $L_k^{\mathsf{pub}} = (L_k^{\triangleright j} \cap A_j^{\mathsf{pub}}) \cup \{\bar{a}\}$ of $L_k^{\triangleright j}$ is sent from α_j to α_i. For the place-holder action $\bar{a}$ holds

$\mathsf{cost}_k(\bar{a}) = \mathsf{cost}_k^{\mathsf{lm}}(L_k^{\rhd j})$. This ensures, that $\mathsf{cost}_k^{\mathsf{lm}}(L_k^{\mathsf{pub}}) = \mathsf{cost}_k^{\mathsf{lm}}(L_k^{\rhd j})$ even if the least-cost action is not public. When L_k^{pub} is received, $L_k^{\rhd i} \leftarrow L_k^{\rhd i} \cup L_k^{\mathsf{pub}}$, retaining the least-cost $\bar{a}$. From the definition of $\mathsf{cost}_k^{\mathsf{lm}}$ follows $\mathsf{cost}_k^{\mathsf{lm}}(L_k^{\rhd i}) = \mathsf{min}(\mathsf{cost}_k^{\mathsf{lm}}(L_k^{\rhd i}), \mathsf{cost}_k^{\mathsf{lm}}(L_k^{\mathsf{pub}}))$. Therefore, when L_k^{pub} is received from all agents $\alpha_{j\neq i}$ and $L_k^{\rhd i}$ is completed, $\mathsf{cost}_k^{\mathsf{lm}}(L_k^{\rhd i}) = \mathsf{min}_{0<j\leq n}(L_k^{\rhd j}) = \mathsf{cost}_k^{\mathsf{lm}}(L_k)$. □

Finally we conclude that:

Theorem 75. *(Equality of centralized and distributed LM-Cut) For any state s and any agent α_i, $h_{\mathsf{LM\text{-}Cut}}(G, s) = h^{\mathsf{G}i}_{\mathsf{LM\text{-}Cut}}(G, s)$.*

Proof. From Theorems 68 and 69 the result of distributed computation of $h^{\mathsf{G}i}_{\max}(G, s)$ is equal to the centralized $h_{\max}(G, s)$ for any state s, any agent α_i and for all facts $p \in P$ and therefore also for all actions $a \in A$. For each step k of the algorithm, a distributed justification graph $J^{\mathsf{G}i} = (J^{\rhd i}, \{J^{\triangledown j}\}_{j\neq i})$ can be constructed such that Lemma 72 holds for reachability and 0-reachability. Also, from Lemma 74 the cost of the projected landmark $L_k^{\rhd i}$ constructed by the distributed algorithm initiated by α_i equals the cost of the landmark L_k constructed in step k by the centralized algorithm. The cost is then shared with all agents $\alpha_{j\neq i}$ and all actions in $L_k^{\rhd i}$ and all $L_k^{\rhd j}$, which are all actions in L_k (from Lemma 73), have their costs updated. Therefore, the updated cost function in the $k+1$ step of the centralized algorithm equals the cost function in the $k+1$ step of the distributed algorithm for all agents and all actions. □

The proof of Theorem 75 concludes the section on the admissible distributed landmark heuristic. By showing, that the distributed heuristic returns values equal to a centralized heuristic computed on the respective global problem, we also show that the heuristic is admissible for the multi-agent problem (trivially by Theorem 21).

6.3 Distributed Search with Additive Heuristics

In the techniques presented in Chapter 4 and in this chapter so far, the distributed computation of heuristic estimates requires cooperation of all (or at least most of) the agents and incurs a substantial amount of additional communication. In many scenarios, the communication may be very costly (multi-robot systems) or prohibited (military) and even on high-speed networks, communication takes significant time compared to local computation. In such cases it may pay off to use the projected heuristic instead of its better informed counterpart. In [Nissim and Brafman, 2014], the authors mention an idea of an additive heuristic such that projected estimates of two agents could be added together and still maintain admissibility. In this section, we take the idea a step further, formalize two variants of additive heuristics, and detail out a modification of the MAD-A* utilizing such heuristics. Moreover, in Section 6.4 and Section 6.5 we present heuristics and distributed computation techniques which satisfy the properties we define here.

Definition 76. (Agent-additive heuristic) A global heuristic h estimating the global problem Π^{G} is agent-additive iff for any agent $\alpha_i \in \mathcal{A}$ it can be represented as

$$h(s) = h^{\mathsf{pub}}(s^{\rhd}) + \sum_{\alpha_j \in \mathcal{A}} h^j(s^{\rhd j})$$

where h^{pub} is a heuristic computed on the public projection problem $\Pi^{\rhd}$ and h^j is a heuristic computed on the j-projected problem $\Pi^{\rhd j}$.

Informally, the definition states that each part of an agent-additive heuristic can be computed by each respective agent separately an then added together. A heuristic is agent-additive even without the public part, that is, if $h^{\mathsf{pub}}(s^{\rhd}) = 0$ for all states.

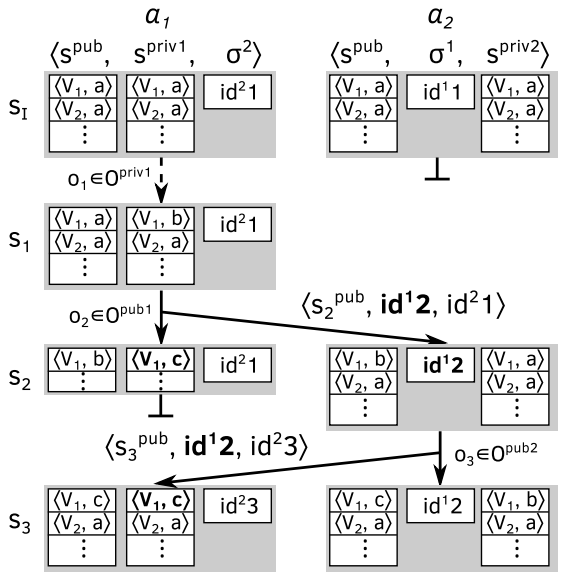


Figure 6.3.1: Search using separate public and private parts. Preserved private information $s_2^{\mathsf{priv}_1}[V_1] = \mathsf{c}$ of agent α_1 in form of identifier $\sigma_2^2 = \mathsf{id}^1 2$ used by α_2 is emphasized.

Definition 77. (Agent-agnostic heuristic) An agent-additive global heuristic

$$h(s) = h^{\mathsf{pub}}(s^{\triangleright}) + \sum_{\alpha_i \in \mathcal{A}} h^i(s^{\triangleright i})$$

is agent-agnostic iff for each two global states s and s', s.t. $s' = s \circ o$, where $o \in \mathcal{O}^i$ of some agent α_i, holds $h^j(s^{\triangleright j}) = h^j(s'^{\triangleright j})$ for all $j \neq i$.

In other words, in an agent-agnostic heuristic, no agent can influence the parts of the heuristic computed by other agents. The principle of the multi-agent heuristic search presented in this section is based on the MAD-A* algorithm (Multi-Agent Distributed A*) [Nissim and Brafman, 2012], which is thoroughly described in Section 6.3, here, we briefly rephrase the main principle using the MA-MPT formalism (Section 3.2). The MAD-A* algorithm is a simple extension of classical A*. The agents search in parallel, possibly in a distributed setting (i.e. communicating over a network). Each agent $\alpha_i \in \mathcal{A}$ searches using its operators from $\mathcal{O}^i$ and if a state s is expanded using a public operator $o \in \mathcal{O}^{\mathsf{pub}_i}$, the resulting state s' is sent to other agents (the agents may be filtered in order to send the state only to the relevant ones). When some other agent α_j receives the state s', s' is added to the OPEN list of α_j and expanded normally when due. The original MAD-A* uses only projected heuristics. Each state sent by α_i is also accompanied with its i-projected heuristic estimate and when received, the receiving agent α_j computes j-projected heuristic estimate of the received state s' and takes $h(s) = \mathsf{max}(h^{\triangleright i}(s^{\triangleright i}), h^{\triangleright j}(s^{\triangleright j}))$.

In MA-MPT, each agent α_i can work only with its set of variables $\mathcal{V}^i$. In order to use the MAD-A* search on the MA-MPT formalism, each search state has all variables private to other agents $\alpha_{j \neq i}$ replaced by a unique identifier σ^j. This identifier refers to the last state on the search path modified by agent α_j. No other agent can reconstruct the private part from the identifier.

The search process is illustrated in Figure 6.3.1. When an agent receives a state from another agent, it uses this identifier to retrieve the proper private part. Formally, agent α_i internally represents state s as a tuple $\left\langle s^{\mathsf{pub}}, \sigma^1, ..., s^{\mathsf{priv}_i}, ..., \sigma^n \right\rangle$, where s^{pub} is the public part of the state (i.e. assignment to variables in $\mathcal{V}^{\mathsf{pub}}$), $s^{\triangledown i}$ is the private part of α_i (i.e. assignment to variables in $\mathcal{V}^{\mathsf{priv}_i}$) and $\sigma^1, ..., \sigma^{i-1}, \sigma^{i+1}, ...\sigma^n$

represents the private parts of other agents. When sending a state, the private part is replaced by the respective σ^i, when received by α_j, the σ^j is replaced by $s^{\triangledown j}$ from the state determined by σ^j.

Let us now consider how can the agent-additive and agent-agnostic properties utilized in the search to reduce heuristic computation and communication. According to Defition 76, an agent-additive heuristic for agent α_i can be expressed as

$$h(s) = h^{\mathsf{pub}}(s^{\triangleright}) + \sum_{\alpha_j \in \mathcal{A}} h^j(s^{\triangleright j})$$

Now from the point of view of a single agent α_i, the value of the private parts of all other agents can be expressed as

$$\sum_{\alpha_j \in \mathcal{A} \setminus \{\alpha_i\}} h^j(s^{\triangleright j}) = h(s) - h^{\mathsf{pub}}(s^{\triangleright}) - h^i(s^{\triangleright i})$$

Thanks to the agent-additive property, by subtracting the public part and private part of agent α_i from the heuristic value, we obtain the sum of private parts of other agents. Moreover, the following holds:

$$h(s') = h(s) - h^{\mathsf{pub}}(s^{\triangleright}) - h^i(s^{\triangleright i}) + h^{\mathsf{pub}}(s'^{\triangleright}) + h^i(s'^{\triangleright i}) \tag{6.3.1}$$

where s' is a successor state of s. This means, that the heuristic estimate of a state s' can be easily determined from the heuristic estimate of its predecessor s. When a state is received from some other agent α_j, it is accompanied with its global heuristic estimate computed by α_j. When a state s is expanded, the heuristic estimate of its successor s' can be computed using the above equation.

The difference between agent-agnostic and agent-additive heuristic is that in the case of agent-agnostic heuristic, Equation 6.3.1 holds for all actions of agent α_i, that is, $s' = s \circ o$ for some $o \in \mathcal{O}^i$. In the case of agent-additive heuristic which is not agent-agnostic, Equation 6.3.1 holds only for private action of agent α_i, that is, $s' = s \circ o$ for some $o \in \mathcal{O}^{\mathsf{priv}_i}$. If the agent uses a public action, the heuristic must be computed by all agents from scratch, as in general the heuristic $h^j(s^{\triangleright j})$ of any agent α_j might be influenced by a public action $o' \in \mathcal{O}^{\mathsf{pub}_i}$.

Example 78. (Logistics) In Figure 6.3.2 we illustrate the principle of the agent-agnostic heuristic in a distributed search on the Logistics example. In the first step, the agents exchange the private heuristic values of the truck ($h^t = 3$) and the plane ($h^e = 1$). Then the search can continue using only the private actions of the agents without any communication, but still obtaining global heuristic values. Finally, after using the unload public action, the public heuristic changes and is sent alongside the state as in standard MAFS or MAD-A*. In the case of agent-additive heuristic which is not agent-agnostic, the use of the public unloadTB action could have influenced the private heuristic value of the plane and thus the truck would have to request the value from plane.

6.4 Distributed Potential Heuristics

In this section we propose an agent-additive (Definition 76) and agent-agnostic (Definition 77) distributed heuristic based on the idea of potential heuristics [Pommerening et al., 2015]. We use the MPT and MA-MPT formalism (Section 3.2) throughout this section.

6.4.1 Potential Heuristics

Potential heuristics are a family of admissible heuristics introduced in [Pommerening et al., 2015]. Here we describe the original centralized version. A potential heuristic (denoted as h_{pot}) associates a numerical potential with each fact. The potential heuristic for a state s is simply a sum of potentials of the facts in s, formally:

$$h_{\mathsf{pot}}(s) = \sum_{V \in \mathcal{V}} \mathsf{pot}(\langle V, s[V] \rangle)$$

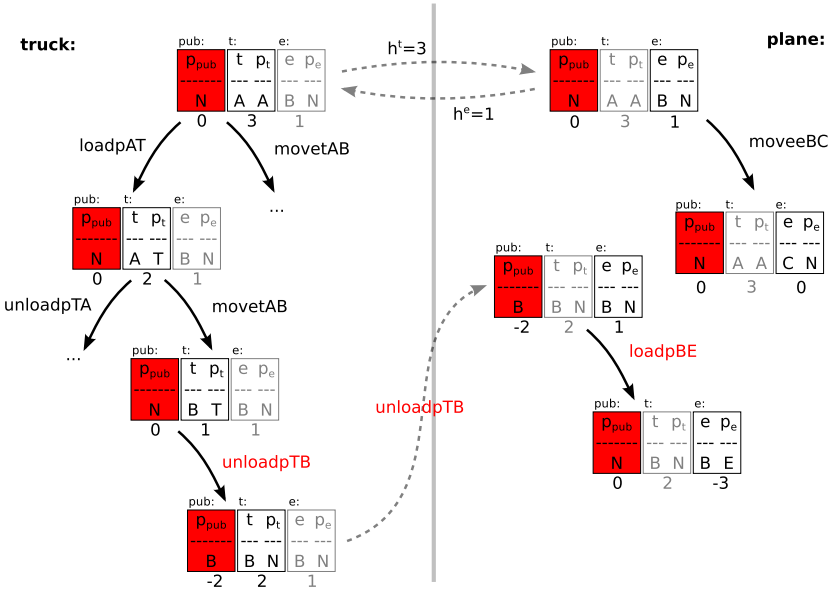


Figure 6.3.2: Example of additive A* computation on the Logistics problem.

where $\mathcal{V}$ is a set of variables and $\mathsf{pot}(\langle V, s[V]\rangle) \in \mathbb{R}$ is a potential for the fact representing the assignment for V in s.

The potentials can be determined as a solution to a linear program (LP). In this work, we use a formulation described in [Pommerening et al., 2014a]. The objective function of the LP is simply the sum of potentials for a state (or average for a set of states). The simplest variant is to use the initial state s_I as the optimization target. Another option is to use the set of all syntactic states[1] (S), as described in [Seipp et al., 2015], that is for all facts the coefficient associated with the potential variable of fact $\langle V, v\rangle$ is $1/|\mathsf{dom}(V)|$.

For a partial variable assignment p, let $\mathsf{maxpot}(V, p)$ denote the maximal potential that a state consistent with p can have for variable V, formally:

$$\mathsf{maxpot}(V, p) = \begin{cases} \mathsf{pot}(\langle V, p[V]\rangle) & \text{if } V \in \mathsf{vars}(p) \\ \max\limits_{v \in \mathsf{dom}(V)} \mathsf{pot}(\langle V, v\rangle) & \text{otherwise} \end{cases}$$

The LP will have a potential LP-variable $\mathsf{pot}(\langle V, v\rangle)$ for each fact (that is each possible assignment to each variable) and a maximum potential LP-variable maxpot_V for each variable in $\mathcal{V}$. The constraints ensuring the maximum potential property are simply

$$\mathsf{pot}(\langle V, v\rangle) \leq \mathsf{maxpot}_V \tag{6.4.1}$$

for all variables V and their values $v \in \mathsf{dom}(V)$. To ensure goal-awareness of the heuristic, i.e., $h_{\mathsf{pot}}(s) \leq 0$ for all goal states s, see Definition 41(iii), we add the following constraint

$$\sum_{V \in \mathcal{V}} \mathsf{maxpot}(V, s_\star) \leq 0 \tag{6.4.2}$$

restricting the heuristic of any goal state to be less or equal to 0. The final set of constraints ensures consistency. A consistent heuristic is such h that for each two states s, s' and all operators s.t. $s' = s \circ o$

[1] Such LP formulation may be unbounded. A common solution we adopt is to use a large-enough upper bound for each LP variable.

holds $h(s) \leq h(s') + \mathsf{cost}(o)$, see Definition 41(v). Consistency together with the goal-awareness implies admissibility. For each operator o in a set of operators $\mathcal{O}$ we add the following constraint

$$\sum_{V \in \mathsf{vars}(\mathsf{eff}(o))} (\mathsf{maxpot}(V, \mathsf{pre}(o)) - \mathsf{pot}(\langle V, \mathsf{eff}(o)[V] \rangle)) \leq \mathsf{cost}(o) \tag{6.4.3}$$

A solution of the LP yields the values for potentials which are then used in the heuristic computation.

Example. (pot) Let us consider a concrete example with two agents α_1, α_2, where the variables are the following (the actions are not important for the presentation):

α_1 **private:** $V_1 \in \{d_1, d_1'\}$,

α_2 **private:** $V_2 \in \{d_2, d_2'\}$

public: $V_{\mathsf{pub}} \in \{d_{\mathsf{pub}}, d_{\mathsf{pub}}'\}$

For now we consider the equivalent global problem with all variables and no agents. Assume that the computed potentials are the following:

$$\begin{array}{lll} \mathsf{pot}(\langle V_1, d_1 \rangle) = 1 & \mathsf{pot}(\langle V_2, d_2 \rangle) = 2 & \mathsf{pot}(\langle V_{\mathsf{pub}}, d_{\mathsf{pub}} \rangle) = 3 \\ \mathsf{pot}(\langle V_1, d_1' \rangle) = -2 & \mathsf{pot}(\langle V_2, d_2' \rangle) = 0 & \mathsf{pot}(\left\langle V_{\mathsf{pub}}, d_{\mathsf{pub}}' \right\rangle) = -1 \end{array}$$

In the initial state s_I holds $s_I[V_1] = d_1$, $s_I[V_2] = d_2$ and $s_I[V_{\mathsf{pub}}] = d_{\mathsf{pub}}$. The potential heuristic for the initial state can be computed as

$$h_{\mathsf{pot}}(s) = \sum_{V \in \mathcal{V}} \mathsf{pot}(\langle V, s[V] \rangle) = 1 + (-2) + 2 + 0 + 3 + (-1) = 3$$

6.4.2 Potential Heuristics for Multi-Agent Planning

Let us first examine the global potential heuristic $h^{\mathsf{G}}_{\mathsf{pot}}$ computed on Π^{G}. For now, assume we already have the potentials for the global problem. For a state s, the global potential heuristic is

$$h^{\mathsf{G}}_{\mathsf{pot}}(s) = \sum_{V \in \mathcal{V}^{\mathsf{G}}} \mathsf{pot}(\langle V, s[V] \rangle)$$

which can be rewritten as

$$h^{\mathsf{G}}_{\mathsf{pot}}(s) = \sum_{V \in \mathcal{V}^{\mathsf{pub}}} \mathsf{pot}(\langle V, s[V] \rangle) + \sum_{\alpha_i \in \mathcal{A}} \sum_{V \in \mathcal{V}^{\mathsf{priv}_i}} \mathsf{pot}(\langle V, s^{\triangleright i}[V] \rangle)$$

which is the sum of potentials of public facts plus the sum of potentials of private facts of each agent. Further on, we will denote

$$h^{\mathsf{pub}}_{\mathsf{pot}}(s^{\triangleright}) = \sum_{V \in \mathcal{V}^{\mathsf{pub}}} \mathsf{pot}(\langle V, s^{\triangleright}[V] \rangle)$$

and

$$h^{\mathsf{priv}_i}_{\mathsf{pot}}(s^{\triangleright i}) = \sum_{V \in \mathcal{V}^{\mathsf{priv}_i}} \mathsf{pot}(\langle V, s^{\triangleright i}[V] \rangle)$$

thus the global heuristic can be rewritten as

$$h^{\mathsf{G}}_{\mathsf{pot}}(s) = h^{\mathsf{pub}}_{\mathsf{pot}}(s^{\triangleright}) + \sum_{\alpha_i \in \mathcal{A}} h^{\mathsf{priv}_i}_{\mathsf{pot}}(s^{\triangleright i}). \tag{6.4.4}$$

Note, that an i-projected potential heuristic can be expressed as

$$h^{\triangleright i}_{\mathsf{pot}}(s) = h^{\mathsf{pub}}_{\mathsf{pot}}(s^{\triangleright}) + h^{\mathsf{priv}_i}_{\mathsf{pot}}(s^{\triangleright i})$$

Example. (pot) In the case of the example:

$$h^{\mathsf{pub}}_{\mathsf{pot}}(s^{\triangleright}) = \mathsf{pot}(\langle V_{\mathsf{pub}}, d_{\mathsf{pub}}\rangle) + \mathsf{pot}(\langle V_{\mathsf{pub}}, d'_{\mathsf{pub}}\rangle)$$
$$h^{\mathsf{priv}_1}_{\mathsf{pot}}(s^{\triangleright 1}) = \mathsf{pot}(\langle V_1, d_1\rangle) + \mathsf{pot}(\langle V_1, d'_1\rangle)$$
$$h^{\mathsf{priv}_2}_{\mathsf{pot}}(s^{\triangleright 2}) = \mathsf{pot}(\langle V_2, d_2\rangle) + \mathsf{pot}(\langle V_2, d'_2\rangle)$$

Now we show the desirable properties of the multi-agent potential heuristic.

Theorem 79. *(Properties of the potential heuristic) The global potential heuristic*

$$h^{\mathsf{G}}_{\mathsf{pot}}(s) = h^{\mathsf{pub}}_{\mathsf{pot}}(s) + \sum_{\alpha_i \in \mathcal{A}} h^{\mathsf{priv}_i}_{\mathsf{pot}}(s^{\triangleright i})$$

is admissible (Definition 41), agent-additive (Definition 76) and agent-agnostic (Definition 77).

Proof. Admissibility follows directly from the construction of the LP which is equal to the LP in the centralized case. The agent-additivity of the potential heuristic for any agent α_k follows from setting $h^{\mathsf{pub}}(s^{\triangleright}) = h^{\mathsf{pub}}_{\mathsf{pot}}(s^{\triangleright})$ and $h^i(s^{\triangleright i}) = h^{\mathsf{priv}_i}_{\mathsf{pot}}(s^{\triangleright i})$ for all $\alpha_i \in \mathcal{A}$.

Let $i \neq j$, the agent-agnostic property holds for $h^{\mathsf{G}}_{\mathsf{pot}}(s)$, because for each $o \in \mathcal{O}^i$, $\mathsf{eff}(o) \cap \mathcal{V}^{\mathsf{priv}_j} = \emptyset$. The part of the state s private to agent α_j, that is partial assignment $p = s \cap \mathcal{V}^{\mathsf{priv}_j}$, is equal to the part of the successor state $s' = s \circ o$ private to agent α_j (that is partial assignment $p' = s' \cap \mathcal{V}^{\mathsf{priv}_j}$). As both $h^{\mathsf{priv}_j}_{\mathsf{pot}}(s^{\triangleright j})$ and $h^{\mathsf{priv}_j}_{\mathsf{pot}}(s'^{\triangleright j})$ are computed only on the respective parts of the states private to agent α_j, the heuristic estimates are equal. □

By application of Theorem 79, the global heuristic estimate of a successor state s' after application of an operator $o \in \mathcal{O}^i$ can be effectively computed by the agent α_i by computing the public part of the heuristic estimate, the part private to agent α_i and adding the private parts of other agents from the predecessor state s. The multi-agent distributed A* search using this principle is outlined in Figure 6.4.1.

Example. (pot) Let us, again, have a look on the example. In the initial state s_I holds $s_I[V_1] = d_1$, $s_I[V_2] = d_2$ and $s_I[V_{\mathsf{pub}}] = d_{\mathsf{pub}}$, therefore $h^{\mathsf{priv}_1}_{\mathsf{pot}}(s_I^{\triangleright 1}) = 1$, $h^{\mathsf{priv}_2}_{\mathsf{pot}}(s_I^{\triangleright 2}) = 2$ and $h^{\mathsf{pub}}_{\mathsf{pot}}(s_I^{\triangleright}) = 3$ and thus

$$h_{\mathsf{pot}}(s_I) = h^{\mathsf{pub}}_{\mathsf{pot}}(s_I^{\triangleright}) + h^{\mathsf{priv}_1}_{\mathsf{pot}}(s_I^{\triangleright 1}) + h^{\mathsf{priv}_2}_{\mathsf{pot}}(s_I^{\triangleright 2}) = 6$$

If the agent α_1 applies an action $a_1 \in \mathcal{O}^{\mathsf{pub}_1}$ which changes both V_1, V_{pub} so that $s_1[V_1] = d'_1$ and $s_1[V_{\mathsf{pub}}] = d'_{\mathsf{pub}}$, the heuristic value of the resulting state s_1 computed by α_1 is

$$h_{\mathsf{pot}}(s_1) = h^{\mathsf{pub}}_{\mathsf{pot}}(s_1^{\triangleright}) + h^{\mathsf{priv}_1}_{\mathsf{pot}}(s_1^{\triangleright 1}) + h^{\mathsf{priv}_2}_{\mathsf{pot}}(s_I^{\triangleright 2}) = -1$$

The state s_1 is then sent to agent α_2 together with the value of $h^{\mathsf{priv}_1}_{\mathsf{pot}}(s_1^{\triangleright 1}) = -2$. When agent α_2 applies action $a_2 \in \mathcal{O}^{\mathsf{priv}_2}$ which modifies only V_2 so that $s_2[V_2] = d'_2$, α_2 can compute

$$h_{\mathsf{pot}}(s_2) = h^{\mathsf{pub}}_{\mathsf{pot}}(s_2^{\triangleright}) + h^{\mathsf{priv}_1}_{\mathsf{pot}}(s_1^{\triangleright 1}) + h^{\mathsf{priv}_2}_{\mathsf{pot}}(s_2^{\triangleright 2}) = -3$$

using the value of $h^{\mathsf{priv}_1}_{\mathsf{pot}}(s_1^{\triangleright 1})$ received from α_1.

In addition to the method shown in Figure 6.4.1, thanks to the agent-agnostic property of the potential heuristic and Equation 6.3.1, the potential heuristic can be computed form the parent state as

$$h^{\mathsf{G}}_{\mathsf{pot}}(s') = h^{\mathsf{G}}_{\mathsf{pot}}(s) - h^{\mathsf{pub}}_{\mathsf{pot}}(s^{\triangleright}) - h^{\mathsf{priv}_i}_{\mathsf{pot}}(s^{\triangleright i}) + h^{\mathsf{pub}}_{\mathsf{pot}}(s'^{\triangleright}) + h^{\mathsf{priv}_i}_{\mathsf{pot}}(s'^{\triangleright i}) \tag{6.4.5}$$

where $s' = s \circ o$ for some $o \in \mathcal{O}^i$. This means, that the sum in Equation 6.4.4 does not have to be explicitly computed, thus any privacy concerns of the sum computation are avoided.

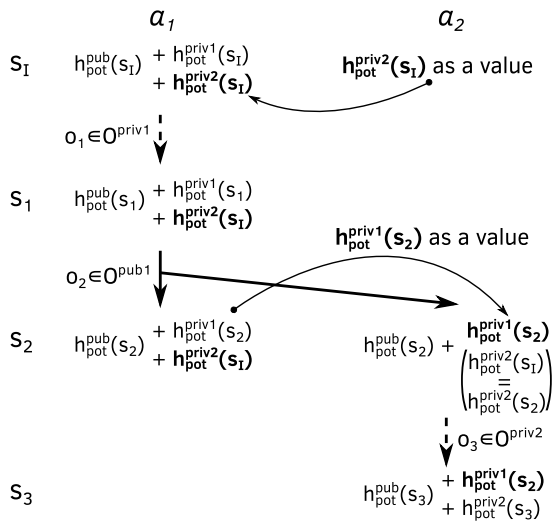


Figure 6.4.1: Sequence of $h^{\mathsf{G}}_{\mathsf{pot}}$ computations. The emphasized heuristics are not computed, but used as a heuristic value from the other agent.

Example. (pot) Again, referring to the running example, when agent α_1 applies the public action $a_1$in s_I, resulting in s_1, it sends s_1 to α_2. Instead of sending the value of $h^{\mathsf{priv}_1}_{\mathsf{pot}}(s_1^{\triangleright 1})$ as suggested previously, it can send only the value of $h^{\mathsf{G}}_{\mathsf{pot}}(s_1)$. After application of a_2, the agent α_2 can compute the heuristic estimate of s_2 simply by

$$h^{\mathsf{G}}_{\mathsf{pot}}(s_2) = h^{\mathsf{G}}_{\mathsf{pot}}(s_1) - h^{\mathsf{pub}}_{\mathsf{pot}}(s_1^{\triangleright}) - h^{\mathsf{priv}_2}_{\mathsf{pot}}(s_1^{\triangleright 1}) + h^{\mathsf{pub}}_{\mathsf{pot}}(s_2^{\triangleright}) + h^{\mathsf{priv}_2}_{\mathsf{pot}}(s_2^{\triangleright 2}) = -1 - (-1) - 2 + (-1) + 0 = -3$$

which equals the result obtained by the original computation.

6.4.3 Distributed Computation of Potentials

So far we have not considered the way how the potentials are obtained in the case of the distributed heuristic. As in the classical planning version we can build a linear program (or a set of linear programs) consisting of the constraints in Equation 6.4.2 and Equation 6.4.3. There is a number of ways how the LP can be formulated and solved in a distributed way. In this section we present the techniques, but we leave the discussion of their implications on privacy to Section 7.3.5 after we have established the theory which allows us to analyze privacy properly.

Projections

The simplest idea is to let each agent compute the LP on the projected problem and to use the resulting potentials in the global heuristic. Unfortunately, this approach does not result in an admissible heuristic. One of the reasons lies in the goal-awareness constraint, which in the global LP contains either potential or maximum potential LP-variable for each variable in $\mathcal{V}^{\mathsf{G}}$. In the projected problem, the private variables of other agents are missing. This allows the remaining variables to have higher values and results in higher potentials making the sum of private parts inadmissible.

An admissible variant is to take the maximum of the projections. The i-projected heuristic can be

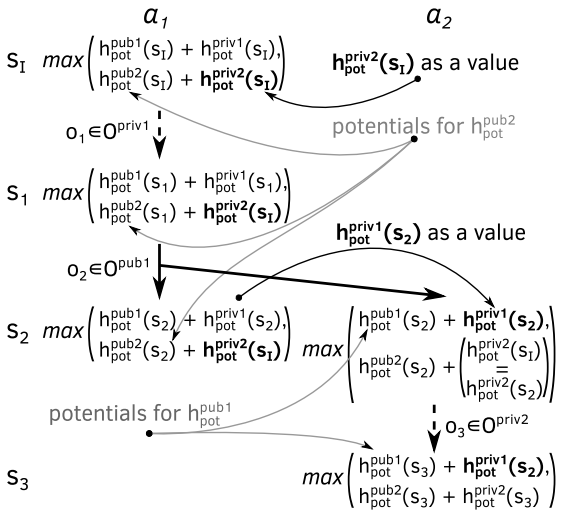


Figure 6.4.2: Sequence of $h^{\mathsf{maxproj}}_{\mathsf{pot}}$ computations. The emphasized heuristics are not computed, but used as a heuristic value from the other agent. The gray potentials are communicated from the other agent and computed as $h^{\mathsf{pub}_i}_{\mathsf{pot}}(s)$.

rewritten as

$$\begin{aligned} h^{\triangleright i}_{\mathsf{pot}}(s) &= h^{\mathsf{pub}_i}_{\mathsf{pot}}(s^{\triangleright}) + h^{\mathsf{priv}_i}_{\mathsf{pot}}(s^{\triangleright i}) \\ &= \sum_{V \in \mathcal{V}^{\mathsf{pub}}} \mathsf{pot}^i(\langle V, s^{\triangleright}[V] \rangle) + \sum_{V \in \mathcal{V}^{\mathsf{priv}_i}} \mathsf{pot}^i(\langle V, s^{\triangleright i}[V] \rangle) \end{aligned}$$

where $\mathsf{pot}^i(\langle V, v \rangle)$ is the potential for fact $\langle V, v \rangle$ computed on the i-projected problem. Although $\mathcal{V}^{\mathsf{pub}}$ is the same for all agents the potentials in $h^{\triangleright i}_{\mathsf{pot}}$ for variables in $\mathcal{V}^{\mathsf{pub}}$ may differ as they are computed using different LPs. Nevertheless, the potentials $\mathsf{pot}^i(\langle V, v \rangle)$ for public variables $V \in \mathcal{V}^{\mathsf{pub}}$ computed by agent α_i can be communicated to all other agents α_j.

This means that the public part of the i-projected heuristic can be computed by each agent α_j separately on its projected problem $\Pi^{\triangleright j}$ using the potentials of public variables shared from other agents. Thanks to the agent-agnostic property of the potential heuristic shown by Theorem 79 (which trivially holds also for projections), the private parts of the i-projected heuristic are not changed by actions of other agents. The private part can be computed by the respective agent α_i and sent along with each state s.

This means, that unlike a general projected heuristic, each agent α_i can compute the $h^{\triangleright j}_{\mathsf{pot}}$ projected heuristics of all agents without any additional communication and take the maximum. We denote the resulting heuristic as

$$h^{\mathsf{maxproj}}_{\mathsf{pot}}(s) = \max_{\alpha_i \in \mathcal{A}} h^{\triangleright i}_{\mathsf{pot}}(s^{\triangleright i}).$$

To compute the $h^{\mathsf{maxproj}}_{\mathsf{pot}}(s)$ heuristic, the agent computes all public parts, its own private part, sums the corresponding public and private parts and takes the maximum, as shown in Figure 6.4.2. Obviously, $h^{\mathsf{maxproj}}_{\mathsf{pot}}(s)$ is always at least as informed as $h^{\triangleright i}_{\mathsf{pot}}(s^{\triangleright i})$, but never more informed than $h^{\mathsf{G}}_{\mathsf{pot}}(s)$.

Example. (pot) Let us consider the running example again. Now each agent has its potentials (including the public ones) computed independently, for the example:

α_1	α_2
$\mathsf{pot}^1(\langle V_{\mathsf{pub}}, d_{\mathsf{pub}}\rangle) = 4$	$\mathsf{pot}^2(\langle V_{\mathsf{pub}}, d_{\mathsf{pub}}\rangle) = 3$
$\mathsf{pot}^1(\left\langle V_{\mathsf{pub}}, d'_{\mathsf{pub}}\right\rangle) = 0$	$\mathsf{pot}^2(\left\langle V_{\mathsf{pub}}, d'_{\mathsf{pub}}\right\rangle) = 1$
$\mathsf{pot}^1(\langle V_1, d_1\rangle) = 1$	$\mathsf{pot}^2(\langle V_2, d_2\rangle) = 2$
$\mathsf{pot}^1(\langle V_1, d'_1\rangle) = -2$	$\mathsf{pot}^2(\langle V_2, d'_2\rangle) = 1$

The potentials may be higher as the LPs of the projected problems are less constrained. By sharing the public potentials, both agents can compute

$$h^{\mathsf{maxproj}}_{\mathsf{pot}}(s_I) = \mathsf{max}(h^{\mathsf{pub}_1}_{\mathsf{pot}}(s_I^{\triangleright}) + h^{\mathsf{priv}_1}_{\mathsf{pot}}(s_I^{\triangleright 1}), h^{\mathsf{pub}_2}_{\mathsf{pot}}(s_I^{\triangleright}) + h^{\mathsf{priv}_2}_{\mathsf{pot}}(s_I^{\triangleright 2})) = \mathsf{max}(4+1, 3+2) = 5$$

After the application of a_1 by α_1, sending the resulting state s_1 to α_2 together with the value of $h^{\mathsf{priv}_1}_{\mathsf{pot}}(s_1^{\triangleright 1})$ and application of a_2 by α_2, the agent α_2 can compute

$$h^{\mathsf{maxproj}}_{\mathsf{pot}}(s_2) = \mathsf{max}(h^{\mathsf{pub}_1}_{\mathsf{pot}}(s_2^{\triangleright}) + h^{\mathsf{priv}_1}_{\mathsf{pot}}(s_1^{\triangleright 1}), h^{\mathsf{pub}_2}_{\mathsf{pot}}(s_2^{\triangleright}) + h^{\mathsf{priv}_2}_{\mathsf{pot}}(s_2^{\triangleright 2})) = \mathsf{max}(0-2, 1+1) = 2$$

Plain Global LP

A baseline approach to the global LP computation is to compute it plainly as it is. The principle of the computation is simple. One agent is selected to be the master, all other agents send their private parts of the LP (that is, optimization function, LP-variables, and constraints) to the master. The master then solves the complete LP and sends the computed values of the LP-variables back to their respective owners.

Securely Computed Global LP

Another approach is the use of a privacy-preserving transformation of the whole LP, which is often used in the secure multi-party computation. Representative examples of such transformation were published in [Mangasarian, 2011] and [Dreier and Kerschbaum, 2011].

We base the secure LP computation on [Mangasarian, 2011], a more general approach can be found in [Dreier and Kerschbaum, 2011]. The transformation is applicable only on vertically partitioned data, that is data partitioned based on the variables. This means that each agent owns a disjoint subset of the LP variables and the respective parts of constraints containing them, that is, each constraint either falls completely into one partition or is partitioned according to the variables (may span over multiple partitions, e.g., the goal-awareness constraint). The potential heuristic LP can be partitioned in $n+1$ partitions, where the i-th partition comprises of the pot and maxpot LP variables for $V \in \mathcal{V}^{\mathsf{priv}_i}$ and the *public* $(n+1)$-th partition contains the pot and maxpot LP variables for $V \in \mathcal{V}^{\mathsf{pub}}$. The public partition, which may span over constraints of multiple agents, does not have to be encrypted and thus can be treated separately. Thus, each agent knows complete constraints for its own actions, the public part of constraints for public actions and its private and public part of the goal-awareness constraint. The secure LP computation according to [Mangasarian, 2011] proceeds as follows.

Let $\mathsf{max}\, c^T x$ be the optimization function and $Ax \leq b$, $A \in \mathbb{R}^{l \times m}$ the global set of constraints, which means that the global problem consists of m LP-variables and l constraints. The whole computation proceeds as follows:

1. All agents agree on some $k \geq m$. A master agent α_j which will compute the LP is selected.
2. Each agent α_i s.t. $i \neq j$ generates a random matrix $B_i \in \mathbb{R}^{k \times m_i}$, where m_i is the number of LP-variables private to agent α_i and m_{pub} is the number of public LP-variables. We define $B = [B_1 \dots B_n] \in \mathbb{R}^{k \times m}$, where B_j is a unit matrix $k \times (m_{\mathsf{pub}} + m_j)$, as α_j does not have to encrypt its part of the LP.
3. Each agent α_i sends matrix product $A_i B_i^T$ and cost coefficient product $B_i c_i$ to agent α_j, where A_i and c_i are the parts of the global LP problem private to agent α_i.

4. The linear program maximize $c^T B^T u$ subject to $AB^T = A_1 B_1^T + ... + A_n B_n^T \leq b$ is computed by agent α_j and the result vector u is sent to all other agents.

5. Each agent α_i reconstructs the solution as $x_i = B_i^T u$.

The LP for the potential heuristic differs in two features. First, there is a public part, which does not have to be encrypted. Second, some of the constraints are private-only and other agents are not aware of them. Therefore, in the Step 1 above, the agents inform the master agent about the number of constraints in the form of $k_i \geq m_i$ and k is chosen subsequently as $k = \sum_{\alpha_i \in \mathcal{A}} k_i$. This allows the agents to hide the real number of LP variables and thus also the number of variables in the private part of the planning problem (the constant k_i gives an upper bound on the number of private variables). In Step 3 in addition to the encrypted private part, the agents send to the master also the unencrypted public part and the part of vector b respective to the private constraints (this vector encodes the costs of private actions), which are combined to form the public part of the LP and the cost vector.

6.5 Multi-Agent Cost Partitioning

All distributed heuristics presented in Chapter 4 and in this chapter so far are based on more-or-less ad-hoc techniques to distribute each particular heuristic or a family of heuristics. The distributed computation of heuristic estimate often requires the cooperation of all (or at least most of) the agents and incurs a substantial amount of communication, which, as already said in Section 6.4, may be ineffective or prohibitive. In the previous Section 6.4, we have presented an example an additive heuristic such that projected estimates of two agents could be added together and still maintain admissibility, based on the family of potential heuristics. In this section, we apply general results of additive heuristic research, namely the approach of cost-partitioning, to the case of distribution of heuristics for multi-agent planning. This way we obtain a fully general approach allowing us to compute any heuristic additively in a distributed way. Also, it allows us to combine different heuristics, which adheres to the idea of independent agents (that is, each agent can use the heuristic it sees most fit). Last but not least, the presented approach allows us to compute an admissible sum of admissible heuristics.

In classical planning, cost partitioning is typically computed for each state evaluated during the planning process. In PP-MAP, such approach does not make much sense as we want to keep local as much computation as possible. The envisioned use of such cost-partitioning is to compute it once at the beginning of the planning process, use the cost-partitioned problems to evaluate heuristics locally and sum the local heuristics to obtain a global estimate.

Example. (CP) Here we present a small running example with two agents α_1 and α_2. The problem of agent α_1 is Π^1:

$$
\begin{array}{lcl}
\mathcal{V}^{\mathsf{pub}} & = & \{V_3 \in \{u, g\}\} \\
\mathcal{V}^{\mathsf{priv}_1} & = & \{V_1 \in \{i_1, p_1\}\} \\
\mathcal{O}^{\mathsf{pub}_1} & = & \{b_1\} \\
\mathcal{O}^{\mathsf{priv}_1} & = & \{a_1\} \\
s_I^1 & = & V_1 \mapsto i_1, V_3 \mapsto u \\
s_\star^1 & = & V_3 \mapsto g
\end{array}
$$

a	$\mathsf{pre}(a)$	$\mathsf{eff}(a)$	$\mathsf{cost}^1(a)$
a_1	$V_1 \mapsto \mathsf{i}_1$	$V_1 \mapsto \mathsf{p}_1$	$\mathsf{cost}^1(a_1) = 1$
b_1	$V_1 \mapsto \mathsf{p}_1$	$V_1 \mapsto \mathsf{i}_1, V_3 \mapsto g$	$\mathsf{cost}^1(b_1) = 2$

The problem of agent α_2 is Π^2:

$$
\begin{array}{lcl}
\mathcal{V}^{\mathsf{pub}} & = & \{V_3 \in \{u, g\}\} \\
\mathcal{V}^{\mathsf{priv}_2} & = & \{V_2 \in \{i_2, p_2\}\} \\
\mathcal{O}^{\mathsf{pub}_2} & = & \{b_2\} \\
\mathcal{O}^{\mathsf{priv}_2} & = & \{a_2\} \\
s_I^2 & = & V_2 \mapsto i_2, V_3 \mapsto u \\
s_\star^2 & = & V_3 \mapsto g
\end{array}
$$

a	$\mathsf{pre}(a)$	$\mathsf{eff}(a)$	$\mathsf{cost}^2(a)$
a_2	$V_2 \mapsto \mathsf{i}_2$	$V_2 \mapsto \mathsf{p}_2$	$\mathsf{cost}^2(a_2) = 1$
b_2	$V_2 \mapsto \mathsf{p}_2$	$V_2 \mapsto \mathsf{i}_2, V_3 \mapsto g$	$\mathsf{cost}^2(b_2) = 2$

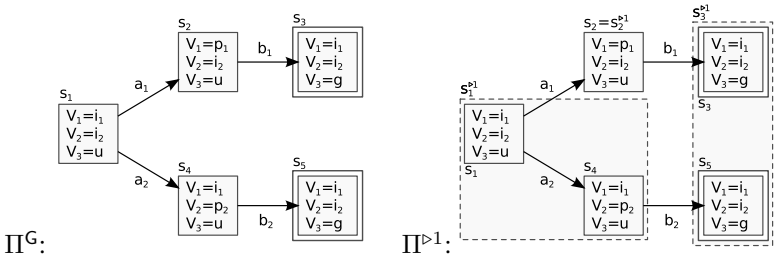


Figure 6.5.1: a) Transition system of the global problem Π^{G} respective to the example. b) Example transition system, 1-projection (abstraction).

In addition, the actions of projected problem $\Pi^{\triangleright 1}$ are $\mathcal{O}^{\triangleright 1} = \{a_1^{\triangleright 1}, b_1^{\triangleright 1}, b_2^{\triangleright 1}\}$, where $a_1^{\triangleright 1}, b_1^{\triangleright 1}$ are unchanged and $b_2^{\triangleright 1}$:

a	$\mathsf{pre}(a)$	$\mathsf{eff}(a)$	$\mathsf{cost}^1(a)$
$b_2^{\triangleright 1}$	$\emptyset$	$V_3 \mapsto g$	$\mathsf{cost}^1(b_2^{\triangleright 1}) = 2$

Analogously, the actions of projected problem $\Pi^{\triangleright 2}$ are $\mathcal{O}^{\triangleright 2} = \{a_2^{\triangleright 2}, b_2^{\triangleright 2}, b_1^{\triangleright 2}\}$, where $a_2^{\triangleright 2}, b_2^{\triangleright 2}$ are unchanged and $b_1^{\triangleright 2}$:

a	$\mathsf{pre}(a)$	$\mathsf{eff}(a)$	$\mathsf{cost}^2(a)$
$b_1^{\triangleright 2}$	$\emptyset$	$V_3 \mapsto g$	$\mathsf{cost}^2(b_1^{\triangleright 2}) = 2$

Figure 6.5.1 shows the global and projected transition systems of the example problem. A global solution to the problem is either (a_1, b_1) or (a_2, b_2), both of cost 3. The optimal solution of $\Pi^{\triangleright 1}$ is $(b_2^{\triangleright 1})$ with the cost of 2 and symmetrically for $\Pi^{\triangleright 2}$. Thus if we take the baseline approach and maximize the two optimal costs we obtain 2 which is a bound on the value any two admissible heuristics can give as a maximum of projected heuristics.

6.5.1 Cost Partitioning

In this section, we describe the idea of cost-partitioning [Katz and Domshlak, 2010] as used in classical planning and define a novel notion of multi-agent cost-partitioning. We consider non-negative cost-partitioning, where the costs of actions are not allowed to be less than 0, but all notions and techniques generalize to the case of general cost-partitioning without such restriction.

Definition 80. (Cost partitioning). Let Π be a planning task with operators $\mathcal{O}$ and cost function cost. A cost partitioning for Π is a tuple $\mathsf{cp} = \langle \mathsf{cp}_1, ..., \mathsf{cp}_k \rangle$ where $\mathsf{cp}_l : \mathcal{O} \to \mathbb{R}_0^+$ for $1 \leq l \leq k$ and $\sum_{l=1}^{k} \mathsf{cp}_l(o) \leq \mathsf{cost}(o)$ for all $o \in \mathcal{O}$.

As shown in [Katz and Domshlak, 2010], a sum of admissible heuristics computed on the cost-partitioned problem is also admissible, formally

Proposition 81. *(Katz and Domshlak 2010). Let Π be a planning task, let $h_1, ..., h_k$ be admissible heuristics for Π, and let $\mathsf{cp} = \langle \mathsf{cp}_1, ..., \mathsf{cp}_k \rangle$ be a cost partitioning for Π. Then $h_{\mathsf{cp}} = \sum_{l=1}^{k} h_l(s)$ where each h_l is computed with cp_l is an admissible heuristic estimate for a state s.*

Based on the particular cost partitioning cp, the heuristic estimate can have varying quality. By optimal cost-partitioning (OCP) we mean a cost-partitioning which maximizes h_{cp}.

Now we proceed with the definition of a multi-agent variant of cost-partitioning, which differs in that the partitions are defined apriori by the set of i-projected problems.

Definition 82. (Multi-agent cost partitioning). Let $\mathcal{M}^{\triangleright} = \{\Pi^{\triangleright i}\}_{i=1}^{n}$ be the set of all i-projected problems with respective cost functions $\mathsf{cost}^{\triangleright i}$. A multi-agent cost-partitioning for $\mathcal{M}^{\triangleright}$ is a tuple of functions $\mathsf{cp} = \langle \mathsf{cp}_1, ..., \mathsf{cp}_n \rangle$ where $\mathsf{cp}_i : \mathcal{O}^{\triangleright i} \to \mathbb{R}_0^+$. For $1 \leq i \leq n$ and for each $o \in \mathcal{O}^{\mathsf{G}}$ holds $\sum_{i=1}^{n} \mathsf{cp}_i(o^{\triangleright i}) \leq \mathsf{cost}^{\triangleright j}(o^{\triangleright j})$ where α_j is the owner of o, that is $o \in \mathcal{O}^j$.

Theorem 83. *Let $\mathcal{M}^{\triangleright} = \{\Pi^{\triangleright i}\}_{i=1}^{n}$ be the set of all i-projected problems, Π^{G} the global problem respective to $\mathcal{M}$ and cp a multi-agent cost-partitioning for $\mathcal{M}^{\triangleright}$. Then cp is a cost-partitioning for Π^{G}.*

Proof. The theorem follows from Definition 80, Definition 82 for all public actions and from setting $o^{\triangleright i} = \epsilon$ for all $o \in \mathcal{O}^{\mathsf{priv}_j}$ s.t. $j \neq i$. As $\mathsf{cost}^{\triangleright i}(o^{\triangleright i}) = \mathsf{cost}^{\triangleright i}(\epsilon) = 0$ and $\mathsf{cost}^{\triangleright j}(o^{\triangleright j}) = \mathsf{cost}^{j}(o)$, the cost-partitioning property $\sum_{i=1}^{n} \mathsf{cp}_i(o^{\triangleright i}) \leq \mathsf{cost}^{j}(o)$ holds also for private operators. □

Thanks to Theorem 83 we can apply the Proposition 81 also in the multi-agent setting using a multi-agent cost-partitioning. Thus, each agent α_i can compute its part of the heuristic locally on $\Pi^{\triangleright i}$ using cp_i instead of cost^i as the cost function. To obtain the global heuristic, the individual parts can be simply summed

$$h_{\mathsf{G}}(s) = \sum_{i=1}^{n} h^{\triangleright i}_{\mathsf{cp}_i}(s^{\triangleright i}) \tag{6.5.1}$$

where $h^{\triangleright i}_{\mathsf{cp}_i}$ is an i-projected heuristic computed on $\Pi^{\triangleright i}$ using cp_i. Trivially

Theorem 84. *Let $\mathcal{M}^{\triangleright} = \{\Pi^{\triangleright i}\}_{i=1}^{n}$ be the set of all i-projected problems, Π^{G} the global problem respective to $\mathcal{M}$ and cp a multi-agent cost-partitioning for $\mathcal{M}^{\triangleright}$. Then the heuristic*

$$h_{\mathsf{G}}(s) = \sum_{i=1}^{n} h^{\triangleright i}_{\mathsf{cp}_i}(s^{\triangleright i})$$

is agent-additive (Definition 76).

Proof. Follows trivially from Definition 76 by having the public part equal to zero, that is, $h^{\mathsf{pub}}(s^{\triangleright})$ for all i and all states. □

It is also trivial to see, that the heuristic from Equation 6.5.1 is not agent-agnostic (Definition 77). We contrast the cost-partitioning based approach to the current state of the art, which is taking the maximum of the heuristics computed by individual agents, formally

$$h_{\mathsf{max}}(s) = \mathsf{max}_{1 \leq i \leq n} h^{\triangleright i}(s^{\triangleright i}) \tag{6.5.2}$$

where $h^{\triangleright i}$ is any (admissible) heuristic computed on $\Pi^{\triangleright i}$ using the original cost^i.

6.5.2 Optimal Cost Partitioning

To compute the optimal cost partitioning (OCP) for i-projections, based on Theorems 36 and 83 we can readily apply the results of optimal cost partitioning for abstractions [Pommerening et al., 2014b].

The idea behind the following linear program (LP) formulation is to encode the abstract transition systems and possible shortest paths in it. The LP variables used for each $\alpha_i \in \mathcal{A}$ are $\bar{h}^{\triangleright i}$ encoding the i-projected heuristic value (given the cost-partitioning), $\bar{s}'^{\triangleright i}$ representing the cost of shortest path from a state s (or actually $s^{\triangleright i}$) to $s'^{\triangleright i}$ in the i-projected problem given the cost partitioning and $\bar{a}^{\triangleright i}$ representing the cost-partitioned cost of action $a^{\triangleright i} \in \mathcal{O}^{\triangleright i}$. The LP is formulated as follows:

Maximize $\sum_{i=1}^{n} \bar{h}^{\triangleright i}$ subject to

$$\begin{array}{rcll} \bar{s}' & = & 0 & \text{for all } s' = s \\ \bar{s}'' & \leq & \bar{s}' + \bar{a}^{\triangleright i} & \text{for all } \left\langle s'^{\triangleright i}, a^{\triangleright i}, s'' \right\rangle \in T^{\triangleright i} \\ \bar{h}^{\triangleright i} & \leq & \bar{s}' & \text{for all } \bar{s}' \in s^i_\star \\ \sum_{j=1}^{n} \bar{a}^{\triangleright j} & \leq & \mathsf{cost}^i(a) & \text{for all } a \in \mathcal{O}^{\mathsf{pub}_i} \\ \bar{a}^{\triangleright i} & \leq & \mathsf{cost}^i(a) & \text{for all } a \in \mathcal{O}^{\mathsf{priv}_i} \end{array}$$

where the first set of constraints sets all states equal (in the i-projection) with the current state s to have zero cost of shortest path. The second set of constraints encode the actual (abstracted) transitions and their costs (transitions where $s'^{\triangleright i} = s''$ can be ignored), the third set of constraints places an upper bound on the actual heuristic estimate to keep it admissible. The fourth and fifth sets of constraints represent the cost partitioning of public and private actions respectively. Note, that private actions of agent α_i always occur only as i-projections and are not partitioned (i.e. any other projection of such action has the cost of 0).

Example. (CP) Let us show how the optimal cost partitioning is computed on the running example. The global transition system is shown in Figure 6.5.1 a) and the transition system projected to agent α_1 in Figure 6.5.1 b) (transition system projected to α_2 is symmetrical). The LP is built based on the projected problems as follows:

Maximize $\bar{h}^{\triangleright 1} + \bar{h}^{\triangleright 2}$ subject to

$$\begin{aligned}
\bar{s}_1^{\triangleright 1} &= 0 \\
\bar{s}_2^{\triangleright 1} &\leq \bar{s}_1^{\triangleright 1} + \bar{a}_1^{\triangleright 1} \\
\bar{s}_3^{\triangleright 1} &\leq \bar{s}_2^{\triangleright 1} + \bar{b}_1^{\triangleright 1} \\
\bar{s}_3^{\triangleright 1} &\leq \bar{s}_1^{\triangleright 1} + \bar{b}_2^{\triangleright 1} \\
\bar{h}^{\triangleright 1} &\leq \bar{s}_3^{\triangleright 1} \\
&\ldots \\
\bar{a}_1^{\triangleright 1} &\leq 1 \\
\bar{b}_1^{\triangleright 1} + \bar{b}_1^{\triangleright 2} &\leq 2 \\
&..
\end{aligned}$$

where the omitted parts are defined for agent α_2 analogously. The solution gives $h^{\triangleright 1} + h^{\triangleright 2} = 3$ as the value of the objective function and $\bar{b}_1^{\triangleright 1} = 1, \bar{b}_1^{\triangleright 2} = 1, \bar{b}_2^{\triangleright 1} = 2, \bar{b}_2^{\triangleright 2} = 0$ as the values of (relevant) LP variables. The values directly give the cost partitioning. When applied, the optimal solutions of $\Pi^{\triangleright 1}$ and $\Pi^{\triangleright 2}$ has the cost of 1 and 2 respectively resulting in the sum of 3, which is the maximal value so that the sum does not violate admissibility.

In contrast to the use in classical planning, we intend to compute the cost-partitioning LP only once at the beginning of the planning process. Obviously, this results in a possibly sub-optimal cost-partitioning for other states than the initial one but still, should give better-informed heuristics than just taking maximum of the projections.

Unfortunately, even computing such OCP once may be intractable in general, as the i-projected problems may be as large and as hard as the global problem e.g., in a scenario where all (or most of) actions and variables are public. Even though typically the projected problems are significantly smaller and thus it is reasonable to experimentally evaluate this approach. An alternative option might be to create a smaller abstraction using some of the classical planning algorithms (e.g., Merge&Shrink [Helmert et al., 2007] for each agent and compute OCP only on those smaller abstractions.

6.5.3 Approximate Optimal Cost Partitioning

Another approach is to approximate the optimal cost-partitioning. As already mentioned, the most obvious approach is to compute a smaller abstraction of each of the $\Pi^{\triangleright i}$ and compute the OCP as above on that set of smaller abstractions. Moreover, a cost-partitioning LP formulation is known also for other heuristics, such as LM-Cut and even more heuristics can be expressed as an LP [Pommerening et al., 2014b] and modified to compute the cost-partitioning. In the following text, we describe three such examples, a LM-Cut [Helmert and Domshlak, 2009] based cost-partitioning, a cost partitioning modification of the State Equation (SEQ) heuristic LP formulation [Van Den Briel et al., 2007], and a

cost-partitioning based on the potential heuristic LP [Pommerening et al., 2015]. Finally, we describe a number of ad-hoc cost-partitioning techniques which are very easy to compute (without the use of LP) and still may lead to interesting results.

Landmarks

The LM-Cut heuristic proceeds by computing disjunctive action landmarks in the relaxed problem and iteratively reducing their cost, see Definition 70 (informally a disjunctive action landmark is a set of actions out of which at least one must be in each valid plan) and Section 6.2 for detailed description of the LM-Cut algorithm. The LP formulation [Pommerening et al., 2014b] starts with a set $\mathcal{L}$ of landmarks and assigns an LP variable to the cost of each $L \in \mathcal{L}$ respective to each cost-partitioning. Also, cost of each action respective to each CP is represented by an LP variable. The LP maximizes the sum of all landmark costs subject to

$$\sum_{a \in L} \bar{L} \leq \bar{a}$$

for each a and the cost-partitioning constraints. The LP variables $\bar{a}$ and $\bar{L}$ represent the costs of respective actions and landmarks.

Example. (CP) In the running example, there is one disjunctive landmark for agent α_1, that is $L_1^{\triangleright 1} = \{b_1^{\triangleright 1}, b_2^{\triangleright 1}\}$ and symmetrically $L_2^{\triangleright 2} = \{b_1^{\triangleright 2}, b_2^{\triangleright 2}\}$ for α_2. The private actions do not form a disjunctive landmark as the plan $(b_2^{\triangleright 1})$ solves $\Pi^{\triangleright 1}$ without using a_1. The LP is then formulated as follows:

Maximize $\bar{L}_1^{\triangleright 1} + \bar{L}_2^{\triangleright 2}$ subject to

$$\begin{aligned} \bar{b}_1^{\triangleright 1} &\leq \bar{L}_1^{\triangleright 1} \\ \bar{b}_2^{\triangleright 1} &\leq \bar{L}_1^{\triangleright 1} \\ &\dots \\ \bar{b}_1^{\triangleright 1} + \bar{b}_1^{\triangleright 2} &\leq 2 \\ \bar{b}_2^{\triangleright 1} + \bar{b}_2^{\triangleright 2} &\leq 2 \end{aligned}$$

where the constraints for $\bar{L}_1^{\triangleright 2}$ are analogous to those for $\bar{L}_2^{\triangleright 1}$. The solution is $\bar{L}_1^{\triangleright 1} = 2, \bar{L}_2^{\triangleright 2} = 0, \bar{b}_1^{\triangleright 1} = 2, \bar{b}_1^{\triangleright 2} = 0, \bar{b}_2^{\triangleright 1} = 2, \bar{b}_2^{\triangleright 2} = 0$, from which is clear that given the resulting CP, the optimal solution for $\Pi^{\triangleright 1}$ is $(b_2^{\triangleright 1})$ with cost 2 and for $\Pi^{\triangleright 2}$ is $(b_1^{\triangleright 2})$ with cost 0. Thus the sum of optimal costs is 2 which is not more than the maximum using the original costs.

It might help to compute the landmarks globally (as in [Štolba et al., 2015a]). It would make no difference in the example above, but in general, including private and public actions of different agents in a single landmark might improve the quality of the resulting cost-partitioning.

State Equation

State equation heuristic (SEQ) [Van Den Briel et al., 2007] builds on the idea of counting the operators necessary to change the values of variables from the initial state values to the goal state values. It is naturally formulated as a LP [Pommerening et al., 2014b] and can be easily modified so that the resulting values can be interpreted as a multi-agent cost-partitioning.

In the original formulation, there is an LP variable for each action, encoding the number of times it has to be used in any optimal plan. There is a constraint for each fact, that is a variable-value pair $\langle V, v \rangle$ for each $v \in V$ and each variable V. In order to formulate the constraint, we need to determine the set $\mathcal{O}^{\mathsf{AP}}$ of actions which always produce the fact (i.e. $\langle V, v \rangle \in \mathsf{eff}(a)$ and $\langle V, v' \rangle \in \mathsf{pre}(a)$ for some $v' \in V$), a set $\mathcal{O}^{\mathsf{SP}}$ of actions which sometimes produce the fact (i.e. $\langle V, v \rangle \in \mathsf{eff}(a)$ and $V \notin \mathsf{vars}(\mathsf{pre}(a))$) and analogously a set $\mathcal{O}^{\mathsf{AC}}$ of actions which always consume the fact (i.e. $\langle V, v \rangle \in \mathsf{pre}(a)$

and $\langle V, v' \rangle \in \mathsf{eff}(a)$ for some $v' \in V$) and a set $\mathcal{O}^{\mathsf{SC}}$ of actions which sometimes consume the fact (i.e. $\langle V, v \rangle \in \mathsf{pre}(a)$ and $V \notin \mathsf{vars}(\mathsf{eff}(a))$). The constraints for each fact $\langle V, v \rangle$ are

$$\begin{aligned} \sum_{a \in \mathcal{O}^{\mathsf{AP}}} \bar{a} + \sum_{a' \in \mathcal{O}^{\mathsf{SP}}} \bar{a'} - \sum_{a'' \in \mathcal{O}^{\mathsf{AC}}} \bar{a''} &\geq L \\ \sum_{b \in \mathcal{O}^{\mathsf{AP}}} \bar{b} - \sum_{b' \in \mathcal{O}^{\mathsf{AC}}} \bar{b'} - \sum_{b'' \in \mathcal{O}^{\mathsf{SC}}} \bar{b''} &\leq U \end{aligned}$$

where the bounds L, U are determined based on the initial and goal state. The optimization function of the LP is minimize $\sum_{a \in \mathcal{O}} \mathsf{cost}(a)\bar{a}$, where $\mathcal{O}$ is a set of all actions.

In order to compute a multi-agent CP based on the SEQ LP, we simply express all constraints respective to the i-projected problem $\Pi^{\triangleright i}$ for each agent and add them to a single LP. The optimization criterion is modified so that it minimizes the sum of $\mathsf{cost}(a)\bar{a}$ for all actions and all agents. The computed values of the LP variables then represent how many times each projection of each action has to be used in a solution of each projected problem. That is, for an action $a \in \mathcal{O}^{\mathsf{pub}_j}$ of some agent $\alpha_j \in \mathcal{A}$, we obtain $\bar{a}^{\triangleright 1}, ..., \bar{a}^{\triangleright n}$. Subsequently, the cost partitioning cp_k for agent $\alpha_k \in \mathcal{A}$ can be computed as

$$\mathsf{cp}_k(a^{\triangleright k}) = \mathsf{cost}^j(a) \frac{\bar{a}^{\triangleright k}}{\sum_{i=1}^{n} \bar{a}^{\triangleright i}} \tag{6.5.3}$$

that is, based on the ratio of the use of the action projections in the respective projected problems. Of course, if $\sum_{i=1}^{n} \bar{a}^{\triangleright i} = 0$, we need to determine the cost partitioning some other way.

Example. (CP) The LP for the example problem is formulated as follows:

Minimize $\bar{a}_1^{\triangleright 1} + 2\bar{b}_1^{\triangleright 1} + 2\bar{b}_2^{\triangleright 1} + \bar{a}_2^{\triangleright 2} + 2\bar{b}_1^{\triangleright 2} + 2\bar{b}_2^{\triangleright 2}$ subject to

$$\begin{array}{lcrcl} \langle V_1, i_1 \rangle & : & \bar{b}_1^{\triangleright 1} - \bar{a}_1^{\triangleright 1} & \geq & -1 \\ \langle V_1, i_1 \rangle & : & \bar{a}_1^{\triangleright 1} - \bar{b}_1^{\triangleright 1} & \leq & 0 \\ \langle V_1, p_1 \rangle & : & \bar{a}_1^{\triangleright 1} - \bar{b}_1^{\triangleright 1} & \geq & 0 \\ \langle V_1, p_1 \rangle & : & \bar{a}_1^{\triangleright 1} - \bar{b}_1^{\triangleright 1} & \leq & 1 \\ & & & \dots & \\ \langle V_3, g \rangle & : & \bar{b}_1^{\triangleright 1} + \bar{b}_2^{\triangleright 1} & \geq & 1 \\ \langle V_3, g \rangle & : & \bar{b}_1^{\triangleright 1} + \bar{b}_2^{\triangleright 1} & \leq & 1 \end{array}$$

where the constraints for V_2 are symmetric to the constraints for V_1. The resulting values are $\bar{b}_1^{\triangleright 2} = 1, \bar{b}_2^{\triangleright 1} = 1$ and 0 for all other LP variables. According to Equation 6.5.3, the costs of $b_1^{\triangleright 1}, b_2^{\triangleright 1}$ is computed as $\mathsf{cp}_1(b_1^{\triangleright 1}) = 2 \cdot 0/1 = 0$ and $\mathsf{cp}_1(b_2^{\triangleright 1}) = 2 \cdot 1/1 = 2$ respectively and analogously for $b_1^{\triangleright 2}, b_2^{\triangleright 2}$, which gives exactly the same results as the solution based on the LM-Cut formulation.

Potential Heuristic-based OCP

The family of potential heuristics was described in-detail in Section 6.4.1, including the LP used to compute the potentials. The LP is, in fact, a dual to the SEQ heuristic LP (see [Pommerening et al., 2015]). There is a constraint for each operator, expressing the facts it produces and consumes, where the cost of the operator is a constant on the right side.

Example. (CP) Let us consider the running example, where

a	$\mathsf{pre}(a)$	$\mathsf{eff}(a)$	$\mathsf{cost}^2(a)$
b_2	$V_2 \mapsto \mathsf{p}_2$	$V_2 \mapsto \mathsf{i}_2, V_3 \mapsto g$	$\mathsf{cost}^2(b_2) = 2$

The consistency constraint of the potential heuristic LP constructed for the public action b_2 is

$$\mathsf{pot}(\langle V_2, p_2\rangle) - \mathsf{pot}(\langle V_2, i_2\rangle) + \mathsf{maxpot}_{V_3} - \mathsf{pot}(\langle V_3, g\rangle) \leq \mathsf{cost}^2(b_2) = 2$$

where for the variable V_2 we use the potential for the precondition and the effect and for V_3 we use the maxpot_{V_3} variable for the precondition as V_3 has no value in the precondition of b_2.

The consistency of the potential heuristic with respect to an action a is obtained by putting the sum of the differences of potentials lower or equal to the cost of the action a. The LP is completed by adding such consistency for each of the actions and by adding the maxpot constraints stating, that each potential concerning a variable V is lower or equal to the maximum potential of V. The optimization function of the LP can be set to the sum of potentials in the initial state.

We can simply obtain a cost-partitioning LP by replacing the action costs with variables, concatenating the respective LPs for each of the agent problems and adding the cost-partitioning constraints. There are separate LP variables even for the potentials of public variables for each of the agents.

Example. (CP) The action b_2 will then be represented by two consistency constraints, one for b_2 in the context of Π^1 and one for $b_2^{\triangleright 2}$ in the context of $\Pi^{\triangleright 2}$. The constraints for b_2 (including the cost-partitioning constraint) are as follows.

$$\begin{aligned}\mathsf{pot}(\langle V_2, p_2\rangle)^{\triangleright 2} - \mathsf{pot}(\langle V_2, i_2\rangle)^{\triangleright 2} + \mathsf{maxpot}_{V_3}^{\triangleright 2} - \mathsf{pot}(\langle V_3, g\rangle)^{\triangleright 2} &\leq \bar{b}_2^{\triangleright 2}\\ \mathsf{maxpot}_{V_3}^{1} - \mathsf{pot}(\langle V_3, g\rangle)^{\triangleright 1} &\leq \bar{b}_2^{\triangleright 1}\\ \bar{b}_2^{\triangleright 1} + \bar{b}_2^{\triangleright 2} &\leq \mathsf{cost}^2(b_2) = 2\end{aligned}$$

Other constraints are formulated analogously. There are multiple possibilities for the optimization function, if we base the function on the initial state, we obtain the following

$$\mathsf{Maximize}: \mathsf{pot}(\langle V_1, i_1\rangle)^{\triangleright 1} + \mathsf{pot}(\langle V_2, i_2\rangle)^{\triangleright 2} + \mathsf{pot}(\langle V_3, u\rangle)^{\triangleright 1} + \mathsf{pot}(\langle V_3, u\rangle)^{\triangleright 2}$$

The resulting cost-partitioning is $\bar{b}_1^{\triangleright 1} = 1, \bar{b}_1^{\triangleright 2} = 1, \bar{b}_2^{\triangleright 1} = 2, \bar{b}_2^{\triangleright 2} = 0$ which gives $h^{\triangleright 1} + h^{\triangleright 2} = 3$, that is, the same value as OCP.

Orthogonal Abstractions

Let us, again, have a closer look on the i-projections as abstractions. What is the reason, that the i-projections cannot be admissibly summed by default? It is the use of the same actions (the public ones) in multiple abstractions. This means, that the abstractions are not orthogonal, formally:

Definition 85. (Orthogonal Abstractions) Let $\mathcal{T}_1, \mathcal{T}_2$ be two abstractions of a planning task Π with transition system $\mathcal{T}$ and let σ_1, σ_2 be their respective abstraction functions. The abstractions $\mathcal{T}_1, \mathcal{T}_2$ are orthogonal if for each transition $\langle s, l, s'\rangle$ in $\mathcal{T}$ holds $\sigma_k(s) = \sigma_k(s')$ for at least one $k \in \{1, 2\}$.

Orthogonal abstractions can be admissibly summed, according to the following proposition taken from Helmert et al. [2007].

Proposition 86. *(Helmert, Haslum & Hoffmann 2007) Let $\mathcal{T}_1, ..., \mathcal{T}_k$ be pairwise orthogonal abstractions of the same transition system $\mathcal{T}$ and let $h_1, ..., h_k$ be admissible heuristics computed on the respective transition systems. Then $\sum_{l=1}^{k} h_l$ is an admissible heuristic.*

This means, that in order to be admissibly summed, each action has to be represented by a loop in all but one abstraction. In the i-projected problems, the only problematic actions are projections of public actions, which are counted (non-loop) in each of the projections. Instead, we can consider the i-private projections and the public projection (Definition 32). By computing an admissible heuristic on the public projection and on each of the i-private projections and summing the results, we obtain an admissible heuristic.

Theorem 87. Let $\mathcal{T}(\Pi^{\mathsf{G}})$ be the transition system of the global problem Π^{G}, $\mathcal{T}(\Pi^{\triangledown i})$ the transition system of the i-private projected problem $\Pi^{\triangledown i}$ for each $1 \leq i \leq n$ and $\mathcal{T}(\Pi^{\triangleright})$ the transition system of the public projection $\Pi^{\triangleright}$. Let $h^{\triangledown 1}, ..., h^{\triangledown n}, h^{\triangleright}$ be admissible heuristics computed on the respective projections. Then $h^{\triangleright} + \sum_{i=1}^{n} h^{\triangledown i}$ is an admissible heuristic.

Proof. The public and i-private projections are abstractions by the same reasoning as in Theorem 36. They are (pairwise) orthogonal from definition and from $\mathcal{O}^i \cap \mathcal{O}^j = \emptyset$ for each $j \neq i$ and $\mathcal{O}^{\mathsf{pub}_i} \cap \mathcal{O}^{\mathsf{priv}_i} = \emptyset$ for each i, thus by application of Proposition 86 the theorem holds. □

A question remains, whether the i-private and public projections can be expressed in the form of cost partitioning of the i-projected problems. An obvious answer is yes, they can. By setting $\mathsf{cp}_i(a^{\triangleright i}) = 0$ for all i-projections of public actions a, the heuristic computed on $\Pi^{\triangleright i}$ using cp_i as a cost function ignores the public actions as if they were self-loops and thus computes the heuristic on $\Pi^{\triangledown i}$. Similar treatment of private actions (i.e. retaining costs only of i-projections of public actions) leads to computation of the heuristic on the public projection $\Pi^{\triangleright}$.

In order to maintain only the n cost-partitioned problems, one of the agents (say α_j) may keep the problem not partitioned, resulting in a heuristic $h^{\triangleright j} + \sum_{i=1, i\neq j}^{n} h^{\triangledown i}$, where $h^{\triangleright j}$ is an admissible heuristic computed on the j-projected problem $\Pi^{\triangleright j}$. In such case, the orthogonality of abstractions still holds and thus the resulting heuristic is also admissible (and possibly more informative).

Example. (CP) Let us now apply this approach on the running example. A public projection $\Pi^{\triangleright}$ of the problem reflects only the variable V_3 and actions $b_1^{\triangleright}, b_2^{\triangleright}$. The transition system $\mathcal{T}(\Pi^{\triangleright})$ has two states, an initial state $s_I^{\triangleright}$ where $V_3 = u$ and a goal state $s_\star^{\triangleright}$ where $V_3 = g$. There are two transitions (one for each actions) from $s_I^{\triangleright}$ to $s_\star^{\triangleright}$ with cost 2, thus the cost of optimal solution is $h^{\triangleright}(s_I^{\triangleright}) = 2$. A 1-private projection $\Pi^{\triangledown 1}$ reflects only the variable V_1, thus has two states, which are both goal states (the goal condition is empty). Thus $h^{\triangledown 1} = 0$ and similarly $h^{\triangledown 2} = 0$, resulting in total estimate of $h^{\triangleright} + h^{\triangledown 1} + h^{\triangledown 2} = 2$. Using $h^{\triangleright 1}$ instead of $h^{\triangleright} + h^{\triangledown 1}$ does not help in this particular case as $h^{\triangleright 1} = 2$.

Ad-hoc Cost Partitioning

So far, we have presented a number of more or less involved multi-agent cost-partitioning schema, but more trivial approaches should not be omitted. First is the very baseline uniform cost-partitioning, where

$$\mathsf{cp}_j(a^{\triangleright j}) = \frac{\mathsf{cost}^i(a^{\triangleright i})}{n}$$

for each action $a \in \mathcal{O}^{\mathsf{pub}_i}$ and each agent $\alpha_j \in \mathcal{A}$. Private actions are not partitioned as in the other cases.

Often, the costs of plans using projections of other agent's actions are underestimated as the cost of their private preconditions (that is the cost of private actions achieving them) is not reflected. The aim of presented cost-partitioning techniques is to balance this out. Instead of complex optimization, a simple rule of thumb may work in many cases. We denote such simple approach as projection-compensating cost-partitioning and base it on the following equation

$$\begin{array}{rcll} \mathsf{cp}_j(a^{\triangleright j}) & = & \frac{1-k}{n-1}\mathsf{cost}^i(a^{\triangleright i}) & \text{for } j \neq i \\ \mathsf{cp}_i(a^{\triangleright i}) & = & k\mathsf{cost}^i(a^{\triangleright i}) & \end{array} \tag{6.5.4}$$

where $k \in \langle 0, 1\rangle$. For $k = 1/n$, we obtain the uniform cost-partitioning. For $k = 0$, the cost of action $a^{\triangleright i}$ s.t. $a \in \mathcal{O}^i$ in $\Pi^{\triangleright i}$ is 0 and the cost is uniformly distributed among all other agents. For $k = 1$, the cost is retained by the owner agent and the costs of projections are 0. In general, as k is the same for all actions, the OCP cannot be achieved.

Example. (CP) On the running example, the uniform CP results in $\mathsf{cp}_1(b_1^{\triangleright 1}) = 1, \mathsf{cp}_1(b_2^{\triangleright 1}) = 1$ and $\mathsf{cp}_2(b_1^{\triangleright 2}) = 1, \mathsf{cp}_2(b_2^{\triangleright 2}) = 1$. The sum of optimal costs computed on such cost-partitioning is 2. We

obtain the same result for $k = 0$, where $\mathsf{cp}_1(b_1^{\triangleright 1}) = 0, \mathsf{cp}_1(b_2^{\triangleright 1}) = 2$ and $\mathsf{cp}_2(b_1^{\triangleright 2}) = 2, \mathsf{cp}_2(b_2^{\triangleright 2}) = 0$ and 0 for $k = 1$. In this particular example, we can express the OCP by setting $k = 3/4$, where $\mathsf{cp}_1(b_1^{\triangleright 1}) = 0.5, \mathsf{cp}_1(b_2^{\triangleright 1}) = 1.5$ and $\mathsf{cp}_2(b_1^{\triangleright 2}) = 1.5, \mathsf{cp}_2(b_2^{\triangleright 2}) = 0.5$ and the resulting cost of the sum of optimal solutions is 3.

6.6 Evaluation

In this section, we evaluate the presented admissible heuristics. In general, the most important comparison is with the projected versions of the heuristics as the distributed heuristics are typically expected to be better informed, but may incur more communication and thus be slower to compute.

The admissible heuristics were evaluated on implementation of the MAD-A* [Nissim and Brafman, 2012] algorithm in the MAPlan Planner [Fišer et al., 2015][2]. MAPlan is a multi-agent planner implemented in C which can run both in multi-threaded setting (somewhat similar to the MADLA Planner described in Chapter 5) and in a truly distributed setting, where each agent runs on its own machine (as in the CoDMAP competition distributed track). Unlike the MADLA Planner, MAPlan implements the optimal MAD-A* algorithm. The MAPlan planner takes as input either the un-factored or factored MA-PDDL definition of the planning problem and domain (see Appendix A for details on MA-PDDL), performs a distributed translation to create an MPT (or SAS+) representation for each agent and finally runs the MAD-A* algorithm. This is a significant difference from the original MAD-A* implementation by Nissim&Brafman, where, like in the MADLA Planner, the translation from PDDL to MPT is centralized and the factorization is automated based on additional agent definition (ADDL) file.

6.6.1 Evaluation of the Distributed LM-Cut Heuristic

In this section, we evaluate the distributed and projected variants of the LM-Cut heuristic. Each run (per problem) of the planner was limited to 60 min. and 4GB of memory (total for all agents) on a 16 core machine. The used benchmarks are described in Section 3.6 in full detail.

The results of the experiments are summarized in Table 6.1. The coverage results (the number of problems solved for each domain) show that except for three domains, the distributed $h^{\mathsf{G}i}_{\mathsf{LM\text{-}Cut}}$ solves more (or the same) problems and solves also a more of problems in total. The depot, driverlog, and sokoban domains are tightly coupled (as in [Brafman and Domshlak, 2008]) and most of the information is public, which means that the projected $h^{\triangleright i}_{\mathsf{LM\text{-}Cut}}$ has the same information as $h^{\mathsf{G}i}_{\mathsf{LM\text{-}Cut}}$, moreover, $h^{\mathsf{G}i}_{\mathsf{LM\text{-}Cut}}$ has to handle a lot of projected actions.

As expected, both variants of the $h_{\max}$ heuristic perform significantly worse (total coverage of 59 for the projected and 53 for the distributed version) and are not presented in the table. Except for the sokoban domain (coverage 7 for $h^{\triangleright i}_{\max}$ and 0 for $h^{\mathsf{G}i}_{\max}$) and elevators domain (coverage 0 for $h^{\triangleright i}_{\max}$ and 2 for $h^{\mathsf{G}i}_{\max}$), the difference between $h^{\triangleright i}_{\max}$ and $h^{\mathsf{G}i}_{\max}$ is not significant.

To understand the cause of the behavior of $h^{\triangleright i}_{\mathsf{LM\text{-}Cut}}$ and $h^{\mathsf{G}i}_{\mathsf{LM\text{-}Cut}}$ better, we have extracted the heuristic values computed for the initial state by both heuristics (for $h^{\triangleright i}_{\mathsf{LM\text{-}Cut}}$ taking an average for all agents), computed a ratio $h^{\triangleright i}_{\mathsf{LM\text{-}Cut}}/h^{\mathsf{G}i}_{\mathsf{LM\text{-}Cut}}$ for each problem and averaged the ratios per domain. The results are in Table 6.1 in the column labeled $\hat{h}_{\mathsf{LM-Cut}}$ (computed from all problems for which the init. state heuristic values were obtained). The results for coverage show that, in the tightly coupled domains, the distributed evaluation does not improve the heuristic estimate enough to justify the communication overhead. On the other hand, as the ratio drops below approx. 0.8, the improved heuristic accuracy overweight the communication overhead.

For more detailed view, the heuristic ratios aggregated in the column labeled $\hat{h}_{\mathsf{LM-Cut}}$ are plotted per-problem in Figure 6.6.1. The plot raises a question whether the distributed heuristic should not dominate the projected one as was shown for $h_{\max}$ in Lemma 64. Against intuition, the answer is no.

[2] http://github.com/danfis/maplan

domain	$h^{\triangleright i}_{\text{LM-Cut}}$	$h^{\mathsf{G}i}_{\text{LM-Cut}}$	$\hat{h}_{\text{LM-Cut}}$	$\hat{e}_{\text{LM-Cut}}$	$\hat{t}_{\text{LM-Cut}}$	$\hat{t}^s_{\text{LM-Cut}}$
elevators08 (20)	**2**	**2**	0.18	39.9	0.15	0.01
logistics00 (20)	6	**12**	0.26	2521.7	4.78	0.01
zenotravel (18)	6	**10**	0.41	142.4	0.74	0.03
rovers (18)	**6**	**6**	0.52	33.7	0.45	0.15
blocksworld (30)	17	**20**	0.54	45.5	0.56	0.09
satellites (18)	5	**10**	0.55	63.9	0.52	0.03
driverlog (20)	**13**	12	0.8	4.4	0.22	0.12
depot (20)	**7**	4	0.88	1.4	0.13	0.11
woodwork.08 (20)	6	**8**	0.88	12	1.11	0.35
sokoban (10)	**8**	5	1	1	0.15	0.14
total (194)	76	**89**	-	-	-	-

Table 6.1: Coverage and average of $h^{\triangleright i}_{\text{LM-Cut}}/h^{\mathsf{G}i}_{\text{LM-Cut}}$ ratios for initial state heuristic ($\hat{h}_{\text{LM-Cut}}$), expanded states ($\hat{e}_{\text{LM-Cut}}$), total planning time ($\hat{t}_{\text{LM-Cut}}$) and time per expanded state ($\hat{t}^s_{\text{LM-Cut}}$).

The reason lies in the inherent variance of the $h_{\text{LM-Cut}}$ heuristic depending on the tie-breaking behavior of the precondition choice function (pcf). Although in the proofs of equality of $h_{\text{LM-Cut}}$ and $h^{\mathsf{G}i}_{\text{LM-Cut}}$ it was possible to fix the tie-breaking behavior (thanks to the use of distributed h_{max}), it is not the case with $h^{\triangleright i}_{\text{LM-Cut}}$. The fact p which maximizes $h_{\max}$ in $h^{\triangleright i}_{\text{LM-Cut}}$ may not maximize it in $h^{\mathsf{G}i}_{\text{LM-Cut}}$ (or vice versa) therefore the same fact p could not be chosen in both. If most of the actions in the problem are public, this may lead to a situation that for some state $h^{\triangleright i}_{\text{LM-Cut}} > h^{\mathsf{G}i}_{\text{LM-Cut}}$, as can be seen in Figure 6.6.1 for some of the depot problems.

The quality of heuristic estimates can be assessed by the number of expanded states. In Table 6.1, the column $\hat{e}_{\text{LM-Cut}}$ shows the average ratio of expanded states, restricted to problems solved by both heuristics. The most significant improvement is in the logistics domain, where $h^{\triangleright i}_{\text{LM-Cut}}$ expands over $2500\times$ more states than $h^{\mathsf{G}i}_{\text{LM-Cut}}$, followed by zenotravel and satellites with approx. $140\times$ and $60\times$ increase of expanded states over the distributed heuristic respectively. The limiting factor is $30\times$ in rovers domain where the quality of the distributed heuristic just eliminates the overhead of the distributed heuristic, below this factor, the projected heuristic exhibits better performance.

The total planning time ($\hat{t}_{\text{LM-Cut}}$ in Table 6.1) and time per expanded state ($\hat{t}_{\text{LM-Cut}}$ in Table 6.1) were treated similarly, computed only from problems solved by both heuristics. The results show that, except for the logistics and woodworking domains, the projected heuristic leads to approx. $2\times$–$10\times$ faster solution, which was not unexpected. The average time spent on an expanded state shows that the projected heuristic is $10\times$–$100\times$ faster. Even though, the added value of better heuristic estimates is crucial in many domains, most notably logistics. Notice that in domains with the largest difference in the number of expanded states, the projected heuristic is significantly faster. This suggests that the projected problem is much simpler, but ignores a lot of important information thus makes the resulting heuristic estimate much less accurate.

A specific case is the woodworking domain. Even though the distributed heuristic estimates are only slightly better than the projected ones, the distributed heuristic solves more problems as the projected heuristic is only $3\times$ faster and ignores important information.

The structural properties causing the success of the $h^{\mathsf{G}i}_{\text{LM-Cut}}$ heuristic are closely related to the motivation example in the introduction. An example is the logistics domain, where trucks and planes are moving packages from starting to goal locations. In the MA-STRIPS formulation, the location of a package is public only when it is at an intermediate (or goal) location, it is not known when loaded onto some vehicle. The unload action of a vehicle is seen by other agents with a precondition only on the location of the vehicle, having the package actually loaded is not required. The cost of getting the package to a location where it can be loaded and loading it is lost in the projected problem, enabling the

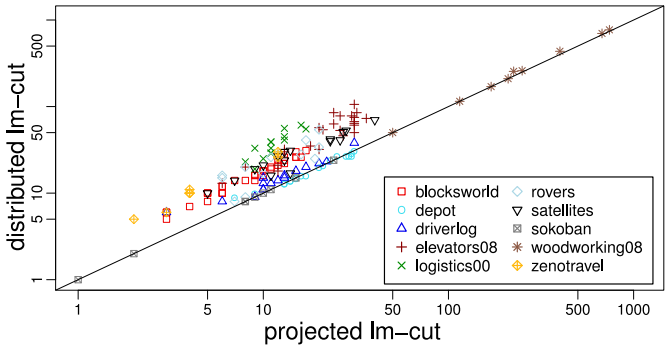


Figure 6.6.1: Per-problem $\hat{h}_{\mathsf{LM-Cut}} = h^{\triangleright i}_{\mathsf{LM\text{-}Cut}}/h^{\mathsf{G}i}_{\mathsf{LM\text{-}Cut}}$ ratios for initial state.

agents to have a package cheaply unloaded at their loading site by a projected unload action of some other agent. This is exactly the principle demonstrated in the motivation example. A similar situation occurs in the elevators domain and other loosely coupled domains.

6.6.2 Evaluation of the Distributed Potential Heuristics

In this section, we evaluate the distributed potential heuristic. For this evaluation, we exactly replicate the distributed track setup of the CoDMAP[3] [Komenda et al., 2016] competition including all 12 benchmarks, described in Section 3.6. Each agent runs on a separate machine with i5-4460 3.4GHz processor and 8GB memory and has its own problem and domain input files. The agents communicate via TCP/IP on Gigabit Ethernet. Each run is limited to 30min.

We compare the following approaches to the computation of the potential heuristic in the multi-agent setting (as an LP solver we use CPLEX 12.6.1):

$h^{\triangleright}_{\mathsf{pot}\text{-}s_I}$ The projected heuristic, that is each agent α_i computes the heuristic on its own α_i-projected problem $\Pi^{\triangleright i}$ (as in MAD-A*). The LP is optimized for the initial state s_I.

$h^{\triangleright}_{\mathsf{pot}\text{-}S}$ The projected heuristic, the LP is optimized for all syntactic states.

$h^{\mathsf{maxproj}}_{\mathsf{pot}\text{-}S}$ The heuristic computed as a maximum of projections. The public potentials are shared. The LP is optimized for all syntactic states.

$h^{\mathsf{G}}_{\mathsf{pot}\text{-}s_I}$ The distributed global heuristic, the LP is optimized for the initial state s_I and with no encryption.

$h^{\mathsf{G}}_{\mathsf{pot}\text{-}S}$ The distributed global heuristic, the LP is optimized for all syntactic states and with no encryption.

$h^{\mathsf{G-sec}}_{\mathsf{pot}\text{-}S}$ The distributed global heuristic, the LP is optimized for all syntactic states and with encryption based on [Mangasarian, 2011].

In the case of the secure computation based on [Mangasarian, 2011], several additional matrix multiplications are performed, which does not pose significant computational overhead, as the LP computation itself is a minor part of the planning process. Also, the amount of communication is the same as when using the plain LP approach. We have measured the overhead of the secure LP computation with the following results. Over all problems, the plain LP computation takes on average 450ms, with maximum 4.5s, while the secure LP computation takes on average 520ms, with maximum 5s. This means that the

[3] http://agents.fel.cvut.cz/codmap

domain	$\|\mathcal{A}\|$	$h^{\triangleright}_{\text{pot-}S}$	$h^{\text{maxproj}}_{\text{pot-}S}$	$h^{\mathsf{G}}_{\text{pot-}S}$	$h^{\mathsf{G-sec}}_{\text{pot-}S}$	$h^{\triangleright}_{\text{LM-Cut}}$	$h^{\mathsf{G}}_{\text{LM-Cut}}$
blocksworld	4	4	4	**13**	6	2	1
depot	$5-10$	6	6	**7**	4	6	2
driverlog	$2-8$	**15**	14	**15**	13	**15**	10
elevators08	4	2	2	2	2	**2**	0
logistics00	$3-7$	4	6	**7**	6	5	5
rovers	$4-10$	1	1	1	1	1	1
satellites	$3-8$	1	1	1	1	2	**3**
sokoban	$2-4$	**13**	**13**	**13**	12	**13**	4
taxi	$4-10$	**20**	19	**20**	**20**	**20**	14
wireless	$6-10$	2	2	2	2	**4**	3
woodw.08	7	4	4	4	4	**4**	5
zenotravel	$2-6$	6	6	6	6	6	6
total		78	78	**91**	77	80	54

Table 6.2: The number of solved problems (out of 20 per domain) ($h^{\triangleright}_{\text{pot-}s_I}$ and $h^{\mathsf{G}}_{\text{pot-}s_I}$ solved 74 and 90 problems respectively).

secure LP computation is on average only $1.15\times$ slower than the plain LP computation. The absolute numbers show the impact on the 30min time limit is negligible for both variants of the LP computation.

This somewhat contrasts with the results shown in Table 6.2. The coverage of secure $h^{\mathsf{G-sec}}_{\text{pot-}S}$ is nearly 15 problems less than the non-secure $h^{\triangleright}_{\text{pot-}S}$. Although the secure LP transformation guarantees to return optimal solution, it does not guarantee to return the same values for the LP variables (the values depend on the randomly generated matrices B_i), which may differ but still yield the same optimization function value. As different values of potentials give different heuristic estimates for the same states, the overall performance of the planner may also differ. We present a detailed statistical evaluation of this phenomenon later in this section.

The secure computation based on [Dreier and Kerschbaum, 2011] was not part of the experimental evaluation. Although this variant requires more matrix operations than [Mangasarian, 2011], we still assume the overhead to be minimal in comparison with the MAD-A* search. Note, that vast majority of the unsolved problems are unsolved due to memory consumption and not due to reaching the time limit.

In Table 6.2, we present the numbers of solved problems for the variants of the potential heuristics for each competition domain ($h^{\triangleright}_{\text{pot-}s_I}$ and $h^{\mathsf{G}}_{\text{pot-}s_I}$ solved 74 and 90 problems in total respectively). The results show that the heuristics optimized for the set of all syntactic states (S) perform slightly better, as expected. Also, the global heuristics perform better than the projected variants, as they are better informed, but does not cause any communication overheads. Even the $h^{\text{maxproj}}_{\text{pot}}$ variant does not bring any substantial improvement, as the global information is still missing there. The ratio of expanded states of $h^{\triangleright}_{\text{pot-}S}$ vs. $h^{\mathsf{G}}_{\text{pot-}S}$ in Figure 6.6.2 shows that the informativeness of the global and projected heuristics is similar, except for the blocksworld, depot and logistics00 domains, which corresponds with the coverage results. In a few problems, the projected heuristic offers better guidance.

Comparison with the state of the art

Finally, we compare the MAD-A* search using the global ($h^{\mathsf{G}}_{\text{pot-}S}$) potential heuristic with the state of the art. Namely, we compare it with the best performing distributed optimal multi-agent planner [Fišer et al., 2015] in CoDMAP, using a projected ($h^{\triangleright}_{\text{LM-Cut}}$) and global distributed ($h^{\mathsf{G}}_{\text{LM-Cut}}$) versions of the LM-Cut heuristic. Notice that the results of the LM-Cut heuristic significantly differ from those presented in Section 6.6.1. This is because the results in Section 6.6.1 were measured on MAPlan running all agents on a single machine (even though still communication via TCP/IP). The results in this section were, similarly as in the CoDMAP competition measured on MAPlan running on a distributed system

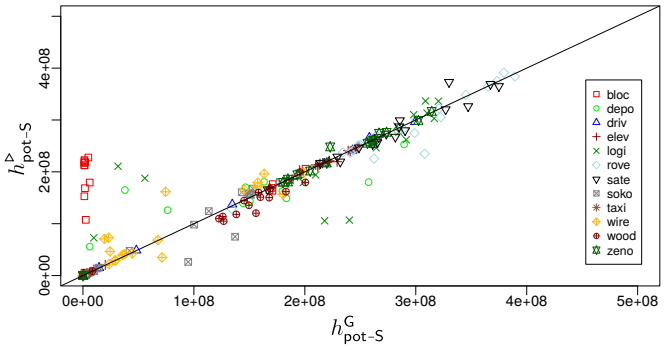


Figure 6.6.2: Ratios $h^{\triangleright}_{\text{pot-}S}/h^{\mathsf{G}}_{\text{pot-}S}$ of expanded states per problem.

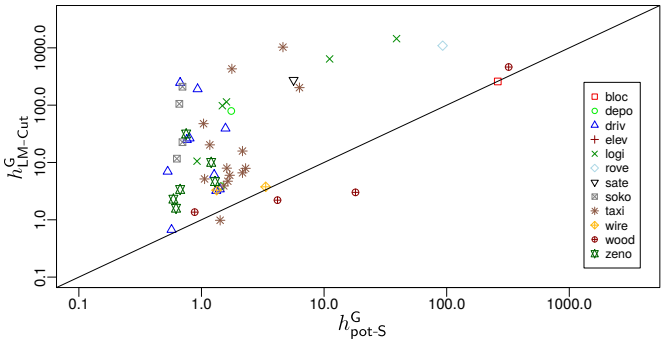


Figure 6.6.3: Time ratios $h^{\mathsf{G}}_{\text{LM-Cut}}/h^{\mathsf{G}}_{\text{pot-}S}$ per problem.

(one agent per machine) communicating over a local network. This setup makes lower communication overhead much more crucial.

Comparison of the number of problems solved by the planners is shown in Table 6.2. Whereas the performance of the projected heuristics $h^{\triangleright}_{\text{pot-}S}$ and $h^{\triangleright}_{\text{LM-Cut}}$ is on par, the global versions indeed show the strength of $h^{\mathsf{G}}_{\text{pot-}S}$, which is more informed than $h^{\triangleright}_{\text{pot-}S}$ but does not incur any additional computation or communication costs as the potential heuristic is additive. This results in a better coverage than other compared heuristics, especially the global version of LM-Cut, where the difference is over 40 problems in total.

In order to emphasize the results, let us compare results for the classical centralized versions of the heuristics in the literature. In [Pommerening et al., 2014b], the LM-Cut is reported to have coverage of 763, whereas in [Seipp et al., 2015] the potential heuristic optimized for initial state and all syntactic states have coverage of 611 and 659 respectively (in the same experimental setting). This illustrates that although in the centralized setting, the LM-Cut heuristic performs significantly better, in the distributed setting, the properties of $h^{\mathsf{G}}_{\text{pot}}$ give it a significant advantage. More advanced techniques for computing the optimization function proposed in [Seipp et al., 2015] would probably improve the results of $h^{\mathsf{G}}_{\text{pot}}$ even more.

In Table 6.3, the IPC Agile scores for each of the configurations are shown. The score is computed as a sum over all problem scores. For a given problem let T^* be the minimum time required by any planner to solve the problem. A configuration that solves the problem in time T gets a score of $1/(1 + \log_{10}(T/T^*))$ for the problem. Search guided by any variant of the potential heuristic is faster (have a higher score) than the projected LM-Cut heuristic and significantly faster than the global LM-Cut heuristic. Results for the global variants of the potential and LM-Cut heuristics are shown in Figure 6.6.3

	$h^{\triangleright}_{\text{pot-}S}$	$h^{\mathsf{G}}_{\text{pot-}S}$	$h^{\mathsf{G-sec}}_{\text{pot-}S}$	$h^{\triangleright}_{\text{LM-Cut}}$	$h^{\mathsf{G}}_{\text{LM-Cut}}$
score	56.5	**57.9**	54.5	48.8	22.7

Table 6.3: IPC Agile Score. A configuration that solves a problem in time T gets a score of $1/(1+\log_{10}(T/T^*))$ where T^* is the best solution time of any configuration. A higher number means faster solutions.

domain	$\lvert\mathcal{A}\rvert$	$h^{\mathsf{G-sec}}_{\text{pot-}s_I}$		$h^{\mathsf{G}}_{\text{pot-}s_I}$		$h^{\mathsf{G-sec}}_{\text{pot-}S}$		$h^{\mathsf{G}}_{\text{pot-}S}$	
blocksworld	4	12	(6.2)	13	(13)	12	(6.6)	13	(13)
driverlog	$2-8$	14	(11.8)	15	(15)	15	(12.2)	15	(14.8)
logistics00	$3-7$	8	(5.8)	8	(7.8)	6	(5.4)	6	(6)
rovers	$4-10$	0	(0)	0	(0)	1	(0.8)	1	(1)
satellites	$3-8$	0	(0)	0	(0)	1	(1)	1	(1)
sokoban	$2-4$	12	(11.8)	12	(11.6)	13	(13)	13	(13)
taxi	$4-10$	20	(20)	20	(20)	20	(19.8)	20	(19.8)
zenotravel	$2-6$	7	(6.2)	9	(9)	6	(6)	6	(6)
total		73	(61.8)	77	(76.4)	74	(63.8)	75	(74.6)

Table 6.4: Coverage results of the following variants of the potential heuristic:
$h^{\mathsf{G-sec}}_{\text{pot-}s_I}$ - secure variant optimized for the initial state
$h^{\mathsf{G}}_{\text{pot-}s_I}$ - non-secure variant optimized for the initial state
$h^{\mathsf{G-sec}}_{\text{pot-}S}$ - secure variant optimized for all syntactic states
$h^{\mathsf{G}}_{\text{pot-}S}$ - non-secure variant optimized for all syntactic states
The numbers in brackets denote the average coverage over the 5 runs. The numbers without the brackets show the coverage where a problem was counted as solved if it was solved in at least one of the runs. The domains where no configuration solved any problem were excluded from the table.

as per-problem ratios (restricted to problems solved by both). With a small number of exceptions, the $h^{\mathsf{G}}_{\text{pot-}S}$ heuristic guided the search much faster.

Statistical Evaluation

As the nature of the privacy-preserving algorithm is stochastic, we provide a statistical evaluation in addition to the original evaluation published in [Štolba et al., 2016a]. Although we aim for a statistical significance, it is important to mention that the evaluation technique based on the CoDMAP competition takes an enormous time and cannot be very well outsourced to established cloud computing services because of the need of precise control of the network and the machines used for computation. This means that we could run only a relatively small number of experiments, but nevertheless, such evaluation brought more light into the issue. This round of experiments was run on 40 machines[4] with Intel i5-4460 CPUs at 3.2 GHz, 16GB of RAM per agent, 5 runs per configuration (that is, all combinations of secure, not secure, optimized for the initial state, and optimized for all syntactic states).

The Table 6.4 shows that in terms of coverage, the secure variant of the heuristic is on average about 15% worse than the non-secure variant. This confirms the results shown in Table 6.2 as being not caused by chance. If we count any problem which was solved in at least one run as solved, we can see, that the results of the secure variant are still worse in the initial state variant and practically equal in the all syntactic states variant. Nevertheless, we conclude that some of the problems are often not solved by the secure variant.

As we have already discussed, lower coverage of the secure variant cannot be caused by the time overhead of secure computation because a) it is negligible, and b) most of the problems are not solved because of running out of memory. This clearly points to a difference in the quality of the heuristic.

[4]Note that the machines and the network setup were different than in the previous experiments.

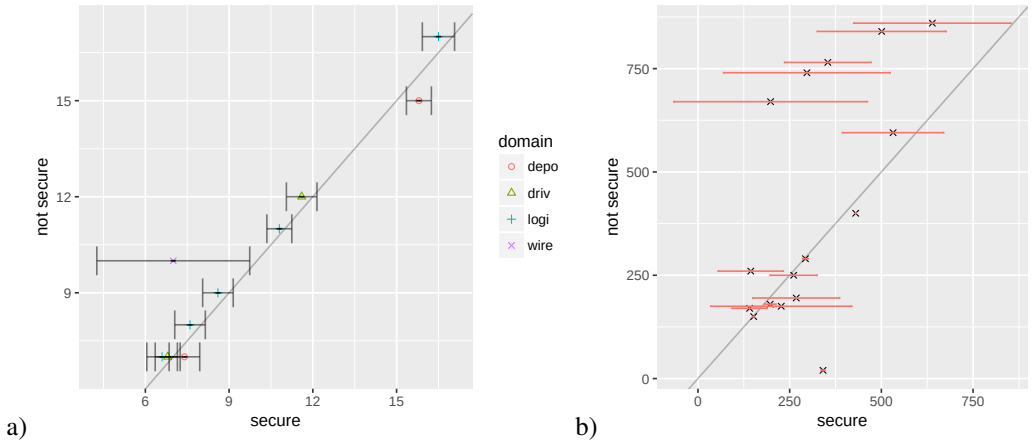


Figure 6.6.4: Heuristic values of the initial state as computed by the secure $h^{\mathsf{G-sec}}_{\text{pot-}S}$ and non-secure $h^{\mathsf{G}}_{\text{pot-}S}$ heuristics. The heuristic values (for the secure variant) include error bars ($+/-$ standard deviation) and are shown only for the problems where the values are not equal. Plot a) include all domains except for woodworking, b) shows the woodworking domain.

Let us now focus on the heuristic values of the initial states. In the case of the heuristics $h^{\mathsf{G}}_{\text{pot-}s_I}$ and $h^{\mathsf{G-sec}}_{\text{pot-}s_I}$, the heuristic values are equal for all runs. This verifies, that both the secure and non-secure LPs return the same optimal value. The case of the secure $h^{\mathsf{G-sec}}_{\text{pot-}S}$ and non-secure $h^{\mathsf{G}}_{\text{pot-}S}$ is different in that different optimal results of the LP computation might result in different heuristic values. Such values are shown in Figure 6.6.4, divided into two plots because of the order of magnitude difference of the heuristic values in woodworking. The figure shows that in multiple domains there are problems where the secure heuristic gives worse estimates and a couple of problems where it gives better estimates. The most significant case is the woodworking domain where is the largest number of problems where the values differ, but again, not always is the secure heuristic worse.

To understand the situation even better, we look at the number of expanded states, Figure 6.6.5. The results show that, ignoring the noise caused by the distributed computation, the only significant cases are the blocksworld and depots domains and a couple of isolated problems from other domains. It is safe to say, that in all significant cases, the secure heuristic provides worse guidance. This conveys with the coverage results, where the largest difference between the secure and non-secure variant is in the blocksworld domain. The results show, that against all expectations, the secure computation provides worse heuristic guidance.

The most reasonable explanation for this behavior is that due to the randomization, the resulting LPs are less numerically stable and the computed potentials are more likely (in some domains) to be prune to rounding errors. Similarly to the classical potential heuristic implemented in Fast Downward, the heuristic values are rounded up before use in the search, the difference is, that in the distributed computation, we use Equation 6.4.5 to compute the heuristic value from the parent state. In the current implementation in MAPlan, the parent heuristic is integer as in MAD-A* search. Therefore, the new heuristic value has to be rounded down, otherwise, the rounding error might accumulate. For some values of the potentials, rounding down might cause deterioration of the heuristic value throughout the search. Alternatively, sending the non-integer value together with the state and rounding up only after the use of Equation 6.4.5 might solve the issue.

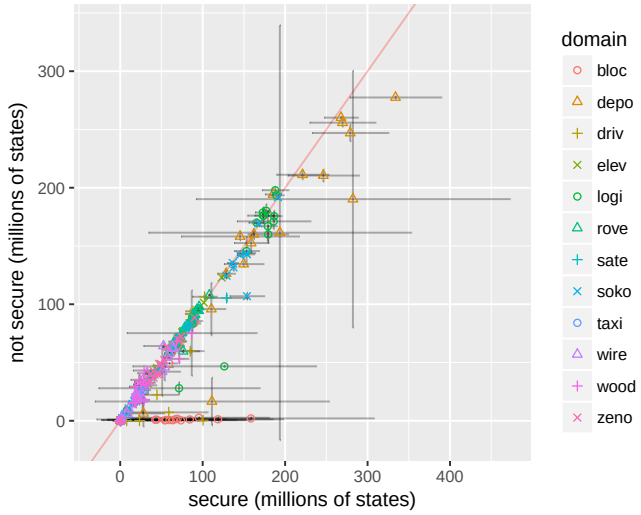


Figure 6.6.5: Expanded states in MAD-A* search using the secure $h^{\mathsf{G}-\mathsf{sec}}_{\text{pot-}S}$ and non-secure $h^{\mathsf{G}}_{\text{pot-}S}$ heuristics (also for unsolved instances). Each point also includes error bars ($+/-$ standard deviation) over all runs.

6.6.3 Evaluation of Multi-Agent Cost Partitioning

In this section, we present the evaluation of the multi-agent cost-partitioning based approach to computing distributed heuristic presented in Section 6.5. We aim to evaluate the feasibility of the approach by evaluating the quality of the heuristic given by the cost-partitioning. In order to do so, we compute the optimal heuristic h^* (that is, the true cost of the optimal solution) for each agent problem Π^i. As a baseline, we use the maximum of the optimal projected heuristics for the initial state, that is,

$$h^*_{\mathsf{max}}(s_I) = \mathsf{max}_{1\leq i\leq n} h^{*\triangleright i}(s_I^{\triangleright i})$$

and compare it to the sum of the optimal heuristics for a given cost partitioning cp, that is,

$$h^*_{\mathsf{G}}(s_I) = \sum_{i=1}^{n} h^{*\triangleright i}_{\mathsf{cp}_i}(s_I^{\triangleright i})$$

In order to compare the heuristic values of different problems better, we use the ratios $h^*_{\mathsf{max}}(s_I)/h^*(s_I)$ and $h^*_{\mathsf{G}}(s_I)/h^*(s_I)$ where $h^*(s_I)$ is the optimal heuristic computed by a centralized planner on the global problem Π^{G}.

We start the evaluation with the optimal cost partitioning (OCP) described in Section 6.5.2. Figure 6.6.6 shows the resulting comparison of the heuristic values. Even though we were able to compute the OCP only for a small subset of the benchmark problems, the results show, that for a significant portion of the problems, the cost-partitioning based approach improves over the maximum of i-projected heuristics.

Unfortunately, the OCP computation is not feasible for such large abstractions as the i-projections typically are (in the worst case, they can be as big as the global problem). Next, we evaluate the cost partitioning computed on smaller abstractions. Figure 6.6.7a) shows the evaluation of an OCP computed on abstractions of Π^i with 100 states each, computed based on the Merge&Shrink algorithm [Helmert et al., 2007]. Clearly, this approach works well for some of the domains such as elevators, where we are able to obtain the global optimal heuristic h^*, but not so well for some domains such as driverlog. Overall, most of the domains benefit from this cost-partitioning based approach. We assume that it may

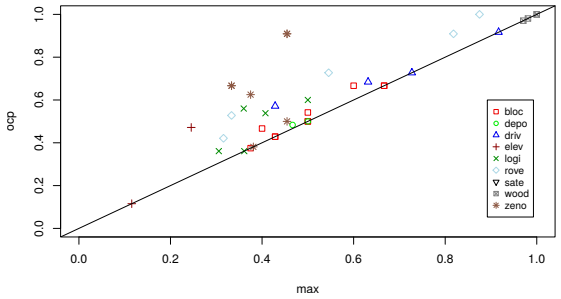
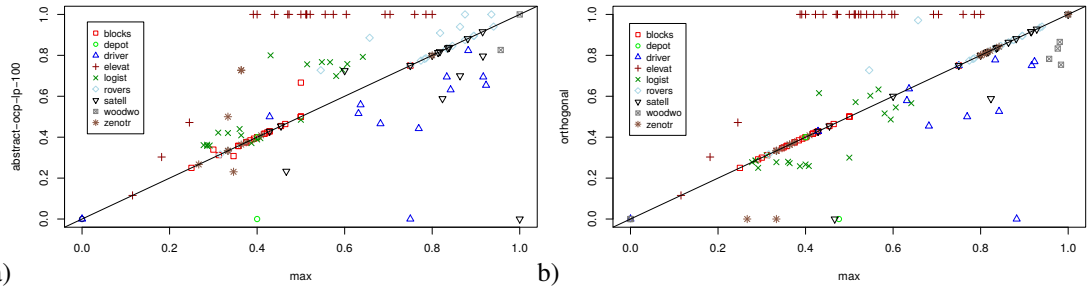


Figure 6.6.6: Comparison of $h^*_{\mathsf{max}}(s_I)/h^*(s_I)$ and $h^*_{\mathsf{G}}(s_I)/h^*(s_I)$ for optimal cost partitioning.



a) b)

Figure 6.6.7: Comparison of $h^*_{\mathsf{max}}(s_I)/h^*(s_I)$ and $h^*_{\mathsf{G}}(s_I)/h^*(s_I)$ for a) optimal cost partitioning on abstractions with 100 states and b) cost-partitioning based on orthogonality of i-private and public projections.

be possible to improve the results by providing a better abstraction or tailoring the abstraction specifically for the purpose of the multi-agent cost-partitioning. Simply increasing the number of abstract states does not seem to work, similar evaluation with abstractions containing 1000 states did not show any improvement. Figure 6.6.7b) shows the evaluation of the approach based on the orthogonality of the i-private and public projections. The results show that this approach is much less successful, even though it is still able to achieve the perfect global heuristic in the elevators domain.

Next, let us have a look on the approximations of the OCP based on other heuristics expressed as LPs. Figure 6.6.8a) shows the result for a cost-partitioning based on landmarks generated by the LM-Cut heuristic [Helmert and Domshlak, 2009]. In this case, only the elevators domain seems to be affected positively. Figure 6.6.8b) shows the results of a cost-partitioning generated based on the LP of the state equation heuristic (SEQ) [Van Den Briel et al., 2007]. Again, we can see that the results are not very promising, as only the logistics domain is affected overall positively. Finally, Figure 6.6.8c) shows the effect of using the cost-partitioning based on the potential heuristic LP. Out of the results presented in Figure 6.6.8, the potential heuristic based cost-paritioning performs the best. The only negative results are for most of the driverlog domain and a number of other isolated cases. For a number of problems not limited to the elevators domain, the cost-partitioning gives an optimal heuristic. Coupled with the ease of the LP formulation (in comparison to, e.g., computing an abstraction first), this approach seems to be practically the most promising.

We conclude the experiments by providing the results for the most simple cost-partitionings, the uniform and the projection-compensating, described in Section 6.5.3. Figure 6.6.9a) shows the results for the uniform cost-partitioning. The problems are indeed quite uniformly distributed among the problems where the cost-partitioning is beneficial and where they are worsening the heuristic estimate. An exception is the elevators domain as the uniform cost-partitioning is enough to provide the global optimum. The projection-compensating cost-partitioning shown in Figure 6.6.9b) is a rather extreme case.

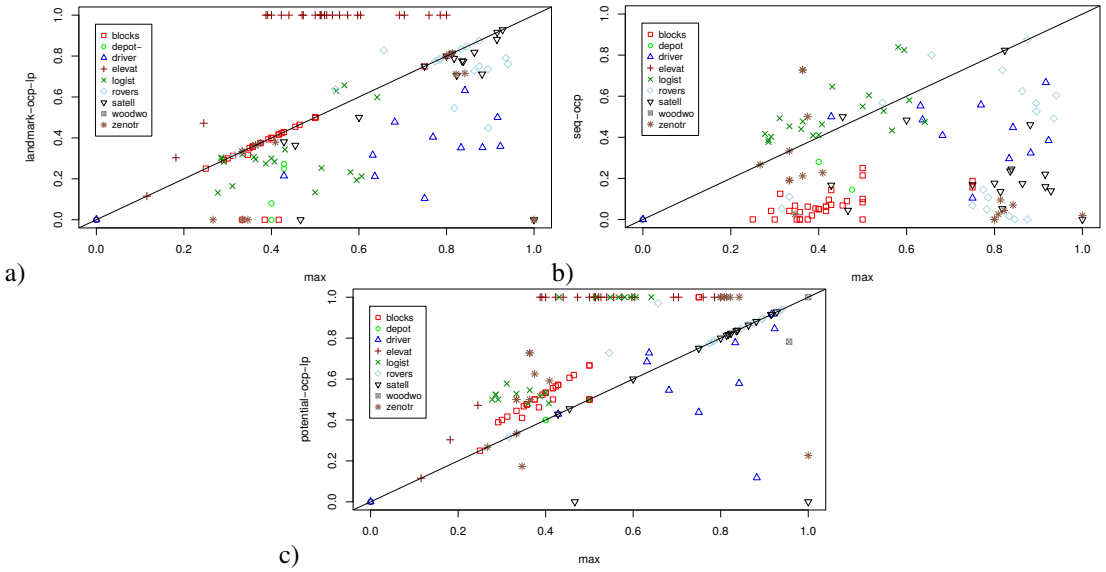


Figure 6.6.8: Comparison of $h^*_{\mathsf{max}}(s_I)/h^*(s_I)$ and $h^*_{\mathsf{G}}(s_I)/h^*(s_I)$ for a) cost-partitioning based on landmarks b) cost-partitioning based on the state equation heuristic LP and c) cost-partitioning based on the potential heuristic LP.

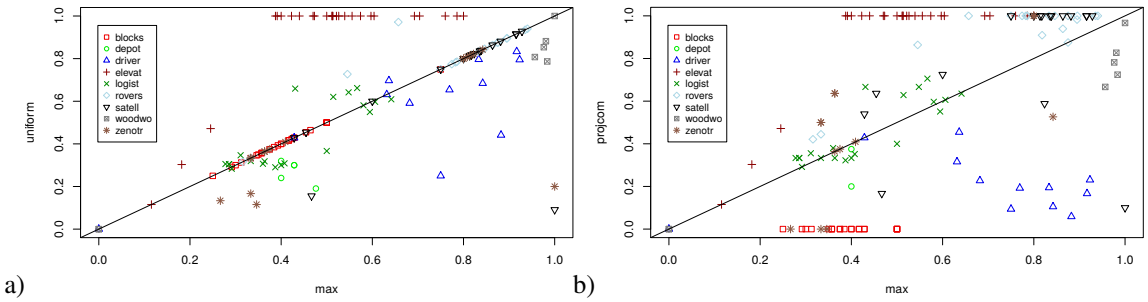


Figure 6.6.9: Comparison of $h^*_{\mathsf{max}}(s_I)/h*(s_I)$ and $h^*_{\mathsf{G}}(s_I)/h*(s_I)$ for a) uniform cost partitioning and b) projection-compensating cost-partitioning

There is a significant number of problems (and nearly entire domains) where it provides optimal global heuristic, e.g., the elevators, satellites (some problems), and depot. But in the other extreme, there are domains where it results in a zero heuristic, most prominently the blocksworld domain. In blocksworld, the behavior is caused by the fact that each of the agents is able to solve the problem on its own, thus when computing the heuristic, each agent considers the problem to be solvable by the other agents with a zero cost. It might be possible to mitigate such extreme cases by putting $k > 0$ in Equation 6.5.4.

6.7 Summary

Together with Chapter 4, this chapter provides a number of answers to the question of **(Objective 1)**, that is, how to compute heuristics in a distributed way. In this chapter, we have focused on admissible heuristics and shown that the distributed variants maintain admissibility, which is necessary for optimal multi-agent planning. The first heuristics we have treated are the max and LM-Cut heuristics, which are examples of admissible relaxation heuristics (the max heuristic was already treated in Chapter 4 but in

an inadmissible way). The approach for the relaxation heuristics was again based on the communication of reachable facts, found landmarks, etc. for each evaluated search state.

Next, we have taken a different approach in distributed computation of potential heuristics. We have shown, that the properties of potential heuristics allow us to compute them in an additive way, that is, each agent can compute its part separately and the partial results can be summed up to obtain the global estimate without any additional communication. The only distributed computation is the evaluation of the linear program which is necessary to compute the potentials. We have shown that such computation can be done in a privacy-preserving way using existing techniques.

Finally, we have proposed a general approach to heuristic computation based on cost partitioning, which is also additive and thus can reduce the amount of communication necessary. This approach maintains admissibility but can be used for inadmissible heuristics as well.

We have experimentally evaluated the $h_{\max}$, LM-Cut and potential heuristics both in the terms of projected and distributed computation and against each other. The results show that in a fast communication setting (such as local computer communication) the distributed LM-Cut heuristic beats its projected counterpart, but the same does not hold in a real network situation. On the contrary, the distributed potential heuristic can utilize its better estimates in a real network setting as well as there is no additional communication during the planning process. The evaluation of the general cost-partitioning approach showed that this approach is a very promising direction for research.

Chapter 7

Privacy

In the previous chapters, we have focused on analyzing what information can be shared in order to speed-up the search for a solution and to improve the quality of the heuristic estimates. In this chapter, we focus on the contrary, that is, how to prevent unintentional communication of information which was not meant to be shared.

Multi-agent planning is an appropriate approach in a number of situations. One adequate reason why to use multi-agent planning instead of classical planning is caused by the locality of information. This means that each of the agents has access to some information which would be too costly to communicate. Take, for example, a multi-robotic team where each robot receives a huge amount of sensoric data, but the communication is limited. A similar case might be the cost of formalization, that is, it may be too costly for an agent to formalize its inputs so that they can be shared with other agents (or a common formalism for a diverse set of agents might incur substantial complexity). These points are valid in general but are not applicable to the particular case of multi-agent planning, where the common formalism is defined and the inputs (formulated either as PDDL, STRIPS or MPT) are already concise representations and thus can be easily communicated.

In multi-agent planning, one of the reasons for distributed computation might be the aim of improving performance by exploiting the given factorization. Indeed, the parallel version of the MAD-A* algorithm denoted as MAP-A* [Nissim and Brafman, 2012] improves over the state of art in multi-core parallelized search, especially on well-decoupled domains such as satellites. Importantly, in the case of parallel computation, there is no need to compute the heuristic distributedly as it can be computed over the whole problem and thus provide better guidance.

We proceed to the most important reason which prevents any centralized computation and thus justifies the approach to multi-agent planning adopted in this thesis and that is the preservation of privacy. Take for example a scenario where two logistics companies would like to cooperate in order to improve their coverage of customers but which do not want to reveal sensitive information such as the locations of their customers or the costs of their transportation to each other. Similarly, in a military coalition mission, coordinated planning is necessary, but some information needs to be kept secret.

From cryptography and secure multi-party computation in particular, we learn that it is not enough to prevent the communication of private data as private information might be learned even from the execution of the algorithms and communication protocols (e.g., the received messages) themselves. In this chapter, we fulfill the **(Objective 3)** of this thesis

(Objective 3) How to formalize privacy and quantify privacy leakage and how to apply secure multi-party computation techniques in multi-agent planning?

by the precise definition of privacy in the context of MAP (Section 7.1), the definition of privacy leakage measure for privacy-preserving MAP (Section 7.2). We analyze privacy leakage of the MAD-A* and Secure-MAFS algorithms (Section 7.3), which is also applicable on the MADLA Search presented in

Chapter 5. Moreover, in Section 7.3 we provide a novel analysis and insight in privacy leakage of the distributed heuristics described in Chapters 4 and 6. We conclude this chapter by providing theoretical results about PP-MAP in general (Section 7.4).

7.1 Formal Definition of Privacy in Multi-Agent Planning

What exactly are the agents in privacy-preserving multi-agent planning (PP-MAP) trying to hide and why? Let us consider the following example. A company has a secret recipe for a well-known beverage. In order to work effectively, it wants to optimize its process of logistics and its use of subcontractors. In this example, parts of the recipe can be represented as actions either private (the most secret parts) or public (the "interface" with other companies). Other agents provide the actions for logistic transportation and providing ingredients. The beverage agent wants to hide all its private actions and the private parts of the public actions, namely the "signature" of the actions, that is, their preconditions and effects regardless of renaming (formally, an isomorphic model).

Any leakage of such information is undesirable, but the value of the information decreases with the increasing uncertainty of the exact isomorphic model. We now formalize the described concepts.

We borrow the formal treatment of privacy-preserving planning from [Nissim and Brafman, 2014] as follows:

Definition 88. Let $\mathcal{M} = \{\Pi_i\}_{i=1}^n$ be a MAP problem for a set $\mathcal{A}$ of agents. For each agent $\alpha_i \in \mathcal{A}$, the private part of its problem Π^i is

(i) the set of private variables $\mathcal{V}^{\mathsf{priv}_i}$, the number of private variables[1] $|\mathcal{V}^{\mathsf{priv}_i}|$, their respective domains $\mathsf{dom}(V)$ and their sizes $|\mathsf{dom}(V)|$ for each $V \in \mathcal{V}^{\mathsf{priv}_i}$,

(ii) the set of private operators $\mathcal{O}^{\mathsf{priv}_i}$, the number of private operators $|\mathcal{O}^{\mathsf{priv}_i}|$, the number and values of variables in $\mathsf{pre}(o)$ and $\mathsf{eff}(o)$ and the value of $\mathsf{cost}(o)$ for each $o \in \mathcal{O}^{\mathsf{priv}_i}$, and

(iii) the private parts of the public operators in $\mathcal{O}^{\mathsf{pub}_i}$, i.e., the number and values of private variables in $\mathsf{pre}(o) \cap \mathcal{V}^{\mathsf{priv}_i}$ and $\mathsf{eff}(o) \cap \mathcal{V}^{\mathsf{priv}_i}$ for each $o \in \mathcal{O}^{\mathsf{pub}_i}$.

We refer to the agent trying to hide information as agent α (the agent). We model all other agents as a single agent β (the adversary), which is common in secure MPC, as all the agents can collude and combine their knowledge in order to infer more private information. The public part of the agent's problem is the public projection $\Pi^{\triangleright}$ and can be shared with the adversary including public projections of the operators in $\mathcal{O}^{\mathsf{pub}_i}$. In this chapter, we refer to the view of the agent $\alpha = \alpha_i$ on the global problem $\mathcal{M}$, that is, the i-projected problem $\Pi_i^{\triangleright}$ as the problem Π, formally

$$\Pi = \Pi_i^{\triangleright} \qquad \text{for } \alpha = \alpha_i \in \mathcal{A}$$

In general, a single public projection $a^{\triangleright}$ can represent multiple actions a, a' such that $\mathsf{pre}(a)^{\triangleright} = \mathsf{pre}(a')^{\triangleright}$ and $\mathsf{eff}(a)^{\triangleright} = \mathsf{eff}(a')^{\triangleright}$. For the sake of simplicity of the presentation of some concepts and the proofs in the later sections, we use the label-preserving projection $\Pi^{\trianglerighteq}$ which preserves the labels of actions, formally $a^{\trianglerighteq} = \langle \mathsf{pre}(a)^{\triangleright}, \mathsf{eff}(a)^{\triangleright}, \mathsf{lbl}(a) \rangle$. In Section 7.2.2 we show that the label preserving projection leaks a significant amount of information and thus it is not reasonable to use in PP-MAP. Nevertheless, the label-preserving projection is commonly assumed in PP-MAP planners and heuristics and was not identified as a source of significant loss of privacy before. For disambiguation, we sometimes refer to $\Pi^{\triangleright}$ as a label non-preserving projection.

Example. (UAV) As an example running throughout the chapter, we use the most simple case of the coalition surveillance mission problem with one UAV and two secret locations (see Figure 7.1.1). We

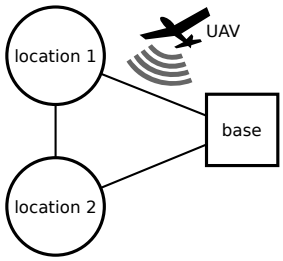


Figure 7.1.1: UAV surveillance scenario example.

omit the movement actions for simplicity (movement between the surveyed locations would be private, the movement to the coalition base would be public).

The problem in the running example consists of two agents $\mathcal{A} = \{\alpha_{\mathsf{UAV}}, \alpha_{\mathsf{base}}\}$ and can be described using the following sets of variables:

Variable in	Description	Variable	Values	s_I	$s_\star$
$\mathcal{V}^{\mathsf{pub}}$:	UAV has fuel	$\mathbf{f}$	T/F	F	-
	mission is complete	$\mathbf{c}$	T/F	F	T
$\mathcal{V}^{\mathsf{priv}}_{\mathsf{UAV}}$:	location 1 is complete	l1	T/F	F	-
	location 2 is complete	l2	T/F	F	-
$\mathcal{V}^{\mathsf{priv}}_{\mathsf{base}}$:	base has enough supplies	s	T/F	T	-

For further analysis, we use binary variables with T/F values, but all principles can be easily applied to general domain sizes. The problem consists of the following actions:

Actions (α_{UAV})	lbl(a)	pre(a)	eff(a)
survey location 1	SL1	$\{\mathbf{f} = \mathsf{T}\}$	$\{\mathsf{l1} = \mathsf{T}, \mathbf{f} = \mathsf{F}\}$
survey location 2	SL2	$\{\mathbf{f} = \mathsf{T}\}$	$\{\mathsf{l2} = \mathsf{T}, \mathbf{f} = \mathsf{F}\}$
complete mission	C	$\{\mathsf{l1} = \mathsf{T}, \mathsf{l2} = \mathsf{T}, \mathbf{c} = \mathsf{F}\}$	$\{\mathbf{c} = \mathsf{T}\}$

Actions (α_{base})	lbl(a)	pre(a)	eff(a)
refuel	R	$\{\mathbf{f} = \mathsf{F}, \mathsf{s} = \mathsf{T}\}$	$\{\mathbf{f} = \mathsf{T}, \mathsf{s} = \mathsf{F}\}$
refuel and resupply	RR	$\{\mathbf{f} = \mathsf{F}, \mathsf{s} = \mathsf{F}\}$	$\{\mathbf{f} = \mathsf{T}, \mathsf{s} = \mathsf{T}\}$

All actions in the problem are public for the ease of presentation, but private actions can be analyzed using the presented model as well. For further analysis, let us assume that the final global solution to the example problem together with its public projection is $\pi_{\mathsf{UAV}} = \{\mathsf{R}, \mathsf{SL1}, \mathsf{R}, \mathsf{SL2}, \mathsf{C}\}$, $\pi_{\mathsf{base}} = \{\mathsf{R}, \mathsf{SL}, \mathsf{RR}, \mathsf{SL}, \mathsf{C}\}$ and $\pi^{\triangleright}_{\mathsf{UAV}} = \pi^{\triangleright}_{\mathsf{base}} = \{\mathsf{R}, \mathsf{SL}, \mathsf{R}, \mathsf{SL}, \mathsf{C}\}$.

The complete transition system of the example global problem $\mathcal{M}$ is shown in Figure 7.1.2 (a), together with the transition system of the public projection $\Pi^{\triangleright}$(b). The transition system for $\Pi^{\trianglerighteq}$ looks exactly the same, except for the labels being SL1, SL2 instead of just SL and R, RR instead of R.

7.1.1 Cryptographic Assumptions

As is common in the cryptography literature, we place a number of assumptions on the agents, the environment and the algorithms used.

(Assumption 1) Semi-honest agents This is an assumption often used in secure MCP, stating that the agents do not diverge from the algorithm and communication protocol in order to exploit it, but

[1]Here we mean the exact number of variables. Later on, we assume an upper bound on the number of private variables is publicly known.

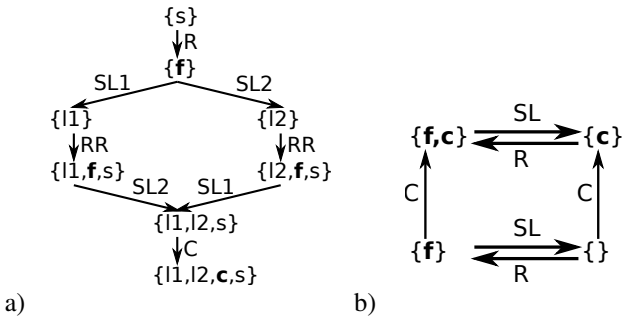


Figure 7.1.2: a) Global transition system (its reachable part) of $\mathcal{M}$. Variables shown are set to true, all other variables to false. b) Transition system of the public projection.

deduce as much information as possible from the execution and from the communicated data. Such agents are also known as "honest but curious" agents.

(Assumption 2) The adversary knows the algorithm and the communication protocol This means that the adversary can place assumptions based on the algorithm itself, such as that for each expanded state, all its successors are created and sent before expanding any further state.

(Assumption 3) Messages are received in-order That is, in the order in which they were sent. We also assume that the communication is lossless and all messages are received in finite time.

(Assumption 4) Upper bound on the size of the global problem We assume that there is an upper bound on the size of the global problem including private parts of all agents (e.g., the number of private variables) and this bound is publicly known.

The Assumption 1 limits the power of the adversaries so that we can assume that the protocol of computation is correctly followed. A stronger form of an adversary is a malicious agent. The malicious agent can not only infer information from the communicated data but can also alter its behavior in order to increase its chance of receiving more information. In the context of MAP, this might mean, for example, to force the agent to explore the complete search space by sending states generated systematically by breadth first search instead of A* and not confirming the found solution. Thus removing the first assumption would increase the information leakage in existing algorithms and complicate the development of new secure algorithms as there is much less existing cryptographic primitives secure in the presence of malicious agents and they incur much more computation overheads.

On the contrary, the other three assumptions limit the uncertainty in the system and thus increase the amount of information the adversary can learn about the planning problems of other agents. The Assumption 2 allows the adversary to analyze the received information with respect to the inner workings of the agent. In the context of MAP, this means that, for example, when a state is received by the adversary, it can analyze the context why it was sent based on the principles of the MAD-A* algorithm (if that is the protocol).

The Assumption 3 is a common assumption on the communication channel (similar assumptions were already placed in Chapter 5 in order to prove soundness and completeness of the MADLA Search algorithm. By removing this assumption, the adversary cannot infer knowledge based on the order of received messages, such that some state was expanded by the agent before some other state.

The Assumption 4 has a twofold effect. First, it is often used in secure MPC and cryptography in order to allow the information about the size of the problem to be used in the cryptographic protocols or to leak during the computation. For example, the size of the problem can be induced from the running time of a cryptographic protocol. In that case, if the size was not a public information, such leakage

would make the protocol not secure, but if it is already assumed public, no private information has leaked. The other effect of this assumption is that the adversary can bound its inference, as utilized in Section 7.2.

7.1.2 Weak and Strong Privacy

In the MAP and PP-MAP literature, the concept of privacy has been mostly reduced to the idea of weak privacy, as stated in [Nissim and Brafman, 2014]. Here, we rephrase the (informal) definition:

> We say that an algorithm is *weak privacy-preserving* if, during the whole run of the algorithm, the agent does not communicate (unencrypted) private parts of the states, private actions and private parts of the public actions. In other words, the agent openly communicates only the information in $\Pi^{\triangleright}$.

Obviously, the weak privacy does not give any guarantees on privacy whatsoever, as the adversary may deduce private knowledge from the communicated public information. Nevertheless, not all weak privacy-preserving algorithms are equal in the amount of privacy leaked. We have proposed measures of privacy leakage and techniques allowing computation of such measures in [Štolba et al., 2016c,d, 2017]. Summarized and extended results are presented in Section 7.2 and in Section 7.3.

In [Nissim and Brafman, 2014], the authors define also strong privacy, which is in accordance with the cryptographic and secure MPC model. Here we informally rephrase the definition of Nissim&Brafman:

> A *strong privacy-preserving* algorithm is such a distributed algorithm that no agent α_i can deduce an isomorphic (that is differing only in renaming) model of a private variable, a private operator and its cost, or private precondition and effect of a public operator belonging to some other agent α_j, beyond what can be deduced from the public input ($\Pi^{\triangleright}$) and the public output (projection of the solution plan $\pi^{\triangleright}$).

A more precise formal definition can be stated based on the definition of privacy in secure MPC, where privacy is typically defined with respect to the ideal world in which a trusted third party exists.

Definition 89. (Strong privacy MPC) Let $p_1, ..., p_n$ be n parties computing an algorithm P which takes n private inputs $\mathsf{in}_1, ..., \mathsf{in}_n$ respective to the parties, and produces n private outputs $\mathsf{out}_1, ..., \mathsf{out}_n$ respective to the parties. Let T be a trusted third-party. An algorithm P is *strong privacy-preserving* if the parties $p_1, ..., p_n$ do not learn more information from executing P without the third-party (in a distributed way), than by sending their respective private inputs in_i via a secure channel to T and receiving their respective outputs out_i via a secure channel.

Formally, the definition is bound to the existence of a simulator which, based solely on the input and output of a party i, can simulate the execution of the protocol so that the simulated execution (its distribution, in the probabilistic case) is indistinguishable from the execution of the protocol by the party i. See Goldreich [2009] for detailed formal definitions. Now we rephrase the definition in the context of MAP, where the difference is that there is a public part of the input and also a public part of the output.

Definition 90. (Strong privacy MAP) Let $\mathcal{M} = \{\Pi_i\}_{i=1}^n$ be a MAP problem for a set $\mathcal{A}$ of n agents. Let T be a trusted third-party. A MAP planner P is strong privacy preserving, if the agents $\alpha_1, ..., \alpha_n$ do not learn more information from solving $\mathcal{M}$ with P without the third-party (in a distributed way), than by sending their respective agent planning problems Π_i via a secure channel to T and receiving the i-projections $\pi^{\triangleright i}$ of the global plan via a secure channel.

The Definition 90 is compatible with the definition of Nissim&Brafman published in [Nissim and Brafman, 2014, Brafman, 2015].

7.2 Quantifying Privacy Leakage

Our approach is based on research in the area of secure Multi-Party Computation (MPC) [Yao, 1982], where the amount of privacy loss in general algorithms (or functions) has been recently studied as information leakage [Smith, 2009, Braun et al., 2009]. In this section, we take this general approach and apply it to the problem of PP-MAP. The information leakage is quantified based on the difference in the probability of guessing the right input of a function before and after the distributed computation. A high-level formula defined in [Smith, 2009] is

$$\text{information leaked} = \text{initial uncertainty} - \text{remaining uncertainty}, \tag{7.2.1}$$

where the initial uncertainty is related to the probability of guessing the right input without any additional knowledge gained from the execution of the algorithm, whereas the remaining uncertainty is the probability of guessing the right input given the output of the particular execution. In PP-MAP, we base the probabilities on the number of transition systems representing the global planning problem including all private information which is possible with respect to the public information known to the adversary agent (and the information obtained during the planning process).

We focus on a well-defined class of distributed state-space search algorithms and determine the lower bound for the MAFS and Secure-MAFS algorithms. We also present a running example, compute a worst-case leakage and lower bound on leakage of an execution of the MAFS (or MAD-A*) and Secure-MAFS algorithms on a running example. This work builds on [Štolba et al., 2016c,d, 2017], where the concept of information leakage based on uncertainty about the agent's transition system was first introduced. We extend the work by providing more precise and formal definitions and also a general method of computing the leakage bounds.

7.2.1 Privacy Leakage

One of the threat models studied in the literature on secure MPC is the threat that an attack will allow the adversary to guess the private information of the agent. In the case of PP-MAP, this means that the adversary may be able to guess the actual transition system of the agent.

Based on [Smith, 2009], let us have an algorithm which takes a private input H and produces a public output L. What we are interested in is, how much information about H can be deduced by an adversary who sees the output L. We assume that there is an a priori, publicly-known probability distribution of a random variable H with a finite space of possible values $\mathcal{H}$. We denote the a priori probability that H has a value $h \in \mathcal{H}$ by $P[H = h]$, and we assume that each element h of $\mathcal{H}$ has nonzero probability. Similarly, we assume that L is a random variable with a finite space of possible values $\mathcal{L}$, and with probabilities $P[L = l]$. We assume that each output $l \in \mathcal{L}$ is possible, in that it can be produced by some input $h \in \mathcal{H}$.

In the case of PP-MAP, the private input H the adversary is attempting to guess is the transition system $\mathcal{T}(\Pi)$ of the agent's problem (see Definition 33). In agreement with the assumptions above (known distribution and finite space of values), we assume that an upper bound on the size (number of states) of $\mathcal{T}(\Pi)$ is publicly known as n (see Assumption 4 in Section 7.1.1). This bound together with the public projection $\mathcal{T}(\Pi)^{\triangleright}$ of the transition system and public projection $\pi^{\triangleright}$ of the final plan limits the number of the transition systems possible with respect to $\mathcal{T}(\Pi)^{\triangleright}$ and $\pi^{\triangleright}$. We denote the number as t_{apriori}. We also assume that all the possible transition systems are equally probable, which gives us a uniform distribution with the probability $P[H = \mathcal{T}(\Pi)] = 1/t_{\mathsf{apriori}}$.

The public output consists of all information obtained from the agent during the planning process, that is, the public projection of the transition system $\mathcal{T}(\Pi)^{\triangleright}$, the resulting plan π and all additional information obtained during the planning process, such as reachable states, possible transitions, etc. We will represent the output as a mapping $O_\Pi : \mathcal{O} \mapsto \mathbb{N}$ from the actions of the agent to the number of possible transition sub-systems it represents according to the obtained information.

The vulnerability $V(X)$ of a random variable X is defined as the worst-case probability that an adversary could correctly guess the value of X in one try, which is simply the maximum of the probabilities of values of X, formally $V(X) = \mathsf{max}_{x \in \mathcal{X}} P[X = x]$, where $\mathcal{X}$ is the space of possible values of X. In the uniform distribution case, we obtain again $V(H) = 1/t_{\mathsf{apriori}}$.

By converting the vulnerability (which is a probability) to an information measure, we obtain a measure known as min-entropy $H_\infty(X) = \log \frac{1}{V(X)}$ where $\log$ represents a base-2 logarithm (and will so further on in the text). In our (uniform) case, the measure coincides with the Shannon entropy and gives the result of $H_\infty(H) = \log t_{\mathsf{apriori}}$. We use $H_\infty(H)$ as the measure of the initial uncertainty. To measure the remaining uncertainty, we consider the notion of conditional vulnerability, which gives the probability of guessing X in one try, given Y:

$$V(X|Y) = \sum_{y \in \mathcal{Y}} P[Y = y] \mathsf{max}_{x \in \mathcal{X}} P[X = x | Y = y].$$

In our (uniform) case, $P[H = \mathcal{T}(\Pi) | L = O_\Pi]$ is the probability of guessing the transition system $\mathcal{T}(\Pi)$ given the output O_Π reflecting all the information revealed by the algorithm. Altogether, $P[H = \mathcal{T}(\Pi) | L = O_\Pi] = 1/t_{\mathsf{post}}$, where t_{post} is the number of possible transition systems (on n variables) given the output O_Π. As we consider only deterministic algorithms and the output distribution can be also considered uniform, the resulting vulnerability is

$$\begin{aligned} V(H|L) &= \sum_{O_\Pi \in \mathcal{L}} P[L = O_\Pi] \mathsf{max}_{\mathcal{T}(\Pi) \in \mathcal{H}} P[H = \mathcal{T}(\Pi) | L = O_\Pi] \\ &= \sum_{O(\Pi) \in \mathcal{L}} P[L = O(\Pi)] \frac{1}{t_{\mathsf{post}}} = |\mathcal{L}| \frac{1}{|\mathcal{L}|} \frac{1}{t_{\mathsf{post}}} = \frac{1}{t_{\mathsf{post}}} \end{aligned}$$

The remaining uncertainty can be then measured by the conditional min-entropy

$$H_\infty(H|L) = \log \frac{1}{V(X|Y)} = \log t_{\mathsf{post}}$$

In summary, we obtain information leakage measures based on min-entropy $H_\infty(H)$ as:

initial uncertainty: $H_\infty(H) = \log t_{\mathsf{apriori}}$

remaining uncertainty: $H_\infty(H|L) = \log t_{\mathsf{post}}$

information leaked: $H_\infty(H) - H_\infty(H|L) =$

$$\log t_{\mathsf{apriori}} - \log t_{\mathsf{post}} = \log \frac{t_{\mathsf{apriori}}}{t_{\mathsf{post}}} \tag{7.2.2}$$

Where t_{apriori} is the number of possible transition systems known a priori, that is, based on the public projection $\mathcal{T}(\Pi^\triangleright)$, the assumption of the maximum n nodes and the public projection of the resulting plan $\pi^\triangleright$. Conversely, t_{post} is the number of transition systems based on the a posteriori knowledge, that is, the public projection of the transition system, public projection of the resulting plan and all other information obtained from the run of the planning algorithm.

According to [Smith, 2009], the remaining uncertainty gives a security guarantee as the expected probability of guessing H, that is, the transition system of the agent, given the public output of the planning algorithm L. The expected probability is

$$2^{-H_\infty(H|L)} = 2^{-\log t_{\mathsf{post}}} = 1/t_{\mathsf{post}} \tag{7.2.3}$$

where, again, t_{post} is the number of possible transition systems given the public output. Thanks to the determinism and uniform distributions, the result conveys with an intuition that the privacy preservation decreases by lowering the number of possible transition systems. In the next section, we will focus on how to estimate both t_{apriori} and t_{post}.

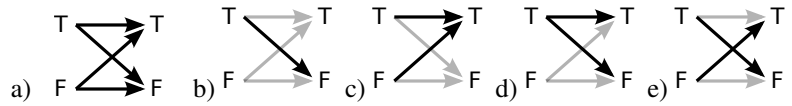


Figure 7.2.1: **a)** All possible transitions represented by a single action for a single binary variable $V \in \{\mathsf{T}, \mathsf{F}\}$, where T represents true and F represents false. Left side represents a precondition, right side an effect. Possible transition systems of the action are all subsets of the depicted transitions.
b) Transition system of a "resource consuming" action with precondition $V = \mathsf{T}$ and effect $V = \mathsf{F}$.
c) Transition system of an action which does not depend on the value of V in precondition and always sets $V = \mathsf{T}$ as an effect.
d) Transition system of a nondeterministic action which is applicable if $V = \mathsf{T}$ and has both effects $V = \mathsf{T}$ and $V = \mathsf{F}$. This can occur in the label non-preserving projection if $a_1^{\triangleright} = a_2^{\triangleright}$ and effect of a_1 is $V = \mathsf{T}$ and of a_2 is $V = \mathsf{F}$.
e) Transition system of an action with conditional effects or a nondeterministic action.

7.2.2 Leakage Quantification in PP-MAP

In the previous section, we have placed an assumption that an upper bound n on the total number of the states of the agent's transition system is publicly known. This results in n^2 possible transitions (including loops) and 2^{n^2} possible transition systems. Moreover, we assume that bounds on the number of private variables $|\mathcal{V}^{\mathsf{priv}}| \leq p$ and on the size of the private variable domains $|\mathsf{dom}(V)| \leq d$ for all $V \in \mathcal{V}^{\mathsf{priv}}$ are also publicly known.

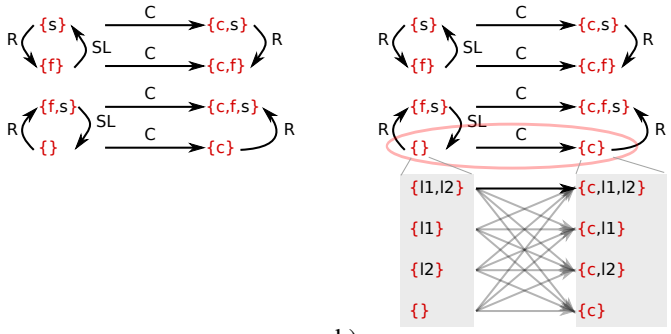
Similarly to [Brafman, 2015], for the simplicity of presentation, we assume that $\mathcal{O}^{\mathsf{priv}=\emptyset}$. This assumption can be stated without loss of generality as each sequence of private actions followed by a public action can be compiled as a single public action (with a potential exponential blow-up in the number of public actions). From the perspective of privacy, it is clear that when adhering to the weak privacy at least, private action is never communicated and thus can never leak. Any information about private action always leaks only in the sense of the described compilation, that is, it appears as if each new public action created by the compilation had some additional private preconditions or effects.

Let us first consider the public projection of the agent's transition system $\mathcal{T}(\Pi)^{\triangleright}$. Based on the above bounds, the number of private states $s \in S$ represented by a single public state $s^{\triangleright} \in S^{\triangleright}$ is d^p. The number of possible private transitions $\langle s, l, s' \rangle \in T$ represented by a single public transition $\langle s^{\triangleright}, l^{\triangleright}, s'^{\triangleright} \rangle \in T^{\triangleright}$is $(d^p)^2$, that is, from each private state s there is a transition to a private state s'.

For a single variable ($p = 1$), there are d private states represented by each public state and for a single action a, the respective public transition $\langle s^{\triangleright}, \mathsf{lbl}(a)^{\triangleright}, s'^{\triangleright} \rangle \in T^{\triangleright}$ represents d^2 possible private transitions between two public states. As demonstrated in Figure 7.2.1a), for each of the left d private states there either is or is not a transition to each of the right d private states, based on the private preconditions and effects of a. This set of transitions represents a transition system of the action a where the left to right direction represents the application of a. The upper part represents $V = \mathsf{T}$ before (left) and after (right) the application of the action and the lower part represents $V = \mathsf{F}$ analogously. For example, an action represented as Figure 7.2.1b) is applicable only if $V = \mathsf{T}$ and switches the value to $V = \mathsf{F}$, that is, consumes the resource represented by V which is no longer available. An action represented as Figure 7.2.1c) is applicable for both $V = \mathsf{T}, V = \mathsf{F}$ with the effect of setting $V = \mathsf{T}$, as both arrows end in $V = \mathsf{T}$.

An upper bound on the number of all such transition systems for a is a constant $t_a = 2^{d^2}$, that is, the number of subsets of the d^2 transitions (subsets of arrows in Figure 7.2.1a)). A public transition encodes only existing transitions (actions applicable in at least one private state). This means that an empty transition system is not an option, thus $t_a = 2^{d^2} - 1$, for a binary variable $t_a = 15$. As the variables are independent, for p variables, we obtain $t_a = (2^{d^2} - 1)^p$, or $t_a = 15^p$ if $d = 2$.

In the case of a label preserving projection where $\mathsf{lbl}(a) = \mathsf{lbl}(a^{\trianglerighteq})$ all actions $a \in \mathcal{O}^{\trianglerighteq}$ have unique



a) b)

Figure 7.2.3: a) The projection of transition system of the UAV example to the agent α_{base} (the adversary). b) Possible transitions represented by the public projection $\mathsf{C}^{\triangleright}$, the actual transition represented by the action C is in bold.

labels, because each public transition $\langle s^{\trianglerighteq}, l^{\trianglerighteq}, s'^{\trianglerighteq} \rangle \in T^{\trianglerighteq}$ represents exactly one action $a \in \mathcal{O}^{\trianglerighteq}$. As a single (deterministic STRIPS) action can never produce multiple states when applied in a single state (as in Figure 7.2.1d)), the number of possible transition systems t_a is significantly reduced. For a single variable with d values, we get the number of partial functions between two sets of size d, which is $t_a = (1+d)^d - 1$ (again -1 for empty transition system), for STRIPS $t_a = 8$, without conditional effects shown in Figure 7.2.1e), the number is reduced to $t_a = 7$. Again, as the variables are independent, the numbers can be multiplied for each variable. These 7 transition systems for a single variable V s.t. $\mathsf{dom}(V) = \{\mathsf{T}, \mathsf{F}\}$ are schematically shown in Figure 7.2.2.

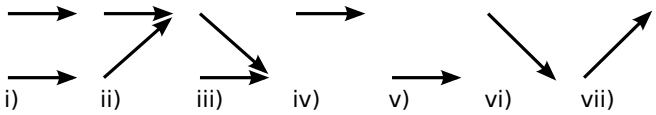


Figure 7.2.2: Possible transition systems of a STRIPS action a represented by a label-preserving projection $a^{\trianglerighteq}$. The upper arrow represents application of a when $V = \mathsf{T}$, the lower arrow when $V = \mathsf{F}$. A horizontal arrow represents unchanged value, diagonal arrow represents a change from $V = \mathsf{T}$ to $V = \mathsf{F}$ (↘) or from $V = \mathsf{F}$ to $V = \mathsf{T}$ (↗). A missing arrow means that a is not applicable when $V = \mathsf{T}$ (upper arrow missing) or $V = \mathsf{F}$ (lower arrow missing).

In the case of the label non-preserving projection, the number of transition systems (as already stated above) for one variable is 15, this number is obtained by adding non-deterministic transitions to the Figure 7.2.2, as is for example shown in Figure 7.2.1d), meaning that multiple actions with the same public projection can be applied in $V = \mathsf{T}$ and the result can be both $V = \mathsf{T}$ and $V = \mathsf{F}$, depending on the actual action applied.

Based on the above, a complete public projection $\mathcal{T}(\Pi)^{\triangleright}$ of the agent's transition system restricts the number of possible private transition system to $t_\Pi = {t_a}^{|T^{\triangleright}|}$, where t_a is the number of transition systems represented by a single public transition (again, we can multiply as the actions are independent).

Example. (UAV) The example problem has binary variables, two of which are private to the α_{UAV}. Figure 7.2.3a) shows the projection of transition system of the UAV example to the agent α_{base} and Figure 7.2.3b) shows all possible transitions represented by the public projection of a single action. There are 2 private variables, that is, $p = 2$. All subsets except for the empty set of the transitions are possible transition systems in the case of label non-preserving projection, thus we get $t_\Pi = 15^{2p} = 15^4 \cong 5 \times 10^4$

possible transition systems. In the case of label-preserving projection, the nondeterministic and conditional actions are not possible, in which case $t_\Pi = 7^{2p} = 7^4 \cong 2 \times 10^3$ possibilities. Already we can see a loss of privacy in the orders of magnitude only by exposing the label-preserving projection.

7.2.3 Sources of Leakage

Before we analyze complex algorithms, let us first focus on elements which have a major impact on the privacy leakage. From Equation 7.2.1 and its particular instance Equation 7.2.2 follows that the source we need to focus on is the information which is communicated in addition to the necessary public information, i.e., the projected problem $\Pi^\triangleright$ and the projected plan $\pi^\triangleright$. We refer to such information as *superfluous* information. In other words, in an ideal world a trusted third party would solve the planning problem and securely communicate the resulting plan to the agents, in the real world, the agents solve the problem distributedly and any private information deduced from the public information exchanged between them which the adversary would not be able to deduce from the information it receives in the ideal world is a superfluous information. In particular, the sources of information we focus on are superfluous distinct states and superfluous action applicability information.

There are also other possible sources of information leakage and their combinations, but we base our analysis on these three prominent sources, thus providing a lower bound on the information leaked. In the following, we provide more detailed description of the aforementioned sources of leakage. We use the label-preserving projection for the ease of presentation, but in the further analysis, we focus mainly on the label non-preserving projection, as it is more reasonable and occurs naturally in the distributed heuristic search.

In the following propositions, we assume a STRIPS model with p private binary variables. We start with the definition of the superfluous distinct states.

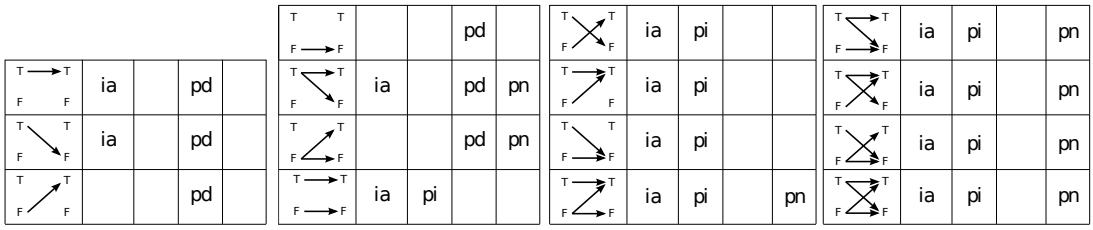
Definition 91. (Publicly equivalent states) Two states s, s' such that $s \neq s'$ are *publicly equivalent* if $s^\triangleright = s'^\triangleright$.

Definition 92. (Superfluous distinct states) Two publicly equivalent states s, s' are *superfluous distinct states* if it is revealed to the adversary agent β that $s \neq s'$.

The possibilities how superfluous distinct states can leak are algorithm-dependent and will be discussed in detail later. The superfluous distinct state information itself does not reduce the number of possible transition systems, but can be used for further deduction of superfluous action applicability.

Computing the information leakage based on action applicability is based on the following method. All transition systems corresponding to a particular type of action are shown in Figure 7.2.2 for the case of label-preserving projection and in Table 7.1 for the case of label non-preserving projection. Now we need to calculate the number of transition systems represented by an action of a certain type. For example, let us assume that an action a has a private precondition (such action is later on formally defined as privately-dependent, see Definition 95). Knowing that a has a private precondition on some private variable, we can compute which transition systems are not possible for a for that particular variable and subtract this number from the base number of transition systems for a, which is 7 for label-preserving projection an 15 for label non-preserving projection per variable (see Section 7.2.2). The number of transition systems which do not satisfy the property (e.g., a has a private precondition) can be counted in Figure 7.2.2 or Table 7.1. In the first case, this means the transition systems (i), (ii), and (iii) for which the action is applicable for both values of the variable, that is, 3 transition systems. In the second case, those are all transition systems without the pd tags, that is, 9 transition systems. Finally, as the variables are independent, we simply subtract the transition systems, where the property does not hold, from the base number of all transition systems, thus we obtain the following equation:

$$t_a^X = b^p - x^p \tag{7.2.4}$$



	ia		pd	
	ia		pd	
			pd	

			pd	
	ia		pd	pn
			pd	pn
	ia	pi		

	ia	pi		
	ia	pi		
	ia	pi		
	ia	pi		pn

	ia	pi		pn
	ia	pi		pn
	ia	pi		pn
	ia	pi		pn

Table 7.1: All 15 transition systems possibly represented by $a^{\triangleright}$. The tags describe which of them are possible for a given type of action:
ia – init-applicable action (Definition 93),
pd – privately-dependent action (Definitnion 95),
pi – privately-independent action (Definition 97), and
pn – privately-nondeterministic (Definition 99) action.

where X is the property (e.g., that a is privately-dependent, denoted as pd), b is the base number of possible transition system, e.g., 7 or 15, and x is the number of transition systems which do not satisfy the property X out of the base transition systems (5 and 9 respectively). In our example the result is $t_a^{\mathsf{pd}} = 7^p - 5^p$ for the label-preserving projection and $t_a^{\mathsf{pd}} = 15^p - 9^p$ for the label non-preserving projection.

The first case of superfluous action applicability we focus on is the situation, where an action is applicable on the initial state s_I, or a state created from s_I by a sequence of actions of the adversary β.

Definition 93. (Init-applicable action) Let $\pi_{\mathsf{sup}} = (a_1, ..., a_k)$ be a a s_k-plan such that π_{sup} is not a prefix of the final plan π and π_{sup} is revealed to the adversary agent β. Let a_l be the first action of agent α in π_{sup}. Then a_l is an *init-applicable* action.

Proposition 94. *Let $\pi_{\mathsf{sup}} = (a_1, .., a_l, ..., a_k)$ be an s_k-plan and let a_l be an init-applicable action. The number of transition systems represented by a label-preserving projection $a_l^{\triangleright}$ is $t_a^{\mathsf{ia}} = 5^p$ and by a label non-preserving projection $a_l^{\triangleright}$ is $t_a^{\mathsf{ia}} = 12^p$.*

Proof. Let $\pi_{\mathsf{sup}} = (a_1, .., a_l, ..., a_k)$ be a a s_k-plan and let a_l be an init-applicable action. From the definition of a s_k-plan, a_1 is applicable in the initial state s_I. Without loss of generality, we can assume that in s_I, all variables have a particular value (e.g., T for STRIPS), as the values can be arbitrarily renamed. The values of s_I private to the agent α are not changed by the actions of the adversary β. Let a_l be the first action of α in π_{sup}, applicable in state s_{l-1}. From the above, s_{l-1} has the same particular values of variables private to α. Again, without loss of generality, we can assume that the value is T. This fixes uncertainty about a_l in that it is now not possible that a_l is not applicable in s_{l-1}. But since we know that $s_{l-1}[V] = \mathsf{T}$, for a single binary variable V this means that a_l cannot have a precondition in the form of $V = \mathsf{F}$. In Equation 7.2.2 this corresponds to $\rightarrow$ and $\nearrow$ and thus only the remaining 5 transition systems are possible. The same holds for each variable independently. The resulting number of transition systems is thus $t_{a_l} = 5^p$ for p variables. For the label non-preserving projection, also the combination of $\rightarrow$ and $\nearrow$ is not possible, which results in removing 3 transition systems out of the total 15 for each variable, thus $t_{a_l} = 12^p$ for p variables. □

In the following, we focus on the information about the applicability of an action a on two superfluous distinct states s, s'. We define two types of the superfluous action applicability information.

Definition 95. (Privately-dependent action) Action $a \in \mathcal{O}$ is a *privately-dependent* action iff $\mathsf{vars}(\mathsf{pre}(a)) \cap \mathcal{V}^{\mathsf{priv}} \neq \emptyset$, that is, a has some private preconditions.

If the adversary learns about some action a that it is privately-dependent, it has obtained a superfluous action applicability information.

Proposition 96. *Let $a \in \mathcal{O}$ be a privately-dependent action. The number of transition systems represented by a label-preserving projection $a^{\trianglerighteq}$ is $t_a^{\mathsf{pd}} = 7^p - 3^p$ and by a label non-preserving projection $a^{\triangleright}$ is $t_a^{\mathsf{pd}} = 15^p - 9^p$.*

Proof. Let us first consider the label-preserving projection $\Pi^{\trianglerighteq}$. For $p = 1$ the following transition systems in Equation 7.2.2 $\rightarrow, \nearrow, \searrow$ are not possible to be represented by $a^{\trianglerighteq}$, because such transition systems do not depend on the variable, leaving only four, in particular $\overrightarrow{\rightarrow}, \searrow, \rightarrow, \nearrow$. For p variables, such situation must occur for at least one variable. We can formulate the total number of possible transitions, by taking $t_a = 7^p$ and subtracting the number of transition systems which does not satisfy the condition, that is, for all p variables, the action is applicable in both values, as in $\rightarrow, \nearrow, \searrow$. By Equation 7.2.4 we obtain $t_a^{\mathsf{pd}} = 7^p - 3^p$. The result for a label non-preserving projection is achieved by the same reasoning applied to all the 15 possible transition systems. □

Example. (UAV) In the running example, a privately-dependent action is C (see Table 7.2), because it depends on both private variables l1 and l2. This means that the number of possible transition systems represented by C is reduced from $7^2 = 49$ to $7^2 - 3^2 = 40$ in the case of label-preserving projection and from $15^2 = 225$ to $15^2 - 9^2 = 144$ in the case of label non-preserving projection.

A somewhat analogous situation arises, when a single action a is applicable in two publicly equivalent, but superfluous distinct states s, s'. In this case, the information learned is that the action does not depend on the private variable which distinguishes s and s', thus reducing the number of possible transition systems for a.

Definition 97. (Privately-independent action) Action $a \in \mathcal{O}$ is a *privately-independent* action iff $\mathsf{vars}(\mathsf{pre}(a)) \neq \mathcal{V}^{\mathsf{priv}}$, that is, there is a private variable $V \in \mathcal{V}^{\mathsf{priv}}$ on which a does not have a precondition.

Proposition 98. *Let $a \in \mathcal{O}$ be a privately-independent action. The number of transition systems represented by a label-preserving projection $a^{\trianglerighteq}$ is $t_a^{\mathsf{pi}} = 7^p - 4^p$ and by a label non-preserving projection $a^{\triangleright}$ is $t_a^{\mathsf{pi}} = 15^p - 6^p$.*

Proof. As in Proposition 98, for the label-preserving projection $\Pi^{\trianglerighteq}$, let $p = 1$. The transition systems $\rightarrow, \overrightarrow{\rightarrow}, \nearrow, \searrow$, where a is applicable only in one of the states are not possible. Thus, based on Equation 7.2.4, the resulting number is $t_a^{\mathsf{pi}} = 7^p - 4^p$ for label preserving projection and $t_a^{\mathsf{pi}} = 15^p - 6^p$ for label non-preserving projection. □

Example. (UAV) In the running example, such action is SL2 (see Table 7.2), because it does not depend on any of the variables l1 and l2. This means that the number of possible transition systems represented by SL2 is reduced from $7^2 = 49$ to $7^2 - 4^2 = 33$ in the case of label-preserving projection and from $15^2 = 225$ to $15^2 - 6^2 = 189$.

Notice that a single action a may be privately-dependent on some private variable and privately-independent on some other variable, but not both on a single variable. In the case of a label non-preserving projection, there is one more possible leakage. The application of a projected action $a^{\triangleright}$ (representing in fact multiple STRIPS actions) may result in two publicly equivalent but distinct states.

Definition 99. (Privately-nondeterministic action) Let $a \neq a' \in \mathcal{O}$ be two actions such that $a^{\triangleright} = a'^{\triangleright}$. Then $a^{\triangleright}$ is a *privately-nondeterministic* action iff there is a private variable $V \in \mathcal{V}^{\mathsf{priv}}$ such that $\mathsf{eff}(a)[V] \neq \mathsf{eff}(a')[V]$, that is, a and a' differ in a private effect.

Proposition 100. *Let $a^{\triangleright} \in \mathcal{O}^{\triangleright}$ be a privately-nondeterministic action. The number of transition systems represented by a label non-preserving projection $a^{\triangleright}$ is $t_a^{\mathsf{pn}} = 15^p - 8^p$.*

Proof. From Definition 99, $a^{\triangleright}$ represents at least two actions a, a' which differ in the private effects, that is, $\mathsf{eff}(a) \neq \mathsf{eff}(a')$. Now for at least one variable, the transition system represented by $a^{\triangleright}$ must contain both $\rightarrow, \nearrow$ or $\overrightarrow{\rightarrow}, \searrow$ transitions. This is not satisfied by 8 out of the 15 possible transition systems, thus based on Equation 7.2.4, $t_a^{\mathsf{pn}} = 15^p - 8^p$. □

Actions (α_{UAV})	$\mathsf{lbl}(a)$	ia	pd	pi	pn
survey location 1	SL1	✓	×	✓	-
survey location 2	SL2	✓	×	✓	-
survey location (no label)	SL	✓	×	✓	✓
complete mission	C	✓/×	✓	×	×

Table 7.2: Classification of actions in the UAV example: ia – init-applicable action (Definition 93), pd – privately-dependent action (Definition 95), pi – privately-independent action (Definition 97), pn – privately-nondeterministic action (applicable only on label non-preserving projection, Definition 99).

Example. (UAV) In the running example, such action is SL representing both SL1 and SL2. This means that the number of possible transition systems represented by SL is reduced from $15^2 = 225$ to $15^2 - 8^2 = 161$.

Even more information can leak from a combination of the properties. We know that an action cannot be both privately-dependent and privately-independent for a single variable, but other combinations are possible. Unfortunately, it is not always possible to simply infer more information from a combination of properties. Consider for example a privately-independent and privately-nondeterministic action. We cannot simply compute $t_a^{\mathsf{pn}\times\mathsf{pi}} = 15^p - 9^p$ (based on Table 7.1) as the each of the properties might be due to a different variable. Of course, we can use this value if we know (from some other leaked information) that both properties are due to the same private variable or if $p = 1$.

More information can leak from the init-applicable actions, where the property affects all variables and thus the "base" number of transition systems is lowered to 12^p. Other applicability properties then reduce the possible transition systems for a single variable. In the following, we apply the Equation 7.2.4 on the combinations of init-applicable actions with other applicability properties, thus the base number b for label non-preserving projection is only 12.

Proposition 101. *Let $a^{\triangleright} \in \mathcal{O}^{\triangleright}$ be a first action in a superfluous plan π_{sup} and let $a^{\triangleright}$ be privately-independent. The number of transition systems represented by a label non-preserving projection $a^{\triangleright}$ is $t_a^{\mathsf{ia}\times\mathsf{pi}} = 12^p - 3^p$.*

Proof. By counting the possible transition systems in Table 7.1 of a privately-independent action restricted to those applicable on $s_I[V] = \mathsf{T}$ for all $V \in \mathcal{V}^{\mathsf{priv}}$ and Equation 7.2.4. □

Proposition 102. *Let $a^{\triangleright} \in \mathcal{O}^{\triangleright}$ be a first action in a superfluous plan π_{sup} and let $a^{\triangleright}$ be privately-nondeterministic. The number of transition systems represented by a label non-preserving projection $a^{\triangleright}$ is $t_a^{\mathsf{ia}\times\mathsf{pn}} = 12^p - 6^p$.*

Proof. The same technique as above. □

Example 103. (UAV) In particular, for SL in the running example holds both, in the first case, the result is $12^2 - 3^2 = 135$ possible transition systems and in the second case $12^2 - 6^2 = 108$ possibilities.

Other sources of leakage

There are also other sources of leakage which are not used in the analysis of search-based algorithms. We present some of them here and use them later on in the analysis of other algorithms.

Definition 104. (Private-effect action) An action $a \in \mathcal{O}$ is a *private-effect* action iff $\mathsf{vars}(\mathsf{eff}(a)) \cap \mathcal{V}^{\mathsf{priv}} \neq \emptyset$, that is, a has some private effects.

Proposition 105. *Let $a \in \mathcal{O}$ be a private-effect action. The number of transition systems represented by a label-preserving projection $a^{\trianglerighteq}$ is $t_a^{\mathsf{pe}} = 7^p - 3^p$ and by a label non-preserving projection $a^{\triangleright}$ is $t_a^{\mathsf{pe}} = 15^p - 3^p$.*

Proof. Based on Equation 7.2.4, Table 7.1, and Equation 7.2.4. □

When the adversary learns a plan which is not (a part of) the final plan, such as in the PSM Planner [Tožička et al., 2016] briefly described in Section 7.4.1, the adversary can learn some information additional to that the first action in the plan is init-applicable.

Definition 106. (Superfluous plan) A *superfluous plan* is a s_k-plan $\pi_{\mathsf{sup}} = (a_1, ..., a_k)$ such that π_{sup} is not a prefix of the final plan π and π_{sup} are revealed to the adversary agent β.

Proposition 107. *Let π_{sup} be a superfluous plan containing l actions of the agent α. The number of transition systems represented by a label-preserving projection $\pi^{\rhd}_{\mathsf{sup}}$ is $t_\pi = 5^{pl}$.*

Proof. From the definition of a s_k-plan, a_1 is applicable in the initial state s_I. From Proposition 94 we know that $t_{a_1} = 5^p$ for p variables and the label-preserving projection. But since we know that $s_I[V] = \mathsf{T}$ and application of a single action always results in a single state, $s_1 = a_1 \circ s_I$, the resulting state s_1 has also a fixed value of V (although we do not know which one). This means that a_2 has also reduced number of possible transition systems according to Proposition 94. As the actions are independent, for $l = |\pi|$ actions of α, the resulting number of transition systems is $t_\pi = 5^{pl}$. □

Example. (UAV) Let us assume that in the example, the adversary has acquired the information that the sequence of actions $\pi' = \{\mathsf{R}, \mathsf{SL1}\}$ is valid (i.e., partial plan). As R does not change the values of variables private to α_{UAV}, the assumption above still holds for $\mathsf{SL1}$ and thus the number of possible transition systems represented by $\mathsf{SL1}$ is reduced from $7^2 = 49$ to $5^2 = 25$.

Clearly, this does not hold for the label non-preserving projection as in that case the application of the first action may result in multiple different states as the projection may represent multiple actions with different private effects.

7.2.4 Leakage Estimate

Let the number of transition systems represented by a single action a without any information leaked be a constant t_a. The particular value of t_a depends on the number of variables, their domain size and other factors, such as the possibility of non-deterministic transitions. Let $\mathcal{O}^{\mathsf{ia}}$ denote the set of init-applicable actions and t_a^{ia} the number of transition systems represented by a single action a reduced by the information learned from its applicability. Similarly, we define the set $\mathcal{O}^{\mathsf{pd}}$ of privately-dependent actions, the set $\mathcal{O}^{\mathsf{pi}}$ of privately-independent actions and the set $\mathcal{O}^{\mathsf{pn}}$ of privately-nondeterministic actions. The respective number of transition systems represented by each such action a is t_a^{pd}, t_a^{pi} and t_a^{pn} respectively (also the combined information $t_a^{\mathsf{ia}\times\mathsf{pi}}, t_a^{\mathsf{ia}\times\mathsf{pn}}$ may be used), each reduced according to the revealed information (the particular numbers were discussed in Section 7.2.3). Thus for each action $a \in \mathcal{O}^{\mathsf{pub}}$, we can define the number of transition systems it represents as

$$\tau_{\mathsf{post}}(a) = \mathsf{min} \begin{cases} t_a & \text{always} \\ t_a^{\mathsf{ia}} & \text{if } a \in \mathcal{O}^{\mathsf{ia}} \\ t_a^{\mathsf{pd}} & \text{if } a \in \mathcal{O}^{\mathsf{pd}} \\ \dots & \\ t_a^{\mathsf{ia}\times\mathsf{pn}} & \text{if } a \in \mathcal{O}^{\mathsf{ia}} \cap \mathcal{O}^{\mathsf{pn}} \\ \dots & \end{cases} \tag{7.2.5}$$

that is, the minimum of the number of transition systems represented by a based on its membership in the $\mathcal{O}^{\mathsf{ia}}, \mathcal{O}^{\mathsf{pd}}, \mathcal{O}^{\mathsf{pi}}$, and $\mathcal{O}^{\mathsf{pn}}$ sets.

Example. (UAV) For example if the action a_1 is revealed as privately-independent and it is in some communicated plan, $\tau_{\mathsf{post}}(a_1) = \mathsf{min}(t_a, t_a^{\mathsf{ia}}, t_a^{\mathsf{pi}})$ where for a STRIPS variables, single private variable and label-preserving projection the constants are $t_a = 7, t_a^{\mathsf{ia}} = 5, t_a^{\mathsf{pi}} = 3$ and thus $\tau_{\mathsf{post}}(a_1) = 3$.

The knowledge obtained about particular actions can be combined to compute the total number of possible transition systems

$$t_{\mathsf{post}} = \prod_{a \in \mathcal{O}^{\mathsf{pub}}} \tau_{\mathsf{post}}(a) \tag{7.2.6}$$

giving us an upper bound on the remaining uncertainty as

$$H_\infty(H|L) \quad = \quad \log(t_{\mathsf{post}}) = \log \textstyle\prod_{a \in \mathcal{O}^{\mathsf{pub}}} \tau_{\mathsf{post}}(a) \tag{7.2.7}$$

In the above formula, the number of possible transition systems include not only the superfluous (and thus leaked) information, but also the a priori known information. The a priori formula for initial uncertainty can be constructed similarly, but using only the information from the projected problem (that is t_a) and the projection $\pi^{\triangleright} = (a_1^{\triangleright}, ..., a_l^{\triangleright}, ..., a_k^{\triangleright})$ of the solution plan, where a_l is the init-applicable action and thus $\mathcal{O}^{\mathsf{ia}=}\{a_1\}$ resulting in

$$\tau_{\mathsf{apriori}}(a) = \mathsf{min} \begin{cases} t_a & \text{always} \\ t_a^{\mathsf{ia}} & \text{if } a^{\triangleright} \in \mathcal{O}^{\mathsf{ia}} \end{cases}$$

The final formula for the leaked information is obtained as the difference of the two, that is,

$$H_\infty(H) - H_\infty(H|L) \quad = \quad \sum_{a \in \mathcal{O}^{\mathsf{pub}}} \log \tau_{\mathsf{apriori}}(a) - \sum_{a \in \mathcal{O}^{\mathsf{pub}}} \log \tau_{\mathsf{post}}(a) \tag{7.2.8}$$

where for the actions with no additional information revealed we obtain $\log \tau_{\mathsf{apriori}}(a) - \log \tau_{\mathsf{post}}(a) = 0$ and for actions with leaked information we obtain $\log \tau_{\mathsf{apriori}}(a) - \log \tau_{\mathsf{post}}(a) > 0$. Since always $\tau_{\mathsf{apriori}}(a) \geq \tau_{\mathsf{post}}(a)$ as no information can be lost, only obtained, the number of possible transition systems can only decrease. This is important, because it shows that the more actions is revealed by the superfluous plans or superfluous applicability (itself following from superfluous distinct states), the more private information is leaked.

7.3 Privacy Analysis of Algorithms

In this section, we take a closer look on how the algorithms presented in the thesis behave from the point of view of privacy leakage analysis. We mainly focus on the Multi-Agent Forward Search (MAFS) [Nissim and Brafman, 2014] family of algorithms, that is, the optimal variant Multi-Agent Distributed A* (MAD-A*) [Nissim and Brafman, 2012], a secure variant Secure-MAFS [Brafman, 2015] and Macro-MAFS [Maliah et al., 2016c] and other similar planners. The MADLA Planner also falls into this category. Moreover, we analyze the relaxation-based heuristics and the LP-based heuristics. In the algorithm analysis, we aim to compute a lower bound on the privacy leakage but also to present general results. This means the amount of information that leaks during the execution of the algorithm for sure.

A significant principle, which undermines the privacy in MAFS-based algorithms is the use of label-preserving projection $\Pi^{\trianglerighteq}$, which is typically used to compute heuristic estimates. In theory, there is no need to preserve the labels of actions, although the quality of the heuristic estimate may suffer as multiple actions with the same public projection may have different costs (in a cost-aware version of MAP). As we aim for a lower bound on the information leakage, we focus on the label non-preserving projection $\Pi^{\triangleright}$ throughout the algorithm analysis. The label-preserving projection can only leak more information when used.

In the next sections, we analyze which parts of the search space are explored in an actual execution of particular algorithms and what effect does it have on the leakage of the private information.

7.3.1 General Method for Search-based Algorithms

In this section, we provide a general high-level method for computing the privacy leakage of MAFS-based algorithms, shown in Algorithm 15. Before we get into more details, let us remind the reader of the basic principle of multi-agent best-first search (detailed in full in Section 5.1). In the MAFS algorithm, each agent searches the state space using its own actions $a \in \mathcal{O}^{\mathsf{pub}_i} \cup \mathcal{O}^{\mathsf{priv}_i}$, an ordered open list, and a closed list, as in a textbook best-first search. If a public action $a \in \mathcal{O}^{\mathsf{pub}_i}$ is used to expand a state s, the resulting state $s = a \circ s$ is sent to all other agents $\alpha_{j \neq i}$ which have some relevant action ($a' \in \mathcal{O}^{\mathsf{pub}_j}$ s.t. $\mathsf{pre}(a')$ is satisfied in s'). The state s' is sent in the form of public projection $s'^{\triangleright}$, together with its heuristic value and an encrypted[2] private part of s' for each agent, that is, $\mathsf{enc}_\alpha(s')$ for the agent and $\mathsf{enc}_\beta(s')$ for the adversary.

Any symmetric encryption algorithm can be used to encrypt the private parts, as only the sender will ever need to decrypt the part of the state and thus the key does not have to be shared. Moreover, the public part of the state can be used as so called salt [Gauravaram, 2012], that is, random data which are concatenated with the actual data before encryption to improve security.

When a state message is received, the encrypted information is used to reconstruct the private part of the received state respective to the receiving agent α_i. Moreover, the heuristic is recomputed from the point of view of agent α_i and the maximum is used (this step may be omitted in the case of a distributed or an inadmissible heuristic).

More formally (based on [Brafman, 2015]), we say that s_0 is an i-parent of s_k if s_k is obtained from s_0 by the application of a sequence $a_1, ..., a_l$ of actions of agents other than α_i and the last action in this sequence, a_l, is public. This means that s_k is a state that would be sent by some agent to α_i, and α_i was not involved in the generation of any state between s_0 and s_k.

Algoritmus 15: Privacy leakage for MAFS-based algorithms

```
1 Algorithm computePrivacyLeakage(M)
2     Reconstruct the search tree
3         Determine possible parent states
4         Determine possible applied actions
5         Determine distinct states
6     Assign actions to the respective sets O^ia, O^pi, O^pd, and O^pn
7     Compute the leakage
```

Here, we present a high-level method for computing the privacy leakage of MAFS-based algorithms, shown in Algorithm 15. We focus on the particular instantiation of the steps in the following sections. In general, the first step is to attempt to reconstruct the search tree of the particular execution of the search-based algorithm (an example of the search tree of MAD-A* is shown in Figure 7.3.1). Reconstructing the search tree means that the parent states and applied actions are identified. Moreover, it is important to determine as much publicly equivalent but distinct states as possible in order to be able to later identify privately-dependent, privately-independent and privately-nondeterministic actions and compute the leaked information based on the computation in Section 7.2.4.

In our examples, we use the simple Equation 7.2.8. This equation can be used only in the case where the applicable action is determined unambiguously, that is, there is at most one applicable action for a given transition. In general case, a set of actions might be possibly responsible for a transition and thus it cannot be determined which of the actions actually has the determined properties (e.g., is privately-independent). Nevertheless, the problem of computing the leakage with sets of actions can be formulated as a linear program with disjunctive constraints, or a mixed-integer linear program. In

[2]Note that there are no implementation details on the encryption algorithm in the description of any of the actual planners, as none of them internally uses actual encryption.

the rest of this section, we present the former as it is conceptually simpler and there is a standard technique for transformation to the latter which can be then solved using off-the-shelf solvers such as IBM CPLEX[3].

In order to compute Equation 7.2.8 we need to obtain the sum of (base 2) logarithms of the a priori and posteriori numbers of possible transition systems for each public action. Let $\mathcal{O}$, $\mathcal{O}^{\mathsf{ia}}$, $\mathcal{O}^{\mathsf{pi}}$, $\mathcal{O}^{\mathsf{pd}}$, and $\mathcal{O}^{\mathsf{pn}}$ be the respective sets of all, init-applicable, privately-independent, privately-dependent, and privately-nondeterministic actions and let t_a, t_a^{ia}, t_a^{pi}, t_a^{pd}, and t_a^{pn} be the respective number of transition systems represented by the particular type of action. Each action $a \in \mathcal{O}^{\mathsf{pub}}$ then represents $\tau_{\mathsf{apriori}}(a)$ transition systems before the algorithm is executed and $\tau_{\mathsf{post}}(a)$ transition systems after it is executed. We focus on the computation of $\tau_{\mathsf{post}}(a)$, the computation of $\tau_{\mathsf{apriori}}(a)$ is analogous. In order to formulate a linear program (LP), we define a LP variable $\bar{a}$ for each action $a \in \mathcal{O}^{\mathsf{pub}}$ such that $\bar{a} = \log \tau_{\mathsf{post}}(a)$ so that we can use the sum variant of Equation 7.2.5. Each LP variable $\bar{a}$ is bounded by $1 \leq \bar{a} \leq \log t_a$ as each action represents at least one transition system (the actual transition system in the global problem) and the maximum number of transition systems an action can represent is that of an action with no additional information on action applicability. We must use logarithm of t_a as $\bar{a}$ represents logarithm of $\tau_{\mathsf{post}}(a)$. We can simply apply logarithm to both sides of an inequality as we use logarithm with base ≥ 1. We formulate the following objective function to be maximized:

$$\mathsf{Maximize} \sum_{a \in \mathcal{O}^{\mathsf{pub}}} \bar{a}$$

resulting in the second part of Equation 7.2.8. The Equation 7.2.5 is easily translated into the following set of constraints:

$$\begin{aligned} \bar{a} &\leq \log t_a^{\mathsf{ia}} && \forall a \in \mathcal{O}^{\mathsf{ia}} \\ \bar{a} &\leq \log t_a^{\mathsf{pi}} && \forall a \in \mathcal{O}^{\mathsf{pi}} \\ &\dots \end{aligned}$$

and similarly for $\mathcal{O}^{\mathsf{pd}}$, $\mathcal{O}^{\mathsf{pn}}$, other types of actions, and their combinations. Together the constraints encode the minimization of Equation 7.2.5 as all must hold at once. In order to infer more than from Equation 7.2.5 we consider also more complex situations. For example if there is a set of actions $\{a_1, a_2, a_3\}$ and we know that at least one of the actions is privately-independent we can add the following disjunctive constraint:

$$\bar{a_1} \leq \log t_a^{\mathsf{pi}} \vee \bar{a_2} \leq \log t_a^{\mathsf{pi}} \vee \bar{a_3} \leq \log t_a^{\mathsf{pi}}$$

More cases where the disjunctive formulation is useful are presented in the next sections. Such a disjunctive formulation can either be directly solved by a branch-and-bound algorithm or transformed to mixed-integer linear program (MILP) for example by using the Big-M reformulation. In order to do so, for each constraint in the disjunction we define a binary variable, e.g., $y_1, y_2, y_3 \in \{0, 1\}$, and enforce that only one of them holds true by

$$y_1 + y_2 + y_3 = 1$$

Then we define a large enough constant M and reformulate the above disjunction as

$$\begin{aligned} \bar{a_1} &\leq \log t_a^{\mathsf{pi}} + M(1 - y_1) \\ \bar{a_2} &\leq \log t_a^{\mathsf{pi}} + M(1 - y_2) \\ \bar{a_3} &\leq \log t_a^{\mathsf{pi}} + M(1 - y_3) \end{aligned}$$

which significantly expand all but one of the bounds thus rendering them ineffective. Such transformation then allows to use a standard solver with MILP capabilities, such as the mentioned CPLEX.

[3] http://www.ibm.com/us-en/marketplace/ibm-ilog-cplex

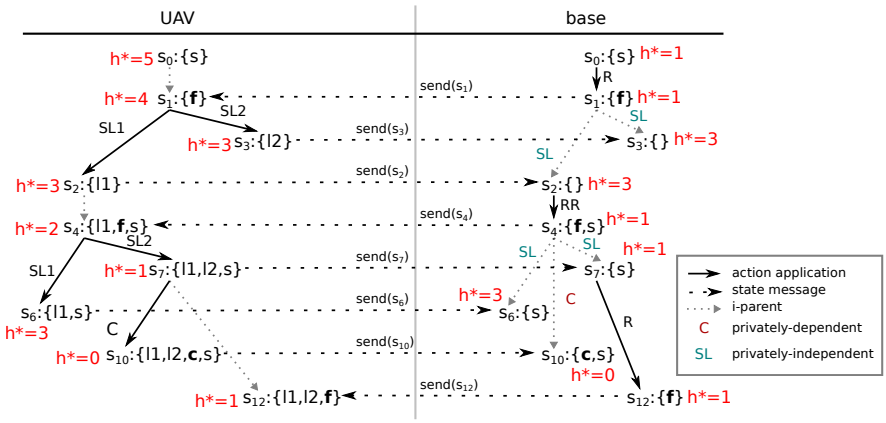


Figure 7.3.1: Search tree of a particular run of MAD-A* (or MAFS) using the optimal projected heuristic h^*.

7.3.2 MAFS and MAD-A*

According to [Nissim and Brafman, 2014], MAFS is at least weak privacy-preserving, as no private information is explicitly sent to other agents. But MAFS is not strong privacy-preserving as some additional information can be deduced from the search. Here we analyze what information it is.

Example. (UAV) For the analysis of the example, we use the MAD-A* algorithm. As MAD-A* is a heuristic search, we need to assign a heuristic value to each state. We will use the optimal projected heuristic, that is, h^* computed on the projected problem (of each agent). Figure 7.3.1 shows a portion of the search space explored by the algorithm using this heuristic estimate (shown as h^* in the figure).

Let us now focus on the individual steps of Algorithm 15 and their relation to the results of Section 7.2.3.

Search tree reconstruction in MAFS

According to Algorithm 15, the first step in privacy leakage analysis of a search-based algorithm such as MAFS or MAD-A* is to reconstruct the search tree. The search tree is a tree determined by the parent/child relation of the states during the particular search execution, as shown if Figure 7.3.1.

The important information in the search tree is which state s_{k-1} is the parent of a state s_k and what action a was applied by the agent to generate s_k from s'_{k-1}. As both the agent and the adversary are using the same algorithm (e.g., MAFS), each state s_k has also a private part of the adversary β, that is, $\mathsf{enc}_\beta(s_k)$. This part can be a simple pointer to the i-parent state s_0, which contains the right private part (possibly encrypted before sending). If a sequence of (public) actions was applied by the agent α between s_0 and s_k, then $s_0 \neq s_{k-1}$. To determine the actual parent, we state the following.

Proposition 108. *For an agent α and an adversary β, let s_k be a state sent by α and received by β and let s_0 be the i-parent of s_k known to β. Then the parent state s_{k-1} of s_k was received by β before s_k and after s_0 which is also an i-parent of s_{k-1}.*

Proof. Trivial if $s_{k-1} = s_0$. Otherwise a sequence $a_1, ..., a_k$ of actions of agent α must have been applied by α on s_0 to generate s_k. The state generated by action a_{k-1} is the parent s_{k-1} of s_k. As we assume only public actions, all intermediate states between s_0 and s_k are also sent to β and thus the same holds for the parent s_{k-1}. Based on the MAFS algorithm, a state is fully expanded before expanding the next state from the open list. As we assume that all messages retain the order in which they were sent, then clearly s_{k-1} was sent before s_k and thus also received before s_k. □

Based on the Proposition 108, the parent s_{k-1} of the state s_k is some of the states received before s_k such that s_0 is an i-parent of s_{k-1}. Thus the parent state might not be determined unambiguously, but instead the adversary knows a set of potential parent states of s_k.

Determining the applied action is even trickier. For two states s', s, where s' is a (potential) parent of s, we need to find a projected action $a^{\triangleright} \in O^{\triangleright}$ for which $\mathsf{pre}(a^{\triangleright})$ holds in s' and $\mathsf{eff}(a^{\triangleright})$ is consistent with s. This means that we can only determine the public projection $\langle \mathsf{pre}(a^{\triangleright}), \mathsf{eff}(a^{\triangleright}) \rangle$ of the used action, which corresponds to the label non-preserving projection. In the case of a label-preserving projection, if there are multiple actions in $O^{\trianglerighteq}$ with the same public projection, the particular action $a^{\trianglerighteq}$ cannot be determined. In the case where we know a set of possible parent states, also the result is a set of possibly applied actions and the exact parent state might not be known (for each possible action we, again, have a subset of possible candidate parent states).

Example. (UAV) In the example in Figure 7.3.1, the public projections of actions of agent α_{UAV} are $\langle \{\mathsf{f} = \mathsf{T}\}, \{\mathsf{f} = \mathsf{F}\} \rangle$ and $\langle \{\}, \{\mathsf{c} = \mathsf{T}\} \rangle$. As in the state s_3 holds $\mathsf{c} = \mathsf{F}$, the second projection is not applicable. The first projection is applicable in s_1 and the effect holds in s_3. The corresponding projected action is SL. The same holds for the parent s_1 and child s_2 and similarly for s_4 and s_7, s_6. The i-parent of state s_{10} is s_4 and the potential parent states are thus s_4, s_6, s_7. As we know, there is no single action which would change $\{\mathsf{f}\}$ to $\{\mathsf{c}\}$, then clearly s_4 is not the parent. For the parents s_7, s_6 and child s_{10} is applicable only the projection $\langle \{\}, \{\mathsf{c} = \mathsf{T}\} \rangle$ corresponding to the projected action C. The only remaining uncertainty is which of the two states is the true parent state.

Superfluous distinct states in MAFS

The goal of encryption in MAFS and MAD-A* is twofold. The first is to hide the parent of the state and the second is to disallow the adversary to determine whether two states are globally equal or not. The first goal is achievable without any effect on the algorithm performance. Parents of states are necessary only for the plan reconstruction process and do not have to be shared to do so. To achieve the second goal is much trickier.

In order to maintain soundness, the agent has to be able to reconstruct the private part of each state unambiguously, this means that two different private parts of a state can never result in the same encrypted value $\mathsf{enc}_{\alpha}(s) = \mathsf{enc}_{\alpha}(s')$.

The simplest approach is to have the private part of each state s encryption result in an unique value $\mathsf{enc}_{\alpha}(s)$. In that case, no information is leaked as the adversary cannot determine which states are the same. On the other hand, this approach may seriously influence the algorithm performance or even make it incomplete. The reason is that each agent has its own closed list and needs to determine the membership of a state s in the closed list. In order to maintain completeness of MAFS, the membership has to be determined over the global state in order to prevent cycles over multiple agents. MAD-A* does not loose completeness as BFS and thus also A* is complete even on infinite graphs (which we effectively get when allowing infinite loops) with finite branching factor (which we retain as there is still finite number of actions). Nevertheless, both algorithms suffer a major blowup of the size of the search space.

Clearly, in practice, a middle ground can be determined. An option is to bound the number of different possibilities how the private part of a state can be encrypted. This way the blowup is bounded, but also the gain in privacy is bounded. Notice also that in fact only the private parts of a state with the same public part has to differ. As in our analysis we are aiming for a lower bound of the privacy leakage, we can safely assume that the agents use the unique encryption method and no information about superfluous distinct states leaks from the encrypted private parts. By bounding the possible encryption, the leakage can only increase.

Both MAFS and MAD-A* expand the states in the open list in the order defined by a function $f(s)$. In the case of MAD-A* the function is defined as $f(s) = g(s) + h(s)$ where $g(s)$ is the distance (number of actions) from the initial state s_I to s and $h(s)$ is an admissible heuristic estimate of the remaining

distance from s to the nearest goal state s_G. In the case of MAFS, the $g(s)$ function can be omitted to obtain a Greedy Best-First Search (GBFS), where $f(s) = h(s)$, so that the states are ordered solely by the heuristic information. Also the heuristic in MAFS does not have to be admissible in general. Based on the heuristic function, we can state the following to infer different states.

Proposition 109. *Let s, s' be two states and $h(s)$ a deterministic heuristic function. If $f(s) \neq f(s')$ or $g(s) \neq g(s')$ or $h(s) \neq h(s')$ then $s \neq s'$.*

Proof. The computation of a heuristic function, the distance from the initial state, or their sum, never results in a different value for the same state. □

Also the set of successor states can be used to determine equivalence.

Proposition 110. *Let S, S' be the sets of all successors of the states s, s' respectively. If $S \neq S'$ then also $s \neq s'$.*

Proof. Let us assume $s = s'$, but $S \neq S'$. But each state was expanded using the same set of actions and thus $S = S'$. □

We can continue the inference with the parents of s, s' and further on.

Example. (UAV) Now let us focus on the example in Figure 7.3.1. We can distinguish s_6 and s_7 based on Proposition 109 as $h(s_6) \neq h(s_7)$ and also s_3 and s_7 as again $h(s_3) \neq h(s_7)$. But based on the information that $s_3 \neq s_7$ and Proposition 110, we can infer that also $s_4 \neq s_1$ as their child states are different.

Superfluous action applicability in MAFS

So far, we have analyzed the search tree of a MAFS or MAD-A* run according to Algorithm 15. We have reconstructed the parent states (line 3), determined which actions were applied (line 4), and determined which publicly equivalent states are in fact distinct (line 5). Now we proceed to combine all this information in order to lower the uncertainty of the adversary about the actual transition system of the agent.

In general, MAFS and MAD-A* use a label non-preserving projection, thus each action a (with a distinct public projection) represents $t_a = 15^p$ transition systems, where p is the number of (binary) private variables (see Section 7.2.3 for details). In our example with two private variables the number is $15^2 = 225$.

An action, which is known to be applied in the initial state s_I (or a state s_l resulting from the application of a sequence of actions of the adversary actions on s_I) reveal the information that it is applicable in some particular state where we can assume a fixed value of all variables. Each such init-applicable action ($\mathcal{O}^{\mathsf{ia}}$) represents at most $t_a^{\mathsf{ia}} = 12^p$ transitions (Proposition 94).

An action, which is known to be applied on two states s, s' known to be distinct is privately-independent (Definition 97) in at least one variable, as it does not depend on the variable in which s and s' differ. Such action represents $t_a^{\mathsf{pi}} = 15^p - 6^p$ transitions (Proposition 98). Conversely, an action, which is known to be applied on a state s and not on a state s' such that s, s' are publicly equivalent, but known to be distinct is a privately-dependent action (Definition 95) as it depends on the variable in which s and s' differ. Such action represents $t_a^{\mathsf{pd}} = 15^p - 9^p$ transitions (Proposition 96).

Furthermore, if we observe that the application of a single action on a single state results in two distinct states, we found a privately-nondeterministic action (Definition 99), that is, a projected action which represents at least two actions with private effects different in at least one variable. Such action represents $t_a^{\mathsf{pn}} = 15^p - 8^p$ transitions (Proposition 100).

Example. (UAV) In the example in Figure 7.3.1, the action SL is applicable in the state s_1 and thus is init-applicable and represents $\tau(\mathsf{SL}) = t_a^{\mathsf{ia}} = 12^2 = 144$ transitions. We can also see that the action SL can be applied in both s_1 and s_4, which are known to be different states and thus is privately-independent in at least one variable. Moreover, we can observe that the action C is applied on state s_6 or s_7, but is never applied on state s_2, which is publicly equivalent and for which all child states were already generated (it is only s_4). This can only mean that either s_6 or s_7 (or both) are distinct from s_2 and this distinction affects the applicability of C. Thus C is privately-dependent. Finally, the action SL, is applied in s_4 and results in two distinct states s_6 and s_7 and thus is privately-nondeterministic.

Based on the above information and on the fact that SL is the first action of agent α_{UAV} in the resulting plan, the combined knowledge of the action properties can be used and the final number of transition systems represented by the actions SL and C computed based on Equation 7.2.5 as

$$\begin{aligned} \tau(\mathsf{SL}) &= \mathsf{min}(t_a, t_a^{\mathsf{ia}}, t_a^{\mathsf{pi}}, t_a^{\mathsf{pn}}, t_a^{\mathsf{ia}\times\mathsf{pi}}, t_a^{\mathsf{ia}\times\mathsf{pn}}) = \mathsf{min}(144, 189, 161, 135, 108) = 108 \\ \tau(\mathsf{C}) &= \mathsf{min}(t_a, t_a^{\mathsf{pi}}) = 144 \end{aligned}$$

Information leakage in MAFS

Now we can combine the results to obtain the total information leaked. Based on Equation 7.2.5 and Equation 7.2.8 we conclude that $\tau_{\mathsf{post}}(\mathsf{SL}) = 108$ and $\tau_{\mathsf{post}}(\mathsf{C}) = t_a^{\mathsf{pd}} = 144$ and because the only information learned from the projection $\Pi^{\triangleright}$ and the plan $\pi^{\triangleright}$ is that the action SL is init-applicable, the a priori information is $t_{\mathsf{apriori}} = 12^p 15^p = 32400$ whereas the a posteriori information is computed using the values obtained in the previous section as

$$t_{\mathsf{post}} = \prod_{a \in \mathcal{O}^{\mathsf{pub}}} \tau_{\mathsf{post}}(a) = \tau_{\mathsf{post}}(\mathsf{SL})\tau_{\mathsf{post}}(\mathsf{C}) = 108 \cdot 144 = 15552$$

Thus, the lower bound on the information leakage of this particular execution of the MAFS algorithm on this particular example is

$$H_\infty(H) - H_\infty(H|L) = \log \frac{t_{\mathsf{apriori}}}{t_{\mathsf{post}}} \cong \log(2) = 1$$

bit of information. Also, based on Equation 7.2.3, we obtain the expected probability of guessing the correct transition system as

$$2^{-H_\infty(H|L)} = 2^{-\log t_{\mathsf{post}}} = \frac{1}{t_{\mathsf{post}}} = \frac{1}{15552} \cong 6 \times 10^{-5}$$

Notice that in this particular example, all the information about action applicability we have considered has leaked.

Information leakage in the MADLA Search

One of the main contributions of this thesis is the MADLA Search described in Chapter 5. How much information does this novel variant of search leak? If similarly to the analysis above, we disregard the heuristic values (and heuristic computation) as a source of information leakage, we can state the following proposition.

Proposition 111. *The MADLA Search leaks on average the same amount of information as MAFS.*

Informally, the proof of Proposition 111 must consider two differences of MADLA Search with respect to MAFS. First, the search trace will differ even if using the same heuristic values as MADLA uses two open lists and thus may expand states in a different order. In both cases, the order of state

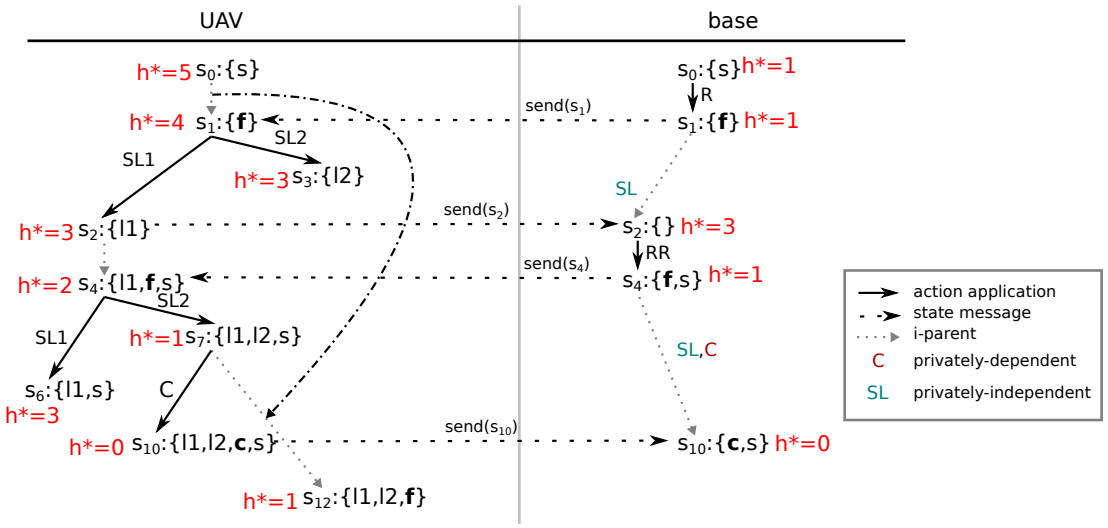


Figure 7.3.2: Search tree of the Secure-MAFS algorithm

expansions depends on the heuristic value and tie-breaking, but the sets of all possible search trees are the same in both cases.

The other difference between MADLA and MAFS is that in MADLA, there is additional information sent in the message stating whether the state was evaluated using the distributed or local heuristic. But this additional information does in no way depend on the problem being solved or the search tree, but depends solely on the particular execution of the algorithm (e.g., how long is the distributed heuristic waiting for replies from other agents influences how many states are evaluated using the local heuristic). This means that no information regarding the search tree or the problem itself can be inferred.

7.3.3 Secure-MAFS

The theoretical Secure-MAFS [Brafman, 2015] algorithm and its (generalized) implementation Macro-MAFS [Maliah et al., 2016c] are variants of the MAFS algorithm aiming at improving the privacy of the planning process. The authors provide a proof, that on a particular constrained variant of a logistics problem the algorithm is strongly private, meaning no information leakage. The authors do not provide much information about other problems and domains but suggest that the privacy should be improved in general.

The main difference of Secure-MAFS in comparison to MAFS is that it sends only states, which differ from previously sent states in the private part of the other agents. In other words, each state with a unique public part and other agents' private parts is sent at most once. The transitions caused by the actions of other agents are stored (in the form of action macros) and applied instead of sending the state.

An execution of the Secure-MAFS algorithm on the running example is shown in Figure 7.3.2. When the agent α_{UAV} receives the state s_1, it stores the transition $\{\mathsf{f} = \mathsf{F}\} \rightarrow \{\mathsf{f} = \mathsf{T}\}$ together with information about the private part of α_{base}. The learned (macro) transition can be later applied in any state which satisfies the left side and the α_{base} private part condition, in other words, the agent α_{UAV} has learned that α_{base} can make f true for certain values of the private part of agent α_{base}. Thus any further state equal to $\{s\}$ (the private part of α_{base} is encrypted) does not have to be sent, the same holds later for $\{\}$. This significantly reduces the need for communication to communicating only one more state (s_2) together with the goal state $s_{10} = \{\mathsf{c}, \mathsf{s}\}$ after the application of the action C, as the agent α_{UAV} has learned the complete functionality of agent α_{base}. Now let us analyze what effect the reduction of communication has on the privacy leakage.

Superfluous distinct states in Secure-MAFS

One major difference of security handling from MAFS in Secure-MAFS is the role of encryption of the private states. In order to be able to determine that two states s, s' do not differ in the public part and the private part of other agents, the encryption of the private parts must be bijective. If it is not so, the effect of the Secure-MAFS principle is lessened. Therefore any two states, which are publicly equivalent, but differ on the private part of some agent can be distinguished.

On the contrary, no two states s, s' which differ only in the private part of an agent α_i are sent to other agents. This means that even though the information about state equality is revealed, its effect for the information leakage is significantly reduced.

Notice that in the example in Figure 7.3.2, only the publicly equivalent states which differ in the private part of the agent α_{base} are sent. In particular, this holds for the states s_0 and s_2.

Search tree reconstruction in Secure-MAFS

As described above, the search tree reconstruction is simplified by the fact that the equality of states is revealed, but is complicated by the fact that significant parts of the search tree are not communicated. Also as the received state s' might be a result of the application of a sequence of multiple public actions on the (known) β-parent state s, Proposition 108 does not hold for Secure-MAFS. This renders inference of the applied action (or in fact a sequence of actions) significantly more complicated (and computationally more intensive). If a sequence of actions of length k was applied, all possible sequences of length up to k must be considered. Moreover, the adversary does not know the exact number of the agent's actions (unless a label-preserving projection was provided) and the actions can repeat in the sequence. An upper bound of the number of actions can be placed based on the bound on the number of private variables. For p binary private variables, each public transition represents at most 2^{2^p} possible actions. Note that this bound is very overestimating, typical problems have much fewer actions.

Basically, to determine a possible sequence of public actions is equal to find a plan in the public projection of the agent's problem with $s^{\triangleright}$ as the initial state and $s'^{\triangleright}$ as the goal state. As the agent may use a different heuristic than the adversary agent, we cannot put any assumption on the length of the sequence and thus to reveal all information, all possible sequences of actions need to be determined, which is clearly not tractable and thus the adversary can never be sure that it has found all possible sequences. In general, even finding a single sequence may not be tractable and thus if we assume that at most polynomial time can be spent on the leaked information analysis, it may not be possible to determine even a single sequence of actions represented by the parent-child transition.

In practice, we can put a bound on the number of actions considered in a sequence (even a single action), thus reducing the complexity. In some cases, this may lead to reasonable results, such as in the example in Figure 7.3.2, where the only possible sequence of actions responsible for the transition system is $\langle \mathsf{SL} \rangle$ for the first one and $\langle \mathsf{SL}, \mathsf{C} \rangle$ for the second transition.

Superfluous action applicability in Secure-MAFS

Let us now analyze what information about action applicability can be inferred from the execution of Secure-MAFS. As we have already seen, in general, it may not even be possible to infer the sequence of actions used to generate the transitions. But even if we are able to determine a single sequence of actions (or even a single action) for each transition, we might not learn much more.

As no two states which differ only in the private part of the agent α are ever communicated, a privately-independent action can be determined only in the case of two states which differ in both private parts of the agent and of the adversary. To do so, the adversary would need to observe the action applied on at least two such states. In the example, the candidate states are s_1 and s_4. As the heuristic estimate is equal for both states, the difference of s_1, s_4 can be determined only if the uniqueness of their private parts is retained. Assuming so, the action SL can be determined to be privately-independent.

Similar restriction holds for privately-dependent actions, again, the adversary would need to observe such action applied on a state s and never applied on a publicly equivalent but distinct state s', where s and s'also differ in the private part of the adversary. In the example in Figure 7.3.2, the adversary only knows that C is applied on some state s'_4 resulting from the application of SL on s_4. There is no information about its applicability on any other state and thus its private dependency is not revealed.

Information leakage in Secure-MAFS

Based on the above analysis, we can compute the total information leaked. Similarly to Section 7.3.2 we first compute the transition systems represented by each action based on the leaked (and a priory) information as $\tau_{\mathsf{post}}(\mathsf{SL}) = \mathsf{min}(t_a, t_a^{\mathsf{pi}}, t_a^{\mathsf{ia}}, t_a^{\mathsf{ia}\times\mathsf{pi}}) = \mathsf{min}(189, 144, 135) = 135$ and $\tau_{\mathsf{post}}(\mathsf{C}) = \mathsf{min}(t_a) = 225$. Notice that the action C now leaks no information. The a priori information is the same as in Section 7.3.2, that is, $t_{\mathsf{apriori}} = 12^p 15^p = 32400$. The a posteriori information now differs

$$t_{\mathsf{post}} = \prod_{a \in \mathcal{O}^{\mathsf{pub}}} \tau_{\mathsf{post}}(a) = \tau_{\mathsf{post}}(\mathsf{SL})\tau_{\mathsf{post}}(\mathsf{C}) = 135 \cdot 225 = 30375$$

Thus, the lower bound on the information leakage of this particular execution of the Secure-MAFS algorithm on this particular example is

$$H_\infty(H) - H_\infty(H|L) = \log \frac{t_{\mathsf{apriori}}}{t_{\mathsf{post}}} \cong \log(1.1) = 0.1$$

bits of information. And again, we obtain the expected probability of guessing the correct transition system as

$$2^{-H_\infty(H|L)} = 2^{-\log t_{\mathsf{post}}} = \frac{1}{t_{\mathsf{post}}} = \frac{1}{30375} \cong 3 \times 10^{-5}$$

which is practically the same as when having only the a priori information.

In the particular example, the Secure-MAFS algorithm leaked some information (applicability of the SL action), but this leakage is very low in the final computation result. Notice that if the adversary α_{base} had no private variables, the algorithm would leak no information at all. We have shown that the tools which were able to deduce a significant amount of information from the execution of plain MAFS are much less powerful in the case of Secure-MAFS, but strong enough to show a non-zero leakage even on such a small example. Thus, according to the presented analysis, the Secure-MAFS algorithm is indeed not strong privacy-preserving in general but improves significantly on plain MAFS.

Comparison of leakage in MAFS and Secure-MAFS in general

We conclude the section with stronger and more general statements. We start by showing that Secure-MAFS does not leak more information than MAFS by first showing that Secure-MAFS does not send more states than MAFS.

Theorem 112. *Let $\mathcal{M}$ be a MAP problem. During the executions of the Secure-MAFS and MAFS algorithms on $\mathcal{M}$ assuming the same heuristic function and the same tie-breaking between states with the same heuristic value, the agent α does not send more states in Secure-MAFS than in MAFS.*

Proof. Let us, for contradiction, assume there is a state s sent by agent α in Secure-MAFS, but not in MAFS and let s be the first such state. Let s' be the β-parent of s. The state s' was sent by α in both Secure-MAFS and MAFS executions. In Secure-MAFS, there are two possibilities how s is created from s':

(i) s is created from s' by the application of a sequence of actions $a_1, ..., a_k$ such that a_k is public and thus is sent afterward. But if s' was sent by α in MAFS, the sequence of actions $a_1, ..., a_k$ can be applied as well, resulting in s which is sent because a_k is public.

(ii) s is created from s' by α using a stored transition representing the application of the sequence of actions $a_1, ..., a_k$. As above, if s' was sent by α in MAFS, necessarily the sequence of actions $a_1, ..., a_k$ can be applied resulting in s which is sent because a_k is public.

We have obtained a contradiction in both cases (i) and (ii) and there is no other way the state s can be sent by α in Secure-MAFS. □

From Theorem 112 follows that in Secure-MAFS, the adversary can reconstruct only a sub-tree of the search tree which can be reconstructed by the adversary in MAFS. The conclusion directly follows.

Corollary 113. *On a MAP problem $\mathcal{M}$ with the assumptions of Theorem 112, Secure-MAFS does not leak more information than MAFS.*

Moreover, the information about privately-nondeterministic actions does not leak in Secure-MAFS, formally:

Theorem 114. *Let $a^{\triangleright}$ be a privately-nondeterministic action of agent α. In the Secure-MAFS algorithm, this information never leaks.*

Proof. From Definition 99, the privately-nondeterministic action $a^{\triangleright}$ represents two actions a', a'' such that $\mathsf{eff}(a') \neq \mathsf{eff}(a'')$. Thus the application of such action $a^{\triangleright}$ on a state s results in two publicly equivalent but distinct states $s' = a' \circ s$ and $s'' = a'' \circ s$. As a', a'' do not influence private parts of other agents, the only different parts of s' and s'' are the parts private to the agent α. But this means that in Secure-MAFS, only one of s', s'' is sent by α and thus the information about $a^{\triangleright}$ is never revealed. □

Now, based on Corollary 113 and Theorem 114 we can state the following result.

Corollary 115. *Let $\mathcal{M}$ be a multi-agent planning problem and $a^{\triangleright}$ a privately-nondeterministic action. Assume that Secure-MAFS and MAFS use the same heuristic function and the same tie-breaking between states with the same heuristic value. If the information about $a^{\triangleright}$ leaks in the execution of the MAFS algorithm, then Secure-MAFS preserves strictly more privacy than MAFS on $\mathcal{M}$.*

We can reformulate Corollary 115 in a weaker but more general way as follows.

Corollary 116. *Let $\mathcal{M}$ be a multi-agent planning problem and $a^{\triangleright}$ a privately-nondeterministic action. If the information about $a^{\triangleright}$ leaks in every execution of the MAFS algorithm, then Secure-MAFS preserves strictly more privacy than MAFS on $\mathcal{M}$.*

7.3.4 Relaxation Heuristics

The family of relaxation heuristics includes all distributed heuristics presented in Chapter 4, the admissible max heuristic (Section 6.1), and the LM-Cut heuristic (Section 6.2). In this section, we present an analysis of privacy leakage of relaxation heuristics in general and of the particular algorithms.

The core general technique used in relaxation heuristics is the relaxed reachability analysis (see Algorithm 1) which is used to find facts (or variable-value pairs) reachable in the relaxed problem. Unlike reachability analysis in the original problem, relaxed reachability analysis is linear in the size of the problem due to the monotonicity of adding new reachable facts and actions.

Although there are multiple variants of distributed relaxed reachability analysis (number of them is presented in this thesis, Chapter 4), the underlying idea is always the same–the agents inform other agents about reachable public facts which would otherwise be unreachable for the agent or might be reached earlier by some of the other agents. A basic distributed reachability analysis is shown in Algorithm 2. Now, let us analyze, how much privacy leaks in this basic case.

Privacy leakage of the distributed reachability analysis

The distributed reachability algorithm (Algorithm 2) is iterative it two dimensions. First are the local iterations in the local reachability analysis (Algorithm 1) where the facts are added based on applicable actions of the agent α_i. Second are the communication rounds, that is, once the agent reaches its local fixed point, it sends all reachable public facts to all other agents. This may, in turn, enable some of the other agents to apply new actions and thus reach new public facts. But some public facts may not be reachable due to private preconditions of some of the actions necessary to reach the fact. Let us first formalize the concepts of (relaxed) reachability.

Definition 117. A public proposition p is *reachable* from a set R of propositions (i.e., relaxed state) by a set A_p^+ of relaxed actions, if by applying the relaxed exploration

$$R' \leftarrow \texttt{Relaxed-Reachability}(A_p^+, R)$$

from Algorithm 1, the returned set R' of propositions contains p, that is, $p \in R'$.

Definition 118. A public proposition p is *minimally reachable* from a set R of propositions (i.e., relaxed state) by a set $A^+_{p-\mathsf{min}}$ of relaxed actions, if $A^+_{p-\mathsf{min}}$ is a set of actions from which p is reachable minimal in $|A^+_{p-\mathsf{min}}|$ out of all such sets.

What can be this reachability used for? If, for a given set of propositions R, we compare the set of propositions reachable by the set of projected operators $A_i^{\rhd+}$ and by the set A_j of the actual operators of some agent α_j, we can determine whether some of the operator in A_j has private preconditions and thus is privately-dependent.

In particular, at the beginning of procedure `receiveMessage`$(\alpha_i, M_{\mathsf{REACH}} = \langle \alpha_j, R^{\mathsf{pub}} \rangle)$, we append the following line:

$$R^{\rhd} \leftarrow \texttt{Relaxed-Reachability}(A_j^{\rhd+}, R^{\mathsf{G}}) \cap P^{\mathsf{pub}}$$

where $A_j^{\rhd+} = \{a^{\rhd} | a \in A_j^{\mathsf{pub}+}\}$ is the set of public projections of relaxed public actions of agent α_j and R^{G} is the set of globally reachable propositions before the message M_{REACH} was received. By comparing $R^{\rhd}$, which are the public propositions reachable by the projected actions of agent α_j, with R^{pub}, which are the public propositions reachable by all relaxed actions of agent α_j, we can determine the following.

Proposition 119. *Let $R^{\mathsf{dif}} \leftarrow R^{\rhd} \setminus R^{\mathsf{pub}}$ be the set of propositions reachable from R^{G} by $A_j^{\rhd+}$, but not by A_j^+. Then for each $p \in R^{\mathsf{dif}}$, there is an action $a \in A_j^{p+} \subseteq A_j^{\mathsf{pub}+}$, where p is reachable from R^{G} by A_j^{p+}, such that a is privately-dependent (Definition 95).*

Proof. For a contradiction, let us assume that there is a set $A_j^{p+} \subseteq A_j^{\mathsf{pub}+}$ of actions such tat p is reachable from R^{G} by A_j^{p+} and for all $a \in A_j^{p+}$ holds that a is not privately-dependent, that is, $\mathsf{pre}(a) \cap P_j^{\mathsf{priv}} = \emptyset$. But in that case, for all actions $a \in A_j^{p+}$ applicable in R^{G} holds that $a^{\rhd}$ is also applicable in $R^{\mathsf{G}} \cap P^{\mathsf{pub}}$. Consider the call of `Relaxed-Reachability()` in Algorithm 2, line 5 with A_i^+ and $A_i^{\rhd+}$ respectively. Let $R_0 = R^{\mathsf{G}}$ in the first case and $R_0^{\rhd} = R^{\mathsf{G}} \cap P^{\mathsf{pub}}$ in the second case. Observe that for all $a' \in A_j^{p+}$ in R_0 also $a^{\rhd}$ is applicable in $R_0^{\rhd}$. Then necessarily $R_1 \cap P^{\mathsf{pub}} = R_1^{\rhd}$ and the same applicability implication holds. Let R_k be the final set of reachable propositions, by the described induction, $R_k \cap P^{\mathsf{pub}} = R_k^{\rhd}$ and as $p \in P^{\mathsf{pub}}$ we get a contradiction with the assumption that $p \in R^{\mathsf{dif}} \leftarrow R_k^{\rhd} \setminus R_k^{\mathsf{pub}}$. □

The Proposition 119 is important because as it is based on the most simplistic version of the distributed reachability analysis (i.e., Algorithm 2), all heuristics based on distributed reachability analysis

leak this information or more. Of course, not all distributed relaxation heuristics are built on this algorithm, e.g., the lazy FF variants use only projected reachability analysis. Let us now have a closer look at the particular relaxation heuristics presented in Chapter 4 and Chapter 6 and see how they differ from the simplistic algorithm analyzed so far.

Privacy leakage of the particular relaxation heuristics

The distributed relaxed plan construction of the MAFF heuristic (Section 4.1) aims to construct exactly the same Relaxed Planning Graph (RPG) as in the centralized version. In order to do so, the algorithm does not communicate only the reachability information as in Algorithm 2. First, the algorithm communicates reachable actions, not just propositions, second it communicates the earliest layer of appearance $e(a)$ of each action. The Proposition 119 can be directly applied to actions instead of facts by considering the preconditions or effects of the particular action as the set of reachable facts. Moreover, utilizing the information about the earliest layer of appearance, the adversary can gain more precise information. In particular, if the earliest layer of appearance of the projected reachability analysis is lower than in the case of the distributed one, that is, $e(a) > e(a^{\triangleright})$, the Proposition 119 can be applied even though the sets of reachable propositions or actions is the same. Plainly said, $e(a) > e(a^{\triangleright})$ means that either a, or some action necessary to apply a, is privately-dependent.

The recursive distribution of relaxation heuristics (Section 4.2) replaces the basic Algorithm 2 with the distributed exploration-queue-based Algorithm 3. Here the distribution principle is that if a proposition p is to be enqueued due to a projected action of some other agent α_j, a request for the heuristic value of p is sent to α_j and the proper value is returned when computed by α_j. Again, by applying the principle used in Proposition 119, we can compare the value obtained by the distributed computation with the readily available projected value. If the projected value is lower than the actual value, using the same logic, at least one of the actions necessary to reach p must be privately-dependent.

The Privacy-Preserving Set-Additive Fast-Forward heuristic (Section 4.3) hides the private information by communicating only public actions and heuristic estimates. In the heuristic computation, agent α_i directly asks agent α_j about the applicability of actions. More precisely, if agent α_i is computing a heuristic for state s, it computes a projected relaxed plan π_i^+. If such projected relaxed plan contains a projection of some action $a^+ \in A_j^{\mathsf{pub}}$, α_i requests α_j to provide the heuristic estimate of reaching the private preconditions of a^+ from s. Agent α_j sends a reply containing public actions used to reach the private preconditions and the number of private actions used to reach the private preconditions. On one hand, this directly gives away the existence of private preconditions of a^+. Unless the reply contains no public actions and the number of private actions is 0, the action a^+ is privately-dependent. On the other hand, the requests are sent only for the public actions present in π_i^+ (and received in subsequent replies), that is, only for public actions present in a single relaxed plan.

The admissible $h_{\max}$ heuristic described in Section 6.1 works on a slightly different principle of relaxed reachability as it starts with the projected heuristic and gradually updates the facts and actions to the correct values. If the adversary is the initiator agent, it directly asks the participants to update the heuristic values of public action. Then clearly, as all the preceding public actions are already updated to the correct value if a heuristic value of an action a is updated, that means that there is a private precondition maximizing the $h_{\max}(a)$ value previously not considered. Thus, even without any additional comparison to the projected heuristic (as the algorithm starts with it), the updated value leaks the information that a is privately-dependent, but this information is revealed only if any of the private preconditions maximizes the $h_{\max}$ heuristic (i.e., has the highest $h_{\max}$ value out of the preconditions of a).

The distributed LM-Cut heuristic (Section 6.2) uses a number of the distributed $h_{\max}$ computation and therefore leaks at least as much information as the $h_{\max}$ heuristic. As the cost is changed during the computation of the heuristic, different preconditions of actions may become the $h_{\max}$ maximizing precondition and thus more information may leak even using only the analysis described above.

Moreover, any of the heuristics can be used to determine publicly equivalent but distinct states

beyond the pure heuristic value. Assuming deterministic computation (which is not always the case in distributed computation), by comparing intermediate values (such as $h_{\max}$ values for the actions and facts) the difference between the states may be detected even if the final heuristic values are equal.

7.3.5 Potential Heuristics

In this section, we analyze how much privacy is compromised when using the distributed potential heuristic presented in Section 6.4. We first focus on the use of the heuristic itself, assuming that we already have securely computed potentials. Next, we analyze how such secure computation of potential might proceed and how much security it guarantees.

Privacy of computation of the Heuristic Value

Summing-up the potentials to obtain the heuristic value is not an issue as there exist secure sum algorithms (e.g., [Sheikh et al., 2010]), or the heuristic value can be computed from the parent state by Equation 6.4.5. Now let us have a look at what can be deduced from the public potentials known by all agents. An agent α_j may deduce some information private to α_i by computing the potentials of public variables $\mathsf{pot}^j(\langle V, v\rangle)$ on the j-projected problem $\Pi^{\triangleright j}$ and comparing them with those obtained from the global LP computation (somewhat analogously to Proposition 119). Let us, for now, assume that the LP computation leaks no information at all. Although the potential $\mathsf{pot}(\langle V, s[V]\rangle)$ of a public variable $V \in \mathcal{V}^{\mathsf{pub}}$ is influenced only by constraints respective to public operators and by the goal constraint, a public operator may also have private effects which also influence the constraint as follows.

Theorem 120. *Let $V \in \mathcal{V}^{\mathsf{pub}}$ such that $V \notin \mathsf{vars}(s_\star)$, let $\mathsf{pot}(\langle V, v\rangle)$ be a potential of the fact $\langle V, v\rangle$ in the MAP problem $\mathcal{M}$, and $\mathsf{pot}^j(\langle V, v\rangle)$ be a potential of $\langle V, v\rangle$ in the projected problem $\Pi^{\triangleright j}$, both computed using the LP described in Section 6.4. If $\mathsf{pot}^j(\langle V, v\rangle) \leq \mathsf{pot}(\langle V, v\rangle)$ then for some agent α_i there exist an operator $o \in \mathcal{O}^{\mathsf{pub}_i}$ such that $V \in \mathsf{vars}(\mathsf{eff}(o))$ and o has also some private variable in the effect, that is, o is a private-effect action (Definition 104).*

Proof. As from the definition of MA-MPT, the goal variables are public, each private variable $V' \in \mathcal{V}^{\mathsf{priv}_i}$ is represented in the LP goal constraint as $\mathsf{maxpot}_{V'}$. Let $V \in \mathcal{V}^{\mathsf{pub}}$ such that $V \notin \mathsf{vars}(s_\star)$ and let $\mathsf{pot}^j(\langle V, v\rangle) \leq \mathsf{pot}(\langle V, v\rangle)$ for some $v \in \mathsf{dom}(V)$. This means that some constraint in the global LP forces the potential $\mathsf{pot}(\langle V, v\rangle)$ to be lower than in the projected case. As $V \notin \mathsf{vars}(s_\star)$, this cannot be the goal constraint and thus the only possibility is that for some agent α_i and some operator $o \in \mathcal{O}^{\mathsf{pub}_i}$ the consistency constraint

$$\sum_{V \in \mathsf{vars}(\mathsf{eff}(o))} (\mathsf{maxpot}(V, \mathsf{pre}(o)) - \mathsf{pot}(\langle V, \mathsf{eff}(o)[V]\rangle)) \leq \mathsf{cost}(o)$$

forces $\mathsf{pot}(\langle V, v\rangle)$ to be lower. As for each public variable $V \in \mathsf{vars}(\mathsf{eff}(o)) \cap \mathcal{V}^{\mathsf{pub}}$ the summand respective to V is the same in the projected and global variant of the constraint, the only possible reason is, that there is an additional summand in the global variant. Such summand can only be because of a $V \in \mathsf{vars}(\mathsf{eff}(o)) \cap \mathcal{V}^{\mathsf{priv}_i}$ which is a private effect of o. □

If there is no private-effect operator in the agents problem Π_i, but $V \in \mathsf{vars}(s_\star)$, the agent α_j may only deduce the existence of (at least one) private variable. Based on this analysis we state the following.

Corollary 121. *If no agent α_i has a public operator with a private variable in the effect, the public potentials reveal that $\mathcal{V}^{\mathsf{priv}_i} \neq \emptyset$ at most.*

Proof. Let for some $\alpha_{j \neq i}$ and $V \in \mathcal{V}^{\mathsf{pub}}$ hold $\mathsf{pot}^j(\langle V, v\rangle) \neq \mathsf{pot}(\langle V, v\rangle)$. As no agent α_i has no public operator with a private variable in the effect, the only constraint that can influence the value of $\mathsf{pot}(\langle V, v\rangle)$ is the goal constraint. From the goal constraint, α_j can deduce at most that $\mathcal{V}^{\mathsf{priv}_i} \neq \emptyset$. □

Even though the assumptions of Corollary 121 seem very restrictive, they hold in many benchmark and real-world problems and are the result of compiling out private effects. Example of a domain for which Corollary 121 holds is the satellite domain, where all actions are private except for the goal-achieving actions, which have public effects only.

So far, we have analyzed the privacy leakage of the potentials themselves, let us now have a look at what information can leak from the LP computation. We present a number of techniques which can be used to compute the LP and analyze each one separately.

Plain global LP

The baseline approach is to let one of the agents compute the complete LP plainly as it is. Even in such a simple setting, some privacy is preserved. This is due to the fact that the LP does not reflect preconditions of actions on variables which are not also in the effect. This means that it is not possible to reconstruct the complete isomorphic model of an operator o, if o has some variable in precondition which is not present in effect.

Now let us have a closer look on how much privacy is compromised.

Proposition 122. *The number of variables and sizes of their domains are not compromised by sharing the private part of the LP.*

Proof. As the maximum potential constraints (Equation 6.4.1) are formulated as $\mathsf{pot}(\langle V, v\rangle) - \mathsf{maxpot}_V \leq 0$ they are indistinguishable from a constraint encoding a 0-cost private operator. Thus it is not clear which LP-variable is encoding a fact and which is encoding the maximum potential. There is also no connection between the LP-variables encoding potentials and maxpot_V of private variables, therefore it cannot be determined which facts belong to a single variable. This means that the number of variables in $\mathcal{V}^{\mathsf{priv}_i}$ and the sizes and values of their respective domains cannot be determined. □

The exact number of private operators is not compromised for the very same reason as above, the maximum potential constraints are indistinguishable from 0-cost operator constraints. The privacy of operators with $\mathsf{cost} > 0$ is partially compromised because constraints encoding operator with $\mathsf{cost} > 0$ can be identified as their right-hand side is > 0. For such operator o, the size of $\mathsf{eff}(o)$ (i.e., the number of facts) can be determined as the number of LP-variables with negative coefficients in the constraint. The size of $\mathsf{pre}(o)$ is not compromised, as some of the LP-variables with positive coefficients may be encoding maximum potentials. Also a fact f s.t. $f \in \mathsf{pre}(o) \backslash \mathsf{eff}(o)$ is not reflected in the constraint. The cost of nonzero-cost operators is revealed by the right-hand side of the constraints. Some relation between operators can be deduced from the use of the same LP-variables in precondition and effect of different operators, such as operator a consumes a fact produced by operator b. The reconstructed relations may not be complete for the reasons mentioned before.

The potential heuristic LP represents a planning task in the transition normal form (see [Pommerening and Helmert, 2015]), which is equivalent to the original task except for the prevail conditions, that is, preconditions on variables which are not in effects. If there are no prevail conditions in the original problem, the plans for the original and the transition normal form task differ only in zero cost operators (so called forgetting operators) which are obtained by interpreting the maximum potential constraints as 0-cost operators. For each fact $\langle V, v\rangle$ there is a 0-cost operator with precondition $V = v$ and effect $V = \bot$ where $\bot$ denotes a none-of-those value and corresponds to maxpot_V. We can state the following.

Proposition 123. *Let $o \in \mathcal{O}^{\mathsf{pub}_j}$ such that for all $V \in \mathsf{vars}(\mathsf{pre}(o)) \cap \mathcal{V}^{\mathsf{priv}_j}$ holds $V \notin \mathsf{vars}(\mathsf{eff}(o))$ or $\mathsf{pre}(o)[V] = \mathsf{eff}(o)[V]$. Then the fact that o is privately-dependent (Definition 95) does not leak from the potential heuristic LP.*

Proof. No prevail conditions (i.e., preconditions on some variable V which have a different value assigned in $\mathsf{eff}(o)$) are represented in the potential heuristic LP. If all private preconditions of o are prevail conditions, no private precondition of o is represented in the LP and thus cannot leak. □

Decomposed Global LP

One may attempt to improve the privacy by using a decomposition algorithm, such as Dantzig-Wolfe decomposition, in a similar way as proposed in [Holmgren et al., 2009]. The principle of the Dantzig-Wolfe decomposition is that the problem is decomposed into a *master* part where many variables have non-zero coefficients and independent sub-matrices. For the sub-matrices hold that if a variable has a non-zero coefficient in one sub-matrix, it has zero coefficient in all other sub-matrices. If such decomposition is possible, it is used to compute the solution by iteratively generating columns for the master problem based on the sub-problem solutions.

In the case of the potential heuristic LP, the constraints for private actions, which contain only private variables would be in the sub-problem factors, whereas all other constraints would be in the master problem. In comparison to the Plain Global LP, the decomposition hides the private operators and their costs but does not provide any formal guarantees on the privacy of the computation process (the original aim of the decomposition algorithm is to improve the efficiency of the computation).

Secure LP computation

Another approach is the use of a privacy-preserving transformation of the whole LP, which is often used in secure multi-party computation. There are basically two approaches to secure LP computation. The first approach is to implement an LP algorithm (e.g., simplex) using secure MPC primitives as in [Toft, 2009]. The second approach is to use a classical LP solver, but with securely transformed inputs. The second approach was extensively criticized in [Bednarz et al., 2009], but in later publications, the critical issues were solved, e.g., in [Dreier and Kerschbaum, 2011]. A significant benefit of the second approach is the speed of computation comparable to the plain LP computation, whereas, in the case of the MPC-based implementations, such as [Toft, 2009], the computation is several orders of magnitude slower. A simpler alternative to [Dreier and Kerschbaum, 2011] was published in [Mangasarian, 2011]. This simpler transformation is applicable only on data with specific properties which are, nevertheless, satisfied in the case of the potential heuristic LP. The technical details of the transformation based on [Mangasarian, 2011] and our modifications necessary for the potential heuristic LP computation are described in Section 6.4.3. Here we focus on the privacy properties of the approach.

The secure LP computation based on [Mangasarian, 2011] (with our modifications) reveals plainly only the cost of private operators and an upper bound on their number (more constraints than the number of private actions can be sent), but without the rest of their isomorphic image. The number of variables is hidden by the k value. Any information about private preconditions and effects of public operators is hidden by the random matrix transformation, assuming the random transformation is secure. The privacy leakage analysis presented in Section 7.2.2 is in its current state not applicable on such probabilistic algorithms (i.e., including random matrix generation), but the underlying techniques from [Smith, 2009] can be applied to probabilistic algorithms and thus it is possible to extend our techniques as well.

In [Dreier and Kerschbaum, 2011] the authors generalize the transformation to arbitrarily partitioned problems and provide formal security analysis. In their approach, the agents follow a protocol similar in idea, but rather complex, thus the description of the protocol is out of the scope of the thesis. There is no information openly shared in contrast to the previous case. There is also a low and quantifiable probability[4] of an attacker succeeding in revealing any part of the original LP, although having only inequality constraints as in our case increases the chances. Similar probability can be expected for [Mangasarian, 2011], although it has not been provided in the literature.

7.3.6 Multi-Agent Cost-Partitioning

Similarly to the potential heuristics, the use of multi-agent cost-partitioning consists of two operations. One operation is the computation of the cost-partitioning, the other is the use of the cost-partitioned

[4]The authors provide an example with 180 constraints and 282 variables, where the probabilities are below 10^{-220}.

partial heuristic values to compute the actual value. In the following text, we briefly analyze both cases, starting with the computation of the final heuristic.

Privacy of Additive Heuristics

Any heuristic h computed in a distributed way (and revealed to the adversary agent) can be used to distinguish two states such that $s^{\triangleright} = s'^{\triangleright}$ and $h(s^{\triangleright}) \neq h(s'^{\triangleright})$, based on Proposition 109. In the case of a heuristic computed using the multi-agent cost-partitioning (Equation 6.5.1), where h_{cp_i} is a heuristic computed by α_i on $\Pi^{i^{\triangleright}}$ based on the cost-partitioning cp_i, the situation is a little bit more complex. If the sum is computed plainly, that is, each agent α_j provides agent α_i with the value of $h_{\mathsf{cp}_j}(s^{j^{\triangleright}})$, the information revealed is not only that $s \neq s'$, but also that $s^{j^{\triangleright}} \neq s'^{j^{\triangleright}}$ for each j such that $h_{\mathsf{cp}_j}(s^{j^{\triangleright}}) \neq h_{\mathsf{cp}_j}(s'^{j^{\triangleright}})$.

Algorithms for secure sum computation from the literature [Sheikh et al., 2010] can be easily used to hide the parts of the additive heuristic. Moreover, it may not be necessary to compute such sum altogether. In distributed forward state-space search (e.g., MAD-A* [Nissim and Brafman, 2012]), the value of $h(s)$ is sent together with the state. Then if agent α_i wants to expand $s^{i^{\triangleright}}$ with a private action $a \in \mathcal{O}^{\mathsf{priv}_i}$ such that $s' = a \circ s$, because a does not change any part of $\Pi^{j^{\triangleright}}$ for any $j \neq i$, the heuristic of state s' can be computed as

$$h(s') = h(s) - h_{\mathsf{cp}_i}(s^{i^{\triangleright}}) + h_{\mathsf{cp}_i}(s'^{i^{\triangleright}})$$

so that the only information revealed is any heuristic with the same values. Of course, this approach cannot be used for public actions, except for the orthogonal abstraction based cost-partitioning, and is not practical if $h_{\mathsf{cp}_i}(s^{i^{\triangleright}})$ is computationally intensive.

Privacy of the Cost-Partitioning Computation

Let us first have a look on the cost-partitionings which are not LP-based.

Theorem 124. *(Strongly Private Cost-Partitionings) The computation of the uniform, the orthogonal abstraction based, and the privacy-compensating cost-partitionings is strong privacy-preserving.*

Proof. There is no information exchanged between any agents in the computation of either cost-partitioning. □

Regarding the LP-based cost-partitioning computation, the same secure LP computation as in Section 7.3.5 can be used. The technique of [Mangasarian, 2011] is applicable only on vertically partitioned LPs, which means that each agent has to own a subset of the LP variables. In the case of the OCP, the LP consists of state and heuristic LP variables for each projection thus satisfying the requirement (each projection falls into the partition of the respective agent). The same can be said about the landmark-based, SEQ-based, and potential heuristic-based LPs. The constraints can be shared by multiple partitions (agents) in which case the constraint is split according to the variables. This technique does not encrypt the right-hand side vector of the LP, which is not a problem in the OCP and landmark-based case, where the right-hand side is either 0, or the cost of a public action. In the case of the SEQ-based LP, the right-hand side of the LP represents the upper and lower bounds which may potentially leak some information. The technique of [Dreier and Kerschbaum, 2011] is applicable in the general case and encrypts the whole LP.

7.4 Theoretical Limits of Strong Privacy

In this section, we step aside from the question of privacy leakage and the measurement of such in existing algorithms and instead focus on the very idea of strong privacy. A strong privacy-preserving

planner is such a planner, which does not enable the adversary to deduce any private information of the agent except for what can be deduced from the public input and public output. In other words, the adversary cannot learn more than what it would learn if there was a trusted third-party able to solve the multi-agent problem and return only the final solution to each of the participants, see Definition 90. The question we aim to answer in this section is, whether there can be such a strong-privacy preserving planner and for what cost (e.g., in terms of efficiency and completeness).

Although in Section 7.4.1 we show that such planner exists, we also show that only for the price of either inefficiency, rendering it practically unusable, or incompleteness which may be the only practical approach to strong-privacy preserving planning (Section 7.4.2).

In Section 7.4.3 we propose a fine-grained definition of strong privacy, where some aspects of the problem are known a priori (e.g., that it is a logistics task) and only the details are left strongly private. We show that a complete, efficient, and strong privacy-preserving planner exists for such restricted notion of strong privacy.

7.4.1 A Strong Privacy Preserving Planner

In this section, we propose a PP-MAP algorithm (or rather a family of algorithms) based on a private set intersection (PSI), which is a well known primitive in MPC. Several approaches to computationally secure PSI has been proposed in [Pinkas et al., 2015, Jarecki and Liu, 2010]. An information-theoretic approach was proposed in [Li and Wu, 2007] which provides unconditional security, as long as at least $n/2$ parties are semi-honest. Alternatively, the proposed PP-MAP planner can be based on another MPC primitive, a (computationally) privacy preserving intersection of deterministic finite automata (DFA) [Guanciale et al., 2014].

In particular, the algorithms are based on the PSM structure[5] and the generate and test paradigm. Based on [Tožička et al., 2016], a set S of plans for a (STRIPS) planning problem Π (that is, Π_i or $\mathcal{M}^{\triangleright}$) can be represented by a planning state machine (PSM).

Definition 125. (PSM) Let $\Pi = \langle P, A, s_I, s_\star \rangle$ be a STRIPS planning problem and S a set of solutions of Π. A planning state machine (PSM) $\Gamma(S) = \langle \Sigma, N, s_I, \delta, F \rangle$ is a deterministic finite automaton (DFA) where the alphabet Σ contains the STRIPS labels of the actions in $a \in A$ s.t. $\Sigma = \{\mathsf{lbl}(a) | a \in A\}$, states are sets of facts $N \subseteq 2^P$ with $s_I \in N$, transitions satisfy that $\delta(s, \mathsf{lbl}(a)) = s'$ iff the action a transforms the state s into another state s' and accepting states are $F = \{s \in N | s_\star \subseteq s\}$. The PSM $\Gamma(S)$ accepts a sequence of actions π iff $\pi \in S$.

A plan π is accepted by $\Gamma(S)$ if it is a solution to Π and $\pi \in S$. If $\Gamma(S)$ contains all solution to Π we call it a full PSM and denote it $\Gamma(\Pi)$. An important advantage of the PSM structure over a set of plans is that a PSM remains finite even if S is infinite and it is also possible to construct it in finite time, even though this time can be exponential in the size of the problem.

A public projection of PSM $\Gamma(S)$ is $\Gamma(S)^{\triangleright}$, where each state s is replaced with a public projection $s^{\triangleright}$ and each transition representing a private action is replaced with an ϵ-transition. The ϵ-transitions are then eliminated using standard DFA algorithm and thus the PSM is minimized.

The generic structure of a PSM-based planner is listed in Algorithm 16. If S_i is finite, PSI can be used instead of a DFA intersection. By instantiating each step of this scheme we create several types of PSM planners:

One-shot-PSM planner generates a proper random subset of all solutions in Step 1. and terminates in Step 4. if a solution is not found.

Iterative-PSM planner repeats all steps until the intersection $\bigcap_{i=1}^{n} \Gamma(S_i)^{\triangleright}$ is nonempty, or all agents have constructed a full PSM $\Gamma(\Pi_i)$, in which case if the intersection is empty, there is no solution. In each iteration of Step 1., new plans are added systematically (e.g., ordered by length).

[5]The PSM structure and a planner based on it was originally published in [Tožička et al., 2016]. The use of secure DFA intersection is a novel contribution in [Tožička et al., 2017b], as well as the One-shot-PSM planner.

Algoritmus 16: Generic PSM-based Planner

Algorithm `GenericPSM(`$\mathcal{M}$`)`

1. Each agent $i \in \{1, ..., n\}$ generates a set S_i of local solutions of Π_i, stored in a PSM $\Gamma(S_i)$.
2. Each agent i computes a public projection $\Gamma(S_i)^{\triangleright}$.
3. Initiate secure computation protocol
   (a) All agents compute together the intersection $\bigcap_{i=1}^{n} \Gamma(S_i)^{\triangleright}$ using a secure DFA intersection. The intersection is not revealed to all agents.
   (b) If $\bigcap_{i=1}^{n} \Gamma(S_i)^{\triangleright} \neq \emptyset$, the intersection represents a nonempty set of global solutions to $\mathcal{M}$, continue with Step 4.
   Otherwise, either terminate and report no solution, or continue with Step 1.
   (c) Jointly and securely select one random solution from $\bigcap_{i=1}^{n} \Gamma(S_i)^{\triangleright}$.
4. Reveal the selected solution to all agents.

Full-PSM planner each agent constructs a full PSM $\Gamma(\Pi_i)$ in Step 1. If the problem has a solution, all solutions are found in the first iteration of Step 4.

The Iterative-PSM and Full-PSM planners were published in [Tožička et al., 2016], albeit without the use of secure DFA intersection, whereas One-shot-PSM is a novel variant of the planner. Notice that both One-shot-PSM and Full-PSM planners are computationally strong privacy preserving as the secure DFA intersection by [Guanciale et al., 2014] is computationally strong privacy preserving and no other communication is performed. Also, in the case of One-shot-PSM, an information-theoretic PSI [Li and Wu, 2007] can be used as the used sets of plans can be finite and thus One-shot-PSM can be strong privacy preserving in the information-theoretic sense (without any assumptions). Another promising feature of the One-shot-PSM planner is that there is a trade-off between completeness and efficiency, which can be exploited. The more plans are generated, the more time it takes, but also the higher is the chance of success in the one shot secure PSM intersection.

Before formulating these observations formally, we provide an algorithm for secure selection of a random solution from the intersection of PSMs, as is required in Step 5.

Random solution selection

In [Guanciale et al., 2014] the authors propose an algorithm for a secure intersection of regular languages, which is based on DFA minimization, secure intersection and secure trimming of unreachable states. As PSM is an instance of DFA, this techniques can be used also to securely compute an intersection of the agents' PSMs and securely remove unreachable states, that is, obtain a minimal DFA representing the resulting PSM. The next step we need to perform is to select a random solution, again, without leaking private information (Step 5. of Algorithm 16). To prevent information leakage, the intermediate intersection cannot be revealed and the random solution selection must be an integral part of the secure intersection algorithm.

To select a random solution (a public plan) from the intersection of PSMs $\bigcap_{i=1}^{n} \Gamma(S_i)^{\triangleright}$, we use the Algorithm 17. The algorithm iteratively selects a random transition (action) from δ of $\bigcap_{i=1}^{n} \Gamma(S_i)^{\triangleright}$ leading from its initial state s_I through intermediate states $s \in N$ and eventually terminates in one of the terminals in F (the goal states). In states where we can both continue with a transition from δ or terminate, we will chose randomly whether to continue with one of the randomly selected tran-

sitions or whether we will terminate. The resulting random trace through the PSM will be in form $(\mathsf{lbl}(a_1) \in \Sigma, \ldots, \mathsf{lbl}(a_k) \in \Sigma)$. The extracted plan is then straightforwardly $\pi = (a_1, \ldots, a_k)$.

Algoritmus 17: Secure random DFA trace algorithm.

```
1  Algorithm SecureRandomTrace(Γ(S) = ⟨Σ, N, s_I, δ, F⟩)
2      π ← ∅; //The trace
3      s ← s_I; //Current state
4      while ⊤ do
5          T ← {⟨s' ∈ N, l ∈ Σ⟩ | δ(s, l) = s'}; //Find all transitions from s
6          r ← |T|; //The range to choose the transition
7          if s' ∈ F then
8              r ← |T| + 1; //Add possibility for terminating
9          x ← random integer from ⟨0, r);
10         if x > |T| then
11             return s; //Terminate
12         s ← s ∪ {l}; //Add label to the trace
13         s ← s'; //Proceed with next state
```

The Algorithm 17 can be implemented as a final step of the secure DFA intersection using the SecreC language in the Sharemind [Bogdanov et al., 2008, Bogdanov, 2013] system as a single secure protocol. The iterative concatenation of the trace (which is the public plan), as well as the number of iterations, does not have to be hidden as it is part of the public output. The used random variable has a uniform distribution which does not reveal any additional information as well. Note that Algorithm 17 can work only with minimal DFA (in a non-minimal DFA it could randomly end up in a state which is not a terminal and there is no outgoing transition from it), but the output of the FDA intersection algorithm by [Guanciale et al., 2014] is minimized also as part of the same secure protocol.

Example of privacy-preserving planning

Let us show how the presented PSM-based algorithms work on a simple case of a coalition surveillance mission problem with one UAV and two secret locations (see Figure 7.1.1), where UAVs survey an area and need to be refueled by coalition partners (a coalition base), the surveyed areas and the state of supplies of the base are private (secret). We omit the movement actions for simplicity (movement between the surveyed locations would be private, the movement to the coalition base would be public).

The public projection of the problem is restricted to the public propositions P^{pub} and the public projections of the actions. Note that as $\mathsf{SL1}$ and $\mathsf{SL2}$ have the same projection, they cannot be distinguished and thus are represented by a single projected action $\mathsf{SL}^{\triangleright}$. The same holds for the actions R and RR which are both represented by a projected action $\mathsf{R}^{\triangleright}$.

Full PSMs and their projections for both agents are shown in Figure 7.4.1. Their intersection equals to the full PSM of the α_{UAV} agent. That is how the *Full-PSM planner* works. On the other hand, when using *One-shot-PSM planner*, the α_{base} agent can decide to add only one local solution to its PSM, namely the local plan $\{\mathsf{R}, \mathsf{C}\}$. In that case, the intersection of agents' PSMs is empty and thus the planner ends without finding a solution. Of course, even inthe *One-shot-PSM planner*, both agents may add multiple solutions and find a global solution.

In the case of *Iterative-PSM planner*, the α_{base} agent adds another plan $\{\mathsf{C}\}$ to its PSM, which still yields an empty intersection. In the third iteration, the α_{base} agent adds also $\{\mathsf{R}, \mathsf{SL}, \mathsf{C}\}$ to its PSM and thus represents all necessary plans. The intersection of such PSM with the α_{UAV} PSM is non-empty and contains a solution of the problem. In this case, the α_{base} agent knows that the α_{UAV} agent does not accept plan $\{\mathsf{R}, \mathsf{C}\}$, which leaks private information. Note that the α_{base} agent cannot deduce this

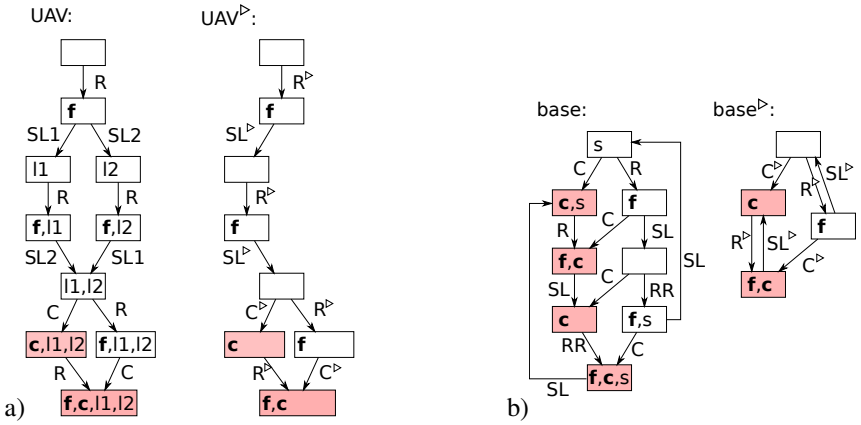


Figure 7.4.1: a) Full PSM of the α_{UAV} agent and its public projection. b) Full PSM of the α_{base} agent and its public projection.

information in the cases of *Full-PSM planner* and *One-shot-PSM planner* because the intersection of PSMs is not known to the agents and the final solution is selected randomly from all solutions encoded by the PSM intersection.

7.4.2 The Limits of Strong Privacy Preserving MAP

In this section, we present theoretical limits of the privacy preserving planner described above and their generalization to other PP-MAP paradigms. For the privacy analysis, we assume that the agents know the algorithm used by all other agents (also including probability distributions of any random variables, e.g., the uniform distributions in the random solution selection) and also that the MAP problems cannot be solved by a single agent alone (in which case a strong privacy-preserving algorithm is trivial). We focus on the following three properties

Definition 126. (MAP Planner properties) A MAP planner P is

(i) Complete if for each MAP problem $\mathcal{M}$ that has a solution (a global plan), P terminates and returns a solution to $\mathcal{M}$ (Definition 38ii).

(ii) Strong privacy preserving if P does not reveal any other information than what can be deduced from the public part of the input and the solution, that is, $\mathcal{M}^{\triangleright}$ and $\pi^{\triangleright}$ (Definition 90).

(iii) Efficient if there exists a MAP problem $\mathcal{M} = \{\Pi_i\}_{i=1}^{n}$ for which P always returns a solution without enumerating all public solutions of each Π_i.

The completeness definition does not require any further explanation and the privacy definition has already been discussed. As already mentioned, based on the presence of computational assumptions, we distinguish two flavors of strong privacy preserving algorithms, computational and information-theoretic. The efficiency definition is somewhat unusual. The aim is to differentiate between algorithms which do have to explore the complete private search spaces of the agents (which, in the worst case, can be as large as the global problem) and those which do not. Although the theoretical complexity class is the same for both, as the worst case is always the complete exploration, in most practical problems, the difference is significant (i.e., the difference between blind breadth-first search and heuristic search such as A*).

The Limits of the PSM-based Planners

In this section, we assign the properties from Definition 126 to the particular PSM-based planner variants.

Theorem 127. *The Full-PSM planner is* complete *and computationally* strong privacy preserving.

Proof. Let $\mathcal{M} = \{\Pi_i\}_{i=1}^n$ be a MAP problem. The Full-PSM generates a full PSM $\Gamma(\Pi_i)$ for each Π_i and a public projection $\Gamma(\Pi_i)^\triangleright$, each representing all local solutions of each respective Π_i and their public projection. Then a PSM intersection $\bigcap_{i=1}^n \Gamma(\Pi_i)^\triangleright$ is computed. If a global solution $\{\pi_i\}_{i=1}^n$ to $\mathcal{M}$ exists, each π_i is a local solution to Π_i and thus is represented by $\Gamma(\Pi_i)$. Because $\pi_i^\triangleright = \pi_j^\triangleright$ for each i, j, $\pi_i^\triangleright$ is represented by the intersection $\bigcap_{i=1}^n \Gamma(\Pi_i)^\triangleright$ and is a public projection of a global solution, privately extensible by all agents. Thus Full-PSM is complete.

If the intersection $\bigcap_{i=1}^n \Gamma(\Pi_i)^\triangleright$ is computed using a privacy preserving DFA intersection by [Guanciale et al., 2014], Full-PSM is computationally strong privacy preserving as no other multiparty computation or communication is performed and no other information is exchanged. □

Corollary 128. *The Full-PSM planner is not* efficient.

Proof. Trivial, as the Full-PSM from definition always generates all local solutions of all agents before computing the intersection. □

Theorem 129. *The Iterative-PSM planner is* complete *and* efficient.

Proof. Let $\mathcal{M} = \{\Pi_i\}_{i=1}^n$ be a MAP problem and let $\{\pi_i\}_{i=1}^n$ be a global solution to $\mathcal{M}$. Even though the number of all possible solutions to $\mathcal{M}$ may be infinite, each such solution $\{\pi_i\}_{i=1}^n$ is finite and has a length l. As Iterative-PSM is adding the plans in a systematic way (that is, a plan of length $k+1$ is added only after all plans of length k were added), all plans of length l are added to each S_i after a finite number of steps. Thus also all the plans in $\{\pi_i\}_{i=1}^n$ are added after a finite many steps and the solution becomes part of the intersection $\bigcap_{i=1}^n \Gamma(S_i)^\triangleright$ and thus Iterative-PSM is complete.

Let l be the length of the shortest global solution. Based on the systematic generation described above, the solution is always found before all solutions of length $l' > l$ and thus Iterative-PSM is efficient according to Definition 126(iii). □

Theorem 130. *The Iterative-PSM planner is not* strong privacy preserving.

Proof. By iterating the PSI or secure DFA intersection, information is leaked. In particular, information that a plan π_i shorter than the solution proposed by agent i is not extensible by some agent $j \neq i$. This reveals the existence of private preconditions of some public actions of agent j used in π_i (that is, privately-dependent action, Definition 95). Note that as we assume the knowledge of the algorithm by all agents, even a less obvious systematic generation of plans leaks information as the particular algorithm can be simulated by other agents and the plans which should have already been generated can be determined. In the case of randomized algorithms, we assume the knowledge of the probability distributions of random variables used in the algorithm as part of the algorithms and thus again, the information about unaccepted public plans leaks. □

Theorem 131. *The One-shot-PSM planner is* strong privacy preserving *and* efficient.

Proof. By computing the secure DFA intersection only once, no additional information can leak and thus One-shot-PSM is computationally strong privacy preserving. If each PSM $\Gamma(S_i)$ is replaced by a finite subset of represented plans S_i, the PSI can be used instead of the DFA intersection. By using an information theoretic secure PSI [Li and Wu, 2007] on finite sets of plans, One-shot-PSM becomes information-theoretic strong privacy preserving. One-shot-PSM is trivially *efficient* according to Definition 126(iii) as it can use an arbitrarily small subset of all possible local solutions. □

Theorem 132. *The One-shot-PSM planner is* not complete.

Proof. As some public solution is not generated by at least one of the agents, it may be the case that the not-generated solution is the one and only solution of the problem and thus such problem would not be solved. □

Impossibility Theorem

Next, we state the main contribution of this chapter. First, we present a formulation for the class of PSM-based planners and later we generalize it to a wider class of planning algorithms.

Theorem 133. *(Impossibility Theorem) A PSM-based MAP planner P cannot have all three properties (Definition 126)* complete*,* strong privacy preserving *and* efficient *together.*

Proof. According to Theorem 127, the Full-PSM is complete and strong privacy preserving, but according to Corollary 128 not efficient as it generates all local solutions. For the sake of contradiction, let us have a complete and strong privacy preserving planner P which is efficient. From Definition 126(iii) follows that there exist a MAP problem $\mathcal{M} = \{\Pi_i\}_{i=1}^n$ for which some of the agents using P do not have to generate all public plans in order to find a global plan $\{\pi_i\}_{i=1}^n$, let j be such agent. Let $\bar{\pi}_j^{\triangleright} \neq \pi_j^{\triangleright}$ be the public plan which is not generated by agent j.

Because we assume that the problem $\mathcal{M}$ cannot be solved by a single agent only, a MAP problem $\bar{\mathcal{M}}$ can be constructed from $\mathcal{M}$ so that the only public plan extensible by all agents is $\bar{\pi}_j^{\triangleright}$. It is enough, if one of the agents rejects all public plans not equal to the public plan $\bar{\pi}_j^{\triangleright}$ and therefore the newly constructed MAP problem $\bar{\mathcal{M}}$ can differ from $\mathcal{M}$ only in the problem of one agent, let that be agent i. The construction is as follows. Let $\bar{\pi}_i$ be a local plan of agent i such that $\bar{\pi}_i^{\triangleright} = \bar{\pi}_j^{\triangleright}$, that is, $\bar{\pi}_i$ can be part of the global plan as it is the extension of $\bar{\pi}_j^{\triangleright}$ by agent i.

We construct $\bar{\Pi}_i$ from $\Pi_i = \left\langle P_i = P^{\mathsf{pub}} \cup P_i^{\mathsf{priv}}, A_i, s_I \cap P_i, s_\star \cap P_i \right\rangle$ by adding a new proposition p_k for each public action $a_k \in \bar{\pi}_i^{\triangleright}$ s.t. $a_k \in A_i$ and by adding a new proposition p_{neg}. We add p_0 to s_I and modify each such a_k so that $\mathsf{pre}(\bar{a}_k) = (\mathsf{pre}(a_k) \cap P^{\mathsf{pub}}) \cup \{p_k\}$, $\mathsf{add}(\bar{a}_k) = (\mathsf{pre}(a_k) \cap P^{\mathsf{pub}}) \cup \{p_{k+1}\}$ or $\mathsf{add}(\bar{a}_k) = \mathsf{pre}(a_k) \cap P^{\mathsf{pub}}$ if a_k is the last action and $\mathsf{del}(\bar{a}_k) = \mathsf{del}(a_k) \cap P^{\mathsf{pub}}$. We modify each $a'_k \in A_i$ s.t. $a_k \notin \bar{\pi}_i^{\triangleright}$ so that $\mathsf{pre}(\bar{a}_k) = (\mathsf{pre}(a_k) \cap P^{\mathsf{pub}}) \cup \{p_{\mathsf{neg}}\}$. The result is that only actions in $\bar{\pi}_i$ are applicable and only in the exact same order, also keeping the public constraints in place, thus $\{\bar{\pi}_i\}_{i=1}^n$ is the only global solution to $\bar{\mathcal{M}}$.

Since P is strong privacy preserving and $\mathcal{M}^{\triangleright} = \bar{\mathcal{M}}^{\triangleright}$ as the public part was not modified, the agent j cannot distinguish between $\mathcal{M}$ and $\bar{\mathcal{M}}$ and thus generates exactly the same PSMs $\Gamma(\Pi_j) = \Gamma(\bar{\Pi}_j)$ for both problems. But then, as the planner P is complete and $\bar{\mathcal{M}}$ has the only solution $\{\bar{\pi}_i\}_{i=1}^n$, the agent j has to generate $\bar{\pi}_j$ also for $\mathcal{M}$. Thus we obtain a contradiction with the assumption that the planner P is efficient (because it has to also generate $\bar{\pi}_j$), in other words, that a strong privacy preserving and complete planner can generate less local plans than Full-PSM which generates all of them and thus violates the efficiency property according to Definition 126(iii). □

Example. (UAV) To illustrate the above proof, we will modify the UAV example. Let α_{UAV} be the agent j and let $\pi_{\alpha_{\mathsf{UAV}}}^{\triangleright} = \{\mathsf{SL}, \mathsf{R}, \mathsf{SL}, \mathsf{R}, \mathsf{C}\}$ be the public plan which is not generated by the α_{UAV} agent. Let the corresponding local plan of the α_{base} agent be $\pi_{\alpha_{\mathsf{base}}} = \{\mathsf{SL}, \mathsf{R}, \mathsf{SL}, \mathsf{RR}, \mathsf{C}\}$. Then the problem of the α_{base} agent can be modified so that $P_{\alpha_{\mathsf{base}}}^{\mathsf{priv}} = \{p_1, p_2, p_{\mathsf{neg}}\}$ and each action in the plan is modified so that $\mathsf{pre}(\mathsf{R}) = \{\neg\mathbf{f}, p_1\}$, $\mathsf{add}(\mathsf{R}) = \{\mathbf{f}, p_2\}$, $\mathsf{pre}(\mathsf{RR}) = \{\neg\mathbf{f}, p_2\}$ and if there was any other action of the agent α_{base}, its private preconditions would be set to $\{p_{\mathsf{neg}}\}$. Also, the private part of s_I is set to $\{p_1\}$ and thus only the action R is applicable.

The properties of PSM-based planners according to Definition 126 and the above theorems are summarized in Figure 7.4.2. Notice that the intersection of all properties is empty, as shown by the

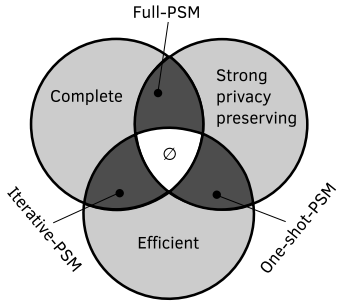


Figure 7.4.2: Properties of PSM-based planners.

Theorem 133. Also, the Full-PSM and One-shot-PSM are the only generally strong privacy preserving planners published up to date, thanks to the novel use of the secure DFA intersection.

Notice that the Theorem 133 holds not only for the PSM Planner [Tožička et al., 2016] and the family of PSM-based planners described in Algorithm 16, but also for algorithms based on similar generate and test paradigm, such as Planning First [Nissim et al., 2010], which uses distributed constraint satisfaction (DCSP) in the place of PSM intersection. The potential use of a secure version of DCSP would result in similar limits as are the limits of the PSM planner.

The Limits of State-Space Search

In this section, we focus on the strong privacy preserving property in general terms. We abstract from the particular secure computations such as PSI, secure DFA intersection or even a single ORAM computation and refer to them collectively as secure primitives. The important property of a secure primitive is that (possibly under computational security assumptions), no information leaks from a single secure primitive and thus, on its own, it is *strong privacy preserving*. In general, combining multiple secure primitives may leak information, as shown e.g., in Theorem 130. Formally:

Definition 134. (Secure primitive) A secure primitive (or a cryptographic primitive) is a (possibly multiparty) computation block which by itself is strong privacy preserving.

We first state two general results applicable to any MAP planning algorithm and then use them to generalize Theorem 133 to wider classes of MAP planners. The definitions are based on the notion of publicly equivalent MAP problems:

Definition 135. (Public equivalence) Two MAP problems $\mathcal{M}$, $\mathcal{M}'$ are publicly equivalent if $\mathcal{M}^{\triangleright} = \mathcal{M}'^{\triangleright}$ and their respective sets of public solutions $S^{\triangleright}$ and $S'^{\triangleright}$ are equal, formally $\mathcal{M} \equiv \mathcal{M}'$.

Theorem 136. *Let P be a MAP planner and $\mathcal{M}$, $\mathcal{M}'$ two publicly equivalent MAP problems $\mathcal{M} \equiv \mathcal{M}'$, such that $\mathcal{M}$ and $\mathcal{M}'$ differ in the private part of agent i. Then if P is strong privacy preserving, it performs the same number of secure primitives on both $\mathcal{M}$ and $\mathcal{M}'$.*

Proof. Trivially, if P performs a different number of secure primitives on $\mathcal{M}$ and $\mathcal{M}'$, the agents other than i can distinguish between $\mathcal{M}$ and $\mathcal{M}'$, which is an information that cannot be learned from the output, as the public projections of both solutions are equal. □

Corollary 137. *The number of secure primitives performed by a strong privacy preserving MAP planner P cannot depend on any private part of the MAP problem $\mathcal{M}$.*

Proof. Direct consequence of Theorem 136. □

Note that the whole planning algorithm can be considered a secure primitive, if it is strong privacy preserving, e.g., the Full-PSM planner.

The Theorem 136 and Corollary 137 are very general, but also provide necessary conditions for any strong privacy preserving MAP planner. This conditions can be used to generalize the Theorem 133 to a wider class of MAP algorithms.

Definition 138. (State-space search) A state-space search (SS) MAP planner is a MAP planner in which each agent searches its own state-space. The agents coordinate themselves by exchanging public projections of reachable states.

An example of a SS MAP planner aiming for secure computation is Secure-MAFS [Brafman, 2015] which is strong privacy preserving for a restricted class of problems.

Corollary 139. *Theorem 133 holds for any SS MAP planner, assuming a bound b on the number of states in the global state space of $\mathcal{M}$.*

Proof. Let us assume that the states in a strong privacy preserving SS MAP planner P are communicated in a secure way, that is, no information is leaked by communicating a single state and thus the communication of a single state can be considered a secure primitive. According to Corollary 137, the number p of secure primitives must depend only on the public part of $\mathcal{M}$. In order to be *complete*, p must be large enough even for the worst case execution, which is if the state space is of size b and all states are expanded and sent. But this corresponds to enumerating all local solutions of $\mathcal{M}$ and thus breaks the *efficiency* property according to Definition 126(iii). □

Based on [Štolba et al., 2016c], the forward-chaining plan-space search as used in FMAP [Torreño et al., 2014] essentially corresponds to the state-space search paradigm and thus the same results apply.

As already mentioned, the question whether a generic MPC technique such as ORAM or BlindTM can be used for efficient strong privacy preserving MAP planning has been discussed in Section 2.3. This concludes the theoretical analysis of the limits of strong privacy preserving multi-agent planning, both in general and in particular case of the PSM-based planners.

7.4.3 Strong Privacy Preserving Equivalence Classes

In [Brafman, 2015], the author proves that Secure MAFS is strong privacy preserving on a restricted class of logistics problems, where the problems have the same set of packages, the same set of public locations, and identical initial public locations for packages and also that every private location is reachable from every other private location. This effectively means that a part of the private problem is irrelevant (that is, it can always be solved) and the rest of the private problem is the same for all instances of the restricted class of problems. In this section, we formalize the notion of privacy used in the proof in [Brafman, 2015] and generalize the idea of privacy on a class of problems.

Definition 140. (Strong privacy on a class) A MAP planner P is *strong privacy preserving on class C* of MAP problems iff from the execution of P on $\mathcal{M} \in C$ and on $\mathcal{M}' \in C$ no agent can distinguish $\mathcal{M}$ and $\mathcal{M}'$.

This definition of strong privacy differs from that in Definition 90, but is reasonable. If solving e.g., a logistics problem, even if the fact that a package is loaded is private, the agents can expect its existence based on how logistics works. Now we formalize the equivalence class of problems based on Definition 135.

Definition 141. (Public equivalence class) A class C of MAP problems is a *public equivalence class* iff for each two $\mathcal{M} \in C$ and $\mathcal{M}' \in C$ holds $\mathcal{M} \equiv \mathcal{M}'$.

This means that $\mathcal{M}^{\triangleright} = \mathcal{M}'^{\triangleright}$ and their respective sets of public solutions $S^{\triangleright}$ and $S'^{\triangleright}$ are equal. For each of the problems $\mathcal{M} \in C$ by itself, the private part of $\mathcal{M}$ poses a constraint and thus reduces the number of public solutions which are also global solutions. But as all problems in C have equal public projection and also the set of public solutions, each of the problems in C differs from the other problems only by such part of the private problem, which does not add more constraints and prevent more solutions, that is, the different private parts of the problems can always be solved.

An example of such class C is the logistics problems used in [Brafman, 2015] and rephrased at the beginning of this section. The particular problems differ by the actual number of private locations, but as the private locations are always connected (not necessarily directly) to a public location, this part of the problem does not constraint the public solutions which are also global solutions of the whole problem. Nevertheless, this does not mean that the private part of $\mathcal{M}$ is unnecessary—the agents still need to cooperate in order to solve $\mathcal{M}$ and some of the public solutions are not extensible because of the private parts of the agent problems.

Based on the Definition 141, we can formulate a general result.

Theorem 142. *A MAP planner P which is complete and efficient by Definition 126 and strong privacy preserving on a public equivalence class C of MAP problems by Definition 140 exists.*

Proof. Let us start with Iterative-PSM, which is complete and efficient by Theorem 129. In each iteration, a plan π_i proposed by agent i is either accepted by all other agents, in which case the algorithm ends and no information leaks, because π_i is part of the solution or π_i is not accepted and thus some information leaks. But in the case of public equivalence class C of problems, π_i is either accepted or not accepted in all $\mathcal{M} \in C$ and thus this information cannot be used to distinguish any two $\mathcal{M}, \mathcal{M}' \in C$. Therefore by Definition 140, Iterative-PSM is strong privacy preserving on class C. □

This theorem generalizes the results of [Brafman, 2015] to all public equivalence classes of MAP problems and also to MAP planners in general. Moreover, we can formulate the following corollary.

Corollary 143. *The public equivalence relation $\equiv$ partitions all MAP problems into classes of equivalence. There exists a MAP planner P, which is complete and efficient by Definition 126 and for each MAP problem $\mathcal{M}$, P is strong privacy preserving on a public equivalence class C of MAP problems, induced by $\mathcal{M}$.*

This means that each MAP problem $\mathcal{M}$ induces the class C of publicly equivalent problems, which can be solved by such planner P (e.g., the Iterative-PSM planner), revealing no other information than that the problem falls in the particular class C. It seems that for some practical applications, this might be enough to consider the planner strong privacy-preserving, as the participating agents already know the class of the planning problems they are solving in advance (e.g., the logistics problems with particular constraints).

Example. (UAV) Considering the UAV example, a problem $\mathcal{M}'$ which does not consider the supplies of the α_{base} agent (there is no private proposition) s falls in the same equivalence class as the original problem as the RR can always be used and provide supplies. Also, all problems which have more complex private parts (e.g., additional private actions for preparing the fuel, etc.) which do not restrict any solutions of the original problem fall in the same equivalence class C.

7.5 Summary

In this section, we have focused on the issue of privacy in multi-agent planning, thus fulfilling the **(Objective 3)** of the thesis. First, we have defined privacy and privacy leakage in the context of multi-agent planning. Second, we have provided a general technique to analyze such privacy leakage and applied in on the MAFS (thus also MADLA) and Secure-MAFS planning algorithms in detail. We

have also applied the techniques and insights gained on the distributed heuristic algorithms presented throughout the thesis, showing their strengths and weaknesses in the context of privacy. Finally, we have provided strong theoretical results, such as the Impossibility Theorem, which for a large class of MAP planning algorithms states that such algorithms can not be strong privacy preserving, efficient and complete at the same time. Moreover, we have formalized a restricted notion of strong privacy for which the Impossibility Theorem does not hold and for which we have presented a theoretical planner having all three mentioned properties.

Chapter 8

Conclusion

In my research, I have set out to understand information sharing in multi-agent planning, in particular, how sharing (or not-sharing) heuristic information influences the performance in heuristic search and how privacy is compromised by sharing information during multi-agent planning. In the thesis, the main research questions (and our answers) are the following:

(Objective 1) How to compute classical planning heuristics in a distributed way?

> We have published a number of works providing distributed variants of classical planning heuristics (mostly relaxation-based), descriptions of which can be found in Chapter 4 and Chapter 6. The latter also provides an example of an additive heuristic, which can provide global estimates without any additional communication during the search (Section 6.3). By analyzing the distributed variants of existing heuristics, we have gained much-needed insights which culminated in the general approach to distributed heuristic computation based on cost-partitioning presented in Section 6.5. We have also gained the understanding that, especially for the relaxation-based heuristics, neither the fully local (projected) approach, where no information is shared, nor the fully distributed approach, dominates the other on all planning domains.

(Objective 2) How to combine local and distributed heuristics?

> Since it turned out, that for the relaxation heuristics, the distributed variant is not always better than the local (projected) one (see Section 4.4.3), we aimed to find out, how to combine the two approaches. One of the first techniques is presented in Section 4.4.2. In Chapter 5 we present the Multi-Agent Distributed and Local Asynchronous (MADLA) Search which is the basis for the MADLA Planner. The MADLA Search is able to combine the local and distributed variants of the Fast-Forward heuristic so that the performance of neither of the two approaches is deteriorated. Thus the MADLA Planner was able to over-perform all state-of-the-art multi-agent planners at that time.

(Objective 3) How to formalize privacy and quantify privacy leakage and how to apply secure multi-party computation techniques in multi-agent planning?

> Finally, one of the most interesting and crucial, but often neglected, topic in multi-agent planning is privacy. As the existing formal tools for privacy analysis were fragmented and incomplete, our first goal was to provide a solid formal base on which we and other authors can build in the future. We have based our formalism described in Chapter 7 on the ideas from secure Multiparty Computation and information leakage quantification. We have used the formalism to analyze several MAP algorithms. The gained understanding also helped us

to provide more general theoretical results, giving us both the first provably strong privacy-preserving planner and the Impossibility Theorem. This theorem states that a wide class of MAP algorithms (entailing all currently used MAP paradigms) cannot be strong privacy-preserving, efficient, and complete at the same time unless computed as a single secure primitive operation. Nevertheless, we have provided a weaker, but still reasonable, concept of privacy for which such MAP planner exists (and was constructed in theory).

As we intended to evaluate the MADLA Planner and compare it with other MAP planners published at that time, we figured out that in order to perform the evaluation rigorously, we need to consolidate the input formalism, benchmarks and the runtime environments of the MAP planners. To do so, we have co-organized the first Competition of Distributed and Multi-Agent Planners (CoDMAP) which, against all odds and the initial uncertainty of how it will be accepted by the community, was a great success. The competition attracted nearly 10 participants from 6 countries, many of them entering with multiple planners or their variants. As this contribution is rather tangential to the main topic of the thesis, we have left the description of the competition to the Appendix A.

We have fulfilled the three main objectives presented in the thesis. The answers to the research questions provide us with many valuable insights into multi-agent planning and distributed heuristic search in particular, but also open a number of new research venues and pose us with many more research questions. Here, we summarize the hottest research topics stemming from the thesis.

The computation of distributed heuristics, i.e., **(Objective 1)**, can be seen as solved by the general approach presented in Section 6.5, but the approach itself firstly needs a much deeper experimental evaluation and secondly can be improved, e.g., by providing better cost-partitioning, but also by closer integration with the distributed search. Still, there might be better heuristic distributed in an ad-hoc manner, or the heuristics presented in Chapter 4 and Chapter 6 may still be improved.

Regarding the **(Objective 2)**, we have provided a solution for satisficing planning, the MADLA Search. An interesting venue for future research is the application of similar approaches to optimal planning, which might be theoretically more challenging. Another option is to provide techniques combining the local and distributed approach specifically for a given heuristic, similarly as we have done in Section 4.4.3 for the case of recursively distributed relaxation heuristics. A promising idea might be to use iterative computation of the LM-Cut heuristic [Helmert and Domshlak, 2009] where the full distributed heuristic can be computed for some states and the local computation can be used for the iterative variant. By reusing the global landmarks and computing new landmarks locally, the communication load could be decreased while possibly retaining better heuristic guidance.

Possibly the largest open field of research in multi-agent planning is the privacy, that is, **(Objective 3)**. The privacy measure presented in Section 7.2.2 can be further refined and extended by tightening both the lower bound and the upper bound. The analysis of algorithms (Section 7.3) can be extended to more algorithms and, more crucially, can be implemented and used for experimental evaluation of leakage for various algorithms and their comparison. Ultimately, the privacy analysis should lead to the development and implementation of planning algorithms which leak less private information according to the presented measure, both in theory and in practice. An extreme case is the implementation of strong privacy-preserving MAP planners based on the secure MAP techniques. The simplest case is to implement the algorithm presented in Section 7.4, a much more challenging option is to implement the whole planning algorithm based on the secure multiparty computation techniques. It is not clear yet, whether it is even possible without significant relaxations of the privacy requirements.

The CoDMAP competition was a great success in consolidating the benchmarking of MAP planners, as many works have since adopted the methodology. A big challenge is to organize a second installment of the competition, possibly as a full IPC track. One significant hurdle is that one of the most important aspects of multi-agent planning identified in the thesis is privacy. It is far from clear, how to enforce its preservation or measure its leakage in the context of the competition, as the techniques presented

in Section 7.3 are mostly algorithm-dependent. But without such measures, a valid solution of the distributed track is to send the factors to a single machine and solve the problem centrally, which is not what would be intended by such competition.

As the world is getting more interconnected, large amounts of information are abundant, and both people and machines are required to plan, act, and interact in such complex environment, the questions of planning and multi-agent cooperative planning is getting more and more imminent. In this thesis, we have progressed the research field in several directions and opened even more for future endeavors, which we are looking forward to undergoing.

Appendix A

The Competition of Distributed and Multi-Agent Planners

Although not one of the main objectives of this thesis, proper and rigorous comparison of multi-agent planners arose as a significant hurdle in the further development of MAP planners and techniques. In order to tackle this issue, we have taken an inspiration in the International Planning Competition (IPC)[1] and organized the first Competition of Distributed and Multi-Agent Planners (CoDMAP)[2]. The competition was organized as preliminary and semi-official as it was not clear whether the format will attract enough participants. For the same reason, we have allowed the organizers (that is, us) and authors affiliated with the organizers to enter the competition as well, which is not a common practice in IPC. As the competition is somewhat tangential to the main topic of the thesis but is a significant contribution nevertheless, we include a description of the competition based on [Komenda et al., 2016, Štolba et al., 2015b] as an appendix.

The planners related to this thesis which entered the competition are the MADLA Planner [Štolba and Komenda, 2015] which entered exactly in the configuration described in Chapter 5 and MA-Plan [Fišer et al., 2015] using the LM-Cut heuristic, Multi-Agent LM-Cut heuristic described in Section 6.2, and a combination of re-implemented FF heuristic from Section 4.3 and a variant of DTG heuristic briefly described in Section 4.4.4. The PSM Planner which is a base of the planners presented in Section 7.4.1 participated in the competition as well.

In this appendix, we describe the decisions we have made, the rules and the language we have designed, and the results of the competition.

A.1 The Aims of the Competition

The first decision we had to make was how to restrict the multi-agent planning problems which would be covered by the competition. We chose an approach similar to that of classical planning, that is, start with the smallest possible subset of features and possibly extend them in the future. We wanted to take classical STRIPS planning and extend it with the smallest possible feature set to transform it into the multi-agent setting. Such an approach was already taken by Brafman&Domshlak in the case of MA-STRIPS formalism [Brafman and Domshlak, 2008].

Before designing the competition, we were aware of about a dozen of multi-agent planners more-or-less compatible with the MA-STRIPS formalism. One of the main focuses of the competition design was to allow as many of them as possible to enter the competition without large-scale modifications. In

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

[2] http://agents.fel.cvut.cz/codmap

order to foster our awareness of the existing planners and their possible extensions, we have conducted a public poll[3].

Out of the poll and other considerations arose three main restrictions of the multi-agent planning model:

STRIPS-like model This means deterministic, non-durative actions and full observability (with respect to privacy, which will be discussed later). This seems to be the simplest model, compatible not only with most of the current multi-agent planners but also with classical planners and classical planning techniques, which is good for comparison, reuse of the techniques and benchmarks.

Cooperative agents This is a very strong assumption, maybe one of the first candidates to be lifted. On the other hand, some competitive problems can be converted to the cooperative by automatic transformation of action costs using mechanism design [Nissim and Brafman, 2013].

Offline planning We have decided to stick to the offline planning paradigm as used in classical planning (input $\rightarrow$ planning $\rightarrow$ plan) in contrast to online planning as used in the probabilistic uncertainty IPC track.

In order to make the transition as smooth as possible for most of the planners, we have decided to run two tracks. The Centralized Track served as a transitional track, where the input is centralized and the planners can be centralized as well, which both contradicts common assumptions of multi-agent planning. Also, most of the language and formalism requirements (described later) can be ignored by the planners (but they have to state that in the description). The other, more ideal, Distributed Track forces stricter rules and also forces the planners to consume distributed (factored) input and run on multiple physical machines in a distributed fashion (each planning agent on one machine). In both tracks, the planning systems are evaluated separately (as in classical IPC), different planners are not interacting.

We have excluded the possibility of a decentralized track, where planners of multiple competitors would plan together cooperatively (or non-cooperatively) in order to find a common plan. Such track would require us to define some common protocol and is far beyond the abilities of most current planners.

A.2 MA-PDDL

Ever since the first IPC in 1998, the base input language of the competition was PDDL (Planning Domain Description Language) [McDermott et al., 1998]. PDDL is based on a subset of predicate logic and uses LISP-like syntax for describing planning domains and problem instances. Over time, the language was gradually enriched with more syntax and expressive power resulting in the latest official version, PDDL3.1. Originally, PDDL described only the STRIPS fragment (with a few ADL constructs). In contrast to STRIPS, PDDL is lifted, allowing parametrization of actions and facts (in form of predicates). Thus PDDL is able to represent transition systems of large planning problems compactly. Nevertheless, most of existing planners still work with a grounded description of the problem, so that during the grounding process, typed PDDL objects or PDDL constants are assigned as arguments of predicates and actions.

Following the minimalistic extension of STRIPS to MA-STRIPS by Brafman and Domshlak in 2008, we wanted a simple extension of PDDL towards multiagent planning, also compatible with MA-STRIPS. There were two existing candidates, MAPL [Brenner, 2003] and MA-PDDL[4] [Kovacs, 2012]. MAPL was published in 2003 and was rather a drastic modification of PDDL2.1, introducing many features not required by MA-STRIPS and missing the partitioning and privacy definitions. MA-PDDL was a more consistent extension of PDDL3.1, but still including many features (inherited mainly from

[3]The poll form can be found at: http://bit.ly/1IsNoqY

[4]The extended BNF can be found at http://agents.fel.cvut.cz/codmap/MA-PDDL-BNF.pdf

PDDL3.1) not needed in the first arrangement of a multiagent planning competition, which was decided to focus primarily on MA-STRIPS planning. Additionally, MA-PDDL did not describe privacy of facts and actions. Therefore, we have extended MA-PDDL with a partitioning definition and a definition of privacy of objects and predicates (and thus implicitly of the privacy of actions). The extension allowed defining agents in various ways: as objects, constants, or not at all. This variability allowed us to reuse many interesting classical planning benchmarks.

The definition of privacy in MA-STRIPS is implicit and follows a simple rule which says that a fact is public if it is required or modified by two or more actions of different agents. An action is public if it requires or modifies at least one public fact. Based on a review of literature and a conducted pre-competition poll, we found that such a definition could be too rigid, especially for future versions of the competition. We slightly relaxed the MA-STRIPS notion of privacy and declared it explicitly in the MA-PDDL description. Our privacy definition follows MA-STRIPS in the sense that facts and actions can be private to particular agents or public among all agents, however, what facts and actions are private and public is determined by a process coined as *maximally concealing grounding* (MCG).

MCG uses privacy defined over predicate, function and constant definitions in a MA-PDDL planning domain, while privacy over MA-PDDL objects was defined in the description of the related planning problem. In order to be able to represent MA-STRIPS problems, privacy was semantically defined by MCG over facts grounded from predicates, based on the following set of rules:

1. A predicate definition declared to be *public* in the domain description, grounded with only public objects/constants, results in a *public fact*.
2. A predicate definition declared to be *public* in the domain description, grounded with at least one object/constant private to agent α, results in a *private fact* of agent α (grounding a single predicate definition with objects private to different agents is not allowed, and an object/constant cannot be private to multiple agents).
3. A predicate definition declared to be *private* in the domain description, grounds to a *private fact* regardless of privacy of the objects used for grounding.

This definition of privacy is very close to the definition of privacy in MA-STRIPS, but allows for more. Starting from everything private to defining everything public, regardless of the use of predicates in actions. A fact used only by one agent can be even declared public or a fact used by multiple agents can be declared private.

By convention (which we have adhered to in the competition), a PDDL object representing an agent is private to that given agent. If it was not, other agents of the same PDDL type would be able to ground and use the other agent's actions. An alternative approach is to use PDDL constants to represent agents and to include only the proper partially grounded actions in an agent's domain description. Or we can represent agents not explicitly with PDDL objects/constants at all and just have all action definitions from the perspective of the particular agent to whom they belong.

For the competition, we have proposed two ways how to encode multiagent planning problems in MA-PDDL. Either as *factored* MA-PDDL, which allows the definition of separate domain and problem descriptions for each planning agent, targeting many for many multiagent planning. Or as *unfactored* MA-PDDL, targeting one for many planning, which allows the definition in a single domain and problem description, incl. partitioning and privacy.

Information-wise the two representations are equivalent. The difference is in the information separation as in the many for many planning case, it is important to provide the respective agents only with information allowed to them by the privacy requirements.

A.2.1 Unfactored MA-PDDL

The unfactored variant of MA-PDDL stems naturally from classical PDDL. It uses a pair of files, one containing the domain information and the other the specification of a problem instance within that

domain.pddl

```
(define (domain logistics)
(:requirements :typing
  :multi-agent :unfactored-privacy)
(:types location vehicle package city - object
  airport - location
  truck airplane - vehicle)
(:predicates (at ?obj - object ?loc - location)
  (in ?obj1 - package ?veh - vehicle)
  (:private ?agent - truck
    (in-city ?agent - truck ?loc - location ?city - city)))
(:action drive-truck
  :agent ?truck - truck
  :parameters (?loc-from - location
    ?loc-to - location
    ?city - city)
  :precondition ...
  :effect ... )
(:action load-truck :agent ?truck - truck ... )
(:action unload-truck :agent ?truck - truck ... )
(:action load-airplane :agent ?apn - airplane ... )
(:action unload-airplane :agent ?apn - airplane ... )
(:action fly-airplane :agent ?apn - airplane ... ))
```

problem.pddl

```
(define (problem logistics-4-0) (:domain logistics)
(:objects obj21 - package
  ...
  pos1 - location
  (:private apn1 apn1 - airplane )
  (:private tru2 cit2 - city tru2 - truck pos2 - location )
  (:private tru1 tru1 - truck cit1 - city))
(:init (at tru1 pos1) ... )
(:goal (and (at obj21 pos1) ... )))
```

domain-tru1.pddl

```
(define (domain logistics)
(:requirements :typing
  :factored-privacy)
(:types location vehice package city - object
  airport - location
  truck airplane - vehicle)
(:predicates (at ?obj - object ?loc - location)
  (in ?obj1 - package ?veh - vehicle)
  (:private
    (in-city ?agent - truck ?loc - location ?city - city)))
(:action drive-truck
  :parameters (?truck - truck
    ?loc-from - location
    ?loc-to - location
    ?city - city)
  :precondition ...
  :effect ... )
(:action load-truck ... )
(:action unload-truck ... ))
```

domain-tru2.pddl ...

domain-apn1.pddl ...

problem-tru1.pddl

```
(define (problem logistics-4-0) (:domain logistics)
(:objects obj21 - package
  ...
  pos1 - location
  (:private tru1 - truck cit1 - city))
(:init (at tru1 pos1) ... )
(:goal (and (at obj21 pos1) ... )))
```

problem-tru2.pddl ...

problem-apn1.pddl ...

Table A.1: An example comparing unfactored (left) and factored (right) MA-PDDL domain and problem descriptions in case of a 3-agent planning problem instance (two trucks, tru1 and tru2; and one airplane, apn1) of the extended CoDMAP version of the logistics00 domain (unessential details are omitted for the sake of simplicity). Bold elements highlight the differences between standard PDDL and extended MA-PDDL.

domain (see an example in Table A.1-left).

PDDL allows a straightforward extension by additional :requirements. Unfactored MA-PDDL is defined in terms of two new additional requirements, **(1)** :multi-agent, and **(2)** :unfactored-privacy. The former informs the planner, that action definitions are annotated with an additional :agent ?agent-parameter - type specification, which acts as any other PDDL parameter, however, a grounded action belongs to the agent named in the specification. Therefore any PDDL object or constant in ?agent-parameter typed by the type is treated as an agent. Using this extension over all actions unambiguously defines action partitioning.

The latter requirement, :unfactored-privacy, marks a PDDL description as privacy defining, and with the help of an additional :private ?agent-parameter - type block, and the above defined MCG rules, it unambiguously defines what facts are private to given agents, and which facts are public. If a predicate, a constant or an object is not defined private, it is treated as public. Privacy definition for actions follows MA-STRIPS, that is, an action is public if it depends on or modifies a public fact.

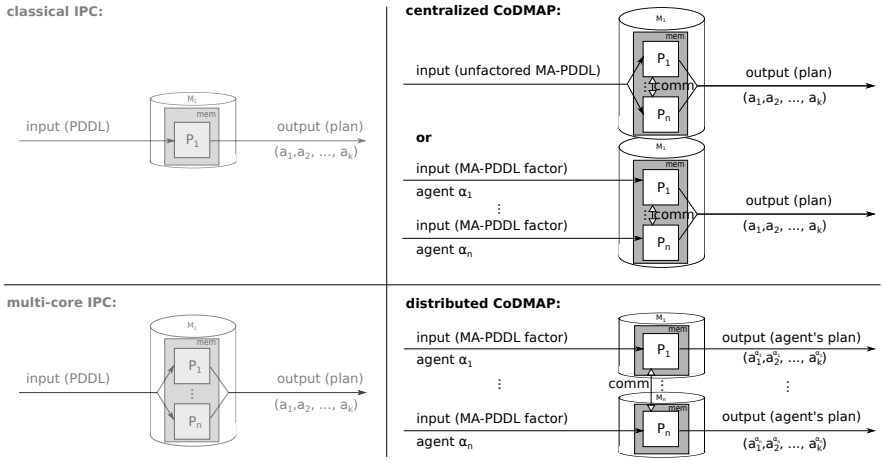


Figure A.3.1: Comparison of IPC and CoDMAP tracks.

A.2.2 Factored MA-PDDL

The motivation for factored MA-PDDL results straightforwardly from the distributed nature of multi-agent systems. Each separate planning agent uses its own pair of domain and problem description files (denoted as a MA-PDDL factor) which define information relevant only to that particular agent (see an example in Table A.1-right).

Action partitioning ensues directly from the decomposition of the input. As each planning agent's factor contains only relevant actions, there is an unambiguous grounding of them. Such partitioning produces single-agent PDDL descriptions, and thus does not need the :multi-agent extension, as in the unfactored variant. However, the planner has to be provided with a name of the agent it plans for (i.e. the name of the PDDL object/constant, which represents the agent) in an additional input. The competition rules defined that additional input as a command-line parameter. Public facts, objects, and constants which were common for more than one agent, were by convention bound over the same names. For instance, the obj21 - package of truck-agent tru1 (see an example in Table A.1-right, file *problem-tru1.pddl*) is by convention the same package as obj21 in the factored problem description of truck-agent tru1, *problem-tru2.pddl*.

To highlight that a factored MA-PDDL is a result of factoring (and thus may contain :private declarations), an additional requirement :factored-privacy was used. The grounding semantics of factors using MCG is the same as in the unfactored variant, the only difference is in the definition of the :private blocks, which do not need an explicit specification of the respective agent now, as the agent's capabilities are clearly defined by the partitioning to MA-PDDL factors.

A.3 Competition Rules

The full rules, used domains and results are published at the competition website[5]. Relation of the centralized and distributed tracks of the CoDMAP competition to the existing IPC tracks is shown in Figure A.3.1. Planners in the classical IPC tracks (both optimal and satisficing) take a pair of PDDL files (domain, problem) as an input, run on a single machine/single-core and output a sequence of actions as the plan. The multi-core track differs in that the planners may run on multiple cores/threads.

CoDMAP consists of two tracks:

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

- Centralized Track, aiming for maximal compatibility with classical IPC and existing multi-agent planners
- Distributed Track, aiming for a proper multi-agent setting.

A.3.1 Centralized Track

In the centralized track, the input of a planner is either a single unfactored MA-PDDL domain and problem description, or a separate factored MA-PDDL domain and problem description for each agent in the planning problem. The planner runs on a single machine, with no other restrictions or requirements (the planner may be as well single-core or multi-core, distributed or not, one thread per agent or multiple threads per agents, etc.). The provided input will have factoring and private separation according to MA-STRIPS, but the planners are not required to adhere to it. This is in order to enable planners built on different multi-agent planning models to enter the competition as well. The output of the planner must be a sound linear plan.

We do not restrict any communication between planning agents (if any), nor do we restrict the exchange of private information.

The rules are intentionally weak not to force the MA-STRIPS formalism and our view of communication and privacy on the competing planners. What we request is to accompany the planner with a short paper explaining all the parameters of the factorization, privacy, inter-agent communication (if any), architecture, etc., so they can be included in the final results and each participant can then derive their own conclusions based on the similarity of their planner with other competitors.

The difference between the centralized track and classical multi-core track is mainly in the input format (MA-PDDL) and also the planners are expected to somehow utilize the multi-agent nature of the given problems.

A.3.2 Distributed Track

The distributed track is much more strict in terms of the rules, but it is rather experimental in the sense that there are currently no planners capable of entering it without significant modification. The aim was to provide a track the way we think a multi-agent planning competition should look like.

The planners have to be truly distributed as shown in Figure A.3.1. Each planning agent of such a distributed planner receives its own factor of the factored MA-PDDL domain and problem, runs on its own dedicated machine and outputs its own plan. The MA-PDDL factorization and privacy definition must be adhered to. In most benchmarks, the factorization and privacy definition will follow the MA-STRIPS model but does not necessarily have to.

The planning agents of a distributed planner can communicate over TCP-IP (IP addresses of other planning agents will be known up-front), but they should avoid exchanging any private information (all such cases should be clearly explained in the accompanying paper).

The output is a linear plan for each agent, which can all be executed in parallel. The actions of all plans in each time step must not be in mutex (mutual exclusion).

A.4 Software Infrastructure

IPC provides an experimental platform for running and evaluating planners by various comparison criteria. Unfortunately, the platform is not appropriate for multiagent planning requiring a distributed run of planners on a cluster of homogeneous computers. The homogeneity requirement also complicates deployment of the experiments to standard cluster solutions as Amazon HPC[6] or MetaCentrum VO[7], as

[6] Amazon High-Performance Computing: http://aws.amazon.com/hpc/

[7] Czech academic computational cluster: https://metavo.metacentrum.cz/en/about/index.html

the assurances on the particular running computers, their physical location and therefore interconnection cannot be easily guaranteed.

The cluster we used for comparing the submitted planners was consisting of 23 identical physical machines interconnected by a 1Gbps network. Each machine had one quad-core AMD processor, at 3.9GHz (at 4.2GHz in case of utilization of only one core, respectively) and comprised 16GB RAM. One half of RAM was dedicated to a RAM-disk containing the operating system (Ubuntu 14.04.2 LTS) with the planners in their various configurations compiled and tested by the respective authors. The other half of RAM, 8GB, was used by the operating system and one planning process solving one planning task. The machines were interconnected into an IP subnet with a 10Gbps switch. The Ethernet cards in the machines were 1Gbps and all the computers were collocated in one room.

Our cluster fulfilled the homogeneity requirement. As we were aware of possible issues with running distributed systems authored by various teams on a cluster without any guarantees on up-time, a practical requirement was *an easy continuation of the experiments* in case of breakdowns. As our time for the experiments was rather limited, the second requirement was a *good utilization of the machines*. The last requirement was to have a possibility to *store all resulting and logging data* from the planners which can be used by the community to analyze the planners and their behavior in the competition environment.

The competition scheduler we have developed was a set of BASH scripts[8] allocating which planner configuration, domain, and problem will run on which machine (or more machines in the distributed case). The scheduler ran on an additional "coordination" machine which did not host planners. In contrast to the machines in the cluster, it had physical hard-drive to store the resulting and logging data. The task generation process and the execution of the tasks were decoupled, so we could reuse one particular randomized allocation of the problems to run the same allocation repeatedly in case of failure of the experimentation process. Additionally, the scheduling script generated a list of successfully run tasks which could be easily subtracted from the complete list to continue only with the pending tasks. The validation of the plans was done on the coordination machine as well. In the centralized case, the scheduler simply locked one machine for each task, therefore the utilization of the cluster was not an issue. In the distributed track, the scheduler greedily allocated more than one machine and synchronously run the planning agents of one multiagent planner. This approach utilized the cluster well enough for the competition requirements and had a straightforward implementation.

A.5 Selected Results

Some of the competing planners were submitted to the competition in several different configurations. For the centralized track, we have received 12 planners in 17 configurations prepared by 8 teams. For the distributed track 6 configurations of 3 planners by 3 teams. Complete, detailed, and interactive results can be found in the official competition web page[9]. In this section we present the results of planners most relevant to the planners presented in this thesis, that is, MA-STRIPS compatible planners which aim to present at least weak privacy by not communicating the private parts of the agent problems. This excludes some of the best performing planners such as ADP [Crosby, 2015], SIW [Muise et al., 2015] and CMAP [Borrajo and Fernández, 2015]. For full results, see the website of the competition. We present the results of an additional summer run where some of the planners have fixed bugs and the results include also the GPPP [Maliah et al., 2014] planner. The relevant planners are put in the context of this thesis in Chapter 2 and are described in detail in the competition proceedings[10].

The selected results of the centralized track are shown in Tables A.2. The table presents the coverage of the planners on the competition domains, that is, the number of problem instances solved given the time and memory limits. The results show that the best performing planner is MAPlan in the satisficing configuration, that is, using the distributed FF heuristic from Section 4.3 and the DTG heuristic initially

[8] Bash is the GNU Project's shell: http://www.gnu.org/software/bash/

[9] CoDMAP results: http://agents.cz/codmap/results

[10] http://agents.fel.cvut.cz/codmap/results/CoDMAP15-proceedings.pdf

Domain	blocksworld	depot	driverlog	elevators08	logistics00	rovers	satellites	sokoban	taxi	wireless	woodworking08	zenotravel	$\sum$
	20	20	20	20	20	20	20	20	20	20	20	20	240
Coverage													
MAPlan	**20**	13	17	11	18	**20**	**20**	**18**	**20**	**4**	16	**20**	**197**
GPPP	12	11	14	**20**	**20**	19	18	9	**20**	3	18	**20**	184
PSM	**20**	**17**	**20**	12	18	12	18	**18**	0	0	**19**	13	167
MADLA	17	4	16	18	19	**20**	19	10	9	1	7	18	158
MH-FMAP	0	2	16	9	4	8	17	4	**20**	0	9	13	102
Quality													
GPPP	10	11	8	**20**	**20**	17	17	9	**18**	3	**18**	**19**	**168**
MAPlan	16	9	11	8	11	**19**	**19**	11	16	**4**	14	11	150
PSM	**17**	**15**	**18**	9	14	11	8	**16**	0	0	15	10	133
MADLA	7	3	11	9	13	14	12	6	7	1	6	11	99
MH-FMAP	0	2	16	9	4	8	17	4	19	0	8	13	99
Time													
MAPlan	**19**	**12**	14	6	15	**19**	**19**	**17**	**20**	**4**	**16**	16	**176**
GPPP	7	8	8	**20**	**18**	9	11	5	14	2	10	**19**	131
MADLA	10	2	12	15	15	15	14	6	7	1	4	13	114
PSM	9	**12**	**17**	7	9	5	12	11	0	0	13	7	103
MH-FMAP	0	1	12	3	2	3	7	2	11	0	4	9	53

Table A.2: Selected CoDMAP centralized track coverage, quality, and planner speed results. The quality score (IPC Score) was computed as Q/Q* where Q is the cost of given solution and Q* is the cost of the best solution found by any of the planners (including those not included). The time score (IPC Agile Score) was computed as T/T* where T is the time of given planner and T* is the time of the best planner (including those not included).

Domain	block.	depot	driver.	elevat.	logist.	rov.	sat.	sok.	taxi	wire.	wood.	zeno.	$\sum$
	20	20	20	20	20	20	20	20	20	20	20	20	240
Coverage													
PSM-VRD	**20**	**16**	**20**	5	**16**	**18**	13	**17**	**20**	0	**19**	16	**180**
MAPlan/FF-DTG	14	10	18	**9**	**16**	**18**	**19**	14	19	**4**	14	**19**	174
MH-FMAP	0	2	18	**9**	4	8	18	4	**20**	0	8	16	107
PSM-VR	12	1	16	2	0	14	13	7	9	0	9	16	99
MAPlan/LM-Cut	2	5	15	2	4	1	2	13	19	3	3	6	75
MAPlan/MA-LM-Cut	1	2	9	0	5	1	4	4	14	2	4	6	52
Quality													
PSM-VRD	**17**	**15**	16	4	**15**	12	5	**13**	16	0	**17**	10	**140**
MAPlan/FF-DTG	7	6	12	6	13	**18**	16	10	15	**4**	13	**15**	135
MH-FMAP	0	2	**17**	**8**	4	8	**18**	4	17	0	7	**15**	100
MAPlan/LM-Cut	2	5	15	2	4	1	2	**13**	**19**	3	3	6	75
PSM-VR	11	1	14	1	0	9	5	6	6	0	8	10	72
MAPlan/MA-LM-Cut	1	2	9	0	5	1	4	4	14	2	4	6	52
Time													
MAPlan/FF-DTG	**14**	9	**17**	**8**	13	**18**	**19**	**13**	**19**	2	9	**18**	**159**
PSM-VRD	**14**	**14**	14	4	**14**	8	6	12	15	0	**18**	8	127
MH-FMAP	0	1	11	4	2	3	7	1	10	0	4	10	52
MAPlan/LM-Cut	1	3	10	1	3	0	1	10	14	**3**	2	5	52
PSM-VR	5	0	7	1	0	7	6	3	3	0	5	8	45
MAPlan/MA-LM-Cut	0	1	4	0	2	0	1	2	7	2	3	4	27

Table A.3: Complete CoDMAP distributed track coverage, quality, and planner speed results. The quality score (IPC Score) was computed as Q/Q* where Q is the cost of given solution and Q* is the cost of the best solution found by any of the planners. The time score (IPC Agile Score) was computed as T/T* where T is the time of given planner and T* is the time of the best planner.

proposed in [Torreño et al., 2014] and reimplemented by us (as described in Section 4.4.4). The MADLA Planner performs competitively only in the loosely coupled domains such as rovers, but it is important to state that the planner was not improved by any means for the competition and was exactly in the same state as for the original publications [Štolba and Komenda, 2014, 2017]. The PSM planner performs also reasonably well and underperformed significantly only on two domains due to a bug which was fixed only after the summer run.

Moreover, the Table A.2 shows the results for quality of the plans. The best performing planner is GPPP but especially PSM outperforms it on a number of domains. Finally, MAPlan dominates the comparison of speeds of the planners. Except for the elevators domain where it is significantly outperformed by the GPPP planner, MAPlan outperforms or matches all other planners on all other domains.

The complete results of the distributed track are shown in Table A.3, including all configurations and the two optimal MAPlan configurations. Unlike the centralized track, the best performing planner in the distributed track in terms of both coverage and quality is the PSM planner. The best performing planner in terms of time is MAPlan. In the distributed track, the extensive communication of the MAD-A* style search and distributed heuristics caused MAPlan to perform slightly worse in terms of coverage. The quality is most probably due to the use of LAMA [Richter and Westphal, 2010] as a local planner in PSM which optimizes for quality, whereas the search in MAPlan is purely greedy best-first search with

an inadmissible heuristic.

An interesting observation is that the MAPlan optimal variant using projected LM-Cut outperforms significantly the distributed variant (MA-LM-Cut) on all metrics. This is due to the intensive communication of the distributed LM-Cut variant and this result also corresponds to the results showing superior performance of the distributed potential heuristics (see Section 6.6.2) for details.

A.6 Summary

The first international Competition of Distributed and Multi-Agent Planners has attracted a surprisingly large number of participants, thus successfully becoming a thorough and nearly complete comparison of existing multi-agent planning systems compatible with the MA-STRIPS model. Although the more populated centralized track was rather permitting in its assumptions, the key motivations of the competition were satisfied. MA-PDDL acted as the base language of the competition, allowing comparisons of various multi-agent planners. The used benchmarks (especially the new wireless domain) were challenging even for centralized state-of-the-art multi-agent planners. Based on positive feedback from the planning community, we conclude our effort to promote multi-agent planning in the wider research community was successful as well.

Future directions for the competition can take advantage of the extensibility of the MA-PDDL language. An obvious direction is to use the looser privacy definition allowed by MA-PDDL and MCG and propose planning problems with complex privacy requirements, which would not be able to formalize using the MA-STRIPS privacy definition. A significant part of this extension would also allow private goals. A partitioning related extension is to allow joint actions (actions which have to be performed by two or more agents at the same time). It is an interesting research question whether transformation or inherent tackling of both private goals and joint-actions is practically better. Probably more distant extensions could follow directions stemming from classical IPC tracks, be it temporal planning, or research in decision making in a competitive or adversarial multi-agent environment. Following multi-agent competitions could shed light on these questions and research topics.

The privacy itself is a challenging task for the future IPC track organization. If the organizers are to enforce some degree of privacy preservation, they could either trust the competitors as we did, or they would need to use some methods to measure privacy leakage (e.g., as described in Chapter 7). Measuring privacy leakage in a competition would be a major endeavor as the techniques presented in this thesis assumed knowledge of the algorithm, which cannot be expected in the competition. Moreover, the planners would probably have to be forced to use a given communication protocol, which could then be used for the analysis.

CoDMAP aimed to serve as a proof-of-concept prototype of a multi-agent competition showing good direction and viability similarly to IPC seventeen years ago. We are highly confident that the way is now clear to make CoDMAP a fully-fledged multi-agent track of the next official International Planning Competition (IPC).

Appendix B

Publications

Related Publications

This section lists the author's publications related to the topic of this thesis. The number in parentheses shows the contribution of the author of the thesis.

Articles in journals, with IF:

1. **M. Štolba,** A. Komenda, "The MADLA Planner: Multi-Agent Planning by Combination of Distributed and Local Heuristic Search", *Artificial Intelligence Journal*, In Press[1], 2017. **IF: 4.80 (70 %)**

2. **M. Štolba**, J. Tožička, and A. Komenda, "Quantifying Privacy Leakage in Multi-Agent Planning", *ACM Transactions on Internet Technology*, vol. In Press, 2017. **IF: 1.49 (70 %)**

3. A. Komenda, **M. Štolba**, D. L. Kovacs, "The International Competition of Distributed and Multi-agent Planners (CoDMAP)", *AI Magazine*, vol. 37, iss. 3, pp. 109-115, 2016. **IF: 0.81 (33 %)**

4. A. Torreño, E. Onaindia, A. Komenda, **M. Štolba**,"Cooperative Multi-Agent Planning: A Survey", *ACM Computing Surveys*, In Press, 2017. **IF: 6.75 (25 %)**

In proceedings, indexed by WoS:

1. **M. Štolba**, J. Tožička, and A. Komenda. "Secure Multi-Agent Planning Algorithms." In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, pp. 1714–1715, 2016. **(40 %)**

2. J. Tožička, A. Komenda, and **M. Štolba**. "ϵ-Strong Privacy Preserving Multiagent Planner by Computational Tractability." in *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART*), pp. 51-57, 2017. **(10 %)**

In proceedings, indexed by Scopus:

1. J. Tožička, **M. Štolba**, and A. Komenda. "The Limits of Strong Privacy Preserving Multi-Agent Planning." in *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS)*, 2017. **(40 %)**

[1] DOI: https://doi.org/10.1016/j.artint.2017.08.007

2. **M. Štolba**, D. Fišer, and A. Komenda. "Potential Heuristics for Multi-Agent Planning." in *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 308-316, 2016. **(45 %)**

3. **M. Štolba**, A. Komenda, and D. L. Kovacs. "Competition of Distributed and Multiagent Planners (CoDMAP)." In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (What's Hot Track)*, 2016. **(33 %)**

4. **M. Štolba**, D. Fišer, and A. Komenda. "Admissible Landmark Heuristic for Multi-Agent Planning." In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 211– 219, 2015. **(50 %)**

5. **M. Štolba** and A. Komenda. "Relaxation Heuristics for Multiagent Planning." In *24th International Conference on Automated Planning and Scheduling (ICAPS)*, 2014. **(65 %)**

In proceedings:

1. **M. Štolba**, J. Tožička, and A. Komenda. "Secure Multi-Agent Planning." In *Proceedings of the 1st International Workshop on AI for Privacy and Security (ACM)*, pp. 11, 2016. **(40 %)**

2. **M. Štolba** and A. Komenda. "Computing Multi-Agent Heuristics Additively." In *Proceedings of the 4th Workshop on Distributed and Multi-Agent Planning (DMAP)*, 2016. **(80 %)**

3. D. Fišer, **M. Štolba**, and A. Komenda. "MAPlan." In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP)*, pages 8–10, 2015. **(5 %)**

4. **M. Štolba** and A. Komenda. "MADLA: Planning with Distributed and Local Search." In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP)*, pages 21–24, 2015.**(80 %)**

5. **M. Štolba**, D. Fišer, and A. Komenda. "Comparison of RPG-based FF and DTG-based FF Disrtibuted Heuristics." In *Proceedings of the 3rd Workshop on Distributed and Multi-Agent Planning (DMAP)*, 2015. **(45 %)**

6. **M. Štolba**, A. Komenda, and D. L. Kovacs. "Competition of Distributed and Multiagent Planners (CoDMAP)." In *Proceedings of the Workshop on International Planning Competition (WIPC), pp. 24,* 2015. **(41 %)**

7. **M. Štolba** and A. Komenda. "Fast-Forward Heuristic for Multiagent Planning." In *Proceedings of the 1st Workshop on Distributed and Multi-Agent Planning (DMAP)*, pp. 75–83, 2013. **(65 %)**

Unrelated Publications

This section lists the author's publications unrelated to the topic of this thesis.

In proceedings, indexed by Scopus:

1. M. Selecký, **M. Štolba**, T. Meiser, M. Čáp, A. Komenda, M. Rollo, J. Vokřínek, and M. Pěchouček, "Deployment of multi-agent algorithms for tactical operations on uav hardware (demonstration)," in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, 2013. **(12.5 %)**

Citations

Below we list all publications that received at least two citation (*excluding auto-citations*). The citation count (also excluding auto-citations) for each publication was obtained from the Google Scholar database on 12.9.2017.

1. **M. Štolba**, A. Komenda, and D. L. Kovacs. "Competition of Distributed and Multiagent Planners (CoDMAP)." In *Proceedings of the Workshop on International Planning Competition (WIPC), pp. 24,* 2015. **(13 citations)**

2. **M. Štolba** and A. Komenda. "Relaxation Heuristics for Multiagent Planning." In *Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*, 2014. **(9 citations)**

3. **M. Štolba**, D. Fišer, and A. Komenda. "Admissible Landmark Heuristic for Multi-Agent Planning." In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 211– 219, 2015. **(5 citations)**

4. **M. Štolba** and A. Komenda. "Fast-Forward Heuristic for Multiagent Planning." In *Proceedings of the 1st Workshop on Distributed and Multi-Agent Planning (DMAP)*, pp. 75–83, 2013. **(5 citations)**

5. **M. Štolba**, D. Fišer, and A. Komenda. "Comparison of RPG-based FF and DTG-based FF Disrtibuted Heuristics." In *Proceedings of the 3rd Workshop on Distributed and Multi-Agent Planning (DMAP)*, 2015. **(2 citations)**

6. **M. Štolba** and A. Komenda. "MADLA: Planning with Distributed and Local Search." In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP)*, pages 21–24, 2015. **(2 citations)**

Bibliography

Abdelrahaman Aly, Edouard Cuvelier, Sophie Mawet, Olivier Pereira, and Mathieu Van Vyve. Securely solving simple combinatorial graph problems. In *Proceedings of the International Conference on Financial Cryptography and Data Security*, pages 239–257. Springer, 2013.

Eyal Amir and Barbara Engelhardt. Factored planning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 929–935, 2003.

Christer Bäckström. Equivalence and tractability results for SAS+ planning. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25-29, 1992.*, pages 126–137, 1992.

Alice Bednarz, Nigel Bean, and Matthew Roughan. Hiccups on the road to privacy-preserving linear programming. In *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, pages 117–120. ACM, 2009.

Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

Marina Blanton, Aaron Steele, and Mehrdad Alisagari. Data-oblivious graph algorithms for secure computation and outsourcing. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 207–218, New York, NY, USA, 2013. ACM.

Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1):281–300, 1997.

Dan Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, 2013.

Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Proceedings of the European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *Proceedings of the European Conference on Planning*, pages 360–372, 1999.

Daniel Borrajo. Plan sharing for multi-agent planning. In *Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13)*, pages 57–65, 2013.

Daniel Borrajo and Susana Fernández. MAPR and CMAP. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, pages 1–3, 2015.

Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'99)*, volume 99, pages 478–485, 1999.

Ronen I. Brafman. A privacy preserving algorithm for multi-agent planning and search. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1530–1536, 2015.

Ronen I. Brafman and Carmel Domshlak. Factored planning: How, when, and when not. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, volume 6, pages 809–814, 2006.

Ronen I. Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 28–35, 2008.

Christelle Braun, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Quantitative notions of leakage for one-try attacks. *Electronic Notes in Theoretical Computer Science*, 249:75–91, 2009.

Michael Brenner. A multiagent planning language. *Workshop on PDDL, ICAPS'03, Trento, Italy, 2003.*, page 33, 2003.

Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pages 236–252. Springer, 2005.

Thomas Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

Mani K. Chandy and Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 3(1):63–75, 1985.

Daniel D. Corkill. Hierarchical planning in a distributed environment. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence, IJCAI 79, Tokyo, Japan, August 20-23, 1979, 2 Volumes*, pages 168–175, 1979.

Matthew Crosby. ADP an Agent Decomposition Planner CoDMAP 2015. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, pages 4–7, 2015.

Matthew Crosby, Michael Rovatsos, and Ronald Petrick. Automated agent decomposition for classical planning. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, pages 46–54, 2013.

Joseph Culberson. Sokoban is PSPACE-complete. 1997.

Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Proceedings of the Annual International Cryptology Conference*, pages 247–264. Springer, 2003.

Mathijs de Weerdt and Brad Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5(4):345–355, 2009.

Mathijs De Weerdt, Adriaan Ter Mors, and Cees Witteveen. Multi-agent planning: An introduction to planning and coordination. In *Handouts of the European Agent Summer*, 2005.

Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003. ISBN 978-1-55860-890-0.

Keith Decker. Distributed problem-solving techniques: A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(5):729–740, 1987.

Keith Decker and Victor Lesser. Generalizing the Partial Global Planning Algorithm. *International Journal on Intelligent Cooperative Information Systems*, 1(2):319–346, June 1992.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1): 269–271, 1959.

Yannis Dimopoulos, Muhammad A. Hashmi, and Pavlos Moraitis. μ-satplan: Multi-agent planning as satisfiability. *Knowledge-Based Systems*, 29:54–62, 2012.

Patrick Doherty and Jonas Kvarnström. TALplanner: A temporal logic-based planner. *AI Magazine*, 22 (3):95–102, 2001.

Jannik Dreier and Florian Kerschbaum. Practical privacy-preserving multiparty linear programming based on problem transformation. In *Proceedings of IEEE 3rd International Conference on Privacy, Security, Risk and Trust (PASSAT) and IEEE 3rd Third Inernational Conference on Social Computing (SocialCom)*, pages 916–924, 2011.

Edmund H. Durfee. Distributed problem solving and planning. In Gerhard Weiß, editor, *A Modern Approach to Distributed Artificial Intelligence*, chapter 3. The MIT Press, San Francisco, CA, 1999.

Edmund H. Durfee and Victor Lesser. Partial Global Planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Sensor Networks*, 21(5):1167–1183, 1991.

Karel Durkota and Antonín Komenda. Deterministic multiagent planning techniques: Experimental comparison (short paper). In *Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13)*, pages 43–47, 2013.

Eric Fabre, Loïg Jezequel, Patrik Haslum, and Sylvie Thiébaux. Cost-optimal factored planning: Promises and pitfalls. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 65–72, 2010.

Boi Faltings, Thomas Léauté, and Adrian Petcu. Privacy guarantees through distributed constraint satisfaction. In *Proceedings of the International Conference on Intelligent Agent Technology*, pages 350–358, 2008.

Richard Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI'71)*, pages 608–620, 1971.

Daniel Fišer, Michal Štolba, and Antonín Komenda. MAPlan. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, pages 8–10, 2015.

Praveen Gauravaram. Security analysis of salt || password hashes. In *Proceedings of the International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pages 25–30. IEEE, 2012.

Daniel Gnad and Jörg Hoffmann. Beating lm-cut with hmax (sometimes): Fork-decoupled state space search. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, pages 88–96, 2015.

Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 182–194, New York, NY, USA, 1987. ACM.

Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

Rachel Greenstadt, Jonathan P. Pearce, and Milind Tambe. Analysis of privacy loss in distributed constraint optimization. In *Proceedings of The Twenty-First National Conference on Artificial Intelligence (AAAI)*, pages 647–653, 2006.

Roberto Guanciale, Dilian Gurov, and Peeter Laud. Private intersection of regular languages. In *Proceedings of the Twelfth Annual International Conference on Privacy, Security and Trust (PST)*, pages 112–120. IEEE, 2014.

Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *Proceedings of Advances in neural information processing systems*, pages 1523–1530, 2002.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Malte Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262, 2003.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006.

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 162–169, 2009.

Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 176–183, 2007.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. doi: 10.1613/jair.855.

Johan Holmgren, Jan A. Persson, and Paul Davidsson. Agent-based Dantzig-Wolfe decomposition. In *Agent and Multi-Agent Systems: Technologies and Applications*, Lecture Notes in Computer Science, pages 754–763. Springer Berlin Heidelberg, 2009.

Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In *Proceedings of the 7th International Conference Security and Cryptography for Networks (SCN)*, pages 418–435, 2010.

Loïg Jezequel and Eric Fabre. A#: A distributed version of A* for factored planning. In *Proceedings of the 51th IEEE Conference on Decision and Control, (CDC'12)*, pages 7377–7382, 2012.

Anders Jonsson and Michael Rovatsos. Scaling up multiagent planning: A best-response approach. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, pages 114–121, 2011.

Michael Katz and Carmel Domshlak. Implicit abstraction heuristics. *Journal of Artificial Intelligence Research*, pages 51–126, 2010.

Henry A. Kautz. Deconstructing planning as satisfiability. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, pages 1524–1526, 2006.

Marcel Keller and Peter Scholl. Efficient, oblivious data structures for mpc. *IACR Cryptology ePrint Archive*, 2014:137, 2014.

Emil Keyder and Héctor Geffner. Heuristics for planning with action costs revisited. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pages 588–592, 2008.

Antonin Komenda and Peter Novak. Multi-agent plan repairing. In *Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities, Proceedings of IJCAI 2011 Workshop*, pages 1–6, 2011.

Antonín Komenda, Michal Stolba, and Daniel L. Kovacs. The international competition of distributed and multiagent planners (CoDMAP). *AI Magazine*, 37(3):109–115, 2016.

Kurt Konolige and Nils J Nilsson. Multiple-agent planning systems. In *Proceedings of the First National Conference on Artificial Intelligence (AAAI)*, volume 80, pages 138–142, 1980.

Daniel L. Kovacs. A multi-agent extension of PDDL3.1. In *Proceedings of the 3rd Workshop on the International Planning Competition (IPC)*, pages 19–27, 2012.

Jonas Kvarnström. Planning for loosely coupled agents using partial order forward-chaining. In *Proceedings of the 21th International Conference on Automated Planning and Scheduling (ICAPS'11)*, pages 138–145, 2011.

Ronghua Li and Chuankun Wu. An unconditionally secure protocol for multi-party set intersection. In *Proceedings of Applied Cryptography and Network Security*, pages 226–236. Springer, 2007.

Nerea Luis and Daniel Borrajo. Plan merging by reuse for multi-agent planning. In *Proceedings of the 2nd ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'14)*, pages 38–44, 2014.

Nerea Luis and Daniel Borrajo. PMR: Plan Merging by Reuse. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, pages 11–13, 2015.

Shlomi Maliah, Guy Shani, and Roni Stern. Privacy preserving landmark detection. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, pages 597–602, 2014.

Shlomi Maliah, Guy Shani, and Roni Stern. Privacy preserving pattern databases. In *Proceedings of the 3rd Distributed and Multiagent Planning (DMAP) Workshop of ICAPS'15*, pages 9–17, 2015.

Shlomi Maliah, Ronen I. Brafman, and Guy Shani. Privacy preserving LAMA. In *Proceedings of the 4th Workshop on Distributed and Multi-Agent Planning, DMAP–ICAPS'16*, pages 1–5, 2016a.

Shlomi Maliah, Ronen I. Brafman, and Guy Shani. Increased privacy with reduced communication and computation in multi-agent planning. In *Proceedings of the 4th Workshop on Distributed and Multi-Agent Planning, DMAP–ICAPS'16*, pages 106–112, 2016b.

Shlomi Maliah, Guy Shani, and Ronen I. Brafman. Online macro generation for privacy preserving planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*, 2016c.

Shlomi Maliah, Guy Shani, and Roni Stern. Stronger privacy preserving projections for multi-agent planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*, pages 216–220, 2016d.

Olvi L Mangasarian. Privacy-preserving linear programming. *Optimization Letters*, 5(1):165–172, 2011.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control,, 1998.

Christian Muise, Nir Lipovetzky, and Miquel Ramirez. MAP-LAPKT: Omnipotent Multi-Agent Planning via Compilation to Classical Planning. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, pages 14–16, 2015.

Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558608567.

Raz Nissim and Ronen I. Brafman. Multi-agent A* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*, pages 1265–1266, 2012.

Raz Nissim and Ronen I. Brafman. Cost-optimal planning by self-interested agents. In *Proceedings of the 28th National Conference on Artificial Intelligence (AAAI)*, 2013.

Raz Nissim and Ronen I. Brafman. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research*, 51:293–332, 2014.

Raz Nissim, Ronen I. Brafman, and Carmel Domshlak. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, pages 1323–1330, 2010.

Frans A Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016. ISBN 978-3-319-28927-4.

Frans A. Oliehoek, Matthijs T. J. Spaan, Philipp Robbel, and João V. Messias. The MADP toolbox: An open-source library for planning and learning in (multi-)agent systems. In *Sequential Decision Making for Intelligent Agents—Papers from the AAAI 2015 Fall Symposium*, pages 59–62, November 2015.

Damien Pellier. Distributed planning through graph merging. In *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence (ICAART 2010)*, pages 128–134, 2010. doi: 10.5220/0002702601280134.

Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*, pages 515–530, Washington, D.C., 2015. USENIX Association.

Florian Pommerening and Malte Helmert. A normal form for classical planning tasks. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 188–192, 2015.

Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning: Proof details. *Technical Report CS-2014-005, University of Basel, Department of Mathematics and Computer Science*, 2014a.

Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. LP-Based heuristics for cost-optimal planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 226–234, 2014b.

Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 3335–3341, 2015.

Miquel Ramirez, Nir Lipovetzky, and Christian Muise. Lightweight Automated Planning ToolKiT. URL `http://lapkt.org/`.

Stefan Rass, Peter Schartner, and Monika Brodbeck. Private function evaluation by local two-party computation. *EURASIP J. Information Security*, 2015:7, 2015.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2010.

Gabriele Röger and Malte Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 246–249, 2010.

Jendrik Seipp, Florian Pommerening, and Malte Helmert. New optimization functions for potential heuristics. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 193–201, 2015.

Rashid Sheikh, Beerendra Kumar, and Durgesh Kumar Mishra. A distributed k-secure sum protocol for secure multi-party computations. *arXiv preprint arXiv:1003.4071*, 2010.

Geoffrey Smith. On the foundations of quantitative information flow. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures*, pages 288–302, 2009.

Sarath Sreedharan, Yu Zhang, and Subbarao Kambhampati. A First Multi-agent Planner for Required Cooperation (MARC). In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, pages 17–20, 2015.

Michal Štolba and Antonín Komenda. Fast-forward heuristic for multiagent planning. In *Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13)*, pages 75–83, 2013.

Michal Štolba and Antonín Komenda. Relaxation heuristics for multiagent planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, pages 298–306, 2014.

Michal Štolba and Antonín Komenda. MADLA: Planning with distributed and local search. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, pages 21–24, 2015.

Michal Štolba and Antonín Komenda. Computing multi-agent heuristics additively. In *Proceedings of the 4th Workshop on Distributed and Multi-Agent Planning, DMAP–ICAPS'16*, 2016.

Michal Štolba and Antonín Komenda. The MADLA Planner: Multi-agent planning by combination of distributed and local heuristic search. *Artificial Intelligence*, 2017.

Michal Štolba, Daniel Fišer, and Antonín Komenda. Admissible landmark heuristic for multi-agent planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 211–219, 2015a.

Michal Štolba, Antonín Komenda, and Daniel L. Kovacs. Competition of distributed and multiagent planners (CoDMAP). *Proceedings of the Workshop on the International Planning Competition (WIPC-15)*, 24, 2015b.

Michal Štolba, Daniel Fišer, and Antonín Komenda. Potential heuristics for multi-agent planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling, ICAPS'16*, pages 308–316, 2016a.

Michal Štolba, Antonín Komenda, and Daniel Kovacs. Competition of distributed and multiagent planners (CoDMAP). In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4343–4345, 2016b.

Michal Štolba, Jan Tožička, and Antonín Komenda. Secure multi-agent planning. In *Proceedings of the 1st International Workshop on AI for Privacy and Security*, pages 11:1–11:8. ACM, 2016c.

Michal Štolba, Jan Tožička, and Antonín Komenda. Secure multi-agent planning algorithms. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI'16)*, pages 1714–1715, 2016d.

Michal Štolba, Jan Tožička, and Antonın Komenda. Quantifying privacy leakage in multi-agent planning. *Transactions on Internet Technology (TOIT)*, 2017.

Tomas Toft. Primitives and applications for multi-party computation. *Doctoral dissertation, University of Aarhus, Denmark*, 2007.

Tomas Toft. Solving linear programs using multiparty computation. *Financial Cryptography and Data Security*, pages 90–107, 2009.

Tomas Toft. Secure data structures based on multi-party computation. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '11, pages 291–292. ACM, 2011.

Alejandro Torreño, Eva Onaindia, and Óscar Sapena. FMAP: Distributed cooperative multi-agent planning. *Applied Intelligence*, 41(2):606–626, September 2014.

Alejandro Torreño, Óscar Sapena, and Eva Onaindia. MH-FMAP: Alternating global heuristics in multi-agent planning. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, pages 25–28, 2015.

Alejandro Torreño, Eva Onaindia, and Oscar Sapena. An approach to multi-agent planning with incomplete information. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, pages 762–767, 2012. doi: 10.3233/978-1-61499-098-7-762.

Alejandro Torreño, Eva Onaindia, and Oscar Sapena. FMAP: A heuristic approach to cooperative multi-agent planning. In *Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13)*, pages 84–92, 2013.

Alejandro Torreno, Oscar Sapena, and Eva Onaindia. Global heuristics for distributed cooperative multi-agent planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 225–233, 2015.

Jan Tožička, Jan Jakubuv, and Antonín Komenda. Generating multi-agent plans by distributed intersection of Finite State Machines. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, pages 1111–1112, 2014.

Jan Tožička, Jan Jakubuv, and Antonín Komenda. PSM-based planners description for CoDMAP 2015 competition. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, pages 29–32, 2015.

Jan Tožička, Jan Jakubuv, Antonín Komenda, and Michal Pechouček. Privacy-concerned multiagent planning. *Knowledge Information Systems*, 48(3):581–618, 2016.

Jan Tožička, Antonín Komenda, and Michal Štolba. ϵ-strong privacy preserving multiagent planner by computational tractability. In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence, ICAART*, pages 51–57, 2017a.

Jan Tožička, Michal Štolba, and Antonín Komenda. The limits of strong privacy preserving multi-agent planning. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling, ICAPS'17*, 2017b.

Menkes Van Den Briel, J Benton, Subbarao Kambhampati, and Thomas Vossen. An LP-based heuristic for optimal planning. In *Proceedings of the Principles and Practice of Constraint Programming–CP 2007*, pages 651–665. Springer Berlin Heidelberg, 2007.

Roman Van Der Krogt. Quantifying privacy in multiagent planning. *Multiagent and Grid Systems*, 5 (4):451–469, 2009.

Andrew Yao. How to generate and exchange secrets. In *Proceedings of the Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, SFCS*, pages 160–164, 1982.

Makoto Yokoo, Edmund H Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on knowledge and data engineering*, 10(5):673–685, 1998.