Journal of Artificial Intelligence Research 51 (2014) 293-332 Submitted 01/14; published 10/14

Distributed Heuristic Forward Search for Multi-agent Planning

Raz Nissim RAZNIS@CS.BGU.AC.IL
Ronen Brafman BRAFMAN@CS.BGU.AC.IL
Ben-Gurion University of the Negev,
Be'er Sheva, Israel

Abstract

This paper deals with the problem of classical planning for multiple cooperative agents who have private information about their local state and capabilities they do not want to reveal. Two main approaches have recently been proposed to solve this type of problem – one is based on reduction to distributed constraint satisfaction, and the other on partial-order planning techniques. In classical *single-agent* planning, constraint-based and partial-order planning techniques are currently dominated by heuristic forward search. The question arises whether it is possible to formulate a distributed heuristic forward search algorithm for privacy-preserving classical *multi-agent* planning. Our work provides a positive answer to this question in the form of a general approach to distributed state-space search in which each agent performs only the part of the state expansion relevant to it. The resulting algorithms are simple and efficient – outperforming previous algorithms by orders of magnitude – while offering similar flexibility to that of forward-search algorithms for single-agent planning. Furthermore, one particular variant of our general approach yields a distributed version of the A* algorithm that is the first *cost-optimal* distributed algorithm for privacy-preserving planning.

1. Introduction

Interest in multi-agent systems is constantly rising, and examples of virtual and real systems abound, with virtual social communities providing many such instances. The ability to plan for such systems and the ability of such systems to autonomously plan for themselves is an important challenge for artificial intelligence, especially as the size of these systems can be quite large. In this context, a fundamental question is how to perform planning for a distributed multi-agent system efficiently, and in many cases, how to do it while preserving privacy.

Distributed planning is interesting for a number of reasons. Scientifically and intellectually, it is interesting to seek distributed algorithms for fundamental computational tasks, such as classical planning. Similarly, it is interesting (and likely very useful in the long term) to seek distributed versions of fundamental tools in computer science, and *search* is definitely such a tool. Moreover, there are pragmatic reasons for seeking distributed algorithms. As an example, imagine a setting in which different manufacturers or service providers can publish their capabilities and then collaborate with each other to provide new products or services that none of them can provide alone. Such providers will certainly need to reveal some sort of public interface, describing what they can contribute, as well as what they require from others. But most likely, they will not want to describe their inner workings: their internal state and how they can manipulate it (e.g., their current stock levels, machinery, logistics capabilities, personnel, other commitments, etc.). This is usually confidential proprietary information that an agent would not want to reveal, although clearly one must reason about it during the planing process.

In principle, the above problem can be addressed using a central trusted party running a suitable planning algorithm. However, such a trusted party may not exist in all settings. Moreover, centralized planning puts the entire computational burden on a single agent, rather than distributing it across the system. Thus, centralized algorithms are less robust, they are prone to agent failures, and are sometimes less efficient. For these reasons, distributed algorithms are often sought, and in our case in particular, distributed, privacy-preserving algorithms. Indeed, this is the main motivation for the field of distributed algorithms, and in particular, the work on distributed constraint satisfaction problems (CSP) (Conry, Kuwabara, Lesser, & Meyer, 1991; Yokoo, Durfee, Ishida, & Kuwabara, 1998; Meisels, 2007).

Moreover, it is often the case that good distributed algorithms formulated for a cooperative team provide the foundation for algorithms and mechanisms for solving similar problems for teams of self-interested agents. For example, the work on planning games (Brafman, Domshlak, Engel, & Tennenholtz, 2009, 2010) suggests modified versions of an earlier algorithm for cooperative multi-agent systems (Brafman & Domshlak, 2008), and work on mechanism design for solving distributed CSPs by self-interested agents (Petcu, Faltings, & Parkes, 2008) is based on earlier work in distributed CSPs for cooperative teams (Petcu & Faltings, 2005). In fact, the work presented in this paper forms the basis of work on mechanism design for privacy-preserving MA planning, where the agents are self-interested (Nissim & Brafman, 2013).

Yet another motivation for developing distributed algorithms is provided by planning domains in which search operators that correspond to actions are implemented using complex simulation software that is accessible only to the relevant agent, because that agent is not interested in sharing it (due to privacy concerns or commercial interests) or because it is not realistic to transfer, integrate, and appropriately execute such software as part of a planning algorithm. As an example, consider planning by a group of robotic agents, each with different capabilities. Each agent has a simulator that can compute the effect of its actions, which the agents do not want to share with each other. Thus, the application of each agent's actions during search can only be done by the agents themselves.

There is a long tradition of work on multi-agent planning for cooperative and non-cooperative agent teams involving centralized and distributed algorithms, often using involved models that model uncertainty, resources, and more (Conry et al., 1991; Ephrati & Rosenschein, 1997; Hansen & Zilberstein, 2001; Bernstein, Givan, Immerman, & Zilberstein, 2002; Szer, Charpillet, & Zilberstein, 2005; Witwicki, Oliehoek, & Kaelbling, 2012), and much work on how to coordinate the local plans of agents or to allow agents to plan locally under certain constraints (Cox & Durfee, 2005; Steenhuisen, Witteveen, ter Mors, & Valk, 2006; ter Mors, Valk, & Witteveen, 2004; ter Mors & Witteveen, 2005). Our work is influenced and motivated by some of the results in that area, but takes as its starting point a more basic model introduced by Brafman and Domshlak (BD) that offers what is possibly the *simplest* model of MA planning – MA-STRIPS (Brafman & Domshlak, 2008). MA-STRIPS minimally extends standard STRIPS (or PDDL) models by specifying a set of agent ids, and associating each action in the domain with one of the agents. Thus, essentially, it partitions the set of actions among the set of agents.

One motivation for exploring MA planning in this simple setting is the belief that simple models make it easier to study fundamental ideas, and that techniques developed can often be generalised to more complex settings. Another closely related reason for working on this model is the recent influential trend in the field of planning, of solving richer, more complex models by *reduction* to the simpler settings and, especially, by using classical planners. This is expected, given that classical planners – like SAT solvers, which are widely used as black boxes for solving various problems (Kautz & Selman, 1992; Clarke, Biere, Raimi, & Zhu, 2001) – have now reached a certain performance level, which makes them capable of (quickly) solving large problems. These planners incorporate a wealth of ideas and techniques which are usually difficult (and sometimes impossible) to export to non-classical models. The use of classical planners has been shown to be efficient in many different settings of planning under uncertainty – Stochastic Shortest Path (Yoon, Fern, & Givan, 2007), Conformant Planning (Palacios & Geffner, 2009), Conformant Probabilistic Planning (Albore, Palacios, & Geffner, 2010; Taig & Brafman, 2013) and Contingent Planning (Albore, Palacios, & Geffner, 2009) – as well as other planning models such as Net-Benefit Planning (Keyder & Geffner, 2009) and Generalized Planning (Srivastava, Immerman, Zilberstein, & Zhang, 2011).

Recently, a number of MA-STRIPS planning algorithms that respect agent privacy and utilize the inherent distributed structure of the system have emerged. The first of these is an approach based on distributed CSP techniques and was introduced in BD's original work.[1] BD formulate a CSP that is particularly suited for MA-STRIPS problems and whose solution is a plan. This algorithm can be transformed into a fully distributed algorithm simply by using a *distributed* CSP solver. BD's work establishes an upper bound on the complexity of solving an MA-STRIPS problem which depends exponentially only on two parameters quantifying the level of agents' coupling in the system. Unfortunately, distributed CSP solvers cannot handle even the smallest instances of MA planning problems. Consequently, a dedicated algorithm, based on the ideas of BD, was developed (Nissim, Brafman, & Domshlak, 2010). While performing well on some domains, this algorithm had trouble scaling up to problems in which each agent had to execute more than a small number of actions. (Indeed, BD's algorithm scales exponentially with the min-max number of actions per agent in a solution plan.) Recently, a new, improved algorithm, based on partial-order planning, MAP-POP, was developed (Torreño, Onaindia, & Sapena, 2012). Yet, this algorithm, too, leaves a serious gap between what we can solve using a distributed planner and what can be solved using a centralized planner. Moreover, neither algorithm attempts to generate a *cost-optimal* plan.

In classical single-agent planning, constraint-based and partial-order planning techniques are currently dominated by heuristic forward search techniques[2]. Thus, it is natural to ask whether it is possible to formulate a distributed heuristic forward search algorithm for privacy-preserving classical planning. This paper provides a positive answer to this question

1. The idea of using distributed CSP techniques for MA planning was first introduced by Conry et al. (1991).
2. Winners of the sequential satisficing (non-optimal) tracks of the last three International Planning Competitions were heuristic forward search planners. Such a planner was also the winner of the *optimal* track in the previous IPC (2011).

in the form of a general approach to distributed state-space search in which each agent performs only the part of the state expansion relevant to it. The resulting algorithms are very simple and very efficient – outperforming previous algorithms by orders of magnitude – and offer similar flexibility to that of forward-search based algorithms for single-agent planning. They respect the natural distributed structure of the system, and thus allow us to formulate privacy-preserving versions.

One particular variant of our general approach yields a distributed version of the A* algorithm, called MAD-A*, which is the first distributed algorithm for privacy-preserving *cost-optimal* planning in MA-STRIPS. MAD-A* solves a more difficult problem than centralized search since in the privacy-preserving setting, each agent has an abstracted (partial) view of the problem. Yet, it is still able to solve problems efficiently, outperforming centralized A* in some cases. As we will show, the main reason for this is an interesting optimality preserving pruning technique that is naturally built into our search approach. This insight has led to a new pruning technique for centralized search that we shall describe later on.

The rest of this paper is organized as follows. We start by providing some background and then describe the model we use. This is followed by a discussion of related work. Section 4 describes our MA forward search algorithm, and Section 5 describes MAD-A*, a modified version which maintains cost-optimality. Then, we present the MA planning framework used, MA-FD, and empirical results evaluating the effectiveness of our algorithms. Section 8 discusses a pruning method "built-in" to our algorithms, and how it can be applied in centralized search. Section 9 provides a discussion of the privacy properties of our algorithms and some open research challenges, and is followed by a conclusion.

2. Background

We describe the MA-STRIPS model for multi-agent planning, and then proceed to define privacy-preserving planning in MA-STRIPS.

2.1 MA-STRIPS

A MA-STRIPS problem (Brafman & Domshlak, 2008) for a set of agents $\Phi = \{\varphi_i\}_{i=1}^k$, is given by a 4-tuple $\Pi = \langle P, \{A_i\}_{i=1}^k, I, G\rangle$, where P is a finite set of propositions, $I \subseteq P$ and $G \subseteq P$ encode the initial state and goal, respectively, and for $1 \leq i \leq k$, A_i is the set of actions agent φ_i is capable of performing. Each action $a = \langle \text{pre}(a), \text{eff}(a), \text{cost}(a)\rangle$ is given by its preconditions, effects and cost. A *plan* is a solution to Π iff it is a solution to the underlying STRIPS problem obtained by ignoring the identities of the agent associated with each action. Since each action is associated with an agent, a plan tells each agent what to do and when to do it. In different planning contexts, one might seek special types of solutions. For example, in the context of planning games (Brafman et al., 2009), *stable* solutions (equilibria) are sought. Our focus is on cooperative multi-agent systems, seeking either a standard solution or a cost-optimal one, which minimizes the sum of action costs of the plan.

The partitioning of the actions to agents yields a natural distinction between private and public propositions and actions. A *private* proposition of agent φ is required and affected only by the actions of φ. An action is *private* if all its preconditions and effects are private. All other actions are classified as *public*. That is, φ's private actions affect and are affected only by φ's actions, while its public actions may require or affect the actions of other agents. For ease of the presentation of our algorithms and their proofs, we assume that all actions that achieve a goal condition are considered *public*. Our methods are easily modified to remove this assumption.

To get a clearer picture of the MA-STRIPS model, consider the well known Logistics classical planning domain, in which packages should be moved from their initial to their target locations using a given fleet of vehicles such as trucks, airplanes, etc. The packages can be loaded onto and unloaded off the vehicles, and each vehicle can move between a certain subset of locations. Propositions are associated with each package location, on the map or in a vehicle, and with every vehicle's location on the map. Possible actions are *drive/fly*, *load*, and *unload*, each with its suitable parameters (e.g., *drive(truck, origin, destination)* and *load(package, truck, location)*). A natural partitioning of this problem associates each vehicle with an agent, assigning this agent all *drive, load* and *unload* actions in which it is involved. MA-STRIPS includes this action assignment as part of the problem description.

In the Logistics domain, since all vehicle locations are private propositions (affect and are affected only by their respective agent's *move* actions), all *move* actions are certainly private, depending on and affecting only private location propositions. Package location propositions can be either private or public, depending on whether they are required/affected by more than one agent. For example, the proposition *at(package,location)* is private if location is accessible only to one agent (e.g. inside the vehicle, or, can be reached only by a single vehicle), and public otherwise. Therefore, *load* (respectively *unload*) actions that require (respectively affect) public package location propositions are considered public.

We note that while the notion of private/public is natural to the MA-STRIPS encoding, it can easily be applied in models having multi-valued variables. For example, in SAS+, where each variable may have multiple values, the analogue of a (boolean) proposition in MA-STRIPS is a $\langle \text{variable}, \text{value} \rangle$ pair. Such a pair is considered private if it is required, achieved or destroyed only by the actions of a single agent. Consequently, actions which require, achieve or destroy only private $\langle \text{variable}, \text{value} \rangle$ pairs are considered private. For clarity and consistency with previous work we use MA-STRIPS notation when discussing the theoretical aspects of our work. However, the examples given, as well the practical framework we present for MA planning, use the more concise multi-valued variables SAS+ encoding.

2.2 Privacy Preserving Planning

Given the notion of private/public propositions and actions, we can now formalize the problem that is the focus of our work – *Privacy-preserving MA-STRIPS planning.* Privacy-preserving MA-STRIPS planning seeks to find a solution to a MA-STRIPS planning problem without exposing information that is private to an agent. Specifically: its set of private variables and their possible values, its set of private actions and their cost, and the private preconditions and effects of its public actions. Public actions and variables can be considered as the exposed "interfaces" of an agent, while private actions and variables correspond to the agent's local state and the actions that manipulate its local state.

More formally, in a privacy-preserving MA-STRIPS planning problem, each agent has access to a description of its own actions and to a public projection of the public actions of other agents. The public projection of an action consists of the lists of public preconditions and public effects. In a *weak privacy preserving* planning algorithm an agent need not communicate a description of any of its private variables, private actions and their cost, and the private preconditions and effects of public actions, to another agent. In a *strong privacy preserving algorithm*, no agent can deduce an isomorphic model (i.e., the same model, up to renaming) of a private variable, a private action and its cost, or the private preconditions and effects of a public action that cannot be deduced from the initial information available to that agent and the public parts of the plan. We will present a distributed search approach that lies somewhere in-between weak and strong privacy preserving. We will discuss its privacy properties in Section 9.1. At this stage, let us illustrate the meaning and need for such privacy properties.

Consider the classical Logistics domain. Agent φ knows about the existence and value of its (private) *location* propositions, the locations of public packages and of packages that are private to it (only it can load and unload them). Thus, the initial state known to φ will not contain the initial locations of other vehicles, nor package locations reachable only by a single agent that is different from φ. The agent has full knowledge of its actions, but has access to projections of other agents' public actions only. For example, another agent's action a =*unload(package, truck, location)* has preconditions *at(package, in-truck)* and *at(truck, location)* and effects ¬*at(package, in-truck)* and *at(package, location)*. However, since only *at(package, location)* is a public proposition (assuming that *location* is reachable by more than one agent), the projection of a known to φ contains no preconditions and only that single effect. The implication of preserving strong privacy in this example is that agents do not know of packages that are handled by a single agent only (private to it). They do not know the locations served by an agent, nor its actual location in different stages of the plan, or how it changes its location. In fact, they are unaware of the existence of a "location" of an agent, as this is a private variable.

In real life, some of this may seem absurd, as we all know that trucks and planes have a location and can move among locations. Privacy guarantees do not take into account general knowledge we have about how things work. However, even in this example, it is appealing that agents do not know about the set of locations an agent serves, how it moves between them, and at what cost. Nor do they know about packages that are handled only by this agent. We believe that the ability to provide privacy at this or a similar level is a clear advantage for a planning algorithm, facilitating the ability to construct ad-hoc teams of cooperating, but privacy seeking agents.

Many service-oriented domains have similar abstract structure, where one agent does some work towards a goal (getting the package to its destination) that requires the collaboration of multiple agents (e.g. truck in one country, plane, truck in another country). As an example, imagine multiple part-manufacturers, together with an aggregator, that can build a complex object together that none can build on their own, e.g., a laptop. This would require monitors, various cards (graphic card, mother board, wifi), casing, etc. Each of these components, in turn, requires diverse electronic parts, wiring, empty boards, etc. The mother board manufacturer's interface to the external world consists of the action of selling a mother board, and actions for purchasing relevant components – these are its public actions. The public action *sell-mother-board* may have preconditions such as *stock* > 0*, have-payment, have-address* and an effect such as *stock=stock-1, has-mother-*

board(customer), where stock variables are private and other variables are public. It would also have a private action such as *produce-mother-board* which will have various technical preconditions related to availability of parts, workers, machinery. Alternatively, perhaps producing the board requires multiple steps, possibly using different machines and different workers. These will be reflected by its set of private actions. Its public variables will refer to mother board orders, availability, and payments received. Its internal variable will refer to where it produces, who are its workers and what roles they can take, where it stores its inventory, and what are its stock levels at each facility. Its private actions will describe how shipping from warehouse to plant is done, the actions used to actually construct the board, who does what, and much more. A privacy preserving algorithm will neither expose, nor require the knowledge of these inner workings, which are typically considered proprietary information that manufacturers would not share, even with collaborators, unless required.

In Section 1 we discussed an example application of planning by a team of robotic agents, which have complex simulators that generate the outcome of their actions. Fitting this scenario into the MA-STRIPS model, the public variables of an agent could describe elements of the environment affected by an agent, and visible aspects of the robot (e.g., where it is, whether it is standing up or on the ground), whereas the local variables would refer to its local state: the actual state of various motors, its charge level, perhaps various local variables describing its current mode of operation. Public actions would be actions that influence the environment and the observable state of the robot: moving, lifting, climbing a ladder, collecting a rock. Local actions would involve local computations that manipulate its internal, private state, such as charging. Note that public actions will influence many private variables: moving an arm or climbing a ladder would change the internal state of many variables and internal mechanical parts. Privacy preservation here serves two purposes: not requiring one robot to understand and model the intricacies of another robot, and not requiring one robot to disclose this information to other robots, both simplifying the interaction between the robots, and maintaining confidentiality of, possibly proprietary information. Thus, one robot can have public (and thus) abstract description of a move action of anther robot, whose public effect is that the position of the robot changes. Yet, it need not model, nor know about, the inner working involved in this move (or lift, or carry, etc.) action. Most likely, the state of internal parts/motors/joints has to change in various ways to accomplish this move action. Indeed, if ad-hoc robot teams will be as ubiquitous as ad-hoc networks, this ability to work together without the burden and the exposure entailed by sharing full models could be crucial.

3. Related Work

Our work focuses on the classical (deterministic) MA planning setting, in which agents collaborate on a specific task, but prefer not to reveal private information about their local states, their private actions, and the cost of these private actions. This preservation of agent privacy is the main difference between our methods (and distributed planning in general) and other approaches such as factored planning and parallel planning. Below we discuss both related work on multi-agent systems and related work on privacy.

3.1 Multi-agent Planning

Factored planning methods (Amir & Engelhardt, 2003; Brafman & Domshlak, 2006; Fabre & Jezequel, 2009; Fabre, Jezequel, Haslum, & Thiébaux, 2010) seek to utilize the structure of the planning problem, making *centralized* search more efficient. These methods can exploit the MA structure of MA-STRIPS problems (as is done by Brafman and Domshlak, 2008), but are not applicable for *privacy-preserving* settings, as they do not respect the distributed form of the problem, giving a centralized entity access to the entire problem description.

Parallel planning methods (Vrakas, Refanidis, & Vlahavas, 2001; Kishimoto, Fukunaga, & Botea, 2009; Burns, Lemons, Ruml, & Zhou, 2010) aim to speed-up the solution of centralized planning problems given access to a distributed computing environment, such as a large cluster. Parallel planning can be performed in MA-STRIPS by applying existing approaches to the underlying STRIPS problem (i.e., ignoring agent identities). While these methods distribute the *computation* required for solving the planning problem, they ignore privacy concerns, giving all agents (essentially processors) access to the entire planning problem.

Given the privacy-preserving MA-STRIPS model, it is natural to ask how to search for a solution. A well-known example of privacy-preserving search is that of distributed CSPs (Conry et al., 1991; Yokoo et al., 1998), for which various search techniques and heuristics have been developed (Meisels, 2007). Distributed CSPs, where agents cooperate to find a feasible assignment to constrained variables while keeping private internal constraints, model a problem very similar to distributed MA-STRIPS. In fact, planning problems can be cast as CSP problems (given some bound on the number of actions), and the first attempt to solve privacy-preserving MA-STRIPS problems was based on a reduction to distributed CSPs. More specifically, Brafman and Domshlak introduced the *Planning as CSP+Planning* methodology for planning by a system of cooperative agents with private information. This approach separates the public aspect of the problem, which involves finding public action sequences that satisfy a certain distributed CSP, from the private aspect, which ensures that each agent can actually execute these public actions in a sequence. Solutions found are locally optimal, in the sense that they minimize δ, the maximal number of public actions performed by an agent. This methodology was later extended to the first fully distributed MA algorithm for MA-STRIPS planning, *Planning-First* (Nissim et al., 2010). *Planning First* was shown to be efficient in solving problems where the agents are very loosely coupled, and where δ is very low. However, it does not scale up as δ rises, mostly due to the large search space of the distributed CSP. Recently, a distributed planner based on partial order planning was introduced (Torreño et al., 2012), which outperforms Planning First, effectively solving more tightly coupled problems. Both methods are privacy preserving, but do not guarantee cost-optimal solutions.

Our work attempts to solve privacy-preserving MA-STRIPS using *distributed heuristic forward search* algorithms. Heuristic forward search methods are widely used, and have been applied in similar settings. Ephrati and Rosenschein (1994, 1997) showed that given an *a priori* breakdown of the global goal into agent assigned subgoals, agents can plan towards their subgoals, and then merge their possibly conflicting plans. This approach essentially searches in *plan-space*, using heuristic estimates of *states* (which are the results of merged feasible subplans) in order to guide search. This approach does not guarantee optimal solutions, relies on solving an exponential number of plan merging problems and does not scale even in loosely-coupled problems (Cox & Durfee, 2009). As will be made clear in the next section, our approach is a simple, general framework for forward search in *state-space*, which easily specializes to a distributed version of the cost-optimal search algorithm A*, and does not depend on complex plan merging algorithms.

In the setting of planning under uncertainty for distributed agents, forward search has been shown to be effective. Szer, Charpillet and Zilberstein (2005) used a heuristic best-first approach in order to optimally solve Decentralized POMDPs. This approach performs *centralized* forward search in the space of agents' policy vectors. In order to compute heuristic estimate for a single node, the solution of the underlying *centralized MDP* must be computed for all possible states of the system. Both the search algorithm itself (which is centralized) and the heuristic computation require full knowledge of the system and therefore are not applicable in the privacy-preserving setting.

Another notable strategy for solving decentralized POMDP's was proposed by Nair et al. (2003). JESP (Joint Equilibrium Based Search for Policies) computes a locally optimal solution by iteratively modifying the policy of each agent to improve the joint (global) policy. This algorithm is proved to converge to a Nash equilibrium. Subsequent work (Ranjit, Varakantham, Tambe, & Yokoo, 2005) aims to exploit local interactions of the agents, by applying distributed CSP techniques. Locality is exploited by computing the agent's best response only with respect to its neighbors, or the agents which are affected by it, which improves JESP's efficiency. Similarly to Planning as CSP+Planning, these methods generate locally optimal solutions and do not guarantee globally optimal solutions. Additionally, the CSP-based methods suffer from exactly the same scalability drawback as *Planning-First*, and will not be efficient as agent interaction rises.

Recently, in parallel with our own work, heuristic forward search for MA planning under uncertainty was combined with influence-based abstraction (Witwicki & Durfee, 2010; Oliehoek, Witwicki, & Kaelbling, 2012), producing a forward search algorithm for searching the influence-space (Witwicki et al., 2012). Influence-based abstraction reformulates the joint-policy search space into the space of influences, which represent the effect that agent policies have on one another. The notion of influences relates closely to that of private/public actions – the influences can be equated to public actions, and the intricate policies on which the influences depend can be equated to private actions. The aforementioned work shows that heuristic search in influence space can lead to significant improvement in performance, which is mostly due to the pruning of the search space (we arrive at a similar conclusion with respect to our work – this is discussed in Section 8). While the work on heuristic search in influence space is similar to our work, there are two main differences which sets them apart. First, the methods described apply only to *transition-decoupled POMDP* models, which restrict the number of agents affecting a variable to at most one, whereas MA-STRIPS has no limit on the number of agents affecting each variable. Second, the methods are not aimed at preserving privacy or at distributing search, but only at making centralized search more efficient.

Overall, the motivation of most existing work in MA planning under uncertainty resembles that of Factored Planning – utilizing the (MA) structure in order to *centrally* solve the problem more efficiently. Most of the research tackles problems which stem from the models' characteristics (e.g. MDP's stochastic actions, POMDP's partial observability etc.). Therefore, the product of this body of research is usually specialized methods which do not provide a general solution schema applicable in other domains, despite solving more general (and complex) models. While these are important models representing problems which are harder to solve than classical planning – DEC-POMDPs have been shown to be NEXP-complete in the general case (Bernstein et al., 2002), we believe the field of MA planning will benefit from general solutions for its simplest fundamental model of "classical" planning, and this is what is presented in this work.

3.2 Privacy

Privacy is a wide topic of research with many technological, social, and legal aspects. Our interest lies in the much more well defined area of *secure multi-party computation* (Yao, 1982, 1986), a subfield of Cryptography. The goal of methods developed for secure multi-party computation is to enable multiple agents to compute a function over their inputs, while keeping these inputs private. More specifically, agents $\varphi_1, \ldots, \varphi_n$, having *private* data $x_1, \ldots, x_n$, would like to jointly compute some function $f(x_1, \ldots, x_n)$, without revealing any of their private information, other than what can be reasonably deduced from the value of $f(x_1, \ldots, x_n)$.

Formal results in this area provide protocols for computing various functions that are guaranteed to be private under various assumptions, that is, in which no information about the inputs to the functions is obtained beyond that which can be deduced from the output. These assumptions cover three key aspects of the problem. The first aspect is the model of the adversary – the party seeking to violate privacy. For example, a relatively simple assumption is one of "honest but curious", that is each of the agents follows the protocol as laid out, but it is also curious and will try to deduce information about other agents from the information that it obtains during the execution of the protocol. Malicious agents, on the other hand, may deviate from the protocol in order to gain additional information. Furthermore, agents may collude, and so various results will place a cap on the number of malicious or colluding agents. Other aspects of an adversary could be its computational power and memory. The second aspect is the network model. Is communication synchronous or asynchronous? Are the channels secure? Can information get lost in the channel? Finally, the third aspect is the meaning of "secure". This actually has two sub-aspects to it. First, what information exactly is being kept private. Second, how strong are the security guarantees. Here, much like in cryptography, one makes various assumptions under which the protocol is secure, e.g., "factoring is hard."

Privacy-preserving MA-STRIPS planning is clearly an instance of a secure multi-party computation problem. The input to the function is the agents' sets of actions, the initial state, and the goal state, and the output is a plan. Our privacy requirements are a bit weaker than in the standard model, as some of the inputs are public (i.e., public aspects of public actions). On the other hand, some of the output is actually private: the private actions of each agents in the plan. The most natural adversary model is one of *honest-but-curious*, although this requirement is not dictated by the problem definition, and diverse adversary models can be considered.

While in principle, it appears that techniques developed in the area of secure multi-party computation can be extended to our setting of distributed planning, their complexity quickly becomes unmanageable. For example, a common approach for secure multi-party computation uses cryptographic circuits. When solving the shortest path problem (e.g., network routing, Gupta et al., 2012), the size of the circuits created is polynomial in the size of the graph. In our setting the function f computes a shortest path in the implicit graph induced by the descriptions of the agents' actions. As this graph is exponential in the problem description size, it quickly becomes infeasible to construct these circuits given time and memory limitations. While it is true that planning is NP-hard and forward search algorithms do, in general, require exponential time/memory, the purpose of heuristic search is to reduce the search space and to solve large problems in low-polynomial time. Requiring the construction of exponential-sized circuits *a-priori* contradicts the goal of efficiency and feasibility.

The computational problem that most closely resembles privacy-preserving MA-STRIPS planning is privacy preserving constraint satisfaction, which is the subject of the whole sub-field of distributed CSPs, mentioned earlier. In DisCSP, each agent has a single variable, and there exist both binary and unary constraints. Binary constraints are public since more than one agent knows of their existence, while unary constraints are considered private information. In meeting scheduling, an agent has a single variable whose values are possible meeting time slots. A binary constraint could be an equality constraint between the values of two variables belonging to different agents, while a unary constraint represents slots in which the agent cannot hold meetings. Early work on distributed CSP did not explicitly consider privacy issues, except for the type of weak privacy mentioned earlier. Thus, agents were expected to solve the underlying CSP without explicitly revealing their unary constraints.

Later on, work on distributed CSPs (Yokoo, Suzuki, & Hirayama, 2002; Silaghi & Mitra, 2004) identified the fact that even if the algorithm is weakly private, private information may leak during the search process. For example, during search, whenever an agent sends some assignment of its variable to other agents, they can deduce that that value has no unary constraint forbidding it. If this value does not end up being assigned in the solution, the agent revealed some private information that could not have been deduced from only viewing the solution. Thus, followup work focused on the question of how to *measure* this privacy loss (Franzin, Rossi, Freuder, & Wallace, 2004; Maheswaran, Pearce, Bowring, Varakantham, & Tambe, 2006), analyzing how much information specific algorithms lose (Greenstadt, Pearce, & Tambe, 2006), and on the question of how to alter existing DisCSP algorithms to handle stricter privacy demands (Greenstadt, Grosz, & Smith, 2007; Léauté & Faltings, 2009). Very recently, work that provides formal guarantees for some DisCSP algorithms has emerged (Léauté & Faltings, 2009; Grinshpoun & Tassa, 2014). This work builds on techniques from the area of secure multi-party computation.

4. Multi-agent Forward Search

This section describes our distributed variant of forward best-first search, which we call MAFS. We begin with the algorithm itself, including an overview and pseudo-code. Then, we provide an example of the flow of MAFS, and a discussion of its finer points.

4.1 The MAFS Algorithm

Algorithms 1-3 depict the MAFS algorithm for agent φ_i. In MAFS, a separate search space is maintained by each agent. Each agent maintains an *open list* of states that are candidates for expansion and a *closed list* of already expanded states. It expands the state with the minimal f value in its open list, which is initialized with the agent's projected view of the initial state. When an agent expands state s, *it uses its own operators only.* This means two agents (that have different operators) expanding the same state, will generate *different* successor states.

Since no agent expands all relevant search nodes, messages must be sent between agents, informing one agent of open search nodes relevant to it expanded by another agent. Agent φ_i characterizes state s as relevant to agent φ_j if φ_j has a public operator whose public preconditions (the preconditions φ_i is aware of) hold in s, and the creating action of s is public. In that case, Agent φ_i will send s to Agent φ_j.

Algorithm 1 MAFS for agent φ_i

while TRUE **do**
    for all messages m in message queue **do**
        process-message(m)
    $s \leftarrow$ **extract-min**(open list)
    expand(s)

Algorithm 2 process-message($m = \langle s, g_{\varphi_j}(s), h_{\varphi_j}(s)\rangle$)

1: **if** s is not in open or closed list **or** $g_{\varphi_i}(s) > g_{\varphi_j}(s)$ **then**
2:     add s to open list **and** calculate $h_{\varphi_i}(s)$
3:     $g_{\varphi_i}(s) \leftarrow g_{\varphi_j}(s)$
4:     $h_{\varphi_i}(s) \leftarrow max(h_{\varphi_i}(s), h_{\varphi_j}(s))$

The messages sent between agents contain the full state s, i.e. including both public and private variable values (we later discuss how these can be encrypted), as well as the cost of the best plan from the initial state to s found so far, and the sending agent's heuristic estimate of s. When agent φ receives a state via a message, it checks whether this state exists in its open or closed lists. If it does not appear in these lists, it is inserted into the open list. If a copy of this state with higher g value exists, its g value is updated, and if it is in the closed list, it is reopened. Otherwise, it is discarded. Whenever a received state is (re)inserted into the open list, the agent computes its local h value for this state, and then can choose between/combine the value it has calculated and the h value in the received message. If both heuristics are known to be admissible, for example, the agent could choose the maximal of the two estimates, as is done in Line 4 of Algorithm 2. Otherwise, an agent is free to combine the estimates however it sees fit, depending on the known characteristics of the heuristics. How the agent does this does not affect the correctness of the algorithm, but could affect search efficiency. The issue of combining heuristic estimates is discussed further in Section 9.1.

Algorithm 3 expand(s)

1: move s to closed list
2: **if** s is a goal state **then**
3:    broadcast s to all agents
4:    **if** s has been broadcasted by all agents **then**
5:       **return** s as the solution
6: **for all** agents $\varphi_j \in \Phi$ **do**
7:    **if** the last action leading to s was public **and** φ_j has a public action for which all public preconditions hold in s **then**
8:       send s to φ_j
9: apply φ_i's successor operator to s
10: **for all** successors s' **do**
11:    update $g_{\varphi_i}(s')$ and calculate $h_{\varphi_i}(s')$
12:    **if** s' is not in closed list **or** $f_{\varphi_i}(s')$ is now smaller than it was when s' was moved to closed list **then**
13:       move s' to open list

Once an agent expands a *solution* state s, it sends s to all agents and awaits their confirmation, which is sent whenever they expand, and then broadcast state s (Line 3 of Algorithm 3). For simplicity, and in order to avoid deadlocks, once an agent either broadcasts or confirms a solution, it is not allowed to generate new solutions. If a solution is found by more than one agent, the one with lower cost is chosen, and ties are broken by choosing the solution of the agent having the lower ID. When the solution is confirmed (broadcasted) by all agents (Line 4 of Algorithm 3), the agent returns the solution and initiates the trace-back of the solution plan. This is also a distributed process, which involves all agents that perform some action in the optimal plan. The initiating agent begins the trace-back, and when arriving at a state received via a message, it sends a trace-back message to the sending agent. This continues until arriving at the initial state. When the trace-back phase is done, a terminating message is broadcasted and the solution is outputted.

As we will see, this general and simple scheme – apply your own actions/operators only and send relevant generated nodes to other agents – can be used to distribute other search algorithms. However, there are various subtle points pertaining to message sending and termination that influence the correctness and efficiency of the distributed algorithm, which we discuss later.

To better demonstrate the flow of the algorithm, consider the example given in Figure 4.1. In this example, we have two agents who must cooperate in order to achieve a goal. The agents' actions are described on the left-hand side, where every node in the graph depicts an action, and an edge (u, v) indicates that u either achieves or destroys a precondition of v. There are two public actions a_5, a_8, which affect/depend on the only public variable, v_4, while the rest of the actions are private. The central part of the figure depicts the joint search space, i.e., the nodes generated by centralized search. The right-hand site depicts the local search space of the agents, i.e., the nodes generated and sent by each agent when running MAFS. In the initial state, all variable values are zero (i.e., $I = 0000$), and the goal

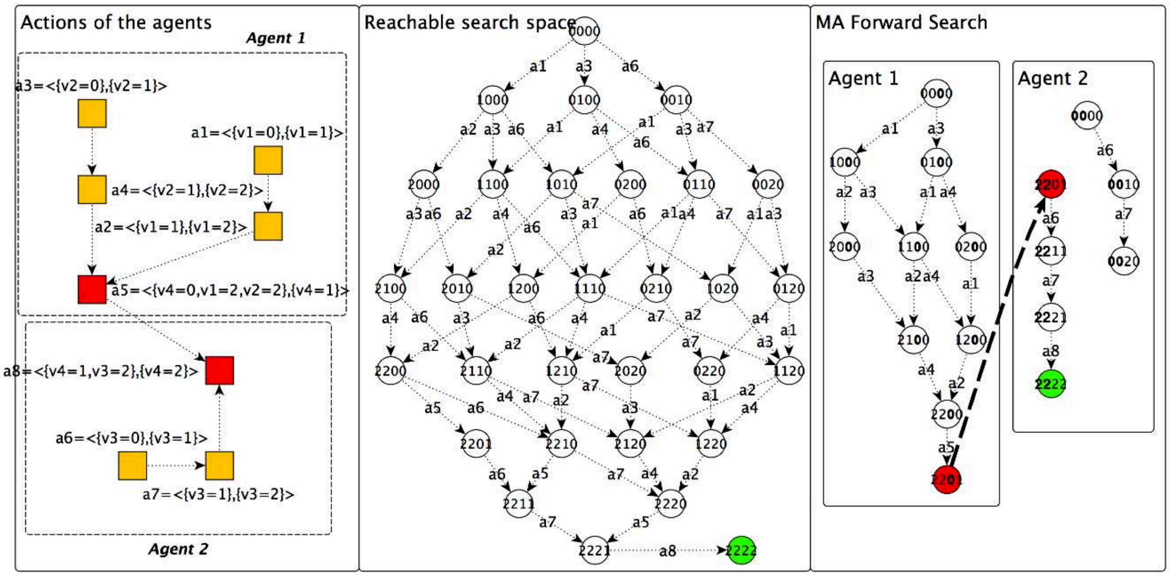

Figure 1: Description of the actions of an example planning problem, its reachable search space, and the search space generated by MAFS. Actions are represented as $<$ *pre*, *eff* $>$ and states are denoted by the values of variables v_1, v_2, v_3, v_4 respectively (For example, 1122 denotes the state where $v_1 = 1, v_2 = 1, v_3 = 2, v_4 = 2$.).

is $G = \{v_4 = 2\}$. The values of private variables *not* belonging to the agents (v_3 for agent 1, v_1, v_2 for agent 2) are shown in bold. These values are not required to be known by the agents, and they can in fact be regarded as don't cares, to be used only as identifiers. When the agents begin searching using MAFS, each applies its own actions only. Therefore, agent 2 quickly exhausts its search space, since as far as it is concerned, state 0020 is a dead end. Agent 1 generates its search space, until it applies public action a_5, which results in state $s = 2201$. s is then sent to agent 2, since all the public preconditions of a_8 hold in s (Line 7 of Algorithm 3). Upon receiving s, agent 2 continues applying its actions, eventually reaching the goal state, which is then broadcasted.

4.2 Discussion

We now discuss some of the more subtle points of MAFS.

4.2.1 Preserving Agent Privacy

If our goal is to preserve privacy, it may appear that MAFS agents are revealing their private data because they transmit their private state in their messages. Yet, in fact, this information is not used by any of the other agents, nor is it altered. It is simply copied to future states to be used only by the agent. Since private state data is used only as an ID, the agents can encrypt this data and keep a table locally, which maps IDs to private states. This encryption can easily be made to generate multiple IDs for each private state, so that the same ID is never used twice, and so other agents cannot identify each other's private states. Consequently, all algorithms based on the distributed search paradigm described are weakly privacy preserving. The issue of privacy is discussed further in Section 9.1.

To compute heuristic estimates of states it receives, an agent must assess the effort required to achieve the goal from them. To do this, it needs some information about the effort required of other agents to construct their part of the plan. In a fully cooperative setting, an agent can have access to the full description of other agents' actions. In the privacy preserving setting, two issues arise. First, agents have only partial information about other agents' capabilities – they only have access to their public interface. Second, different agents may compute different heuristic estimates of the same state because each agent has full information about its capabilities, but not about those of the others. This issue does not affect the actual algorithm, which is agnostic to how agents compute their heuristic estimate, although the fact that agents have less information can lead to poorer heuristic estimates. On the other hand, agents are free to use different heuristic functions, and as we will demonstrate empirically, using the public interfaces only, we are still able to efficiently solve planning problems.

When a state is reached via a message, it includes the sending agent's heuristic estimate. Therefore, the receiving agent now has two (possibly different) estimates it can use. If the heuristics are known to be admissible, then clearly the maximal (more accurate) value is taken, as in line 4 of Algorithm 2. If not, the agent is free to decide how to use these estimates, depending on their known qualities.

4.2.2 Relevancy and Timing of the Messages

State s is considered relevant to agent φ_j if it has a public action for which all public preconditions hold in s and the last action leading to s was public (line 7 of Algorithm 3). This means that all states that are products of private actions are considered irrelevant to other agents. As it turns out, since private actions do not affect other agents' capability to perform actions, an agent need send only states in which the last action performed was public, in order to maintain completeness (and cost-optimality, as proved in the next section). Regarding states that are products of private actions as irrelevant decreases communication, while effectively pruning large, symmetrical parts of the search space. In fact, we will show in Section 8 how this property of mafs can be used to obtain state-space pruning in centralized planning algorithms, using a method called *Partition-based pruning*.

As was hinted earlier, there exists some flexibility regarding when these relevant states are sent. Centralized search can be viewed as essentially "sending" every state (i.e., inserting it to its open list) once it is generated. In mafs, relevant states can be sent when they are expanded (as in the pseudo-code) or once they are generated (changing Algorithm 3 by moving the for-loop on line 6 inside the for-loop on line 10). The timing of the messages is especially important in the distributed setting since agents may have different heuristic estimates. Sending the messages once they are generated increases communication, but allows for states that are not considered promising by some agent to be expanded by another agent in an earlier stage. Sending relevant states when they are expanded, on the other hand, decreases communication, but delays the sending of states not viewed as promising. Experimenting with the two options, we found that the lazy approach, of sending the messages only when they are expanded, dominates the other, most likely because communication can be costly.

4.2.3 Concurrent Solutions

Although the solution plan outputted by MAFS is sequential, i.e. requires agents to execute their actions in turns, it can quite easily be parallelized. Since private actions require/ affect only the agent's private propositions, agents can perform them concurrently without hurting the correctness of the plan, as long as the public actions (interaction points between agents) are performed in correct order. Intuitively, if the execution order of the public actions is maintained, the agents are free to execute their private actions concurrently. This parallelization can be done in time linear in solution length, and requires no joint computation. Another option is using one of the many algorithms known for parallelization of plans (Bäckström, 1998). Given the existence of private actions, these plans have much potential for concurrency.

4.2.4 Search Using Complex Actions

In Section 1, we mentioned a scenario where search operators corresponding to real-world actions are implemented using complex simulation software. This situation can arise, for example, with a team of heterogeneous robotic agents, each of which has a dedicated simulator of its actions. Our approach is well suited for such settings: First, forward search methods are capable of using generative, rather than declarative models of the agent's actions, as their central step involves the generation of successor states and their insertion into appropriate queues. They are oblivious as to how the operators are described or implemented, as long as successor states can be generated. Second, our approach respects the natural system structure, and each agent need only apply its own operators. Thus, there is no need to share the generative models amongst the agents.

One problem, however, is the fact that most contemporary methods for generating heuristic functions require either a declarative STRIPS-like description or a generative model (in the case of sampling methods). Fortunately, our empirical results indicate that the use of an approximate model works quite well in practice. Indeed, our approach assumes that other agents use only the public part of an agent's action model, which is only an approximation. Even if the original action model is generative, a declarative approximate model can be constructed using learning techniques (Yang, Wu, & Jiang, 2007). Alternatively, sampling methods could use a suitably developed simplified simulator.

5. Optimal MAFS

MAFS as presented, is not a cost-optimal planning algorithm. Recall that a cost-optimal plan in single-agent planning is one which achieves the goal with the minimal cost, i.e. minimizing the sum of action costs of the plan. This notion remains in the MA case, where we wish to minimize the sum of cost over all agents participating in the plan. This metric is important in MA systems which describe cooperative agents like MA-STRIPS, where the aim is to minimize the cost of the entire system, and not of specific agents. Moreover, cost-optimal algorithms are required for applying *mechanism design* techniques for planning in

MA systems comprising of *selfish agents* (Nissim & Brafman, 2013). In such settings, a cost-optimal plan constitutes a social-welfare maximizing solution. mafs can be slightly modified in order to achieve cost-optimality. We now describe these modifications, which result in a MA variation of A* we refer to as *Multi-Agent Distributed A** (mad-A*).

As in A*, the state chosen for expansion by each agent must be the one with the lowest $f = g + h$ value in its open list, where the heuristic estimates are admissible. In mad-A*, therefore, *extract-min* (Line 4 in Algorithm 1) must return this state.

5.1 Termination Detection

Unlike in A*, expansion of a goal state in mafs does not necessarily mean an optimal solution has been found. In our case, a solution is known to be optimal only if all agents prove it so. Intuitively, a solution state s having solution cost f^* is known to be optimal if there exists no state s' in the open list or the input channel of some agent, such that $f(s') < f^*$. In other words, solution state s is known to be optimal if $f(s) \leq f_{\text{lower}-\text{bound}}$, where $f_{\text{lower}-\text{bound}}$ is a lower bound on the f-value of the entire system (which includes all states in all open lists, as well as states in messages that have not been processed, yet).

To detect this situation, we use Chandy and Lamport's *snapshot algorithm* (Chandy & Lamport, 1985), which enables a process to create an approximation of the global state of the system, without "freezing" the distributed computation. In this approximation we check whether a state exists whose f value is lower than the value of the candidate solution. Although this check is conducted with respect to an approximate global state, the snapshot algorithm guarantees that the answer is positive *iff* it is true for the global state.

The snapshot algorithm works using marker messages. Each process that wants to initiate a snapshot records its local state and sends a marker on each of its outgoing channels. All the other processes, upon receiving a marker, record their local state, the state of the channel from which the marker just came as empty, and send marker messages on all of their outgoing channels. If a process receives a marker after having recorded its local state, it records the state of the incoming channel from which the marker came as carrying all the messages received since it first recorded its local state.

Although there is no guarantee that the global state computed by the algorithm actually occurred at some point during the run of mad-A*, the approximation is good enough to determine whether a stable property currently holds in the system. A property of the system is *stable* if it is a global predicate which remains true once it becomes true. Specifically, properties of the form "$f_{\text{lower}-\text{bound}} \geq c$" for some fixed value c, are stable when h is a *globally consistent* heuristic function. That is, when f values cannot decrease along a path. In our case, this path may involve a number of agents, each with its h values. If each of the local functions h_φ are consistent, and agents apply the max operator when receiving a state via a message (known as *pathmax*), this property holds[3]. The solution verification procedure using the snapshot algorithm, is run whenever a solution state is expanded (meaning it has minimal f-value), and an agent receives confirmation from all agents for that solution state (In the pseudocode, change Line 5 in Algorithm 3 to *initiate verification of s as a solution.*). This occurs when all other agents have expanded that solution state. Only when

3. Although recent work (Holte, 2010) shows that pathmax does not necessarily make a *bona-fide* consistent heuristic, pathmax does ensure that f-values along a path are non-decreasing.

the snapshot algorithm returns true for the stable property that *no states exist with lower f-value*, does the algorithm return the solution as optimal (In the pseudocode, change Line 1 of Algorithm 1 to *while did not receive* TRUE *from a solution verification procedure.*).

We note that for simplicity of the pseudo-code we omitted the detection of a situation where a goal state does not exist. This can be done by determining whether the stable property *"there are no open states in the system"* holds, using the same *snapshot algorithm.*

5.2 Proof of Optimality

We now prove the optimality of MAD-A*. We must note that as it is presented, MAD-A* maintains completeness (and optimality) only if all actions which achieve some goal condition are considered public. This property is assumed throughout this section, but the algorithm is easily modified to remove it.[4] We begin by proving the following lemma (and a corollary) regarding the solution structure of a MA planning problem. This will demonstrate that there always exists an optimal solution having the structure that is found by MAD-A*. We continue by proving a MA extension of a well-known result for A*, which is required for the completeness of MAD-A*. Finally, before proving MAD-A*'s optimality, we prove the correctness of our termination detection procedure.

Lemma 1. *Let $P = (a_1, a_2 \dots, a_k)$ be a legal plan for a MA planning problem Π. Let a_i, a_{i+1} be two consecutive actions taken in P by different agents, of which at least one is private. Then $P' = (a_1, \dots, a_{i+1}, a_i, \dots, a_k)$ is a legal plan for Π and $P(I) = P'(I)$.*

Proof. By definition of private and public actions, and because a_i, a_{i+1} are actions belonging to different agents, $\text{varset}(a_i) \cap \text{varset}(a_{i+1}) = \emptyset$, where $\text{varset}(a)$ is the set of variables which affect and are affected by a. Therefore, a_i does not achieve any of a_{i+1}'s preconditions, and a_{i+1} does not destroy any of a_i's preconditions. Therefore, if s is the state in which a_i is executed in P, a_{i+1} is executable in s, a_i is executable in $a_{i+1}(s)$, and $a_i(a_{i+1}(s)) = a_{i+1}(a_i(s))$. Therefore, $P' = (a_1, \dots, a_{i+1}, a_i, \dots, a_k)$ is a legal plan for Π. Since the suffix $(a_{i+2}, a_{i+3}, \dots, a_k)$ remains unchanged in P', $P(I) = P'(I)$, completing the proof. □

Corollary 1. *Let $P = (a_1, a_2, \dots, a_k)$ be a solution of the MA planning problem Π. Then, there exists an equal cost solution $P' = (a'_1, a'_2, \dots, a'_k)$ that satisfies the following properties:*

1. *P' contains a permutation of the actions of P.*
2. *If a_i is the first public action in P', then $a_1, \dots, a_i$ belong to the same agent.*
3. *For each pair of consecutive public actions a_i, a_j in P', all actions a_l, $i < l \leq j$ belong to the same agent.*

Proof. Using repeated application of Lemma 1, we can move any ordered sequence of private actions performed by agent φ, so that it would be immediately before φ's subsequent public action and maintain legality of the plan. Clearly, P' is a permutation of P and therefore the cost of P and P' is identical. This implies that if P is cost optimal, so is P'. □

4. In order to remove this assumption, an agent must send as messages all states for which the creating action achieved some goal (which may be private). An agent approves a solution state only if all its private goal are achieved.

Next, we prove the following lemma, which is a MA extension of a well known result for A*. In what follows, we have tacitly assumed a *liveness* property with the conditions that every sent message eventually arrives at its destination and that all agent operations take a finite amount of time. Also, for the clarity of the proof, we assume the atomicity of the *expand* and *process-message* procedures.

Lemma 2. *For any non-closed node s and for any optimal path P from I to s which has properties 2 & 3 of Corollary 1, there exists an agent φ which either has an open node s'* ***or*** *has an incoming message containing s', such that s' is on P and $g_\varphi(s') = g^*(s')$.*

Proof. : Let $P = (I = n_0, n_1, \ldots, n_k = s)$. If I is in the open list of some agent φ (φ did not finish the algorithm's first iteration), let $s' = I$ and the lemma is trivially true since $g_\varphi(I) = g^*(I) = 0$. Suppose I is closed for all agents. Let Δ be the set of all nodes n_i in P that are closed by some agent φ, such that $g_\varphi(n_i) = g^*(n_i)$. Δ is not empty since, by assumption, $I \in \Delta$. Let n_j be the element of Δ with the highest index, closed by agent φ. Clearly, $n_j \neq s$ since s is non-closed. Let a be the action causing the transition $n_j \to n_{j+1}$ in P. Therefore, $g^*(n_{j+1}) = g_\varphi(n_j) + cost(a)$.

If φ is the agent performing a, then n_{j+1} is generated and moved to φ's open list in lines 9-13 of Algorithm 3, where $g_\varphi(n_{j+1})$ is assigned the value $g_\varphi(n_j) + cost(a) = g^*(n_{j+1})$ and the claim holds.

Otherwise, a is performed by agent $\varphi' \neq \varphi$. If a is a public action, then all its preconditions hold in n_j, and therefore n_j is sent to φ' by φ in line 8 in Algorithm 3. If a is a private action, by the definition of P, the next *public* action a' in P is performed by φ'. Since private actions do not change the values of public variables, the public preconditions of a' must hold in n_j, and therefore n_j is sent to φ' by φ in line 8 in Algorithm 3. Now, if the message containing n_j has been processed by φ', n_j has been added to the open list of φ' in Algorithm 2 and the claim holds since $g_{\varphi'}(n_j) = g_\varphi(n_j) = g^*(n_j)$. Otherwise, φ' has an incoming (unprocessed) message containing n_j and the claim holds since $g_\varphi(n_j) = g^*(n_j)$. □

Corollary 2. *Suppose h_φ is admissible for every $\varphi \in \Phi$, and suppose the algorithm has not terminated. Then, for any optimal solution path P which follows the restrictions of Lemma 1 from I to any goal node $s^\star$, there exists an agent φ_i which either has an open node s* ***or*** *has an incoming message containing s, such that s is on P* ***and*** *$f_{\varphi_i}(s) \leq h^*(I)$.*

Proof. : By Lemma 2, for every restricted optimal path P, there exists an agent φ_i which either has an open node s **or** has an incoming message containing s, such that s is on P and $g_{\varphi_i}(s) = g^*(s)$. By the definition of f, and since h_{φ_i} is admissible, we have in both cases:

$$\begin{aligned} f_{\varphi_i}(s) = g_{\varphi_i}(s) + h_{\varphi_i}(s) &= g^*(s) + h_{\varphi_i}(s) \\ &\leq g^*(s) + h^*(s) = f^*(s) \end{aligned}$$

But since P is an optimal path, $f^*(n) = h^*(I)$, for all $n \in P$, which completes the proof. □

Another lemma must be proved regarding the solution verification process. We assume global consistency of all heuristic functions, since all admissible heuristics can be made consistent by locally using the pathmax equation (Méro, 1984), and by using the *max* operator as in line 4 of Algorithm 2 on heuristic values of different agents. This is required since $f_{\text{lower-bound}}$ must be non-decreasing.

Lemma 3. *Let φ be an agent which either has an open node s* ***or*** *has an incoming message containing s. Then, the solution verification procedure for state s^* with $f(s^*) > f_\varphi(s)$ will return false.*

Proof. Let φ be an agent which either has an open node s **or** has an incoming message containing s, such that $f_\varphi(s) < f(s^*)$ for some solution node s^*. The solution verification procedure for state s^* verifies the stable property $p =$ "$f(s^*) \leq f_{\text{lower-bound}}$". Since $f_{\text{lower-bound}}$ represents the lowest f-value of any open or unprocessed state in the system, we have $f_{\text{lower-bound}} \leq f_\varphi(s) < f(s^*)$, contradicting p. Relying on the correctness of the snapshot algorithm, this means that the solution verification procedure will return false, proving the claim. □

We can now prove the optimality of our algorithm.

Theorem 1. MAD-A* *terminates by finding a cost-optimal path to a goal node, if one exists.*

Proof. : We prove this theorem by assuming the contrary - the algorithm does not terminate by finding a cost-optimal path to a goal node. Three cases are to be considered:

1. *The algorithm terminates at a non-goal node.* This contradicts the termination condition, since solution verification is initiated only when a goal state is expanded.

2. *The algorithm does not terminate.* Since we are dealing with a finite search space, let $\chi(\Pi)$ denote the number of possible *non-goal* states. Since there is only a finite number of paths from I to any node s in the search space, s can be *reopened* a finite number of times. Let $\rho(\Pi)$ be the maximum number of times any *non-goal* node s can be reopened by any agent. Let t be the time point when all non-goal nodes s with $f_\varphi(s) < h^*(I)$ have been closed forever by all agents φ. This t exists, since: a) we assume liveness of message passing and agent computations; b) after at most $\chi(\Pi) \times \rho(\Pi)$ expansions of non-goal nodes by φ, all non-goal nodes of the search space must be closed forever by φ; and c) no goal node s^* with $f(s^*) < h^*(I)$ exists[5].

   By Corollary 2 and since an optimal path from I to some goal state s^* exists, some agent φ expanded state s^* at time t', such that $f_\varphi(s^*) \leq h^*(I)$. Since s^* is an optimal solution, if $t' \geq t$, $f_{\text{lower-bound}} \geq f_\varphi(s^*)$ at time t'. Therefore, φ's verification procedure of s^* will return true, and the algorithm terminates.

   Otherwise, $t' < t$. Let φ' be the last agent to close a non-goal state s with $f_{\varphi'}(s) < f_\varphi(s^*)$. φ' has s^* in its open list or as an incoming message. This is true because s^* has been broad-casted to all agents by φ, and because every time s^* is closed by some agent (when it expands it), it is immediately broad-casted again, ending up in the agent's open list or in its message queue. Now, φ' has no more open nodes with f-value lower than s^*, so it will eventually expand s^*, initiating the solution verification procedure which will return true, since $f_{\text{lower-bound}} \geq f_\varphi(s^*)$. This contradicts the assumption of non-termination.

5. This is needed since *goal* node expansions are not bounded.

3. *The algorithm terminates at a goal node without achieving optimal cost.* Suppose the algorithm terminates at some goal node s with $f(s) > h^*(I)$. By Corollary 2, there existed just before termination an agent φ having an open node s', or having an incoming message containing s', such that s' is on an optimal path and $f_\varphi(s') \leq h^*(I)$. Therefore, by Lemma 3, the solution verification procedure for state s will return false, contradicting the assumption that the algorithm terminated.

This concludes the proof. □

As a final note, we point out that the above results can be used to prove that MAFS versions of other search algorithms, such as best-first search, are complete, thanks to Corollary 1 and Lemma 2. Corollary 1 guarantees that we can focus on solutions with certain properties. That is, if a solution exists, then a solution with these properties exists. Lemma 2 ensures that every path with these properties will be generated, eventually, until a solution is found. We note that we cannot provide such guarantees for distributed versions of every search algorithm simply because an arbitrary search algorithm may prune, for some reason, solutions with the properties ensured by the MAFS schema, while keeping other solutions. And of course, completeness guarantees for MAFS require that the liveness properties discussed earlier hold.

6. MA Planning Framework

One of the main goals of this work is to provide a general and scalable framework for solving the MA planning problem. We believe that such a framework will provide researchers with fertile ground for developing new search techniques and heuristics for MA planning, and extensions to richer planning formalisms.

We chose Fast Downward (Helmert, 2006) (FD) as the basis for our MA framework – **MA-FD**. FD is currently the leading framework for planning, both in the number of algorithms and heuristics that it provides, and in terms of performance – winners of the past three international planning competitions were implemented on top of it. FD is also well documented and supported, so implementing and testing new ideas is relatively easy.

MA-FD uses FD's translator and preprocessor, with minor changes to support the distribution of operators to agents. Each agent also receives a projected version of the public operators of the other agents. This information (its own actions and the projected public actions) are provided to the heuristic used by the agent. In addition to the PDDL files describing the domain and the problem instance, MA-FD receives a file detailing the number of agents, their names, and their IP addresses. The agents do not have shared memory, and all information is relayed between agents using messages. Inter-agent communication is performed using the TCP/IP protocol, which enables running multiple MA-FD agents as processes on multi-core systems, networked computers/robots, or even cloud platforms like Amazon Web Services. MA-FD is therefore fit to run as is on *any* number of (networked) processors, in both its optimal and satisficing setting.

Both settings are currently implemented and available[6], and since there is full flexibility regarding the heuristics used by agents, heuristics available on FD are also available on MA-FD, requiring only preprocessing of the agent's view of the problem, creating its projected

6. The code is available at `http://github.com/raznis/dist-selfish-fd` .

view. New heuristics are easily implementable, as in FD, and creating new search algorithms can also be done with minimal effort, since MA-FD provides the ground-work (parsing, communication, etc.).

7. Empirical Results

The following section describes an empirical evaluation of our methods. We begin by evaluating MAFS, comparing it to current state-of-the-art approaches. We then describe results for MAD-A*, compared to centralized cost-optimal search. We also provide scalability results for both methods, scaling up to 40 agents.

7.1 Evaluating MAFS

To evaluate MAFS in its non-optimal setting, we compare it to the state-of-the-art distributed planner MAP-POP (Torreño et al., 2012), and to the Planning-First algorithm (Nissim et al., 2010). As noted in Section 1, another available algorithm for distributed MA planning is via reduction to distributed CSPs using an off-the-shelf disCSP solver. We found this approach was incapable of solving even small planning problems, and therefore we omitted its results from the tables. The problems used are benchmarks from the International Planning Competition (IPC) where tasks can naturally be cast as MA problems. The *Satellites* and *Rovers* domains were motivated by real MA applications used by NASA. Satellites requires planning and scheduling observation tasks between multiple satellites, each equipped with different imaging tools. Rovers involves multiple rovers navigating a planetary surface, finding samples and communicating them back to a Lander. *Logistics*, *Transport* and *Zenotravel* are transportation domains, where multiple vehicles transport packages to their destination. The Transport domain generalizes Logistics, adding a capacity to each vehicle (i.e., a limit on the number of packages it may carry) and different move action costs depending on road length. We consider problems from the Rovers and Satellites domains as loosely-coupled, i.e., problems where agents have many private actions (e.g., instrument warm-up and placement in Rovers, which does not affect other agents), and where a small number of public actions are required in solution plans. On the other hand, we consider the transportation domains as tightly-coupled, having few private actions (only move actions in Logistics) and many public actions (load/unload actions).

For each planning problem, we ran MAFS, using eager best-first search and an alternation open list with one queue for each of the two heuristic functions *ff* (Hoffmann & Nebel, 2001) with preferred actions and the *context-enhanced additive heuristic* (Helmert & Geffner, 2008). Table 7.1 depicts results of MAFS, MAP-POP and Planning-First, on all IPC domains supported by MAP-POP. We also include results for a baseline centralized planner (denoted FD, implemented on top of Fast-Downward) using the same eager best-first search and heuristics used in MAFS. Recall that this planner solves the problem having complete knowledge, unlike the other configurations. We compare the algorithms across three categories – 1) solution cost which reports the total cost of the outputted plan, 2) running time, and 3) the number of messages sent during the planning process. Experiments were run on a AMD Phenom 9550 2.2GHZ processor, time limit was set at 60 minutes, and memory usage was limited to 4GB. For all configurations shown in Table 7.1, experiments were re-

	#	Solution cost				Runtime				Messages		
problem	agents	FD	MAFS	MAP-POP	P-F	FD	MAFS	MAP-POP	P-F	MAFS	MAP-POP	P-F
Logistics4-0	3	21	**20**	**20**	X	≤ 0.1	**0.05**	20.9	X	**340**	375	X
Logistics5-0	3	28	**27**	**27**	X	≤ 0.1	**0.1**	90.4	X	**450**	1565	X
Logistics6-0	3	26	**25**	**25**	X	≤ 0.1	**0.06**	60.6	X	**470**	1050	X
Logistics7-0	4	43	**36**	37	X	≤ 0.1	**0.2**	233.3	X	**2911**	4898	X
Logistics8-0	4	32	**31**	**31**	X	≤ 0.1	**0.16**	261	X	**940**	4412	X
Logistics9-0	4	39	**36**	**36**	X	≤ 0.1	**1.02**	193.3	X	**2970**	3168	X
Logistics10-0	5	50	**45**	51	X	≤ 0.1	**0.43**	471	X	**2097**	14738	X
Logistics11-0	5	**54**	**54**	X	X	≤ 0.1	**2.7**	X	X	**14933**	X	X
Logistics12-0	5	47	**44**	45	X	≤ 0.1	**1.3**	1687	X	**4230**	28932	X
Logistics13-0	7	**85**	87	X	X	≤ 0.1	**0.9**	X	X	**5140**	X	X
Logistics14-0	7	**68**	**68**	X	X	≤ 0.1	**0.67**	X	X	**2971**	X	X
Logistics15-0	7	**94**	95	X	X	≤ 0.1	**0.74**	X	X	**6194**	X	X
Rovers5	2	**22**	**22**	24	24	≤ 0.1	**0.13**	18.7	22.4	**84**	323	590
Rovers6	2	38	**37**	39	X	≤ 0.1	**0.07**	18.2	X	**27**	313	X
Rovers7	3	18	18	18	X	≤ 0.1	**0.07**	44.1	X	**225**	490	X
Rovers8	4	**26**	**26**	27	X	≤ 0.1	**0.2**	744	X	**937**	12102	X
Rovers9	4	40	38	**36**	X	≤ 0.1	**0.82**	222	X	**380**	4467	X
Rovers10	4	**37**	38	X	X	≤ 0.1	**0.41**	X	X	**271**	X	X
Rovers11	4	42	37	**34**	X	≤ 0.1	**0.34**	132.5	X	**299**	2286	X
Rovers12	4	21	21	**20**	X	≤ 0.1	**0.09**	34.4	X	435	**410**	X
Rovers13	4	**48**	49	X	X	≤ 0.1	**0.15**	X	X	**472**	X	X
Rovers14	4	33	**31**	35	X	≤ 0.1	**0.42**	443.8	X	**310**	7295	X
Rovers15	4	**43**	46	44	X	≤ 0.1	**0.33**	164	X	**252**	2625	X
Rovers17	6	53	**52**	X	X	≤ 0.1	**0.57**	X	X	**628**	X	X
Satellites3	2	11	11	11	11	≤ 0.1	**0.01**	4.5	6.8	**7**	78	104
Satellites4	2	26	**17**	20	20	≤ 0.1	**0.17**	6.4	35.2	**36**	109	144
Satellites5	3	21	16	**15**	X	≤ 0.1	**0.15**	15.4	X	**78**	250	X
Satellites6	3	20	20	20	X	≤ 0.1	**0.02**	12.2	X	**30**	323	X
Satellites7	4	28	**22**	**22**	X	≤ 0.1	**0.23**	28.8	X	**248**	543	X
Satellites8	4	27	**26**	**26**	X	≤ 0.1	**0.21**	40.7	X	**133**	678	X
Satellites9	5	37	30	**29**	X	≤ 0.1	**0.35**	93.3	X	**397**	1431	X
Satellites10	5	37	30	**29**	X	≤ 0.1	**0.41**	65.9	X	**355**	942	X
Satellites11	5	36	**31**	**31**	X	≤ 0.1	**0.69**	51	X	**514**	904	X
Satellites12	5	**43**	**43**	49	X	0.2	**1.1**	76.9	X	**390**	1240	X
Satellites13	5	75	**61**	X	X	0.57	**0.88**	X	X	**639**	X	X
Satellites14	6	49	44	**43**	X	0.3	**1.8**	123.4	X	**721**	1781	X
Satellites15	8	64	**63**	X	X	0.66	**3.9**	X	X	**1507**	X	X
Satellites16	10	62	**56**	56	X	0.94	**6.6**	481.2	X	**2279**	4942	X
Satellites17	12	**48**	49	49	X	1.12	**6.7**	2681	X	**2172**	26288	X

Table 1: Comparison of MA greedy best-first search, MAP-POP and Planning-First. Solution cost, running time (in sec.) and the number of sent messages are shown. *"X"* denotes problems which weren't solved after one hour, or in which the 4GB memory limit was exceeded. Best performance entries are in bold.

stricted to run on a *single* processor (core), so running time would be easily comparable[7]. An "X" signifies that the problem was not solved within the time-limit, or exceeded memory constraints.

7. In all other result tables, multiple processors were used.

It is clear that MAFS overwhelmingly dominates both MAP-POP and Planning-First (denoted P-F), with respect to running time and communication, solving *all* problems faster while sending less messages. All problems were solved at least 70× faster than MAP-POP, with several Logistics and Rovers problems being solved over 1000× faster, and the largest Satellites instance being solved over 400× faster. The low communication complexity of MAFS is important, since in distributed systems message passing could be more costly and time-consuming than local computation. Moreover, as messages in MAFS are essentially a state description, message size is always linear in the number of propositions. Although for some problems MAP-POP finds lower-cost solutions, in most cases MAFS outputs better solution quality. We believe that when MAFS finds lower quality solutions, this is mostly because message-passing takes longer than local computation – a subset of agents that have the ability to achieve the goal "on their own", will do so before being made aware of other, less costly solutions including other agents. One possible way of improving solution quality further would be using anytime search methods, which improve solution quality over time. In comparison to the reference centralized planner, MAFS takes longer to compute a solution, but in most cases these solutions have lower or equal cost. The slowdown is expected, both because of communication times and the partial information MAFS agents have, unlike the centralized planner.

7.1.1 Scalability of MAFS

In order to evaluate the scalability of MAFS, we conducted experiments on the Logistics, Rovers and Satellite domains from the IPC. For each of the domains, we generated multiple problem instances, increasing the number of agents k. In Logistics, the number of cities grows linearly with k, the number of airplanes remains constant at 2, and the number of packages is always $2k$. For Satellites and Rovers, the number of targets grows linearly with k, the number of available observations/locations is $2k$, and the instrumentation remains constant. The experiments were run on an Intel Xeon 2.4GHZ, 48-core machine. FD was run on a single processor while each MAFS agent was given a dedicated processor. Cutoff time for both configurations was set at 1 hour (Wall-clock time), and memory limit was set at 100GB, regardless of the number of processors used.

Results of the scalability experiment can be seen in Figure 7.1.1. For every value of k, 5 different problem instances were generated. The runtime values shown are averages over all 5 of them, and the error bars correspond to the standard deviation of the sample[8]. In the loosely-coupled domains Rovers and Satellites, MAFS's increase in runtime is nearly linear, where the largest of the problem instances in both domains which were unsolved by centralized FD, were solved in under 90 seconds. Efficiency (speedup divided by the number of processors) values are always superlinear – 16 for the largest Satellites problem solved by both configurations and > 11 for the largest Rovers problems. While being superior in speed, MAFS outputs lower quality solutions (having higher total cost) in these domains. On average, MAFS solution cost is 14% higher in Rovers and 6% higher in Satellites, with the maximal increase being 20% and 13% respectively. The cause of deterioration in solution quality in these domains is most likely the fact that in many problems, small subsets of agents can reach a solution without other agents. This may lead to MAFS quickly finding

8. The error bars were omitted in cases where standard deviation was too small to be shown on this scale.

a solution involving this subset, but this solution is usually more costly than that is found by centralized search, which considers operators of all agents. In Logistics, a more tightly-coupled domain where many actions are public, problems are still solved much faster (5× faster for the 40-agent problems) by MAFS, but the efficiency is < 1 – on average 0.12 for the largest instances. Here, solution cost is on average 0.5% higher than centralized search, where the maximal increase is 2.5%, and the maximal decrease (improvement in solution quality) is 2%. Overall, we can see that w.r.t. runtime, MAFS is *scalable* – performing with superlinear efficiency in loosely-coupled domains, and exhibiting speedup in the tightly-coupled ones.

7.2 Evaluating MAD-A*

To evaluate MAD-A* with respect to centralized optimal search (A*), we ran both algorithms using the state-of-the-art Merge&Shrink heuristic[9] (Helmert, Haslum, & Hoffmann, 2007). Both configurations were run on the same machine, where for MAD-A*, each agent was allocated a single processor, and A* was run on a single processor. Wall-clock time limit was set at 30 minutes, and memory usage was limited to 4GB, regardless of the number of cores used. We show results up to problems which constitute the limit of either configuration. No more existing IPC problems in these domains were solvable by both configurations. Table 7.2 depicts the runtime, efficiency (speedup divided by the number of processors), number of expanded nodes and the average of the agents' initial state h-values.

When comparing MAD-A* to centralized A*, our intuition is that efficiency will be low, due to the inaccuracy of the agents' heuristic estimates, and the overhead incurred by communication. In fact, the local estimates of the agents are much less accurate than those of the global heuristic, as is apparent from the lower average h values of the initial state, given as approximate measures of heuristic quality – when discussing admissible heuristics (which are required for optimal search), higher values are necessarily more accurate. In the tightly-coupled domains – Logistics, Transport and Zenotravel, we do notice very low efficiency values, mostly due to the large number of public actions, which result in many messages being passed between agents, and the relatively small amount of local (private) search by the agents. However, in the more loosely-coupled domains Satellites and Rovers, MAD-A* exhibits nearly linear and super-linear speedup, solving 2 problems not solved by centralized A*. We analyze the reason for this superlinear speedup and elaborate on this important issue in Section 8.

7.2.1 Scalability of MAD-A*

From the results in Table 7.2, it is clear that MAD-A* does not scale well in tightly-coupled domain such as Logistics, Transport and Zenotravel. In the loosely-coupled domains of Rovers and Satellites, however, MAD-A* exhibited high efficiency, outperforming centralized search in some cases. This section examines the scalability of MAD-A* in these domains compared to that of centralized A*.

As in Section 7.1.1, we created new instances of problems with increasing agents using problems generators. In both domains, the number of available locations and goals increased

9. We used exact bisimulation with abstraction size limit $10K$ (DFP-bop) as the shrink strategy of Merge&Shrink (Nissim, Hoffmann, & Helmert, 2011).

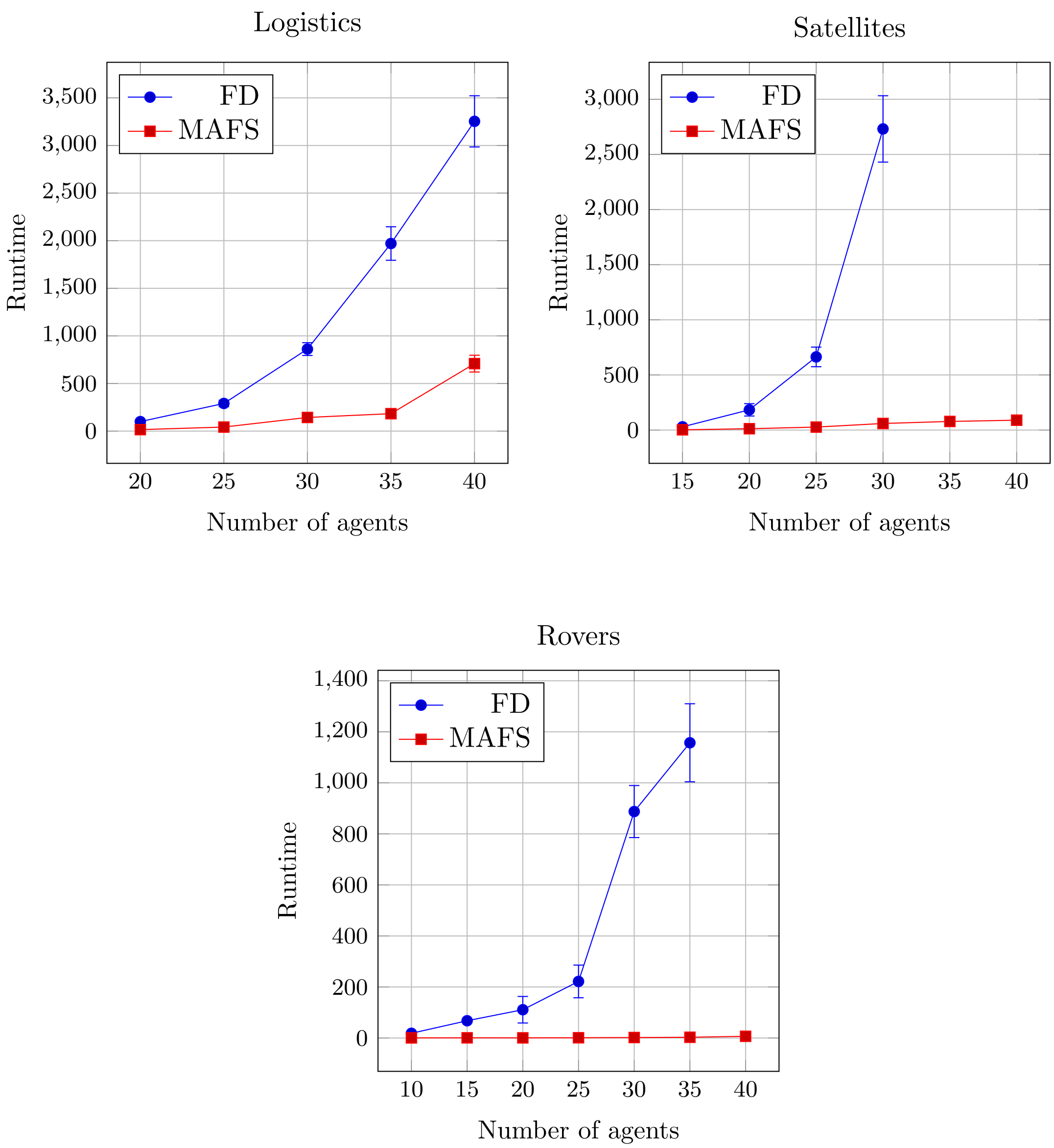

Figure 2: Scalability of MAFS w.r.t. centralized FD. For both configurations, runtime in seconds is shown.

problem	agents	Time			Expansions		init-h	
		A*	MAD-A*	Efficiency	A*	MAD-A*	A*	MAD-A*
Logistics4-0	3	0.07	0.03	0.78	21	2496	20	17
Logistics5-0	3	0.16	0.13	0.41	28	11020	27	23
Logistics6-0	3	0.29	0.15	0.64	547	9722	24	22
Logistics7-0	4	1.42	8.26	0.04	29216	520122	33	29
Logistics8-0	4	1.28	2.89	0.11	16771	157184	27	24
Logistics9-0	4	2.17	11.6	0.05	43283	687572	33	29
Logistics10-0	5	132	X	0.00	4560551	X	37	34
Logistics11-0	5	180	X	0.00	5713287	X	41	38
Rovers3	2	0.2	0.11	0.91	12	86	11	9
Rovers4	2	0.07	0.04	0.88	9	121	8	7
Rovers5	2	8.24	4.4	0.94	213079	307995	12	11
Rovers6	2	X	301	∞	X	**18274357**	24	21
Rovers7	3	32.4	6.82	**1.58**	1172964	**676750**	9	7
Rovers12	4	190.8	27.6	**1.73**	4963979	**2591428**	11	7
Satellites3	2	0.3	0.29	0.52	12	1498	11	5
Satellites4	2	0.6	0.4	0.75	18	5045	17	11
Satellites5	3	16.83	3.9	**1.44**	236647	**42557**	10	7
Satellites6	3	1.93	0.92	0.70	1382	81731	17	12
Satellites7	4	X	18.26	∞	X	**1303910**	10	9
Transport1	2	0.02	0.08	0.13	6	285	54	4
Transport2	2	0.17	0.21	0.40	242	1628	79	4
Transport3	2	1.35	20.64	0.03	69500	546569	34	4
Transport4	2	54.6	335.15	0.08	3527592	4397124	10	10
Transport5	2	X	X	N/A	X	X	12	10
Zenotravel3	2	0.33	0.42	0.39	7	516	6	5
Zenotravel4	2	0.32	0.43	0.37	9	762	7	6
Zenotravel5	2	0.3	0.42	0.36	14	577	10	8
Zenotravel6	2	0.47	0.61	0.39	270	1280	10	8
Zenotravel7	2	0.55	0.8	0.34	621	5681	11	9
Zenotravel8	3	1.22	1.71	0.24	136	5461	9	7
Zenotravel9	3	31.7	315	0.03	775340	6745088	16	14
Zenotravel10	3	9.3	338	0.01	116872	4416461	20	17
Zenotravel11	3	2.99	11.7	0.09	20157	200868	11	9
Zenotravel12	3	X	X	N/A	X	X	16	14

Table 2: Comparison of centralized A* and MAD-A* running on multiple processors. Running time (in sec.), average initial state h-values and MAD-A*'s efficiency values w.r.t. A* are shown. Entries in bold denote super-linear efficiency of MAD-A*.

linearly with the number of agents k. For each value of k, we created 5 problem instances, and the reported runtime values are averages. The experiments were run on the same Intel Xeon machine used in Section 7.1.1, where A* was run on a single processor, and each MAD-A* agent was given a dedicated processor. Runtime limit was set at 90 minutes and memory was limited to 100GB regardless of the number of processors.

In Figure 7.2.1 we can see that in both domains, MAD-A* solves all problems faster than centralized A*, and can solve larger problems within the time limit. However, efficiency varies between the two domains – in Satellites, which is very loosely-coupled, efficiency is superlinear in all cases, reaching up to 7.3 in the largest problem solved by both configura-

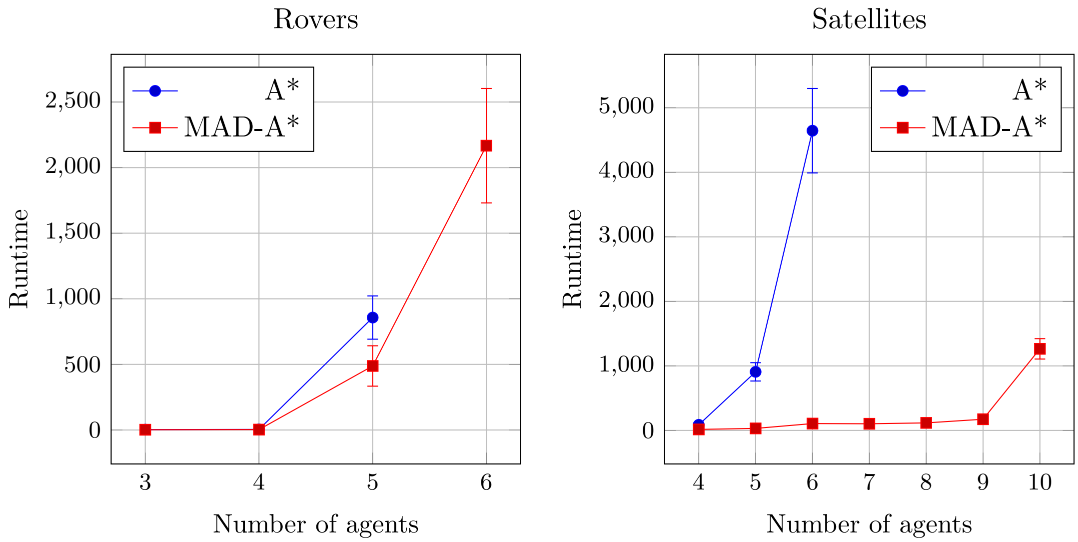

Figure 3: Scalability of MAD-A* w.r.t. centralized A* in loosely-coupled domains. For both configurations, runtime in seconds is shown.

tions. In Rovers, which is more tightly-coupled, efficiency is ≤ 0.5 in all problems, dropping as low as 0.35 in the largest problem solved by both configurations. Recall that the problem solved by MAD-A* is more difficult, since each agent has private information unknown to others, so this drop in efficiency is expected. As mentioned in the previous section, we next elaborate on the reason for MAD-A*'s superlinear efficiency in loosely-coupled problems.

8. Partition-Based Path Pruning

The empirical results presented in the previous section raise an interesting question: *how does MAD-A* achieve > 1 efficiency in weakly-coupled environments?* It is known that when using a consistent heuristic, A* is optimal in the number of nodes it expands to recognize an optimal solution. In principle, it appears that MAD-A* should expand at least the same search tree, so it is not clear, a-priori, why we reach super-linear speedup when comparing to A*. This section provides an explanation to this phenomenon – a *partial order reduction* method inherently "built-in" to MAD-A* (and MAFS as well), which exploits symmetry in the search space to prune effect-equivalent paths. We will describe this pruning method in detail, showing empirical evidence supporting the claim that within this method lies much of the power of MAD-A*.

This exploitation of symmetry utilizes the notion of public and private actions. As we noted in Corollary 1, the existence of private actions implies the existence of multiple effect-equivalent permutations of certain action sequences. A* does not recognize or exploit this fact, and MAFS does. Specifically, imagine that agent φ_i just generated state s using one of its public actions, and s satisfies the preconditions of some public action a of agent φ_j. Agent φ_i will eventually send s to agent φ_j, and the latter will eventually apply a to it. Now, imagine that agent φ_i has a private action a' applicable at state s, resulting in the state $s' = a'(s)$. Because a' is private to φ_i, from the fact that a is applicable at s we deduce that a is applicable at s' as well. Hence, A* would apply a at s'. However, in MAFS, agent φ_j would not apply a at s' because it will not receive s' from agent φ_i. Thus, MAFS does not explore all possible action sequences. This fact can also be clearly seen in the example given in Figure 4.1 – The reachable search space in this example has 31 states, while the number of reachable states using MAFS is only 16.

Since MAFS's inherent pruning of action sequences requires only a partitioning of the actions, it does not pertain only to MA systems, but to any factored system having internal operators. Since the only difference between a MA-STRIPS planning problem and a STRIPS one is the fact that actions are partitioned between agents, why not re-factor the centralized problem into an "artificial" MA one? By mapping all actions into disjoint sets such that $\bigcup_i^k A_i = A$, each representing an "agent", we can now distinguish between private and public operators. Given this distinction, the pruning rule used is simple:

> **Partition-Based (PB) Pruning Rule:** *Following a private action $a \in A_i$, prune all actions not in A_i.*

The fact that this pruning rule is optimality-preserving (i.e., does not prune *all* optimal solutions) follows immediately from Corollary 1 as, if there exists an optimal solution $\pi^\star$, it can be permuted into a legal, optimal plan which is not pruned. This, however, is not enough to maintain the optimality of A* search. We now present a slight modification of the A* algorithm, which allows the application of optimality preserving pruning methods (such as PB-pruning) for the purpose of optimal planning.

8.1 Path Pruning A*

The path pruning A*, (denoted PP-A*), is a search algorithm which receives a planning problem Π and a pruning method ρ as input, and produces a plan π, which is guaranteed to be optimal provided that ρ respects the following properties: (i) ρ is optimality preserving, and (ii) ρ prunes only according to the last action. It is easy to see, for example, that PB pruning respects the second condition, since it fires only according to the last action.

8.1.1 PP-A* versus A*

PP-A* is identical to A* except for the following three changes. First, a different data-type is used for recording an open node. In PP-A*, an open list node is a pair (A, s), where s is the state and A is a set of actions, recording various possible ways to reach s from a previous state. Second, node expansion is subject to the pruning rules of method ρ. Namely, PP-A* executes an applicable action a' in state (A, s) only if there is at least one action $a \in A$ s.t. the execution of a' is allowed after a under ρ's pruning rules. Third, duplicate states are handled differently. In A*, when a state s which is already open is reached by another search path, the open list node is updated with the action of the lower g value, and in case of a tie – drops the competing path. In contrast, ties in PP-A* are handled by preserving the last actions which led to s in each of the paths. Hence, if action a led to an open state s via a path of cost g, and if the existing open list node (A, s) has the same g value, then the node is updated to $(A \cup \{a\}, s)$, thus all actions leading to s with path cost g are saved. Tie breaking also affects the criterion under which closed nodes are reopened. In A*, nodes are reopened only when reached via paths of lower g values. In PP-A*, if an action a leading to state s of some closed node (A, s) is not contained in A, and if the g values are equal, then the node reopens as $(\{A \cup \{a\}\}, s)$. However, when the node is expanded, only actions that are now allowed by ρ and were previously pruned, are executed. We now move to prove the correctness of PP-A*.

8.1.2 Proof of Correctness and Optimality

The next lemma refers to PP-A*, and assumes ρ to be an optimality preserving pruning method, which prunes according to the last action. We say that node (A, s) is *optimal on path* P, if A contains an action a which leads to state s on path P, and $g(s) = g^*(s)$. The notation $s \prec_P s'$ denotes the fact that state s precedes state s' in optimal path P.

Lemma 4. *In* PP-A*, *for any non-closed state s_k and for any optimal non-ρ-pruned path P from I to s_k, there exists an open list node (A', s') which is optimal on P.*

Proof. Let P be an optimal non-ρ-pruned path from I to s_k. If I is in the open list, let $s' = I$ and the lemma is trivially true since $g(I) = g^*(I) = 0$. Suppose I is closed. Let Δ be the set of all nodes (A_i, s_i), optimal on P, that were closed. Δ is not empty, since by assumption, I is in Δ. Let the nodes in Δ be ordered such that $s_i \prec_P s_j$ for $i < j$, and let j be the highest index of any s_i in Δ.

Since the closed node (A_j, s_j) has an optimal g value, it had been expanded prior to closing. From the properties of PP-A*, it follows that the expansion of (A_j, s_j), which is optimal on P, is followed with an attempt to generate a node (A_{j+1}, s_{j+1}) which is optimal on P as well. Generation of (A_{j+1}, s_{j+1}) must be allowed, since under the highest index assumption there can be no closed node containing s which is optimal on P. Naturally, $s_j \prec_P s_{j+1}$.

At this point, we note that actions in A_{j+1} cannot be removed by any competing path from I to s_{j+1}, since (A_{j+1}, s_{j+1}) has an optimal g value. It is possible, though, that additional actions leading to s_{j+1} are added to the node. The updated node can be represented by $(A'_{j+1} \supseteq A_{j+1}, s_{j+1})$, and the property of optimality on P holds. Additionally, node (A'_{j+1}, s_{j+1}) cannot be closed after its generation, since again, this contradicts the highest index property. Hence, there exists an open list node (A', s') which is optimal on P. This concludes the proof. □

Corollary 3. *If h is admissible and ρ is optimality-preserving,* PP-A* *using ρ is optimal.*

Proof. This follows directly from Lemma 4, the optimality preserving property of ρ and the properties of PP-A*, which allow every optimal, non-ρ-pruned path to be generated. □

8.2 Empirical Analysis of PB-Pruning

We set out to check the effect of MAD-A*'s inherent exploitation of symmetry on its efficiency compared to A*. The hypothesis that this is MAD-A*'s main advantage over A* is well supported by the results in Table 8.2, which shows a comparison of MAD-A* and centralized A* using PB pruning. Here, we see that in all problems where MAD-A* achieves superlinear speedup w.r.t. A*, applying partition-based pruning where *partition=agent* reduces runtime and expansions dramatically. In all cases, MAD-A*'s efficiency w.r.t. A* using PB pruning

problem	agents	Time			Efficiency		Expansions		
		A*	$A^{\star}_{pb}$	MAD-A*	A*	$A^{\star}_{pb}$	A*	$A^{\star}_{pb}$	MAD-A*
Logistics4-0	3	0.07	0.06	0.03	0.78	0.67	21	21	2496
Logistics5-0	3	0.16	0.17	0.13	0.41	0.44	28	28	11020
Logistics6-0	3	0.29	0.29	0.15	0.64	0.64	547	527	9722
Logistics7-0	4	1.42	1.07	8.26	0.04	0.03	29216	22425	520122
Logistics8-0	4	1.28	1.07	2.89	0.11	0.09	16771	11750	157184
Logistics9-0	4	2.17	1.59	11.6	0.05	0.03	43283	29953	687572
Logistics10-0	5	132	37	X	0	0	4560551	2132416	X
Logistics11-0	5	180	57.4	X	0	0	5713287	2980725	X
Rovers3	2	0.2	0.2	0.11	0.91	0.91	12	12	86
Rovers4	2	0.07	0.06	0.04	0.88	0.75	9	9	121
Rovers5	2	8.24	3.07	4.4	0.94	0.35	213079	62672	307995
Rovers6	2	X	164.9	301	∞	0.27	X	8107327	18274357
Rovers7	3	32.4	5.22	6.82	1.58	0.26	1172964	235537	676750
Rovers12	4	190.8	10.1	27.6	1.73	0.09	4963979	391372	2591428
Satellites3	2	0.3	0.29	0.29	0.52	0.50	12	12	1498
Satellites4	2	0.6	0.58	0.4	0.75	0.73	18	18	5045
Satellites5	3	16.83	3.15	3.9	1.44	0.27	236647	23503	42557
Satellites6	3	1.93	1.84	0.92	0.70	0.67	1382	385	81731
Satellites7	4	X	26.6	18.26	∞	0.36	X	846394	1303910
Transport1	2	0.02	0.01	0.08	0.13	0.06	6	6	285
Transport2	2	0.17	0.16	0.21	0.40	0.38	242	225	1628
Transport3	2	1.35	0.9	20.64	0.03	0.02	69500	49744	546569
Transport4	2	54.6	29.8	335.15	0.08	0.04	3527592	2589496	4397124
Zenotravel3	2	0.33	0.34	0.42	0.39	0.40	7	7	516
Zenotravel4	2	0.32	0.33	0.43	0.37	0.38	9	9	762
Zenotravel5	2	0.3	0.3	0.42	0.36	0.36	14	14	577
Zenotravel6	2	0.47	0.47	0.61	0.39	0.39	270	220	1280
Zenotravel7	2	0.55	0.56	0.8	0.34	0.35	621	433	5681
Zenotravel8	3	1.22	1.21	1.71	0.24	0.24	136	126	5461
Zenotravel9	3	31.7	12.88	315	0.03	0.01	775340	474180	6745088
Zenotravel10	3	9.3	6.51	338	0.01	0.01	116872	104340	4416461
Zenotravel11	3	2.99	2.02	11.7	0.09	0.06	20157	11565	200868
Zenotravel12	3	X	82.95	X	N/A	0	X	2406708	X

Table 3: Comparison of centralized A* with and without partition-based pruning, and MAD-A* running on multiple processors. Running time, number of expanded nodes, and MAD-A*'s efficiency w.r.t. both centralized configurations are shown.

is sublinear. This is, of course, also due to the fact that MAD-A* solves a more difficult problem – having incomplete information has a negative effect on the quality of heuristics computed by the agents.

We note that although MA structure is evident in some benchmark planning domains (e.g. Logistics, Rovers, Satellites, Zenotravel etc.), in general there isn't always an obvious way of decomposing the problem. In work further exploring PB pruning (Nissim, Apsel, & Brafman, 2012), we describe an automated method for decomposing a general planning problem, making PB pruning applicable in the general classical planning setting. We note that there exist many other *partial-order reduction* methods, such as *Commutativity Pruning* (Haslum & Geffner, 2000), *Stubborn Sets* (Valmari, 1989; Wehrle & Helmert, 2012; Alkhazraji, Wehrle, Mattmüller, & Helmert, 2012; Wehrle, Helmert, Alkhazraji, & Mattmüller, 2013), *Expansion Core* (Chen & Yao, 2009), and more (Coles & Coles, 2010; Xu, Chen, Lu, & Huang, 2011). None of these methods subsumes PB pruning, and the interesting question of how to combine such methods while maintaining optimality remains open in the field of classical planning.

9. Discussion

Our work raises a number of questions, research challenges, and opportunities that we now discuss.

9.1 Privacy

As noted earlier, our algorithms are weakly privacy preserving. That is, no information about private actions, their cost, private variables, and private preconditions and effects is ever communicated by an agent to other agents. Nevertheless, as in the case of DisCSP discussed earlier, information about these elements could be deduced by other agents during the run of the algorithm. And given that privacy preservation is an important goal of our work, it is important to try to understand the extent to which such information can leak.

To examine this, let us consider what is the maximal amount of explicit information that is available to an agent about other agents. In the worse case, we would have to explore the entire search tree rooted at the initial state during the run of the algorithm. However, an agent does not see all the nodes in that tree, but only nodes that correspond to search states obtained following a public action (because only these states are communicated). Thus, the sub-tree visible to an agent corresponds to one obtained from the full search tree by a top-down recursive process where every state obtained following a private action is removed, and the parent of a removed node becomes the parent of its children. Furthermore, the only information available about each state to an agent consists only of the value of its local variables in that state and the value of public variables. (Recall that, as each private state is encrypted using a different identifier, the local states of other agents appear different in every state, so they provide no useful information).

What can an agent learn from this projected search tree? First, it can try to identify different states as identical. Two states whose sub-trees are identical can be deduced as having the same local states for all agents. Notice that this information cannot be deduced from the plan alone. Consequently, we cannot claim that our search-based methods are strong privacy preserving. It is more difficult to identify the fact that two states have the same local state for a particular agent only. Let us, however, assume the worst case scenario in which such states are also identifiable – we will use the term the *projected sub-tree with state ids* to refer to this structure. Thus, the agent knows the possible values, up to renaming, of the local states of an agent that occur following the execution of its public actions. It also knows about the existence of "macros" consisting of a sequence of private actions followed by a public action that allow an agent to move from its private state following a public action to the next private state following the execution of its next public action. It does not, however, know the structure of the local state – it can only deduce some monolithic name. Nor does it know the nature of the local actions comprising these "macros" that enable the transition between two post-public-action states.

But while strong privacy preservation is not guaranteed, it is possible to show – at present on a per domain basis – that distributed forward search algorithms offer more than weak privacy preservation. As an example, we return to the logistics domain. Imagine that the truck has various service stations or rest-stops in locations private to it (i.e., that are served only by that truck). Their existence is not visible to the other agents. This can be proven formally by showing that the projected search tree visible to an external agent is identical regardless of the number of such private locations of the truck. As this projected search tree contains all the information available to other agents, this implies that the information available to other agents is the same in both cases. To see that this projected tree is the same, recall that the public actions of a truck are *load* and *unload*. Furthermore, note that loading or unloading a package in a location that is not accessible to any other agent is a *private* action of the truck because its preconditions and effects are not required or affected by any other agent. Thus, the states visible in the projected search tree are states that follow a *load/unload* action in a location served by other agents as well, such as the airport. In those states, the local state of the truck reflects being in such a location. Consequently, no local state that corresponds to being in a private location is part of the projected search tree. That is, the projected search tree is the same whether the agent has no, one, or any number of private service locations.

Another piece of private information could be the existence of "private" packages. That is, packages being delivered from a private location to another private location. Here, the proof above does not work – the local state of the truck could be different following unloading a (public) package in a public location, simply because the variable denoting the location of the private package could have different values, which are technically part of its local state (because only the truck can influence their value). In fact, the situation is similar in the above case (of private service stations) if packages can be unloaded in the private service stations. Again, local states in which some package is located in such a station would appear in the projected search tree. However, in both cases, external agents cannot infer that the larger set of private states stems from the fact that there is a private package or a private service station. It may very well denote something else, e.g., whether the driver is hungry or thirsty or sleepy, etc. They are only aware of the existence of a larger domain of private states.

Similar techniques can be used to show that the inner processes used by the manufacturers in the laptop example are not visible to other manufacturers: they are very much like the different locations of the truck.

The above discussion makes it clear that distributed forward search is able to provide important privacy guarantees. Future work should seek to go beyond such domain-dependent reasoning and provide general, domain independent conditions under which strong privacy guarantees can be obtained. We believe that this is doable and that the ideas above are a good starting point.

In this context, it is useful to keep in mind the following observations about the projected sub-tree with ids: 1. No practical search algorithm will explore the entire search tree, and as the extent of the search space explored decreases, the difficulty of identifying identical local states increases. Our empirical results indicate that many problems can be solved quickly using distributed forward search, without expanding too many nodes. It is questionable whether it is possible to build reasonable models of agents' private states in such cases.

Here, both empirical and theoretical work – similar to that conducted on distributed CSPs, is desirable. 2. It is not clear whether it is possible to identify two states that share similar local states among only some of the agents, especially given a small portion of the search tree. 3. The number of "macros" consisting of a sequence of private actions followed by a private one is exponential. Thus, in a reasonable search process, only a small portion of them will be visible. As anecdotal evidence we note that we actually attempted to construct such macros in order to improve heuristic computation, but their number exploded so quickly, that this became impractical.

Another interesting alternative is to develop variants of current algorithms that have stronger privacy preserving properties. For example, consider the problem of inferring upper bounds on the minimal cost of applying action a in public state s. In general, private actions which achieve preconditions of a public action do not have to be applied immediately before *that* public action – an agent can perform some of the private actions required for a public action before a previous public action. In other words, an agent can "distribute" the private cost of a public action between different "segments", or parts of the plan between two public actions, making the cost of the first action appear higher and the cost of the second action lower, although with some potential impact on optimality. In the case of non-optimal search, g-values are not disclosed, so this is not an issue.

The above example illustrates a general idea: one can trade-off efficiency for privacy. A similar tradeoff is explored in the area of *differential privacy* (Dwork, 2006). There, some noise is inserted into a database before statistical queries are evaluated, such that the answer to the statistical query is correct to within some given tolerance, ϵ, yet one cannot infer information about a particular entry in the database (e.g., describing the medical record of an individual). Similarly, in our context, one can consider algorithms in which agents refrain from sending certain public states with some probability, or send it with some random delay, or even possibly, generate bogus, intermediate public states. Such changes are likely to have some impact on running time and solution quality, and these tradeoffs would be interesting to explore.

We believe that as this area matures, much like in the area of DisCSP, more attention will be given to the problem of precise quantification of privacy and privacy loss. Our work brings us closer to this stage. It offers algorithms for distributed search that start to match that of centralized search, a general methodology for distributed forward search that respects the natural distributed structure of the system, that can form a basis for such extensions, and some initial ideas on how formal privacy proofs can work.

An additional privacy-related question is the specification of privacy properties. In MA-STRIPS the distinction between private and public variables is derived in a certain way from the domain definition. Recent work that builds on MA-STRIPS suggests a slightly different treatment of privacy. For example, Bonisoli et al. (2014) allow for finer notions of privacy, where variables can be private to a subset of agents, rather than to a single agent. Moreover, the privacy requirements are part of the problem description, so that a variable that would be private in MA-STRIPS could be made public because its privacy is not important to maintain. This extends earlier models, where the set of variables "visible" to each agent is explicitly stated (Torreño, Onaindia, & Sapena, 2014), and where the set of private variables of an agent is not derived, as in MA-STRIPS but rather specified (Luis & Borrajo, 2014). One might also consider distinguishing between public variables that are write-only public variables. For example, an agent may not know the value of a certain variable, except immediately after it assigns it a value. It would be interesting to explore the ability of our algorithms, as well as others, to exploit such notions of privacy.

9.2 Accurate Heuristics Given Incomplete Information

The empirical results presented lead us to what is perhaps the greatest practical challenge suggested by MAFS and MAD-A* – computing an accurate heuristic in a distributed (privacy-preserving) system. Despite theoretical results saying that even having almost perfect heuristics can lead to large (exponential) search spaces (Helmert & Röger, 2008), practically, having an accurate heuristic has a large effect on search efficiency. 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, and/or affecting solution quality. We believe that there are techniques that can be used to alleviate this problem. As a simple example, consider a public action a_{pub} that can be applied only after a private action a_{priv}. For example, in the Rovers domain, a *send* message can only be applied after various private actions required to collect data are executed. If the cost of a_{pub} known to other agents would reflect the cost of a_{priv} as well, the heuristic estimates would be more accurate. Another possibility for improving heuristic estimates is using an additive heuristic. In that case, rather than taking the maximum of the agent's own heuristic estimate and the estimate of the sending agent, the two could be added. To maintain admissibility, this would require using something like cost partitioning (Katz & Domshlak, 2008). One obvious way of doing this would be to give each agent the full cost of its actions and zero cost for other actions. The problem with this approach is that initially, when the state is generated and the only estimate available is that of the generating agent, this estimate is very inaccurate, since it assigns 0 to all other actions. In fact, the agent will be inclined to prefer actions performed by other agents, as they appear very cheap, and we see especially poor results in domains where different agents can achieve the same goal, as in the Rovers domain, resulting in estimates of 0 for many non-goal states. Therefore, how to effectively compute accurate heuristics in the distributed setting is an important research challenge.

One of the first attempts to address this issue is a recent paper by Maliah et al. (2014). Building on the MA-STRIPS model and the MAFS approach discussed here, It suggests a mechanism for propagating information about landmarks among agents without revealing private information. Using this approach each agent can detect more landmarks than those detected on the projected problems, resulting in a more informative heuristic for each agent.

10. Summary

We presented a formulation of heuristic forward search for classical MA planning that respects the natural distributed structure of the system, preserving agent privacy. MAFS dominates the state-of-the-art distributed planners w.r.t. runtime and communication, as well as solution quality in most cases. In this class of privacy-preserving algorithms, MAD-A* is the first cost-optimal distributed planning algorithm, and it is competitive with its centralized counterpart, despite having partial information. We have studied the strengths and weaknesses of these methods, providing empirical evidence of our claims. Both algorithms were shown to be scalable, solving problems with up to 40 agents with high efficiency, especially in loosely-coupled domains. Our distributed planning framework MA-FD, provides researchers with a good starting point for future research into new algorithms and heuristics for the privacy-preserving MA setting.

There are many interesting directions for future research. Recently, the work presented here formed the basis for our work on mechanism design for privacy-preserving MA planning (Nissim & Brafman, 2013). Specifically, mad-a* is used as the underlying cost-optimal planner in a distributed implementation of the Vickerey-Clarke-Groves mechanism. This results in a distributed method for cost-optimal planning by *self-interested* agents with private information. It will be interesting to further explore methods for computing more accurate heuristics, and the inevitable trade-off between privacy, accurate heuristics and communication complexity. One could also explore how to modify our methods to deal with extensions to the ma-strips model (e.g. to include joint actions), and with different solution criteria (e.g. makespan). Finally, the notion of private/public actions can be refined to distinguish between read-write public actions and read-only ones. This distinction could have an effect on search methods and heuristics, and is an interesting avenue for future research.

Acknowledgments

The authors are grateful to the Associate Editor and the anonymous referees for many useful suggestions and corrections. This work was supported in part by ISF grant 1101/07 and the Lynn and William Frankel Center for Computer Science.

References

Albore, A., Palacios, H., & Geffner, H. (2009). A translation-based approach to contingent planning. In Boutilier, C. (Ed.), *IJCAI*, pp. 1623–1628.

Albore, A., Palacios, H., & Geffner, H. (2010). Compiling uncertainty away in non-deterministic conformant planning. In *ECAI*, pp. 465–470.

Alkhazraji, Y., Wehrle, M., Mattmüller, R., & Helmert, M. (2012). A stubborn set algorithm for optimal planning. In *ECAI*, pp. 891–892.

Amir, E., & Engelhardt, B. (2003). Factored planning. In *IJCAI*, pp. 929–935.

Bäckström, C. (1998). Computational aspects of reordering plans. *J. Artif. Intell. Res. (JAIR)*, *9*, 99–137.

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

Bonisoli, A., Gerevini, A., Saetti, A., & Serina, I. (2014). A privacy-preserving model for the multi-agent propositional planning problem. In *ICAPS'14 Workshop on Distributed and Multi-Agent Planning*.

Brafman, R. I., & Domshlak, C. (2006). Factored planning: How, when, and when not. In *AAAI*.

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

Brafman, R. I., Domshlak, C., Engel, Y., & Tennenholtz, M. (2009). Planning games. In *IJCAI*, pp. 73–78.

Brafman, R. I., Domshlak, C., Engel, Y., & Tennenholtz, M. (2010). Transferable utility planning games. In *AAAI*.

Burns, E., Lemons, S., Ruml, W., & Zhou, R. (2010). Best-first heuristic search for multicore machines. *J. Artif. Intell. Res. (JAIR)*, *39*, 689–743.

Chandy, K. M., & Lamport, L. (1985). Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, *3*(1), 63–75.

Chen, Y., & Yao, G. (2009). Completeness and optimality preserving reduction for planning. In *IJCAI*, pp. 1659–1664.

Clarke, E. M., Biere, A., Raimi, R., & Zhu, Y. (2001). Bounded model checking using satisfiability solving. *Formal Methods in System Design*, *19*(1), 7–34.

Coles, A. J., & Coles, A. (2010). Completeness-preserving pruning for optimal planning. In *ECAI*, pp. 965–966.

Conry, S. E., Kuwabara, K., Lesser, V. R., & Meyer, R. A. (1991). Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics*, *21*(6), 1462–1477.

Cox, J. S., & Durfee, E. H. (2005). An efficient algorithm for multiagent plan coordination. In *AAMAS*, pp. 828–835. ACM.

Cox, J. S., & Durfee, E. H. (2009). Efficient and distributable methods for solving the multiagent plan coordination problem. *Multiagent and Grid Systems*, *5*(4), 373–408.

Dwork, C. (2006). Differential privacy. In *ICALP (2)*, pp. 1–12.

Ephrati, E., & Rosenschein, J. S. (1994). Divide and conquer in multi-agent planning. In *AAAI*, pp. 375–380.

Ephrati, E., & Rosenschein, J. S. (1997). A heuristic technique for multi-agent planning. *Ann. Math. Artif. Intell.*, *20*(1-4), 13–67.

Fabre, E., & Jezequel, L. (2009). Distributed optimal planning: an approach by weighted automata calculus. In *CDC*, pp. 211–216. IEEE.

Fabre, E., Jezequel, L., Haslum, P., & Thiébaux, S. (2010). Cost-optimal factored planning: Promises and pitfalls. In *ICAPS*, pp. 65–72.

Franzin, M. S., Rossi, F., Freuder, E. C., & Wallace, R. J. (2004). Multi-agent constraint systems with preferences: Efficiency, solution quality, and privacy loss. *Computational Intelligence*, *20*(2), 264–286.

Greenstadt, R., Grosz, B. J., & Smith, M. D. (2007). SSDPOP: improving the privacy of DCOP with secret sharing. In *AAMAS*, p. 171.

Greenstadt, R., Pearce, J. P., & Tambe, M. (2006). Analysis of privacy loss in distributed constraint optimization. In *AAAI*, pp. 647–653.

Grinshpoun, T., & Tassa, T. (2014). A privacy-preserving algorithm for distributed constraint optimization. In *AAMAS'14*.

Gupta, D., Segal, A., Panda, A., Segev, G., Schapira, M., Feigenbaum, J., Rexford, J., & Shenker, S. (2012). A new approach to interdomain routing based on secure multi-party computation. In *HotNets*, pp. 37–42.

Hansen, E. A., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artif. Intell.*, *129*(1-2), 35–62.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In *AIPS*, pp. 140–149.

Helmert, M. (2006). The fast downward planning system. *J. Artif. Intell. Res. (JAIR)*, *26*, 191–246.

Helmert, M., & Geffner, H. (2008). Unifying the causal graph and additive heuristics. In *ICAPS*, pp. 140–147.

Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, pp. 176–183.

Helmert, M., & Röger, G. (2008). How good is almost perfect?. In *AAAI*, pp. 944–949.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: fast plan generation through heuristic search. *J. Artif. Int. Res.*, *14*(1), 253–302.

Holte, R. C. (2010). Common misconceptions concerning heuristic search. In *SOCS*.

ICAPS. The international planning competition. `http://www.plg.inf.uc3m.es/ipc2011-deterministic/`.

Katz, M., & Domshlak, C. (2008). Optimal additive composition of abstraction-based admissible heuristics. In *ICAPS*, pp. 174–181.

Kautz, H. A., & Selman, B. (1992). Planning as satisfiability. In *ECAI*, pp. 359–363.

Keyder, E., & Geffner, H. (2009). Soft goals can be compiled away. *J. Artif. Intell. Res. (JAIR)*, *36*, 547–556.

Kishimoto, A., Fukunaga, A. S., & Botea, A. (2009). Scalable, parallel best-first search for optimal sequential planning. In *ICAPS*.

Léauté, T., & Faltings, B. (2009). Privacy-preserving multi-agent constraint satisfaction. In *CSE (3)*, pp. 17–25.

Luis, N., & Borrajo, D. (2014). Plan merging by reuse for multi-agent planning. In *ICAPS'14 Workshop on Distributed and Multi-Agent Planning*.

Maheswaran, R. T., Pearce, J. P., Bowring, E., Varakantham, P., & Tambe, M. (2006). Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *Autonomous Agents and Multi-Agent Systems*, *13*(1), 27–60.

Maliah, S., Shani, G., & Stern, R. (2014). Privacy preserving landmark detection. In *ECAI'14*.

Meisels, A. (2007). *Distributed Search by Constrained Agents: Algorithms, Performance, Communication (Advanced Information and Knowledge Processing)*. Springer.

Méro, L. (1984). A heuristic search algorithm with modifiable estimate. *Artif. Intell.*, *23*(1), 13–27.

Nair, R., Tambe, M., Yokoo, M., Pynadath, D., & Marsella, S. (2003). Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *IJCAI*.

Nissim, R., Apsel, U., & Brafman, R. I. (2012). Tunneling and decomposition-based state reduction for optimal planning. In *ECAI*, pp. 624–629.

Nissim, R., & Brafman, R. I. (2013). Cost-optimal planning by self-interested agents. In *AAAI*.

Nissim, R., Brafman, R. I., & Domshlak, C. (2010). A general, fully distributed multi-agent planning algorithm. In *AAMAS*, pp. 1323–1330.

Nissim, R., Hoffmann, J., & Helmert, M. (2011). Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *IJCAI*, pp. 1983–1990.

Oliehoek, F. A., Witwicki, S. J., & Kaelbling, L. P. (2012). Influence-based abstraction for multiagent systems. In *AAAI*.

Palacios, H., & Geffner, H. (2009). Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Intell. Res. (JAIR)*, *35*, 623–675.

Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *IJCAI*, pp. 266–271.

Petcu, A., Faltings, B., & Parkes, D. C. (2008). M-DPOP: Faithful distributed implementation of efficient social choice problems. *J. Artif. Intell. Res. (JAIR)*, *32*, 705–755.

Ranjit, N., Varakantham, P., Tambe, M., & Yokoo, M. (2005). Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *AAAI*.

Silaghi, M.-C., & Mitra, D. (2004). Distributed constraint satisfaction and optimization with privacy enforcement. In *IAT*, pp. 531–535.

Srivastava, S., Immerman, N., Zilberstein, S., & Zhang, T. (2011). Directed search for generalized plans using classical planners. In *ICAPS*.

Steenhuisen, J. R., Witteveen, C., ter Mors, A., & Valk, J. (2006). Framework and complexity results for coordinating non-cooperative planning agents. In *MATES*, pp. 98–109.

Szer, D., Charpillet, F., & Zilberstein, S. (2005). MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *UAI*, pp. 576–590.

Taig, R., & Brafman, R. I. (2013). Compiling conformant probabilistic planning problems into classical planning. In *ICAPS*.

ter Mors, A., Valk, J., & Witteveen, C. (2004). Coordinating autonomous planners. In *IC-AI*, pp. 795–.

ter Mors, A., & Witteveen, C. (2005). Coordinating self interested autonomous planning agents. In *BNAIC*, pp. 383–384.

Torreño, A., Onaindia, E., & Sapena, O. (2012). An approach to multi-agent planning with incomplete information. In *ECAI*, pp. 762–767.

Torreño, A., Onaindia, E., & Sapena, O. (2014). Fmap: Distributed cooperative multi-agent planning. *Applied Intelligence*, *41*(2), 606–626.

Valmari, A. (1989). Stubborn sets for reduced state space generation. In *Applications and Theory of Petri Nets*, pp. 491–515.

Vrakas, D., Refanidis, I., & Vlahavas, I. P. (2001). Parallel planning via the distribution of operators. *J. Exp. Theor. Artif. Intell.*, *13*(3), 211–226.

Wehrle, M., & Helmert, M. (2012). About partial order reduction in planning and computer aided verification. In *ICAPS*.

Wehrle, M., Helmert, M., Alkhazraji, Y., & Mattmüller, R. (2013). The relative pruning power of strong stubborn sets and expansion core. In *ICAPS*.

Witwicki, S. J., & Durfee, E. H. (2010). Influence-based policy abstraction for weakly-coupled dec-pomdps. In *ICAPS*, pp. 185–192.

Witwicki, S. J., Oliehoek, F. A., & Kaelbling, L. P. (2012). Heuristic search of multiagent influence space. In *AAMAS*, pp. 973–980.

Xu, Y., Chen, Y., Lu, Q., & Huang, R. (2011). Theory and algorithms for partial order based reduction in planning. *CoRR*, *abs/1106.5427*.

Yang, Q., Wu, K., & Jiang, Y. (2007). Learning action models from plan examples using weighted MAX-SAT. *Artif. Intell.*, *171*(2-3), 107–143.

Yao, A. C.-C. (1982). Protocols for secure computations (extended abstract). In *FOCS*, pp. 160–164.

Yao, A. C.-C. (1986). How to generate and exchange secrets (extended abstract). In *FOCS*, pp. 162–167.

Yokoo, M., Durfee, E. H., Ishida, T., & Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Knowl. Data Eng.*, *10*(5), 673–685.

Yokoo, M., Suzuki, K., & Hirayama, K. (2002). Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*, pp. 387–401.

Yoon, S. W., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. In *ICAPS*, pp. 352–.