The 26th International Conference on Automated Planning and Scheduling

Proceedings of the 4th Workshop on

Distributed and Multi-Agent Planning (DMAP-16)

Edited by:

Antonín Komenda, Guy Shani

London, UK, 13-14/06/2016

Organising Committee

Antonín Komenda

Czech Technical University in Prague, Czech Republic

Guy Shani

Ben Gurion University, Israel

Roni Stern

Ben Gurion University, Israel

Dániel Laszlo Kovacs

Budapest University of Technology and Economics, Hungary

Christopher Amato

University of New Hampshire, USA

Program Committee

Christopher Amato (University of New Hampshire, USA)

Antonín Komenda (Czech Technical University in Prague, Czech Republic)

Dániel Laszlo Kovacs (Budapest University of Technology and Economics, Hungary)

Shlomi Maliah (Ben Gurion University, Israel)

Eva Onaindia (Universidad Politecnica de Valencia, Spain)

Guy Shani (Ben Gurion University, Israel)

Guni Sharon (Ben Gurion University, Israel)

Matthijs Spaan (Delft University of Technology, Netherlands)

Roni Stern (Ben Gurion University, Israel)

Table of Contents

Privacy Preserving LAMA

Shlomi Maliah and **Guy Shani** and **Roni Stern**

Ben Gurion University of the Negev, Israel

Abstract

In collaborative privacy preserving planning (CPPP), multiple agents collaborate to achieve a goal while keeping certain facts about the world private. A prominent approach in the development of CPPP algorithms is to use components from single agent planners and adapt them to preserve privacy. In this short paper, we show how the components of LAMA, arguably one of the most successful single-agent planners, can be used in a privacy preserving manner. These components include alternating between a landmark heuristic and an FF heuristic, preferred operators and deferred heuristic evaluation. We integrate the components into the Greedy Privacy Preserving Planner, a state-of-the-art CPPP algorithm. The resulting algorithm performs better than other CPPP algorithms from the recent Competition of Distributed and Multiagent Planners.

1 Introduction

Collaborative privacy preserving planning (CPPP) is a recently introduced setting in which multiple agents cooperate to achieve joint goals while concealing certain facts. As a motivating scenario, consider an army organization outsourcing its food supply to external caterers. Caterers unloads packaged food in logistics center and army trucks deliver the packages to the various bases. The army and the caterer must plan together to deliver appropriate amounts of food to the army bases, but the army may not wish to disclose to the caterers the location of its bases or the number of soldiers in each base.

Brafman and Domshlak (2013) proposed an attractive framework for such planning problems called multi-agent STRIPS (MA-STRIPS), which has attracted much attention in recent years (Tozicka, Jakubuv, and Komenda, 2015; Torreno, Sapena, and Onaindia, 2015; Štolba and Komenda, 2014; Maliah, Shani, and Stern, 2014). The first Competition of Distributed and Multiagent Planners (CoDMAP), held last year, already featured many planners from 10 different groups Štolba, Komenda, and Kovacs (2015). Many successful CPPP algorithms borrow or adapt algorithmic components from the single-agent planning literature. For example, privacy preserving versions have been proposed for popular single-agent heuristics such as landmarks (Maliah, Shani, and Stern, 2014; Torreno, Sapena, and Onaindia, 2015; Štolba, Fišer, and Komenda, 2015), FF (Štolba and Komenda, 2014), and pattern databases (Maliah, Shani, and Stern, 2015). In this short paper we propose a CPPP algorithm that successfully adapts and uses key components of LAMA (Richter and Westphal, 2010), a renowned single-agent planner.

LAMA is perhaps one of the most successful single-agent planners. It uses a forward heuristic search algorithm employing both a landmark-based heuristic (Richter, Helmert, and Westphal, 2008) and the FF delete-relaxation heuristic (Hoffmann, 2001), and alternates between them (Röger and Helmert, 2010). In addition, LAMA introduced preferred operators and deferred heuristic evaluation (Richter and Helmert, 2009), which are both common components in state-of-the-art single-agent planning algorithms.

The main contribution of this work is in the integration of these components into the Greedy Privacy Preserving Planner (GPPP) (Maliah, Shani, and Stern, 2014), a state-of-the-art CPPP algorithm. Experiments with the CoDMAP benchmarks show that the resulting algorithm, which we call PP-LAMA, outperforms all previous CPPP algorithms. Some LAMA components were already adapted to preserve privacy by prior work. Štolba and Komenda (2014) and Maliah, Shani, and Stern (2014) proposed a privacy preserving versions of the FF and landmark heuristics, and Torreno, Sapena, and Onaindia (2015) proposed alternating multiple heuristics. We explain how the other components of PP-LAMA operate in a way that preserves privacy. In particular, we highlight the importance of deferred heuristic evaluation, which is especially useful for reducing the collaborative effort in computing privacy preserving heuristics. Moreover, we improve on the landmark detection algorithm used by Maliah, Shani, and Stern (2014), showing a simple modification that allows finding substantially more landmarks.

2 Background

We now briefly describe the privacy preserving planning setting, the heuristics used by LAMA, and the GPPP algorithm.

2.1 Privacy Preserving Planning

An MA-STRIPS problem (Brafman and Domshlak, 2013) is represented by a tuple $\langle P, \{A_i\}_{i=1}^{k}, I, G\rangle$ where k is the number of agents, P is a finite set of facts (can be true

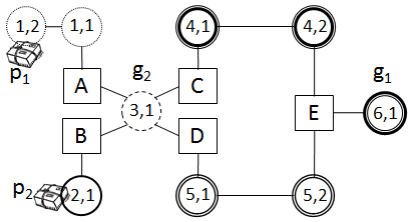


Figure 1: A logistics example.

of false), A_i is the set of actions agent i can perform, I is the start state, and G is the goal condition. Each action $a = \langle pre(a), eff(a) \rangle$ is defined by its preconditions ($pre(a)$), and effects ($eff(a)$). Preconditions and effects are logical formulas over P. A state is a conjunction of facts in P (true or false). The goal G is also a conjunction of facts. A solution is a *plan* that achieves G, i.e., a sequence of actions transforming the initial state (I) to a state that satisfies G.

Privacy-preserving MA-STRIPS extends MA-STRIPS by defining sets of variables and actions as private, known only to a single agent. Formally, a privacy-preserving MA-STRIPS problem defines a set of facts as public, denoted $public(P)$, and a set of actions as public, where $public_i(A) \subset A_i$ are the public actions of agent i. It is assumed that when a public action is executed, all agents are aware of the execution, and view the public effects of the action. A privacy-preserving MA-STRIPS problem also defines for each agent i a set of facts and actions as private, denoted by $private_i(P)$ and $private_i(A)$, respectively. The identity of the private facts and actions must not be revealed to the other agent during planning and during execution. The public and private facts of each agent are disjoint, i.e., $private_i(P) \cap public(P)\emptyset$, and the union of the public facts and all private facts form the entire set of facts in the underlying MA-STRIPS problem, i.e., $\bigcup_{i=1}^{k} private_i(P) \cup public(P) = P$.

For ease of exposition we assume that all goals are public. A solution to a privacy-preserving MA-STRIPS problem, is a sequence of public and private actions. We say that the sequence of public actions in such a solution is a high-level, or public, plan that must be extended to a full plan using private actions of various agents. A high-level plan is said to be *valid* if each agent can plan independently to achieve the private preconditions on the public actions it executes in the high level plan Maliah, Shani, and Stern (2014).

Figure 2.1 illustrates a simple logistic example in which the agents are trucks tasked with delivering packages. The set of facts P represents the location of the two packages and six trucks. Each truck has three actions: move, load, and unload, corresponding to moving the truck between locations, loading a package and unloading it. Each truck is owned by a different company and may have different areas of operation, and no company wants to share its location and coverage (which locations it can reach) with other companies. Thus, all the facts representing the location of trucks are private, while the facts representing whether a package is at a logistic center are public. Only the load/unload actions at the logistic centers are public, while the move actions are private for each agent, as well as loading and unloading packages at private locations.

2.2 Landmark Heuristics

Landmarks, as used in LAMA, are propositional formulas that must be hold at some point during the execution of every successful plan (Richter, Helmert, and Westphal, 2008). A landmark heuristic function evaluates a state by considering the number of landmarks that still needs to be achieved. A popular method for identifying landmarks searches backwards from the goal. At each phase, one landmark is selected for development. All actions that can satisfy this landmark are identified, and a new landmark is constructed from their preconditions. When all these actions share a single fact, that fact is identified as a new landmark. Otherwise, a disjunctive formula is created from some facts that appear in the actions preconditions. When developing a landmark, all actions that take facts in this landmark as preconditions are ignored, in order to avoid circular reasoning.

2.3 The Fast Forward (FF) Heuristic

The FF heuristic begins by computing the relaxed planning graph, where facts are organized into layers, and all edges are between consecutive layers (Hoffmann, 2001). The first layer contains all the facts that hold at the current search state. To construct the next layer the FF heuristic considers the actions that can be executed at the current state, i.e., the actions whose preconditions are satisfied by the facts in the current layer. The next layer in the relaxed planning graph consists all the facts in the previous layer and all the facts that are effects of these set of actions. Importantly, *delete effects*, i.e., effects that remove facts from the state, are ignored. A result of ignoring delete effects is that once an action has participated in the construction of a layer, there is no need to consider it in the construction of future layers. For each fact p, the FF heuristic maintains the action a_p that has achieved it for the first time. Then, it constructs an edge between the preconditions of a_p on the previous level and p. After the graph construction, FF computes a plan over the relaxed graph, starting from the goal and moving backwards: first considering the goal facts and adding all the actions that achieved them to the plan. Then considering the preconditions of these actions, and so forth, ignoring repeated facts.

2.4 GPPP

The GPPP algorithm (Maliah, Shani, and Stern, 2016) is a CPPP algorithm based on heuristic forward search. The first phase in GPPP is the *high-level planning* phase, in which the agents collaboratively perform a best-first search on a relaxed version of the CPPP problem in which only public actions are used. To maintain privacy, states in this search are represented by the set of public facts that hold for that state, and a set of private state identifiers, one for each agent. Every agent can map its private identifiers to a set of private facts. When a state s is expanded, each agent generates new states by applying the public actions it can execute in s. In addition to the effects of the executed public action, the

generated state also includes all private facts that the corresponding agent can achieve by applying private actions and using delete relaxation (Hoffmann, 2001). The high-level planning phase ends when a high-level public plan has been found that enables achieving all goals. Then, all agents compute private plans to achieve the preconditions of the public actions in the high level public plan. If some agent cannot achieve the preconditions of one of its actions in the high-level plan then this second phase fails then the high level planning phase resumes.

Maliah, Shani, and Stern (2014) show that GPPP's efficiency depends on the heuristic function used to guide the high-level search and that effective heuristics can be computed by a collaborative effort of the agents. For example, collaboratively computing a landmark heuristic requires each agent to reports the number of landmarks satisfied in a state. Using such heuristic allowed GPPP to show impressive performance, but the need to collaborate for evaluating the heuristic of each expanded state slows down the search process considerably. This is especially problematic for the FF heuristic, which requires agents to collaboratively find a relaxed plan. Our PP-LAMA algorithm builds upon GPPP and is especially suited to address this limitation by borrowing from LAMA the preferred operators and deferred heuristic evaluation techniques (Section 3.3).

3 Privacy Preserving LAMA

Algorithm 1: The PP-LAMA algorithm

```
1  PP-LAMA()
2      Init all open lists to hold the initial state
3      c_LM ← 0; c_FF ← 0; c_pLM ← 0; c_pFF ← 0;
4      while some open list is not empty do
5          active-list← ChooseList(c_LM,c_FF,c_PLM,c_pFF)
6          s ← best state in active-list
7          Remove s from all open lists
8          if s ⊨ G then
9              P_pub ← the public plan to achieve s
10             P_full ← private extensions for P_pub
11             if P_full is valid then
12                 return P_full
13         Find achievable private facts in s
14         Compute all heuristic values for s
15         if s was generated by a PO and h(s) is the lowest
           so far for either FF or LM then
16             c_pLM←c_pLM + 1000; c_pFF←c_pFF + 1000
17         else
18             c_LM ← c_LM + 1; c_FF ← c_FF + 1
19         foreach public action a_pub applicable in s do
20             s' ← apply(s, a_pub)
21             if a_pub is a preferred operator for s then
22                 Add s' into all open lists w. h(s)
23             else
24                 Add s' into FF and LM open lists w. h(s)
```

Next, we present PP-LAMA, starting by describing its different components.

3.1 Privacy Preserving Landmark Detection

We use the landmark identification process of Maliah, Shani, and Stern (2014), where agents collaborate in identifying landmarks, and augment it with an improved detection of private landmarks. The process begins by adding each fact (or disjunction of facts) in the goal as a landmarks. Then, the agents agree on a landmark and *develop* it, potentially adding more landmarks. This process continues until there are no more landmarks to develop. *Developing* a landmark ϕ means checking which actions can achieve ϕ, and then considering the preconditions of these actions as additional landmarks. To identify which actions achieve ϕ, the agents first check which facts they can achieve without requiring any facts in ϕ. This is done effectively by ignoring the delete effects of agents' actions (i.e., using a delete-relaxation), applying them iteratively (starting from the initial state) and sharing the achieved public facts between the agents. Then, each agent considers which actions it can perform to achieve facts in ϕ given this set of achieved facts. The preconditions of these actions form a new landmark.

A landmark ϕ can be a disjunction of public and private facts: $\phi = \textit{public}(\phi) \vee \phi_1, \ldots, \phi_k$, where $\textit{public}(\phi)$ is the public facts in ϕ and ϕ_i are the private facts of agent $i \in [1, \ldots, k]$ in ϕ. Each agent develops its private facts in ϕ and all agents develop together the public facts. Formally, if $d(\phi)$ denoted developing a landmark ϕ then, $d(\phi) = d(\textit{public}(\phi)) \vee d(\phi_1) \vee \ldots d(\phi_k)$. To preserve privacy, only the public facts are published to the other agents. For private facts, all agents agree on a unique ID for this landmark, and each agent maintains a mapping from this ID to its own set of private facts that participate in this landmark. Our landmark identification process improves on that of Maliah, Shani, and Stern (2014) in that it allows detecting and developing landmarks that contain private facts of multiple agents. Experimentally, this improvement allows finding more than twice as many landmarks in some domains from the CoDMAP centralized planning track Štolba, Komenda, and Kovacs (2015).

The detected landmarks are used to evaluate a state during planning. Each agent publishes which landmarks it identified as satisfied in the state. The number of landmarks that at least one agent can satisfy is used as the landmarks heuristic estimate for that state. We refer to this heuristic as LM.

3.2 Privacy Preserving FF

We use the privacy preserving FF heuristic computation suggested by Štolba, Fišer, and Komenda (2015). The method constructs the relaxed planning graph jointly, with each agent maintaining a part of the graph. In every level of the relaxed planning graph, each agent computes its own set of achievable facts. Then, the public facts in that level are published to the other agents which then inserts these facts into their local current layer. As in regular FF, when a public fact p is achieved for the first time we maintain the action a_p that achieved it, and draw an edge between the preconditions of a_p on the previous layer and p. If a public fact has

Domain	GPPP	MAPR -p	PMR	MAPlan/ FF+DTG	PSM-VRD	PP-LAMA
blocksworld	12	**20**	**20**	**20**	**20**	**20**
depot	11	0	0	13	17	**18**
driverlog	14	**20**	19	17	**20**	**20**
elevators	**20**	19	19	11	12	**20**
logistics	**20**	19	0	18	18	**20**
rovers	19	19	**20**	**20**	12	**20**
satellites	18	**20**	19	**20**	18	**20**
sokoban	9	0	6	**18**	**18**	12
taxi	**20**	0	19	**20**	0	**20**
wireless	3	2	**7**	4	0	4
woodworking	18	0	0	16	**19**	**19**
zenotravel	**20**	**20**	18	**20**	13	**20**
total	184	139	147	197	167	**213**

	FF		LM		FF+PO+DH		FF+LM		PP-LAMA	
Domain	C	T	C	T	C	T	C	T	C	T
blocksworld	18	49.5	12	35.5	18	19.2	20	49.4	20	12
depot	3	25	11	18.6	9	16.6	10	22.5	18	0.6
driverlog	14	155	14	203.4	20	2.8	17	26.8	20	2.7
elevators	20	148	20	25	20	4.4	20	149	20	4.2
logistics	20	7.8	20	2.5	20	0.8	20	7.9	20	0.8
rovers	14	217.2	19	156.7	20	0.8	19	222.9	20	0.8
satellites	16	266.2	18	89.5	20	2.5	17	260.1	20	2.3
sokoban	9	23.3	9	82.5	10	76	11	24.9	12	27
taxi	20	3.4	20	4.4	20	1.1	20	3.1	20	1
wireless	2	0.7	3	0.4	2	0.4	3	0.7	4	0.4
woodworking	5	2.3	18	0.5	16	1.2	15	1.3	19	0.4
zenotravel	20	165.8	20	65.5	20	4.1	20	164.8	20	2.1
total	161		184		195		192		213	

Table 1: (left) coverage results over the CoDMAP instances. (right) Impact of the various PP-LAMA components.

been achieved for the first time by several agents at the same level, one agent is considered to be responsible for achieving it, and the fact is labeled by the public action that this agent has used to achieve the fact.

The construction of the relaxed plan is also done collaboratively, where each agent computes the part of the plan that contains its own actions, starting from the goal and moving backwards. If an action in the plan requires a precondition fact that was generated by another agent then that agent is notified to continue constructing the plan to achieve that precondition. When the plan construction phase has terminated, all agents report the number of actions in their portion of the relaxed plan. The sum of these counts is the FF heuristic estimate for that state.

3.3 Preferred Operators

A preferred operator (PO) is an action that is deemed to be valuable at a given state. These are computed differently for the FF heuristic and for the LM heuristic. For FF, the POs are actions that achieve at least one precondition of an action in the relaxed plan. For LM, the POs are actions that appear prior to the first landmark that the relaxed plan achieves.

LAMA emphasizes POs in domains where they are useful as follows. When a state is generated by a PO, it is added to the regular open list and to an additional open list that contains only states generated by POs. We will refer to the latter open list as the *preferred list*. The search alternates between the regular open list and the preferred list, thus giving priority to states in the preferred list. Moreover, the priority of the preferred lists is boosted whenever a state generated by a PO has the best heuristic value so far when it is expanded. This boosting is embodied by having the next 1000 states to be expanded only from the preferred lists. If several heuristics are used then each heuristic has an open list and a preferred list. Thus, in LAMA as well as our PP-LAMA, we used four priority queues, two for FF and two for LM. In PP-LAMA, we followed this exact use of POs, except that only public actions are considered: a PO for FF is a *public* action that achieves a precondition of a public action in the relaxed plan, and similarly a PO for LM is any public action that is on the relaxed plan for achieving the first landmark.

The PP-LAMA algorithm is listed in Algorithm 1. The best-first search chooses which open list to use at each step according to their priorities (given by the counters c_{LM},c_{FF},c_{pLM}, and c_{pFF}), alternating between open lists with equal priority (line 4). As in LAMA, boosting the preferred lists is done whenever a state generated by a PO has a heuristic value that is the lowest seen so far (line 16). As in GPPP, when a public plan to the goal is found all agents try to extend it to a full plan 10.

Deferred State Evaluation. An important aspect of LAMA that is incorporated in PP-LAMA, and for the first time in a CPPP algorithm, is the *deferred heuristic evaluation* technique. When a state s' is inserted into the open lists, its associated heuristic value is that of its parent (lines 22 and 24). Computing the heuristic functions for s' is deferred until it is expanded (line 14). This deferred heuristic evaluation technique takes advantage of the prioritization obtain by the preferred lists, designed to reduce the number of costly heuristic computation. This is especially useful for improving GPPP, where the heuristic computation requires costly collaboration. Another process that is time consuming during state generation in GPPP is the computation of all achieveable private facts, which is done via delete relaxation. To resolve this, we also defer this process to the time when the state is expanded (line 13).

4 Empirical Evaluation

We evaluate the performance of PP-LAMA over all benchmarks, comparing PP-LAMA to all relevant algorithms from the centralized track of the CoDMAP competition Štolba, Komenda, and Kovacs (2015). All experiments were run on a 2.67 GHz machine with 32GB of memory. Table 1(left) shows how many problem instances each algorithm solved under a 30 minutes timeout (AKA "coverage"). In every domain the best performing algorithm is marked in bold. PP-LAMA solves 16 problems more than its best competitor and in all but two domains it is the best performing planner.

Next, we analyzed the impact of PP-LAMA's different components on planning performance. Table 1(right) presents coverage and runtime results (over problems solved by all variants) when using only the FF, only LM, FF with POs and deferred heuristics (denoted FF+PO+DH), both the FF and the landmark heuristics using alternating lists without deferred heuristics (FF+LM), and the complete PP-LAMA. The computation of the POs for LM requires the FF relaxed planning graph, and was hence not evaluated sepa-

rately. Without POs, FF produces the poorest results, significantly worse than LM. This is because landmarks are faster to compute, as they are identified only once, and then the agents only need to evaluate which landmarks are satisfied in the current state, while FF needs to compute a relaxed plan in every state. The combination of both landmarks and FF is somewhat better than each heuristic alone, but the largest gain comes from applying PO and deferred heuristic evaluation. PP-LAMA (FF+LM+PO+DH) is very fast even though it uses the FF heuristic, because the heuristics are computed only for states that are removed from the open list, and not for all generated states.

5 Conclusion

We propose an adaptation of the renowned LAMA classical planner to CPPP. The resulting algorithm, called PP-LAMA, is built on GPPP and includes privacy preserving versions of the FF heuristic, landmark heuristic, preferred operators and deferred heuristic evaluation. PP-LAMA outperforms all relevant planners from the CoDMAP competition. In the future, we would extend our approach with additional heuristics, as was done for the classical LAMA.

Acknowledgments: We thank the reviewers for their useful comments. This work was supported by ISF Grant 933/13, by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev.

References

Brafman, R. I., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence* 198:52–71.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI magazine* 22(3):57.

Maliah, S.; Shani, G.; and Stern, R. 2014. Privacy preserving landmark detection. In *the European Conference on Artificial Intelligence (ECAI)*, 597–602.

Maliah, S.; Shani, G.; and Stern, R. 2015. Privacy preserving pattern databases. In *ICAPS workshop on Distributed and Multi-Agent Planning (DMAP)*.

Maliah, S.; Shani, G.; and Stern, R. 2016. Collaborative privacy preserving multi-agent planning. *Autonomous Agents and Multi-Agent Systems* 1–38.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *ICAPS*, 273–280.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39(1):127–177.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, volume 8, 975–982.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *ICAPS*, 246–249.

Štolba, M., and Komenda, A. 2014. Relaxation heuristics for multiagent planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Štolba, M.; Fišer, D.; and Komenda, A. 2015. Admissible landmark heuristic for multi-agent planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Štolba, M.; Komenda, A.; and Kovacs, D. L. 2015. Competition of distributed and multiagent planners (codmap). *The International Planning Competition (WIPC-15)* 24.

Torreno, A.; Sapena, O.; and Onaindia, E. 2015. Global heuristics for distributed cooperative multi-agent planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Tozicka, J.; Jakubuv, J.; and Komenda, A. 2015. On internally dependent public actions in multiagent planning. In *ICAPS workshop on Distributed and Multi-Agent Planning (DMAP)*.

Factored Monte-Carlo Tree Search for Coordinating UAVs in Disaster Response

Chris A. B. Baker, Sarvapali Ramchurn, W. T. Luke Teacy, Nicholas R. Jennings
Agents Interactions and Complexity Group, University of Southampton
Southampton SO17 1BJ, United Kingdom
{cabb1g08, sdr1, wtlt}@soton.ac.uk, n.jennings@imperial.ac.uk

Abstract

The coordination of multiple Unmanned Aerial Vehicles (UAVs) to carry out surveys is a major challenge for emergency responders. In particular, UAVs have to fly over kilometre-scale areas while trying to discover casualties as quickly as possible. However, an increase in the availability of real-time data about a disaster from sources such as crowd reports or satellites presents a valuable source of information to drive the planning of UAV flight paths over a space in order to discover people who are in danger. Nevertheless challenges remain when planning over the very large action spaces that result. To this end, we introduce the survivor discovery problem and present as our solution, the first example of a factored coordinated Monte Carlo tree search algorithm to perform decentralised path planning for multiple coordinated UAVs. Our evaluation against standard benchmarks show that our algorithm, Co-MCTS, is able to find more casualties faster than standard approaches by 10% or more on simulations with real-world data from the 2010 Haiti earthquake.

Introduction

The increased prevalence of low-cost, robust, commercially available Unmanned Aerial Vehicles (UAVs) has led to concerted efforts to utilise these platforms in disaster response. Specifically, UAVs have seen widespread use in recent disasters—such as the 2010 Haiti earthquake and the 2015 Nepal earthquake—aiding first responders with collecting imagery and other sensory data without putting human lives at risk (Adams and Friedland 2012; Goda et al. 2015; Meier 2015). In particular, an important body of work has focused on developing UAVs that act as *autonomous systems* to minimise the involvement of overstretched first responders: both to conserve valuable manpower and to ensure their safety (Crisis Mappers 2013; Murphy 2012; United Nations Foundation 2011). Key to this work, is the idea of enabling coordinated UAVs to explore a disaster space to discover the spatial location of casualties: a difficult task given the extent and large number of possible locations to visit.

To enable this exploration, advances in data collection—and specifically crowd-sourcing (Morrow et al. 2011; Goodchild and Glennon 2010)—have created new sources of information about disaster scenarios that contribute to increased awareness of the situation on the ground during a disaster. Such information is vital given the size and scale of the effects of natural disasters (Fowler 2016). For instance, during 2010 a magnitude 7.3 earthquake struck Haiti near the capital, Port-Au-Prince, which caused widespread destruction over thirty kilometres from the epicentre and resulted in the destruction or damage of over three hundred thousand homes; the death or injury of over five hundred and twenty thousand people; and around one point three million displaced persons needing temporary shelter (Government of the Republic of Haiti 2014). A natural extension of the use of UAVs in these events is to fully exploit prior data on the extent and nature of the disaster (whether crowd-sourced or otherwise) in order to maximise the likelihood of rescuing or discovering casualties quickly. However, at present there is no specific work that seeks to use spatial information on the distribution of *people* and the expected *danger*—for instance the likely rate of fatalities due to structural damage to buildings or from the spread of radiation—to inform the paths of UAVs through a disaster space, in order to maximise the number of observations made of possible casualties.

Currently, the state of the art for UAV path planning algorithms focuses on three main areas, the first of which is target tracking for surveillance (Bernardini, Fox, and Long 2014; Hu et al. 2014; Kolling and Kleiner 2013). Now, although these techniques are related to the exploration of a disaster space, they are designed to find a known number of targets that are in motion, rather than an unknown number of survivors distributed over an area. Other developments in path planning focus on trying to reach a set goal location (or state) (Chen et al. 2014; Durkota and Komenda 2013; He 2007; Kothari, Postlethwaite, and Gu 2009) or working with single autonomous UAVs (Cashmore et al. 2014; Kothari and Postlethwaite 2012); neither of which fulfil the need for algorithms that coordinate multiple vehicles in an *explorative* traversal of the disaster space, rather than aiming for a particular final location. Specific challenges exist in coordinating multiple vehicles. For example, there is often no benefit to multiple UAVs providing imagery of the same location: there must be coordination between the vehicles to allow them to find survivors in a disaster, without all attending the same locations.

In line with the terminology used in this field (Bry and Roy (2011), Gan and Sukkarieh (2011), Waharte, Trigoni, and Julier (2009) and others) we henceforth refer to data

distributed over a spatial representation of a disaster area as a *belief map*. This belief map can come from different sources, including crowd-sourcing, but we assume it has the characteristics of mapping spatial locations onto some function that represents numerical data: for instance number of people or radiation levels.

Our work seeks to address these challenges with the following three contributions:

1. We introduce a formulation of the *survivor discovery* problem, informed by several datasets collected following natural disasters;
2. We develop a decentralised algorithm that allows multiple UAVs to coordinate the exploration of a large disaster space with many states;
3. We test and evaluate our approach on real-world data simulating a very large action space, showing consistent gains in survivor discovery of over 10% compared to benchmarks.

The rest of the paper is organised as follows. First we discuss the background of using UAVs in disaster response and the types of problems that existing research has focussed on; next we describe the specifics of the problem we seek to solve by formulating the survivor discovery problem; we then detail our use of a coordinated Monte-Carlo tree-search algorithm (including examples); before showing the benefits of our implementation with three experimental scenarios using real-world datasets; and finally summarising our work and proposing an extension of the algorithm and a removal of some assumptions.

Background

In order to best use UAVs to aid responders in disasters they must be able to plan paths autonomously, as a group. Furthermore, as we have already indicated, it is beneficial to use prior information about the area to inform the flight paths of UAVs in order to maximise the likelihood of discovering survivors. Currently, work on path planning in robotics focusses primarily on reaching goal locations and frequently formulates path planning as a control problem (Goerzen, Kong, and Mettler 2009). Conversely, in a disaster scenario there need not be any final end-point to a UAV's path planning; rather the length of the exploration may be constrained by—for example—battery life, and the number of people to be discovered must be maximised over the length of the path. Alternatively, much work has also been done to enable the use of vision algorithms and belief data to track mobile targets or map an area (Liu and Dai 2010). However, this area of research often focusses on locating a known number of targets, or covering a bounded space for the purposes of mapping. In contrast, the task of searching for casualties in disaster response can be summarised as the localisation of an *unknown* number of people as quickly as possible (Fawcett and Oliveira 2000; Chiu et al. 2002), in order to ensure quick rescue or attention by emergency services and thus the greatest chance of survival (Macintyre, Barbera, and Petinaux 2011). Consequently techniques for known numbers of targets or for mapping environments are not useful.

Against this background, we find closer similarities with work on solving Markov Decision Processes (MDPs); specifically where locality of UAVs can be used to reduce calculation overheads. In particular, the generality of the MDP formulation lends itself well to the construction of a simulation environment given numerical data used for a belief map, as well as having a number of well-established solutions. Specifically, work by Amato and Oliehoek (2015) utilises factored tree-searches for partially observable MDP solutions, and demonstrates performance benefits over state of the art un-factored solver POMCP (Silver and Veness 2010). This performance advantage is obtained by exploiting problem structure to allow factorisation in a way reflective of local state spaces and interactions. In a similar way, we use factored trees in this paper to represent the available actions of UAVs in a disaster environment, factoring the value of locating people between UAVs within spatial proximity of each other. Notably—because of the dimensions of the physical area we consider—we deal with a very large state-space, which must be particularly carefully sampled since complete searches are computationally intractable. We also include—for the first time—real data from a disaster scenario to validate our model.

We next introduce the specific formulation of the problem we tackle, and explain the origin of the large state-space.

The Survivor Discovery Problem

In this section to formulate the problem of finding a number of survivors in a disaster-area, with a team of cooperative UAVs planning their actions and coordinating with each other if required, in order to maximise the number of people discovered. All of this is to be done in the presence of a model of the danger to the casualties expressed as an estimated fatality rate mapped to spatial locations. As such the problem is one of prioritising visits to areas of high expected numbers of people, while also attending areas with high expected fatality rates as quickly as possible. We now describe; in turn; the environment model, the behaviour of UAVs in the simulation, and the specific problem of discovering survivors. Following this we introduce the Bellman equation (used to characterise MDPs) for our scenario.

Environment Model

We begin by discretising the search-space to allow for fast plan creation. Specifically, we formulate the survivor discovery problem as exploring a uniform $x \times y$ sized grid world—C—formed of cells,[1] $c_{ij} \in C$. Each cell c_{ij} contains an *unknown* number of people, $p_{ij} \in \mathbb{Z}^*$, and a scalar value

$$d_{ij} \in [0, 1] \tag{1}$$

denoting the *danger* in the cell as the probability of *any* person in the cell dying during the next time step. Time steps are denoted by an integer value $t \in \mathbb{Z}^*$, with subsequent steps referred to by adding integer values; for example $t+1$. If a value depends on time, this is denoted in parentheses; i.e. $p_{ij}\,(t)$.

[1] Cells are indexed for their horizontal and vertical position respectively by ij.

UAV Behaviour Formulation

The area C is explored by the set of UAVs $U = \{u_1, \ldots, u_m\}$ that traverse C from cell to cell once per timestep. Each UAV is equipped with sensors capable of accurately detecting people inside one cell at any given timestep: i.e. they return the value of p_{ij} deterministically (this is a simplifying assumption we will address in further work). Each UAV, u_k, must select its own trajectory $T_k \subseteq C$ with contiguous borders, through the area, forming the set of all UAV trajectories $\mathbf{T} = \{T_1, \ldots, T_m\}$. To address double-counting the values in each cell,[2] we introduce the union of all the trajectories in the set $\mathbf{T}$ as $\mathcal{T}(\mathbf{T}) = \bigcup_{T_k \in \mathbf{T}} T_k$. Additionally, we impose constraints on the length of each UAV's planned trajectory to account for limits on battery life. If the maximum number of cells that can be traversed due to the battery limitations of a UAV u_k is denoted $b_k \in \mathbb{Z}^+$, then the maximum trajectory length is simply $\mid T_k \mid \leq b_k \forall k \in \{1, \ldots, m\}$; which we denote f.

The action vectors enabling the UAVs to transition from cell to cell are defined as $a_k = (\leftarrow, \rightarrow, \uparrow, \downarrow)$ for each UAV u_k, (each arrow representing the direction of motion: we do not consider the UAVs remaining stationary as according to our model this can never result in more reward than moving). We impose the constraint that the available actions are restricted to UAVs at the edge of the grid world (i.e. explicitly where the UAV occupies a cell c_{ij} where $i = 0$ or x, or $j = 0$ or y) so that they do not have the action available to cross out of the grid area. At any given time, the vector denoting all possible UAV actions is given by $a = (a_1, \ldots, a_m)$, and represents all possible combinations of movement available to all UAVs. The set of all actions available to all UAVs at any arbitrary time forms the *total* action space A, which we now incorporate into the standard Bellman equation.

Exploration Problem Formulation

Having described the environment and UAV behaviour, we formulate the multi-UAV exploration problem as a Markov Decision Process (MDP), comprising a tuple $\langle S, A, R, P \rangle$ of states S, actions A, rewards R, and transition probabilities P, which we define below. Since the UAVs are not aware in advance of the ground-truth values of p_{ij} and d_{ij} in each cell, computations are instead made using prior belief-data about the expected number of people $\bar{p}_{ij} \in \mathbb{R}^*$ and the expectation value of the probability of death $\bar{d}_{ij} \in [0, 1]$ of each person in a cell in a given time step. We also consider a binary variable $v_{ij} \in \{0, 1\}$ denoting the *visibility* of the cell: i.e. whether it has been observed by a UAV. This allows us to construct the tuple s_{ij} to denote the state of a cell c_{ij}:

$$s_{ij} = \langle \bar{p}_{ij}, \bar{d}_{ij}, v_{ij,} \rangle$$

The set of these for all cells in C forms the global state variable $s = \{s_{00}, \ldots, s_{xy}\}$. With this constructed, we next formulate an update procedure for the expected value of the number of people in a given cell after a time step t by computing the product of the current expected number of people and the probability of survival:

$$\bar{p}_{ij}(t+1) = \bar{p}_{ij}(t)(1 - \bar{d}_{ij}) \tag{2}$$

Here, $\bar{d}_{ij}$ for the next time step is dependent on whether a cell has been observed. If so, we consider the danger to reduce to zero, since first responders are now be aware of the need to rescue the people occupying that cell:

$$\bar{d}_{ij}(t+1) = \begin{cases} 0 & if\ v_{ij}\,(t) = 1 \\ \bar{d}_{ij}\,(t) & otherwise \end{cases}$$

This assumption need not hold in general. However, it is a convenient way of indicating that no further utility can be derived from revisiting a cell once it has been observed. In a scenario with trapped survivors it is reasonable to assume there will be no large-scale movement of population during the search. Indeed, in many cases victims are often recovered after being trapped in the same location for days at a time (Macintyre, Barbera, and Petinaux 2011).

We note the logical extension of equation 2 to times at an arbitrary point in the future t', as required for planning future actions:

$$\bar{p}_{ij}\,(t') = \bar{p}_{ij}\,(t)\left(1 - \bar{d}_{ij}\right)^{t'-t} \tag{3}$$

We record the set of positions of each UAV $u_k \in U$ at time t as $g_k\,(t) \in C$, which we denote as members of the vector of all UAV positions:

$$g(t) = (g_1(t), \ldots, g_m(t))$$

where the indices on each g correspond to the indices of the UAV at that location. Additionally, we record the set of unique UAV locations as the union of the elements of g as:

$$G\,(t) = \bigcup_{k=1}^{m} \{g_k\,(t)\}$$

Thus, the state of the map at a time t is a tuple comprising the state of each cell, and the position of each UAV: $\tilde{s}\,(t) = \langle s\,(t)\,, g\,(t) \rangle$.

As a result, we formulate the immediate reward to all UAVs at timestep t as a function of the expected number of people saved due to UAV observation. Specifically, this is the product of the expected number of people in a cell multiplied by the death rate in that cell, summed over all unique cells where a UAV is present (i.e. all members of the set G):

$$R\,(t) = \sum_{G(t)} \bar{p}_{ij}(t) \times \bar{d}_{ij}$$

Over an infinite time horizon, we consider the sum of expected rewards for each time step:

$$\sum_{t=0}^{\infty} R\,(t) = \sum_{t=0}^{\infty} \sum_{c_{ij} \in G(t)} \bar{p}_{ij}(t) \times \bar{d}_{ij}$$

Since G is itself dependent on the trajectory of each UAV (i.e. the cell each UAV occupies at any time t) this can be said to be equivalent to:

$$\sum_{t=0}^{\infty} R\,(t) = \sum_{c_{ij}(t) \in \mathcal{T}(\mathbf{T})} \bar{p}_{ij}(t) \times \bar{d}_{ij}$$

[2] Specifically, we seek to avoid repeated observation of the same high-value cells by different UAVs. Thus, by taking the union of trajectories, we ensure only unique observations contribute to the utility function.

Thus, R must be maximised over the trajectory of all UAVs in order to observe (and subsequently 'save') the maximum number of people. The key challenge is to allow the path planning to be coordinated between UAVs to maximise global, rather than local, reward (shown explicitly by the inclusion of G in Equation). We briefly discuss the implications of the UAVs detecting p_{ij} with certainty, and justify this assumption, at the end of this section.

Bellman Equation

Having formulated the environment, the UAVs, and the discovery problem; we finally formulate the standard MDP Bellman optimality equation, where we denote R's dependence on the actions and state space (which themselves depend on t):

$$Q(a,s) = R\left(a,s\right) + \sum_{s'} P\left(s' \mid a,s\right) \max_{a'} Q\left(a',s'\right)$$

In our case, we do not require the diminishing-returns term γ to ensure $Q\left(a,s\right)$ is finite, since the diminishing value of examining a cell further in the future is expressed in the reduction of the expected number of people as a result of the death-rate term (see Equation 2); which is incorporated in the reward function. In other words, $Q\left(a,s\right)$ cannot exceed the number of people alive in the space, given s.

We also do not require the explicit sum over various states since the transition probability P between states given a fixed action is (as discussed below) deterministic. While this simplifies the form of the action value function, we still require the result be optimal according to:

$$Q(a,s) = R\left(a,s\right) + \max_{a'} Q'\left(a',s'\right) \tag{4}$$

from which we obtain the optimal action: $a = \arg\max_a Q$ to maximise current and future reward. While ostensibly a simplification of a typical MDP formulation, the problem is far from trivial as the joint action space a at any time step grows as an exponent of the number m of UAVs in the system, namely: $\|a\| \propto a_k^m$, while the possible combinations of G grow proportional to:

$$\mid C \mid! = (x \times y)! \tag{5}$$

i.e. a combinatorial function of the dimensions of the environment. The result is a state space S of extremely large size (we give a specific example in our Results section below) even before accounting for the permutations of p, d, and v.

We now briefly discuss the implications of the discoveries in the environment happening deterministically and how this does not affect our planning.

Determinism in our Model

With regards to the rewards obtained in our formulation, we note here the implications of determinism of detection of people in our model. Whilst the construction above relies on expected values of reward for exploration of a cell—since we cannot in advance know the true conditions on the ground (implied above in the unknown quantities p_{ij} and d_{ij})—we can still consider our MDP model deterministic, with the following justification. All planning in our decision making processes can only rely on the expected value of cell reward. Once a cell has been visited, and the true reward discovered, further exploration of the cell in our model yields no further reward (see below). Furthermore, at this stage we do not consider correlation between adjacent cells. This is because we ensure the decomposition results in cells encompassing entire buildings that in Haiti—and in Port au Prince in particular—are often built sporadically and have little relation to the structures in their surroundings (Government of the Republic of Haiti 2014). As a result, planning is not affected upon discovery of the true reward of visiting a cell, and we can therefore consider the prediction of future expected reward deterministic. Having clarified this point, we now introduce our solution to the survivor discovery problem in the environment model just described.

The Coordinated Monte Carlo Tree-Search Algorithm

Our decision to base our algorithm on Monte-Carlo tree search (MCTS) methods is due to their ability to sample very quickly from large state spaces (traditionally used in solving games), and the flexibility with which they can be applied to general problems (including MDPs) (Browne et al. 2012; Amato and Oliehoek 2015). This former point is particularly vital in our scenario as we have already noted the extremely large number of configurations possible in the environment (Equation 5). The principal purpose of our algorithm is to allow UAVs to use MCTS algorithms to calculate coordinated paths without incurring the cost described in Equation 5. To do this we exploit locality between UAVs to factor the search space into local joint-action trees. Furthermore, we allow trees to coordinate over shared factors (that is, shared UAVs). To this end we use the max-sum algorithm (Ramchurn et al. 2010; Rogers et al. 2011) as we note its ability to guarantee neighbourhood maximal rewards, and can do so in relatively few iterations (Farinelli, Rogers, and Jennings 2014).

Specifically, we introduce an additional step to the standard MCTS process of tree growth. This growth is typically summarised: node *selection*, *expansion*, *rollout* or simulation, and *backpropagation* (Browne et al. 2012; Kocsis and Szepesvari 2006). However, in our case we need to allow the UAVs to search their future actions whilst also accounting for the actions of other UAVs and the impact they will have on the reward obtained (that is, on the survivors discovered). Most significantly, we modify the selection process to determine which node to expand by coordinating in parallel between trees via max-sum: resulting in an exploration algorithm factored between multiple subsets of UAVs. This represents the first time a factored tree-search has been applied to a UAV search simulation. We detail our approach in the following subsections.

Tree Construction

At each timestep in the simulation, the coordinated MCTS (Co-MCTS) algorithm begins by calculating which UAVs

require coordination with their neighbours, leading to the form of the UAV-based factor graph constructed in the joint-action creation function $\mathcal{J}$ (Line 3). This is performed to establish whether coordination is needed in a given UAV's locality. In cases where a UAV is spatially isolated from neighbouring UAVs, a local tree is grown. The resulting groups of UAVs will form the basis of the factor graph used in the max-sum calculation (Line 14 in Algorithm 1). The result of $\mathcal{J}$ is represented formally by a set $N = \{n_1, \ldots, n_f\}$ that represents the domain of the factor nodes to be coordinated. Specifically, each member of N contains a set of actions corresponding to a group of UAVs that require coordination.

In more detail, for some set of neighbouring UAVs—for example $\{u_1, u_2, \ldots, u_k\}$—the possibility exists of $g_1(t+1) = g_2(t+1) = \ldots = g_k(t+1)$ at the next time step $t+1$ of a simulation.[3] In this case, the corresponding element in N—say n_i—would be the set of actions available to these UAVs: $n_i = \{a_1, a_2, \ldots, a_k\}$. Notice that since a UAV may interact (that is, potentially occupy the same cell at a future timestep, and thus form a joint tree) with more than one neighbour, the condition $\bigcup_{n_k \in N} n_k = G$ must hold, whereas $\bigcap_{n_k \in N} n_k = \emptyset$ will, in general, not. Trees are grown for each n_i in N, each of which in turn represents the factors in the max-sum graph connected to the variables representing the available actions of the UAVs. Individual nodes in the tree n_i will be indicated as $n_i^{(k)}$ or from any arbitrary tree by $n^{(k)}$.

Having described the construction of the trees, we now detail how each tree is grown in order to sample from the action space.

Tree Growth

Algorithm 1 begins with the creation of the root nodes representative of each factor seen in Line 6, which are recorded in the set N_r (Line 4). Following this, the creation and growth of branches is performed Δ times inside the loop beginning at Line 9. This begins by exploring down each tree, starting from the root node, to determine which node to branch on next. Similarly to standard upper confidence bound MCTS (Kocsis and Szepesvari 2006), this begins by selecting nodes which have hitherto not been fully expanded: that is, there remain neighbouring action states that have not yet been branched to previously. The total number of neighbouring actions ν from a given action node is of order $\nu = \prod_{a_\beta \in n_\alpha} \mid a_\beta \mid$ for a tree corresponding to factor node n_α, simplifying to $\nu = \mid a_\beta \mid^{|n_\alpha|}$ when all UAVs in the set have the same number of available actions.[4] Thus, the operation of the function in Line 18:

$$\text{fullexp}(n^{(k)}) = \begin{cases} True & \text{if } e = \nu \\ False & \text{otherwise} \end{cases}$$

where e is the number of previous expansions of that node. Line 10 introduces the current set of nodes (across all trees) to be expanded next, N_{next}, and Line 11 creates the set of previously expanded nodes N_{prev}. At Line 14 the max-sum algorithm is used to maximise the value of the actions over each n_i, returning a vector of favourable actions $a^* = (a_1^*, a_2^*, \ldots, a_m^* \mid a_k^* \in a_k)$. Since each n_i depends on a subset of actions, the function select$(n^{(k)}, a^*)$ serves to return only the actions corresponding to a given $n^{(k)}$. This is then used as the argument to create the new expansion to a node in N_{next} in Line 17.

Algorithm 1 Coordinated MCTS

```
CoMCTS(G, C, t = 0)
1.  for each in [1, ..., f]
2.    //Creation of factor graphs//
3.    N ← J(G)
4.    N_r ← ∅
5.    for n_i in N
6.      append(N_r) ← n_i^(0)
7.    endfor
8.    //Loop for each iteration of tree growth//
9.    for δ in [1, ..., Δ]
10.     N_next ← N_r
11.     N_prev ← ∅
12.     while N_next ≠ ∅
13.       //Coordination between trees for production of optimal actions//
14.       a* ← maxsum (N, N_next)
15.       for n^(k) in N_next
16.         //Chooses action relevant to the tree//
17.         n_new^(k) ← expand (n^(k), select (n^(k), a*))
18.         if fullexp (n^(k)) = True
19.           remove (n^(k), N_next)
20.           append (n^(k), N_prev)
21.         endif
22.       endfor
23.     endwhile
24.     for n^(k) in N_prev
25.       //Rollout and backpropagation of values//
26.       rollout(n_new^(k))
27.       backpropagate(n_new^(k))
28.   endfor
29.   endfor
30.   t ← t + 1
31. endfor
32. for n_i in N
33.   Return (bestactions (n_i))
34. endfor
```

An example factor graph and trees are shown in Figure 1, for four interacting UAVs. Their actions are shared between two factor graph utility nodes, hence the two joint action trees n_1 and n_2. Four expansions of the root are shown where the action of the shared UAV—u_3—has been coordinated between the trees each time, thus ensuring contradictory actions are not chosen for the same UAV in two different trees. A second depth of growth is shown in Figure 2, where coordination has resulted in a newly created node with common action for a_3 of $\uparrow$.

Rollout

The rollout portion of the MCTS is traditionally a coarse estimate of the affect of future actions as the result of explor-

[3] We note here a slight abuse of notation, since these nodes serve as *functions* within a factor graph rather than simply a set of actions. Since we factor locally, the functions depend only on the actions in each n and so we omit the function notation for clarity.

[4] We use $\mid x \mid$ on any set x to denote cardinality.

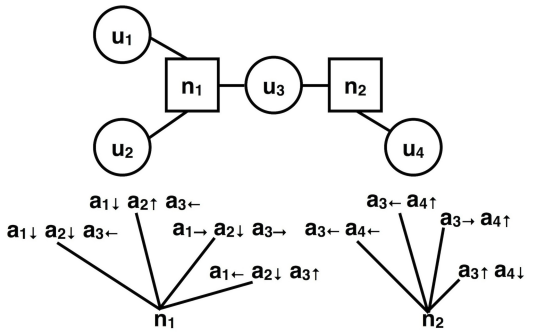


Figure 1: Example factor graph and the first iteration of tree growth for four interacting UAVs. Here u_3 can interact with any of the other UAVs, and is thus common to the two joint-action trees, which coordinate in order to maximise the reward from its actions in conjunction with the other UAVs. Factor and utility nodes are synonymous with variable and function nodes as outlined in (Ramchurn et al. 2010).

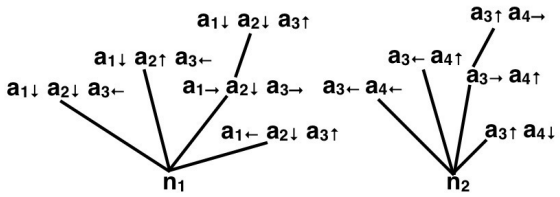


Figure 2: New nodes created at lower depth in the tree, where the actions of u_3 have again been coordinated to be $\uparrow$ in both cases.

ing a particular node in the action space. In this example, we base the rollout on a random-walk through the action space starting at the node just expanded, biased in the direction of the last action taken. This method has the benefit of showing not just the contribution of any random series of actions, but of taking more actions similar to the one represented by the frontier node (for each UAV). Intuitively, a random rollout from one node in a joint action tree will be insignificantly different from a rollout from any similar node because of their spatial proximity. Conversely, our rollout policy contributes to the exploration value of a node by indicating possible future reward through continued tree expansion with a preference for repetitions of the action itself.

Results

To verify the performance of our algorithm on data relevant to real-world disaster scenarios, we used data from the Ushahidi project (Morrow et al. 2011) produced from crowd-sourced information during the 2010 Haiti earthquake.[5] Specifically, we extracted the level of damage and coordinates of buildings in a 2km square centred on the capital, Port-au-Prince. Damage was rated based on crowd reports on a scale from 1 to 5, with 5 being the most severe.

We then constructed a decomposed grid world of size 200×200 of $10m$ cells, with UAVs traversing from the centre of one cell to the centre of an adjoining cell above, below, or to either side at each time step. This is convenient since assuming a UAV speed—typical of quad rotor vehicles—of $10ms^{-1}$ amounts to the traversal of one cell in one timestep of one second.

Damaged buildings represent an estimate of the damage in an area and thus, the danger to the victims on the ground: buildings that have suffered more severe damage will likely lead to more severe casualties and a higher rate of death. We formed a belief map of danger to the populace by summing the total number of buildings above a threshold level (set to a crowd report of damage 3 and above) in each cell, before multiplying by a common factor to convert the data into a map representative of expected fatalities (noting the constraint in Equation 1). The environment is displayed in Figure 3 with a scale showing the value of d in each location.

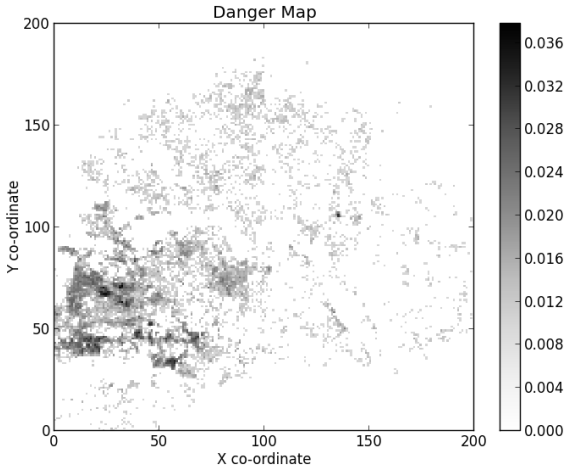


Figure 3: Danger as a function of position, created from Ushahidi dataset centred over Port-au-Prince

The number of unique sets of UAV positions G (that is, the cardinality of the state space) in this environment for (for example) five UAVs, is of the order 8.5×10^{20} (referring to Equation 5); even without the variation in the expected number of people with time.[6] Using this environment, we measure performance by evaluating the number of survivors averaged over each UAV over each timestep. That is: $\frac{1}{f \cdot m} \sum_{t=0}^{f} \sum_{c_{ij} \in G(t)} \bar{p}_{ij}(t)$ where f denotes the final time step (a stopping time introduced simply as a limit of the simulation), and we recall that m is the total number of UAVS, and $G(t)$ is the unique positions of the UAVs at time step t.

Figure 4 shows our results for an initial test of the coordinated Monte Carlo tree search using randomised starting locations for four simulated UAVs. We compare against, a simple "lawnmower" sweep search—a typical strategy employed in search and rescue situations (Goodrich et al.

[5]Available from `http://www.ushahidi.com/`

[6]Calculated using standard multichoose combinatorics (Feller 1968).

2007)—as well as a typical MCTS algorithm (as per the work of Chaslot et al. (2008)) without the factored coordination we introduce. We also demonstrate explicitly the forward planning required in the survivor discovery problem, by benchmarking against Co-MCTS without a rollout policy, and a greedy (but locally coordinated via max-sum) policy. Results demonstrate a minimum performance increase of 10% over MCTS, and higher gains over the other benchmarks. Errors are taken as standard error of the mean over 1000 repeats of each experiment. The poor performance of a greedy one-step lookahead shows that even with coordination, the ability to forward-plan to account for survivor death rates is essential to discovering casualties in our scenario. Indeed, simple un-planned lawnmower-style sweep searches are more successful, but still fall short of the MCTS's ability to plan exploration paths to target (for instance) high-danger areas before casualty rates become too high.

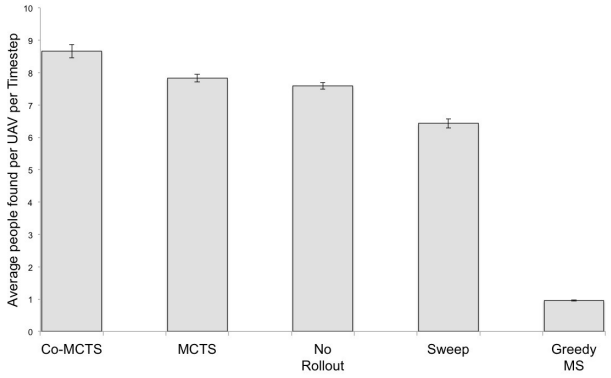


Figure 4: Comparison of performance between: coordinated MCTS, un-coordinated MCTS, simple lawnmower sweep search, Co-MCTS with no rollout policy, greedy policy with max-sum coordination. Start locations were uniformly randomised, $m = 4$, $f = 1000$.

Additionally, we explicitly demonstrate the benefit of the *consistency* afforded by our coordination in a simulation. We do this by varying the number of UAVs present and comparing to the closest benchmark to Co-MCTS's performance from our initial simulation. Good coordination should benefit the overall reward gained by the algorithm (in our case the number of people rescued) with low diminishing returns compared to uncoordinated approaches. Specifically, any additional UAVs should still find close-to the same number of casualties as other UAVs in the system, if they coordinate the exploration task effectively as a group. If they do not, one would expect additional UAVs would explore the same regions of the disaster space as those already present: which, as discussed previously, offers no benefit to the global reward function. This is demonstrated in Figure 5 for varying m, where we note performance improvements of 12% with the inclusion of 6 UAVs (and an average of 6% across the experiment). This is notable after 3 UAVs are introduced, since before this point interaction (and therefore required coordination between the UAVs) was at a minimum since the search space was large enough to accommodate multiple non-intersecting explorative paths. For more UAVs, coordination is more commonplace and with the successful implementation of Co-MCTS there is higher value in the inclusion of further UAVs into the scenario, compared to a situation without coordination.

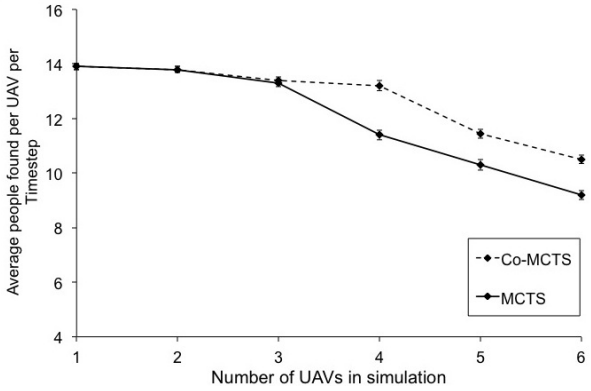


Figure 5: Comparison of coordinated and un-coordinated MCTS in locating survivors with additional UAVs; demonstrating a consistent performance per-UAV in Co-MCTS. Initial UAV starting locations fixed at $c_{0,0} = [0, 0]$; $f = 1000$.

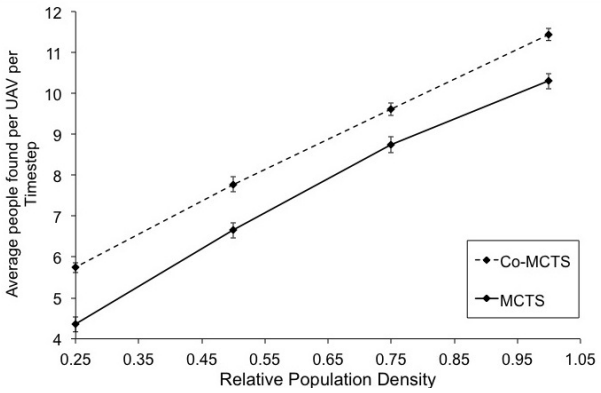


Figure 6: Comparison of coordinated and un-coordinated MCTS in locating survivors over varying densities of population, showing Co-MCTS's consistency in different forms of belief data. Here $c_{0,0} = [0, 0]$; $f = 1000$; and $m = 5$.

Finally, we demonstrate how Co-MCTS performs consistently in varying environments. For the algorithm to be viable, it should show performance benefits in a variety of situations with a variety of distributions of casualties to be discovered. We achieve this by varying the relative population over the belief-space from 1 (as above), down to 0.25 by uniformly sampling from the Ushahidi dataset to the population portion of the belief map. This has the effect of maintaining our standard of using real data with a large action space, while still allowing us to experiment over different state spaces (S). The results presented in Figure 6 demonstrate this consistency with an average 14% improvement over the un-coordinated MCTS benchmark: even at low densities of people; where the coordination requirements would

be intuitively less valuable. We thus provide evidence that the performance gains of Co-MCTS are not simply a by-product of the test environment selected.

Conclusions

Motivated by the recent increased availability of belief-data about disaster environments, we have introduced an implementation of a decentralised, factored, coordinated Monte Carlo tree search algorithm for the purpose of discovering survivors in a simulated UAV path planning scenario. Tests were carried out on real-world data from the 2010 Haiti earthquake via the Ushahidi platform; an environment with a very large (of order 8.5×10^{20}) action space. We demonstrated the capability of our Co-MCTS algorithm in sampling this space and planning paths, and demonstrated consistent performance gains in the number of survivors discovered of $> 10\%$. Future work will seek to extend these solutions to time-varying belief maps to cope with the situation in which data is collected and updated during exploration, and removing the assumption of a cellular disaster area to tackle the issue of discretising (and therefore potentially oversimplifying) the data collected.

References

Adams, S. M., and Friedland, C. J. 2012. A Survey of Unmanned Aerial Vehicle (UAV) Usage for Imagery Collection in Disaster Research and Management. In *Proceedings of the Ninth International Workshop on Remote Sensing for Disaster Response*, volume 9.

Amato, C., and Oliehoek, F. A. 2015. Scalable Planning and Learning for Multiagent POMDPs. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 1995–2002. Austin, Texas: AAAI.

Bernardini, S.; Fox, M.; and Long, D. 2014. Planning the Behaviour of Low-Cost Quadcopters for Surveillance Missions. In *Proc. of 24th Int. Conference on Automated Planning and Scheduling*, 445–453. Portsmouth, NH: AAAI.

Browne, C. B.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *Transactions on Computational Intelligence and AI in Games* 4(1):1–43.

Bry, A., and Roy, N. 2011. Rapidly-Exploring Random Belief Trees for Motion Planning Under Uncertainty. In *2011 IEEE International Conference on Robotics and Automation*, volume 21, 723–730. Shanghai: Massachusetts Institute of Technology, Cambridge, USA.

Cashmore, M.; Fox, M.; Larkworthy, T.; Long, D.; and Magazzeni, D. 2014. AUV Mission Control via Temporal Planning. In *2014 IEEE International Conference on Robotics and Automation*, 6535–6541. Hong Kong, China: IEEE.

Chaslot, G. M. J.-B.; Uiterwijk, J. W. H. M.; Herik, H. J. V. D.; Winands, M. H. M.; and Bouzy, B. 2008. Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation* 04(03):343–357.

Chen, Y.-b.; Luo, G.-c.; Mei, Y.-s.; Yu, J.-q.; and Su, X.-l. 2014. UAV path planning using artificial potential field method updated by optimal control theory. *International Journal of Systems Science* (October):1–14.

Chiu, W.-t.; Arnold, J.; Shih, Y.-T.; Hsiung, K.-H.; Chi, H.-Y.; Chiu, C.-H.; Tsai, W.-C.; and Huang, W. C. 2002. A Survey of International Urban Search-and-rescue Teams following the Ji Ji Earthquake. *Disasters* 26(1):85–94.

Crisis Mappers. 2013. Crisis Mappers - The Humanitarian Technology Network.

Durkota, K., and Komenda, A. 2013. Deterministic Multiagent Planning Techniques: Experimental Comparison (Short paper). In *Proceedings of DMAP Workshop of ICAPS'13*, 43–47. Rome, Italy: AAAI.

Farinelli, A.; Rogers, A.; and Jennings, N. R. 2014. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. In *Autonomous Agents and Multi-Agent Systems*, volume 28, 337–380.

Fawcett, W., and Oliveira, C. S. 2000. Casualty Treatment after Earthquake Disasters: Development of a Regional Simulation Model. *Disasters* 24(3):271–287.

Feller, W. 1968. *An Introduction to Probability Theory and Its Applications. Volume I.* Wiley, 3rd edition.

Fowler, J. 2016. Data curbs earthquake risk in Armenia. Technical report, United Nations Office for Disaster Risk Reduction.

Gan, S. K., and Sukkarieh, S. 2011. Multi-UAV Target Search using Explicit Decentralized Gradient-Based Negotiation. In *2011 IEEE International Conference on Robotics and Automation*, 751–756. Shanghai: Ieee.

Goda, K.; Kiyota, T.; Pokhrel, R. M.; Chiaro, G.; Katagiri, T.; Sharma, K.; and Wilkinson, S. 2015. The 2015 Gorkha Nepal Earthquake: Insights from Earthquake Damage Survey. *Frontiers in Built Environment* 1(April):1–15.

Goerzen, C.; Kong, Z.; and Mettler, B. 2009. A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance. *Journal of Intelligent and Robotic Systems* 57(1-4):65–100.

Goodchild, M. F., and Glennon, J. A. 2010. Crowdsourcing geographic information for disaster response: a research frontier. *International Journal of Digital Earth* 3(3):231–241.

Goodrich, M. A.; Cooper, J. L.; Adams, J. A.; Humphrey, C.; Zeeman, R.; and Buss, B. G. 2007. Using a Mini-UAV to Support Wilderness Search and Rescue: Practices for Human-Robot Teaming. In *IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR)*, 1–6. Rome, Italy: IEEE.

Government of the Republic of Haiti. 2014. Haiti Earthquake PDNA: Assessment of damage, losses, general and sectoral needs. Technical report.

He, R. 2007. *Planning in information space for a quadrotor helicopter in a GPS-denied environment*. Ph.D. Dissertation, MIT.

Hu, J.; Xie, L.; Xu, J.; and Xu, Z. 2014. Multi-agent cooperative target search. *Sensors (Switzerland)* 14(6):9408–9428.

Kocsis, L., and Szepesvari, C. 2006. Bandit based Monte-Carlo Planning. *Machine Learning: ECML 2006* 4212:282–293.

Kolling, A., and Kleiner, A. 2013. Multi-UAV Motion Planning for Guaranteed Search. In *Autonomous Agents and Multiagent Systems*, 79–86.

Kothari, M., and Postlethwaite, I. 2012. A Probabilistically Robust Path Planning Algorithm for UAVs Using Rapidly-Exploring Random Trees. *Journal of Intelligent and Robotic Systems* 71(2):231–253.

Kothari, M.; Postlethwaite, I.; and Gu, D.-W. 2009. Multi-UAV path planning in obstacle rich environments using Rapidly-exploring Random Trees. In *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, 3069–3074. Shanghai: IEEE.

Liu, Y. C., and Dai, Q. H. 2010. A survey of computer vision applied in aerial robotic vehicles. In *OPEE 2010 - 2010 International Conference on Optics, Photonics and Energy Engineering*, number 201, 277–280. Wuhan, China: IEEE.

Macintyre, A. G.; Barbera, J. A.; and Petinaux, B. P. 2011. Survival Interval in Earthquake Entrapments: Research Findings Reinforced During the 2010 Haiti Earthquake Response. *Disaster Medicine and Public Health Preparedness* 5(1):13–22.

Meier, P. 2015. Chapter 6: Uavs And Humanitarian Response. In *Drones And Aerial Observation: New Technologies For Property Rights, Human Rights, And Global Development: A Primer*, number July. www.newamerica.org, online edition. 103.

Morrow, N.; Mock, N.; Papendieck, A.; and Kocmich, N. 2011. Independent evaluation of the Ushahidi Haiti project. Technical report, Ushahidi.

Murphy, R. R. 2012. A Decade of Rescue Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5448–5449. Vilamoura, Portugal: IEEE.

Ramchurn, S. D.; Farinelli, A.; Macarthur, K. S.; and Jennings, N. R. 2010. Decentralized Coordination in RoboCup Rescue. *The Computer Journal* 53(9):1447–1461.

Rogers, A.; Farinelli, A.; Stranders, R.; and Jennings, N. R. 2011. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence* 175(2):730–759.

Silver, D., and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. *Advances in Neural Information Processing Systems* 23:2164–2172.

United Nations Foundation. 2011. Disaster relief 2.0. Technical report, United Nations.

Waharte, S.; Trigoni, N.; and Julier, S. 2009. Coordinated Search with a Swarm of UAVs. In *Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 6th Annual IEEE Communications Society Conference on*, 1–3. Rome, Italy: IEEE.

A Distributed Online Multi-Agent Planning System

Rafael C. Cardoso and **Rafael H. Bordini**
School of Informatics – FACIN-PPGCC
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil
{rafael.caue@acad.pucrs.br, rafael.bordini@pucrs.br}

Abstract

Multi-agent planning is an important capability to have in the development of multi-agent systems, which still remains an open problem, mainly because of the gap between planning and execution. Multi-agent systems often have dynamic environments that require planning to be done during run-time (i.e., online planning). In this paper, we use a platform for the development of multi-agent systems (JaCaMo) in both decentralised multi-agent planning and execution stages, providing a multi-agent system with capabilities to solve online multi-agent planning problems. The contributions shown in this paper are: i) the design of a Distributed Online Multi-Agent Planning System (DOMAPS); ii) the implementation of DOMAPS in JaCaMo; and iii) initial experiments in the Floods domain, a novel planning domain that uses heterogeneous unmanned vehicles to respond to flood disasters.

1 Introduction

Multi-Agent Systems (MAS) are often situated in dynamic environments where new plans of actions need to be constantly devised in order to successfully achieve the system's goals. Therefore, employing planning techniques during run-time of a MAS can be used to improve agent's plans using knowledge that was not previously available, or even to create new plans to achieve some goal for which there was no known course of action at design time.

Research on automated planning has been mostly focused on single-agent planning over the years. Although it is possible to adapt centralised single-agent techniques to work in a decentralised way, such as in (Crosby, Jonsson, and Rovatsos 2014), distributed computation is not the only advantage of using Multi-Agent Planning (MAP).

In MAP, by allowing agents to do their own individual planning (i.e., planning by multiple agents), the search space can be effectively pruned, which can potentially decrease planning time on domains that are intrinsically distributed. This also means that agents get to keep some (or even full) privacy from other agents in the system, as they might have beliefs, goals, and plans that they do not want to share with other agents. The output of a MAP process are plans for multiple agents. Single-agent planning for multiple agents can have no privacy, since the planner needs all the information available, and the agents in the problem representation are usually considered as any other object of the environment. These differences are characterised in Table 1, based on the descriptions found in (Durfee and Zilberstein 2013).

MAS went through a similar process of transitioning from single to multiple agents, albeit at a faster rate. Recent research, as evidenced in (Boissier et al. 2011; Singh and Chopra 2010), shows that considering other programming dimensions such as environments and organisations as first-class entities along with agents allow developers to create more complex MAS.

In this paper, we introduce the design of our Distributed Online Multi-Agent Planning System (DOMAPS). DOMAPS is composed of: i) a formalism for the representation of domains and problems in online multi-agent planning, based on Hierarchical Task Network (HTN); ii) a contract net protocol mechanism for goal allocation; iii) individual planning with the SHOP2 planner; and iv) the use of social laws to coordinate the agents during execution. Some preliminary results from experiments in a novel scenario, the Floods domain, are shown.

Although approaches to online single-agent planning usually involve some kind of interleaving planning and execution, we focus on domains that allow agents some time to plan while the system is still in execution (i.e., anytime planning). DOMAPS allows for the dynamic execution of plans found during run-time, making it easy to transition from planning into execution and vice-versa, while still permitting agents to continue their execution, as long as their actions are believed not to cause any conflict with actions from a possible solution.

In DOMAPS current configuration, during the planning stage the environment is assumed to be deterministic and fully observable (although each agent has its own perspective of the environment). However, in the execution stage we do not make these assumptions, the environment can be non-deterministic and actions can fail, in which case it may be necessary to replan.

The remainder of the paper is structured as follows. In the next section a brief description of the MAS development platform, JaCaMo, used to implement DOMAPS and run the agents and the MAS is given. Section 3 introduces the initial design of the Distributed Online Multi-Agent Planning System. Next, in Section 4, we describe the implementation of

Table 1: Comparisons between single-agent planning and multi-agent planning.

	computation	**privacy**	**agent abstraction**
single-agent planning for a single agent	centralised	not needed	not needed
single-agent planning for multiple agents	centralised	none	objects
multi-agent planning for a single agent	decentralised	none or partial	not needed
multi-agent planning for multiple agents	decentralised	partial or full	first-class entities

DOMAPS in JaCaMo. In Section 5, we describe the Floods domain, and some initial experiments of using DOMAPS in this domain. We follow with a discussion on related and future work, and end the paper with some concluding remarks.

2 Background

DOMAPS was designed for online systems, thus, requiring the use of planning techniques whilst the MAS is running. Therefore, we need a MAS development platform in order to properly implement and evaluate DOMAPS. We chose to use the **JaCaMo**[1] (Boissier et al. 2011) MAS development platform, since it contains programming abstractions that we found to be a suitable match for the implementation of DOMAPS – **organisation**, **environment**, and **agent** abstractions.

JaCaMo combines three separate technologies into a platform for MAS programming that makes use of multiple levels of abstractions, enabling the development of robust MAS. Each technology (Jason, CArtAgO, and Moise) was developed separately for a number of years and are fairly established on their own when dealing with their respective abstraction level (agent, environment, and organisation).

Moise (Hübner, Sichman, and Boissier 2007) handles the organisation level, and how to specify an organisation in a MAS. This level adds first-class elements to the MAS such as roles, groups, organisational goals, missions, and norms. Agents can adopt roles in the organisation, forming groups and sub-groups. Missions are defined to achieve the organisation goals. The behaviour of the agents that adopt roles to execute these missions is guided by norms.

Jason (Bordini, Wooldridge, and Hübner 2007) is responsible for the agent level. It is an extension of the AgentSpeak language, based on the BDI architecture. Agents in Jason react to events in the system by executing actions on the environment, according to the plans available in each agent's plan library.

CArtAgO (Ricci et al. 2009) is based on the A&A (Agents and Artefacts) model (Omicini, Ricci, and Viroli 2008), and deals with the environment level. Artefacts are used to represent the environment, storing information about the environment as observable properties and providing actions that can be executed through operations. Agents can focus on specific artefacts in order to obtain information contained on that artefact. When an agent focuses on an artefact, it receives the observable properties as beliefs, and it is able to execute the artefact's operations.

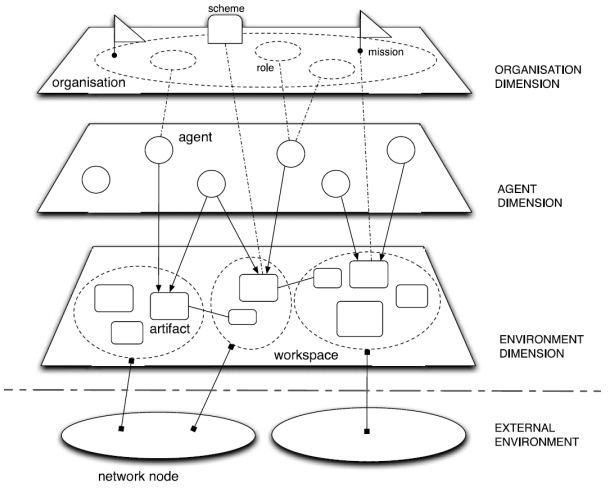


Figure 1: The JaCaMo overview (Boissier et al. 2011).

An overview of how JaCaMo combines these different levels of abstraction can be observed in Figure 1. In the top-most level, the organisation dimension is composed of a scheme, a set of missions, and a set of roles. These roles are adopted by the agents that inhabit the agent dimension. In the bottom-most dimension, the environment houses artefacts that represent objects and information about the environment, grouped by workspaces that agents can access. These workspaces can be distributed across multiple network nodes, providing the distribution of the MAS.

3 The Distributed Online Multi-Agent Planning System

Our framework, the Distributed Online Multi-Agent Planning System (DOMAPS), consists of four main components: **planning formalism** – a formal representation of the information from the planning domain and problem that will be used during planning; **goal allocation** – the mechanism used to allocate goals to agents; **individual planning** – the planner used during each agent's individual planning stage; and **coordination mechanism** – used before or after planning to

[1] `http://jacamo.sourceforge.net/`.

avoid possible conflicts that can be generated during planning.

DOMAPS was made to work as a general-purpose domain-independent system, and as such, we expect to turn it into an open platform where many other alternatives for main components can be added, allowing MAS developers and researchers to pick and choose the ones that work better to solve their online multi-agent planning problem.

The design overview of DOMAPS is shown in Figure 2. Multiple agents $(a_1, a_2, ..., a_n)$ interact with an environment to obtain information and carry out their actions. These agents are part of an organisation where they can adopt roles, follow norms, and receive role-related missions, while pursuing the organisation's goals. These aspects are what represent the MAS part of the application.

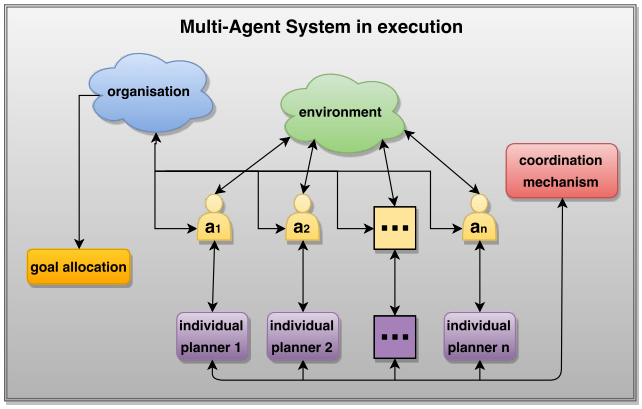


Figure 2: DOMAPS design overview.

Planning input (i.e., domain and problem representation) and output (i.e., the solution) are regulated by a planning formalism. The agents themselves plan individually, using an appropriate planner for the planning problem at hand. Coordination is used in order to achieve the organisational goals of the system. For example, goals that depend on joint plans involving multiple agents, or an agent's actions that can cause conflict with the other agents' plans. Since we are dealing with planning during run-time (online), new organisational goals can emerge (or their conditions can change) during the execution of the MAS. Therefore, we also use a mechanism to allocate these organisational goals to the appropriate agents, that is, allocate them to the agents that have an estimated better chance at solving that particular goal.

During execution, the creation of new organisational goals can start the planning process in DOMAPS, which consists of the following steps:

1. **Allocate goals:** Agents gain access to the organisational goals that will be planned for. Then, a mechanism is used to separate and allocate goals to agents that have the most (estimated) chance of finding a potential solution to the goal.
2. **Obtain up-to-date information needed for planning:** Environment and world information need to be collected from the MAS in execution, and translated into a planning formalism that can be used by the planner. Since planners work individually, each agent passes its input to their respective planner. Thus, the information that a planner has access to is limited to the information that the agent is allowed to access during that exact moment in the execution. Consequently, this ensures, at least, some partial privacy in DOMAPS.
3. **Agents start their individual planner:** The planner starts its search for a solution to the allocated goal, or set of goals.
4. **Coordination before or after planning:** Agents coordinate with each other before or after the planning process, in order to prevent any conflicts or help solve any dependencies.
5. **Translate solution:** Each agent translates the solution found by their respective planners into plans that can be added to their plan library. If we are dealing with coordination after planning, then any points of conflict or dependency found along the way should be sent to the coordination mechanism. Otherwise, when coordination is done before planning, the solution should already be free of conflicts and dependencies.

Another goal that we have with DOMAPS, is to allow the addition of new approaches to each of the four main components more easily, that way it is possible to choose the approach that is more suited for a particular problem. Next, we describe the initial approach used for each component, and discuss alternative approaches in Section 6.

3.1 Planning Formalism

We developed the Multi-Agent Hierarchical Task Network (MA-HTN) formalism, which is an extension of the single-agent HTN formalism used in the SHOP2 planner (Nau et al. 2003). MA-HTN is intended for **online** multi-agent planning problems, since domain and problem information have to be collected during execution. Agents use a **translator** to parse their information about the world into domain and problem specifications that is then passed to their own individual planner. The MA-HTN grammar for the problem and domain representation, as well as the translator's specification, can be found in (Cardoso and Bordini 2016).

Each agent has their own problem and domain specification. This provides a decent level of privacy on its own, since each planner only has access to their respective agent problem and domain specifications. This means that, unlike some of the other multi-agent planning formalisms, MA-HTN does not need to have private or public blocks. Although at some point it might be interesting to add the capability to include private goals, for now we are interested only on searching solutions for organisational goals.

Actions from other agents can cause conflicts, either at the moment that the action is executed (e.g., concurrent actions) or in the future (e.g., durative actions). Actions that can cause **conflict** have to be annotated by the MAS developer, in order for the translator to identify them. Likewise, dependencies between actions can also exist, either as a concurrent action that requires another agent or as actions

that depend on the actions of other agents to happen first. These **dependency** relations also have to be annotated by the MAS developer, so that the translator can add them to the specification. Both conflicts and dependencies specifications are used by the coordination mechanism to coordinate the agents.

3.2 Goal Allocation

A Contract Net Protocol (CNP) mechanism is used to allocate goals to agents in DOMAPS. Our CNP mechanism is based on the original CNP design of Reid G. Smith (Smith 1980), with a few modifications in order to accommodate our needs for a goal allocation mechanism in the context of MAP. The initiator in our case is the organisation. It is the organisation's role to start new auctions for organisational goals that do not have any known plans on how to achieve the goals, or for organisational goals that have plans, but need to be re-planned. The bidders are agents from the organisation that also participate in the planning process.

The logic for determining an agent's bid depends on the rest of the mechanisms being used in DOMAPS and in the MAS development platform, but it is fair to assume that agents have the ability of checking their plan library for plans that are able to decompose, at least at some level, the goal that is being auctioned. Although domain-dependent functions for determining the bid can, generally, provide better results, we supply a simple domain-independent general-purpose function that agents can use to determine their bid, shown in Algorithm 1.

The agent checks if the announcement of the goal came from the organisation and if it is eligible, according to the eligibility criteria provided in the announcement, or otherwise decides not to bid. If the agent chooses to proceed with the bid, then, he keeps decomposing the goal into subtasks and incrementing the bid by 1 for each level that was successfully decomposed, either until it is close to the deadline, or it arrived in an action that could achieve the goal, or it found a dead end (in which case the bet becomes empty). The recursion indicates the backtrack when there are no more levels.

The initiator allocates the goal to the agent with the lowest (not empty) bid. We found that the lowest bid heuristic had better results in our initial experiments, as opposed to using the highest bid heuristic. Our logic behind using the lowest, is that it indicates that lower bids from agents, with different plan libraries and who were able to arrive at final decompositions, means that they can arrive at the solution using the lowest number of actions. However, for homogeneous agents with similar plan libraries and who were not able to fully explore their plan library, the selection of the highest bid may yield better results, since it would represent the agent who could decompose the furthest. We assume here that every goal can eventually be allocated, meaning that there is at least one agent eligible (and capable) for each organisational goal.

Regarding plan decomposition, agents do not check the plan's context (preconditions), nor does it uses any action theory to simulate future states. It simply decomposes into the first plan found in the plan library, and then, proceeds to decompose into any first plan found in the body of the original plan. Once there are no more levels to decompose, the agent backtracks to the original plan and chooses another branch, if there is one.

Algorithm 1 Domain-independent algorithm for determining an agent's bid.

```
function bid (bid-value, from, goal, eligibility, deadline)
if ((from ≠ organisation) or (not eligible)) then
    return bid-value ← ∅
else
    while ((close to deadline) or (no more levels available to decompose)) do
        decompose one level of one task from goal
        bid-value ← bid-value + 1
        if deadend then
            return bid-value ← ∅
        end if
    end while
    if close to deadline then
        return bid-value
    end if
    if there are more levels available to decompose then
        bid (bid-value, from, goal, eligibility, deadline)
    end if
    return bid-value
end if
```

3.3 Individual Planner

SHOP2 (Nau et al. 2003) is a HTN planner with support for anytime planning. No modifications were made to the actual planning algorithm and search heuristics of SHOP2. Instead, we use the mechanisms from the other components to handle the multi-agent part of the planning process. By making little to no modifications to the individual planners, DOMAPS benefits from its multi-layered approach, making it easier to swap components without having to modify the planner's code directly.

Many parameters can be used to tweak the SHOP2 planner. The most relevant to DOMAPS is the parameter that guides which kind of search will be made:

- **first:** depth-first search that stops at the first plan found.
- **shallowest:** depth-first search for the shallowest plan, or the first such plan if there are more than one.
- **id-first:** iterative-deepening search that stops at the first plan found.

3.4 Coordination Mechanism

Social laws can coordinate agents by placing restrictions on the activities of the agents within the system. The purpose of these restrictions are twofold: it can be used to prevent some destructive interaction from taking place; or it can be used to facilitate some constructive interaction.

The design of social laws is domain-dependent, and we require them to be supplied by the system designer offline (i.e., they are provided before planning). We apply the social

laws for the coordination of agents after planning, during execution.

In the original model of Shoham and Tennenholtz (Shoham and Tennenholtz 1995), social laws were used to restrict the activities of agents so as to ensure that all individual agents are able to accomplish their personal goals. We follow a similar idea, although agents here aim to achieve organisational goals, and thus, are naturally compelled to follow the social laws that are present in the system.

We formally define social laws in our model as:

Definition 1 Given a set of agents Ag, a set of actions Ac, a set of states S, a set of preconditions P, and a set of options Θ, a *social law* is a tuple (ag,ac,s,P,Θ) where $ag \in Ag$, $ac \in Ac$, and $s \in S$.

A social law sl constrains a specific action ac of agent ag, considered to be a possible point of conflict (as established in the operator description from the MA-HTN formalism). When the state s satisfies each precondition $\rho_i \in P$, the agent is given all possible options $\theta_i \in \Theta$. Although not explicitly present in this model, the *null* action (i.e., do nothing) can be a possible option, but in order for it to be viable it needs to have been established as an action in the MAS.

4 Multi-Agent System Integration

We implemented the `domaps.plan` internal action to start the DOMAPS planning process – internal actions are actions that Jason agents can execute internally, as opposed to external actions, which are environment-related. These internal actions are implemented as Java classes that agents can call.

To illustrate the run-time of DOMAPS when the `domaps.plan` internal action is executed, consider the overview provided in Figure 3. When an agent calls `domaps.plan`, it goes through phase 1 and activates the contract net protocol mechanism to allocate organisational goals between the agents. Then, in phase 2, each agent knowledge about the world is passed to a MA-HTN translator, that sends the information needed to SHOP2 for the individual planning that takes place in phase 3. The solution found by each agent's planner goes back through the MA-HTN translator again, translating the solution into AgentSpeak Jason plans. Finally, the solution is carried out by the agents, in accordance to the social laws (phase 5) that are associated with the actions that can cause conflicts.

4.1 MA-HTN

Each agent uses a MA-HTN translator in order to parse the current information obtained from the CArtAgO environment artefacts, as well as from their own personal artefact, and the plans from its plan library. The MA-HTN translator generates a problem and domain representation for each agent, as follows:

- **Problem representation:** The name of the problem and the name of the domain are obtained dynamically. The name of the agent is the one who started the translator.

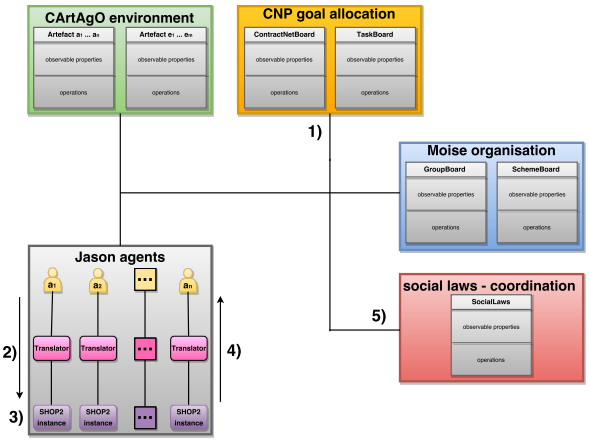


Figure 3: DOMAPS run-time overview of the `domaps.plan` internal action.

The information collected from the CArtAgO artefacts are parsed into the agent's facts and initial states. The goal list is created from the organisational goals that were assigned to this particular agent during the goal allocation phase.

- **Domain representation:** The name of the domain is obtained dynamically. The name of the agent is the one who started the translator. Operators are parsed from all of the artefacts operations that the agent has access to. The preconditions are obtained from any conditional tests in an operation, the delete and add list are acquired from the deletion and addition of observable properties, respectively. The conflict and dependency lists need to be previously annotated into the operation in order for them to be able to be parsed. The methods are parsed from all of the plans in the agent's plan library, with the preconditions parsed from the context of the plan, and the task list parsed from the body of the plan.

4.2 Contract Net Protocol

The contract net protocol artefacts mediate the goal allocation phase of DOMAPS. The mechanism is represented by two artefacts: the *TaskBoard* artefact and the *ContractNetBoard* artefact. All of the agents that will participate in the planning stage take the roles of *bidders*. The *bidders* should always focus on the *TaskBoard*, as that is the artefact in which the organisational goals are announced. The role of *initiator* is restricted to the organisation. When the *initiator* announces an auction for a new organisational goal, it creates a *ContractNetBoard* associated with that goal.

We show the observable properties and operations of the *TaskBoard* and the *ContractNetBoard* artefacts in Figure 4. When a new new task is announced by the *initiator*, a *task* observable property is created. A link interface includes the set of operations that can be executed by other artifacts. Thus, link operations cannot be accessed by agents, but only by linking artifacts. Therefore, only the organisation artefact, as the *initiator*, can announce goals in the *TaskBoard*. When the auction process ends, the *initiator* performs the

clear operation to delete the observable property associated with that goal. This also generates an event in Jason, a belief deletion event, in which agents can perform clean-up and some other necessary activities.

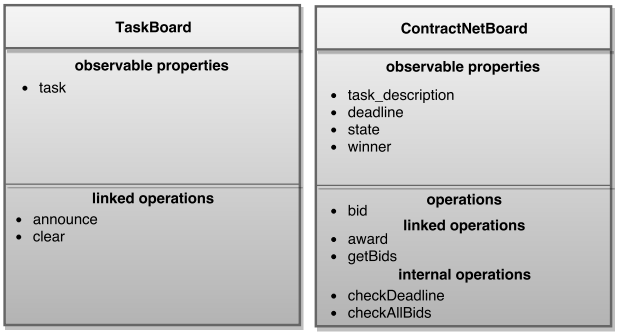


Figure 4: The task board and CNP board artefacts.

A *ContractNetBoard* is created for each goal announced by the *initiator*. The *bidder* agents focus on these new artefacts as soon as they perceive that a new goal was announced. The *task_description* contains an organisational goal, *deadline* is the time (in milliseconds) that the auction will run for, *state* informs if the auction is still open or not, and *winner* is created by the *initiator* once the auction ends with the id of the bid that won the auction.

The operation *bid* is executed by *bidders* in order to place a bid for the goal associated with the artefact. *award* is a linked operation executed by the *initiator*. It updates the *winner* observable property, based on a value function. The *getBids* is a linked operation executed by the *initiator*, returning all bids currently placed by the *bidders*, to be used in the *award* operation.

There are also two internal operations, *checkDeadline* and *checkAllBids*. Both internal operations update the *state* observable property to closed, if the deadline is up or if all agents have already placed a bid. Internal operations are not available to be used by agents or other artefacts. Instead, the artefact's operations themselves can trigger the asynchronous execution of internal operations.

4.3 SHOP2

We did not modify directly any of the SHOP2 code. We provide a *startPlanner* Java class that uses the default Java runtime environment to start a new process that executes an Allegro CL script in order to run SHOP2. The Java class is implemented as an internal action that is executed by Jason agents when they enter their individual planning stage.

4.4 Social Laws

In Figure 5, we show the observable properties and operations of the *SocialLaws* artefact. This artifact is responsible for coordinating the agents during the execution of a solution found during the planning process. It is created during the system's initialisation, one instance for each social law.

The observable properties are: *social_law* contains the name of the social law; *action_name* is the name of the action that is associated with this social law artefact; *precondition_list* is the list of preconditions that make this social law applicable; and *action_options* contains the list of possible actions that an agent may take in order to avoid a conflict, or to solve a dependency.

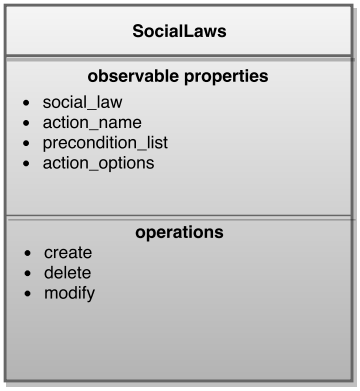


Figure 5: The artefact for social laws.

We also provide operations related to the manipulation of social laws, although there is no mechanism implemented to make use of these operations yet. The *create* operation allows the creation of another instance of *SocialLaws*. The *delete* operation erases the current instance of *SocialLaws*. And the *modify* operation permits to alter the values of the observable properties in the instance of *SocialLaws* that the operation was used.

Regarding the practical usage of the artefact, agents consult the *SocialLaws* artefact associated with the action that they are about to execute. This process is only necessary for actions that are part of the plans adopted as a solution from the planning stage, and only if those actions are annotated with conflict and/or dependency flags.

5 The Floods Domain

The lack of complex multi-agent domains led us to design a new domain, in order to best exploit the advantages of MAP and MAS. The inspiration for this specific domain came from a real-world scenario on using artificial intelligence techniques (e.g., a team of autonomous multi-robots) to help mitigate and prevent natural disasters. This scenario is specifically targeted at flood disasters, often caused by intense hydro-meteorological hazards, that can lead to severe economic losses and in some extreme cases even deaths.

Our domain, the Floods domain, is based on that real-world scenario. Another source of inspiration was the Rover domain, which was used in several past International Planning Competitions (IPCs). In the floods domain, a team of autonomous and heterogeneous robots are dispatched to monitor flood activity in a region. The Centre for Disaster Management (CDM) establishes a base of operation in the region that is being monitored. The base is used to assign goals to the robots, receive and interpret data, and provide some assistance. The CDM is usually operated by humans, but in our JaCaMo+DOMAPS implementation we simulate

them by using agents, capable of creating dynamic goals during run-time.

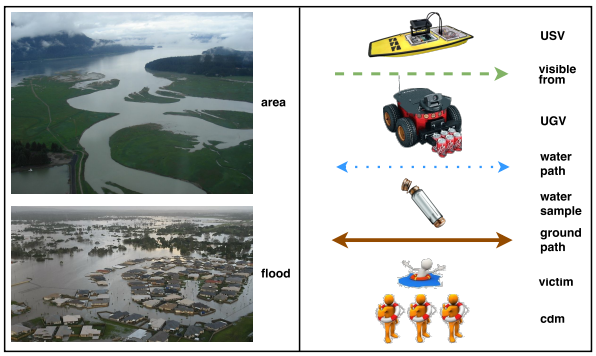


Figure 6: Elements from the Floods domain.

In Figure 6, we show the elements that compose the Floods domain. The domain takes place in a particular region, which is divided into several interconnected *areas*. Movement through the region occurs from traversing these areas. *Flood* events are common in the region, especially during heavy-rain. These floods can be observed from specific areas in the region. The areas can be connected by a *water path*, that can be traversed by naval units, and/or by a *ground path*, that can be traversed by ground units. *Water sample* can be requested to be collected from certain areas. During flood events, *victims* may be detected and in need of assistance. The *CDM* establishes a base of operations in one of the areas in the region.

Finally, the naval units are composed of Unmanned Surface Vehicles (*USVs*) that can move through areas connected by water paths, collect water samples, and take pictures of flood events. Meanwhile, the Unmanned Ground Vehicles (*UGVs*) are ground units that are able to move through areas connected by ground paths, take pictures of flood events, and provide assistance to victims by transporting first-aid kits to first responders close by. The robots can only perceive other robots that are in the same area.

5.1 Initial Experiments

For the initial experiments presented here, we maintained the number of agents and focused on increasing the number of goals. It seems that there is a relation between the number of goals and the number of agents. For most domains, having the number of goals equal to the number of agents, and assuming that each agent is capable of solving its associated goal, appears to result in faster planning times. In Table 2 we show the some initial experiments on this domain for small problems with 4, 8, 16, and 32 goals. As the number of goals surpasses the number of agents, the planning time approximates to that of single-agent SHOP2.

The results are shown in regards to time spent planning, and the number of state expansions and inferences that were made during planning. These results do not depict any of the run-time features of DOMAPS, as we are still investigating how to evaluate it as a whole, and considering what evaluation parameters that could be used both for planning and execution in tandem.

Table 2: Initial experiment results.

	DOMAPS			**SHOP2**
	usv1	usv2	ugv1	
floods 4 pl. time	0.001	0.001	0.001	0.004
exp.	8	8	15	65
inf.	13	13	21	186
floods 8 pl. time	0.001	0.001	0.002	0.011
exp.	15	15	29	129
inf.	21	21	37	360
floods 16 pl. time	0.002	0.002	0.004	0.033
exp.	29	29	57	257
inf.	37	37	69	708
floods 32 pl. time	0.003	0.003	0.005	0.095
exp.	57	57	113	513
inf.	69	69	133	1404

It is natural for DOMAPS to have faster planning times than regular SHOP2 since we assign goals to agents before planning, while SHOP2 does so during planning, in order to try different assignments. In future experiments we want to test scalability and add more domains. We also hope to compare DOMAPS with other frameworks, and provide a full framework evaluation, as well as evaluating each of its components separately.

These experiments show an interesting result in regards to the number of expansions and inferences. DOMAPS requires fewer state expansions and inferences, even if adding all the agents, than SHOP2. The individual planning approach taken in DOMAPS discards many of the predicates that are usually used to assign tasks between different objects, whilst SHOP2 needs those predicates to define the (agent) objects. By considering agents as first-class abstractions in MA-HTN, we are free of the use of these predicates.

6 Related and Future Work

There has been several surveys over the years describing advancements in particular areas of planning. Of interest, and related to this research, there are: in (desJardins et al. 1999), a survey on distributed online (continual) planning is presented, with the state of the art in distributed and online planning at the time, and a conceptual design for distributed online planning; a survey (Meneguzzi and De Silva 2013) that presents a collection of recent techniques used to integrate single-agent planning for a single-agent in BDI-based agent-oriented programming languages, focusing mostly on efforts to generate new plans at run-time; and a multi-agent planning survey (Weerdt and Clement 2009), describing several approaches taken towards multi-agent planning over the last few years.

In (Nissim and Brafman 2014), the authors propose a forward search heuristic for classical multi-agent planning that respects the distributed structure of the system, preserving agents privacy (Brafman 2015). According to their experiments, their system showed the best performance in regards to planning time and communication, as well as the quality of the solution (in most cases), when compared to other offline multi-agent planning systems.

FLAP (Sapena, Onaindia, and Torreño 2015) is a hybrid planner that combines partial-order plans with forward search. The planner uses a parallel search technique that diversifies the search. FLAP exploits delaying commitment to the order in which actions are applicable. This is done to achieve flexibility, reducing the need of backtracking and minimising the length of the plans by promoting the parallel execution of actions. These changes come at an increase in computational cost, but it allows FLAP to solve more problems than other partial-order planner.

In (Clement, Durfee, and Barrett 2007), multi-agent planning algorithms and heuristics are proposed to exploit summary information during the coordination stage, in order to speed up planning time. The authors claim that by associating summary information with plans' abstract operators, it can ensure plan correctness, even in multi-agent planning, while still gaining efficiency and not leading to incorrect plans. The key idea is to annotate each abstract operator with summary information about all of its potential needs and effects. This process often resulted in an exponential reduction in planning time compared to a flat representation. Their approach depends on some specific conditions and assumptions, and therefore cannot be used in all domains.

Multi-Agent Planning Language (MAPL) is proposed in (Brenner and Nebel 2009), for modelling MAP domains in online planning. Plans expressed in this language interleave planning, acting, sensing, and communicating. Their approach is based on sharing knowledge in order to ensure the synchronous execution of joint plans. It is different from our approach, where agents keep their private knowledge and are coordinated through the organisation via social laws. Their interleave mechanism could be used in DOMAPS to shorten the time between planning and execution.

Kovacs proposed an extension for PDDL3.1 that enables the description of multi-agent planning problems (Kovacs 2012) in PDDL. It copes with many of the already discussed open problems in multi-agent planning, such as the exponential increase of the number of actions, but it also approaches new problems such as the constructive and destructive synergies of concurrent actions. Although only the formalism is provided (it is not yet supported by any planner), the ideas expressed by Kovacs are enticing, making it an interesting candidate to add to DOMAPS planning formalisms.

Markov Decision Processes (MDP) are often used when dealing with non-deterministic worlds to coordinate agents during planning. These methods can also be extended to deal with partially observable environments, known as Partially Observable Markov Decision Processes (POMDP). For example, in (Wu, Zilberstein, and Chen 2011), the authors use an online algorithm for planning under uncertainty in multi-agent settings modelled as decentralised POMDPs, requiring little to no communication. While in (Brafman, Shani, and Zilberstein 2013), qualitative decentralised POMDP is proposed as a a qualitative, propositional model for multi-agent planning under uncertainty with partial observability.

In multi-agent POMDPs, the action and observation space grows exponentially with the number of agents. In (Amato and Oliehoek 2015), a scalable approach based on sample-based planning and factored value functions that exploits multi-agent structure to produce a scalable method for Monte Carlo tree search for POMDPs. They formalise a team of agents as a multi-agent POMDP, and introduce an online planner that uses factored statistics and factored trees to reduce the number of joint actions and the number of joint histories considered. Adding these approaches to DOMAPS could allow us to consider domains with non-deterministic actions and partially-observable environments.

Plan repair is the re-use of fragments of an old plan, and can be used to effectively simplify the coordination stage of planning. The authors of (Komenda, Novak, and Pechoucek 2014), argue that in decentralised systems where coordination is required to achieve joint objectives, attempts to repair failed multi-agent plans should lead to lower communication overhead than re-planning from scratch. They also describe three algorithms for domain-independent multi-agent plan repair. At the moment, the re-planning in DOMAS works by restarting the planning process from scratch (but with updated information about the world). Integrating these plan repair algorithms could provide some important improvements to re-planning in DOMAPS.

Techniques for solving multi-agent pathfinding problems are also becoming more common. These problems relate to finding paths from start to goal positions for all agents, while avoiding collisions. For example, in (Sharon et al. 2015), a new multi-agent pathfinding algorithm is presented. The conflict based search algorithm works on two levels: at the high level the search is performed on a conflict tree containing the conflicts between individual agents; and at the low level single-agent searches are performed to satisfy the constraints found at the high level. This conflict tree could be useful for mapping the conflicts for the coordination mechanisms of DOMAPS. Combining path planning with task planning, such as suggested by (Srivastava et al. 2014), could also be useful in order to run real world scenarios with multiple robots.

7 Conclusion

In this paper, we described the design of the Distributed Online Multi-Agent Planning System (DOMAPS). Specifying each of its main components: *i)* the planning formalism – we introduced the MA-HTN formalism, a multi-agent variation of the traditional single-agent HTN formalism; *ii)* the goal allocation mechanism – by using a contract net protocol, the agents that participate in the planning stage can preselect the goals that they believe to be more appropriate to them, this pre-planning can cut the planning time considerably in domains with heterogeneous agents and varied goals; *iii)* the individual planner – the SHOP2 planner is used in each agent for individual planning, so as to make the most of the HTN-like structure of the plan library in Jason agents;

iv) the coordination mechanism – employment of social laws to coordinate the agents during run-time in order to avoid possible conflicts made during planning.

Initial experiments and experience with DOMAPS has presented enough positive incentives to pursue solutions for the limitations and to provide improvements for the framework overall, for example by adding new approaches to each component. The performance of our coordination mechanism is limited to the designer ability of detecting conflicts and formulating suitable social laws, similarly to the way that HTN depends on good methods.

Acknowledgments

We are grateful for the support given by CAPES and by CNPq (grant number 308095/2012-0).

References

Amato, C., and Oliehoek, F. A. 2015. Scalable planning and learning for multiagent pomdps. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 1995–2002.

Boissier, O.; Bordini, R. H.; Hübner, J. F.; Ricci, A.; and Santi, A. 2011. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*.

Bordini, R. H.; Wooldridge, M.; and Hübner, J. F. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons.

Brafman, R. I.; Shani, G.; and Zilberstein, S. 2013. Qualitative planning under partial observability in multi-agent domains. In *Proceedings of the Twenty-Seventh Conference on Artificial Intelligence*, 130–137.

Brafman, R. I. 2015. A privacy preserving algorithm for multi-agent planning and search. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1530–1536.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems* 19(3):297–331.

Cardoso, R. C., and Bordini, R. H. 2016. A multi-agent extension of hierarchical task network. 10th Workshop-School on Agents, Environments, and Applications (WESAAC).

Clement, B. J.; Durfee, E. H.; and Barrett, A. C. 2007. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research (JAIR)* 28:453–515.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A single-agent approach to multiagent planning. In *21st European Conf. on Artificial Intelligence (ECAI'14)*.

desJardins, M. E.; Durfee, E. H.; Ortiz, C. L.; and Wolverton, M. J. 1999. A survey of research in distributed, continual planning. *AI Magazine* 20(4).

Durfee, E. H., and Zilberstein, S. 2013. Multiagent planning, control, and execution. In Weiss, G., ed., *Multiagent Systems 2nd Edition*. MIT Press. chapter 11, 485–545.

Hübner, J. F.; Sichman, J. S.; and Boissier, O. 2007. Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. *Int. J. Agent-Oriented Software Engineering* 1(3/4):370–395.

Komenda, A.; Novak, P.; and Pechoucek, M. 2014. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications* 37:76 – 88.

Kovacs, D. L. 2012. A multi-agent extension of pddl3.1. In *Proceedings of the 3rd Workshop on the International Planning Competition (IPC)*, ICAPS-2012, 19–27.

Meneguzzi, F., and De Silva, L. 2013. Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review* FirstView:1–44.

Nau, D.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Nissim, R., and Brafman, R. I. 2014. Distributed heuristic forward search for multi-agent planning. *J. Artif. Intell. Res. (JAIR)* 51:293–332.

Omicini, A.; Ricci, A.; and Viroli, M. 2008. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 17(3):432–456.

Ricci, A.; Piunti, M.; Viroli, M.; and Omicini, A. 2009. Environment programming in CArtAgO. In *Multi-Agent Programming: Languages, Tools and Applications*, Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer. chapter 8, 259–288.

Sapena, O.; Onaindia, E.; and Torreño, A. 2015. FLAP: applying least-commitment in forward-chaining planning. *AI Commun.* 28(1):5–20.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40 – 66.

Shoham, Y., and Tennenholtz, M. 1995. On social laws for artificial agent societies: Off-line design. *Artif. Intell.* 73(1-2):231–252.

Singh, M., and Chopra, A. 2010. Programming multiagent systems without programming agents. In Braubach, L.; Briot, J.-P.; and Thangarajah, J., eds., *Programming Multi-Agent Systems*, volume 5919 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 1–14.

Smith, R. G. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.* 29(12):1104–1113.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Weerdt, M. D., and Clement, B. J. 2009. Introduction to Planning in Multiagent Systems. *Multiagent Grid Syst.* 5(4):345–355.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence* 175(2):487–511.

Multi-Agent Route Planning Using Delegate MAS

Hoang Tung Dinh, Rinde R. S. van Lon, Tom Holvoet
iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium
{hoangtung.dinh, rinde.vanlon, tom.holvoet}@cs.kuleuven.be

Abstract

Multi-agent route planning (MARP) is a problem that occurs in many applications such as automated guided vehicles, robotics, intelligent transportation networks and airplane taxiing. MARP becomes especially challenging when the application domain is dynamic, large scale and requires continual planning. Due to its decentralized nature, a multi-agent system (MAS) is an ideal candidate for solving dynamic and large scale MARP problems. Delegate MAS is a coordination mechanism based on the idea of intention propagation via the environment inspired by ant behavior. We evaluate delegate MAS on automated guided vehicle routing under realistic conditions. Delegate MAS is compared with context-aware routing, a state-of-the-art centralized approach for dynamic MARP. Two variants of MARP are considered, single-stage where vehicles each have to visit a single destination and multi-stage where a sequence of destinations has to be visited. The experiment results show that delegate MAS and context-aware routing have comparable solution quality while delegate MAS is more scalable for multi-stage routing in dynamic environments and offers higher throughput when continual planning is required.

1 Introduction

Automated guided vehicle (AGV) systems are widely used in many industrial areas, including manufacturing, aviation, retail and transportation logistics (Ullrich 2015). In such systems, a fleet of autonomous vehicles coordinate in order to efficiently transport goods throughout a plant or warehouse. Transportation requests need to be assigned to vehicles and each request contains information about pick-up and delivery locations. Vehicles need to go to battery charging stations several times during execution. The goal of the vehicle fleet is to determine efficient routes that minimize transportation time and maximize throughput. This is a challenging task (Vis 2006). Unexpected obstacles such as humans may block a road; vehicles may temporarily fail; new vehicles may become available and operating vehicles may leave the system for maintenance. This results in unpredictable travel time and transportation requests may change during execution. Additionally, the infrastructure and the number of vehicles can be large; the physical constraints of the infrastructure and vehicles may lead to deadlocks, preventing some vehicles to reach their destinations.

AGV systems, along with robotics, intelligent transportation and airplane taxiing, are typical applications of multi-agent route planning (MARP). In MARP, there is a set of agents in a shared environment. The problem involves planning a conflict-free route for each agent from its current position to one or multiple destinations. Ter Mors (2010, pp. 46–50) proves that MARP is NP-complete, or even PSPACE-complete under additional constraints such as maintaining a minimum distance between agents. Typical challenges of MARP are dealing with dynamism, scalability, communication limitation and deadlock situations. MARP requires a flexible and scalable solution that seamlessly copes with unexpected events and failures. Agents should collaboratively plan routes to resolve conflicts before they occur. These characteristics motivate the feasibility study of a decentralized multi-agent approach.

In this paper we implement and evaluate an online, anytime and continual planning approach for MARP, called delegate MAS. Delegate MAS is an environment centric coordination mechanism for coordination and control applications, inspired by food foraging behavior in ant colonies. It was first introduced in the context of manufacturing control (Holvoet and Valckenaers 2007). Since then, delegate MAS has been applied in different areas such as pick-up and delivery (Hanif et al. 2011) and anticipatory vehicle routing (Weyns, Holvoet, and Helleboogh 2007; Claes, Holvoet, and Weyns 2011). We compare delegate MAS with a state-of-the-art centralized decoupled approach, called context-aware routing (see Section 2). To the best of our knowledge, context-aware routing is the only approach that solves both single-stage (Ter Mors, Zutt, and Witteveen 2007) and multi-stage routing (Ter Mors, Van Belle, and Witteveen 2009), as well as routing in a dynamic environment where unexpected incidents occur regularly (Ter Mors and Witteveen 2009). We evaluate the performance of both approaches in static and dynamic environments, in single as well as multi-stage routing. In single-stage routing, an agent has exactly one destination to go to. In multi-stage routing, an agent has a sequence of destinations. We make abstraction of the way agents get to know their destinations and assume that they are informed when the destinations need to be adapted. The experiment results show that delegate MAS and context-aware routing have comparable solution quality while delegate MAS is more scalable for multi-stage routing in dy-

namic environments and offers higher throughput when continual planning is required.

The rest of this paper is organized as follows. We first discuss related work (Section 2). Then, we formulate the model of the MARP problem (Section 3). After that, we outline the delegate MAS approach (Section 4). We then describe the experiment setup and analyze experiment results (Section 5). Finally, we draw our conclusion and detail possible future work (Section 6).

2 Related Work

There are coupled and decoupled approaches for solving MARP.

Coupled approaches combine the configuration spaces of all individual agents into one composite configuration space which is then searched for a solution. Several coupled approaches (Ryan 2008; Standley and Korf 2011) can solve MARP optimally using the *A** algorithm (Hart, Nilsson, and Raphael 1968). Such approaches do not scale well because the branching factor of search spaces grows exponentially as the number of agents increases. Sharon et al. (2013; 2015) exploit the sparsity in interactions among agents to improve the efficiency in finding the optimal solution. Their approaches perform poorly when there is a high rate of conflicts among agents.

Decoupled approaches decompose the problem of searching for a global solution for all agents into a sequence of individual planning problems. Decoupled approaches offer scalability but are sub-optimal and often incomplete. One type of decoupled approach, called rule-based planning, defines a set of specific movement rules for agents to reduce the computational complexity and guarantee the completeness at the cost of solution quality (De Wilde, Ter Mors, and Witteveen 2013; Wang and Botea 2008). Another type of decoupled approach, prioritized planning, assigns a unique priority to each agent. Agents plan routes in decreasing priority order. An agent finds a route that does not create a conflict with the plans of higher priority agents. Silver (2005) proposed Hierarchical Cooperative *A** and its variant, Windowed Hierarchical Cooperative *A**, where an agent searches for a route in a three-dimensional space-time reservation table. Wang and Goh (2011) proposed an approach where agents search for routes in a two-dimensional map with an adaptive priority re-assignment strategy. These approaches are not complete and their solutions may be far from optimal. Wang and Goh (2013) then proposed the Guided Iterative Prioritized Planning approach that can increase the success rates at the cost of computational time. Hatzack and Nebel (2014), Lee, Lee, and Choi (1998) and Ter Mors et al. (2012) presented the Fixed-Path Scheduling (FPS) approach where each agent, in a given priority, calculates conflict-free schedules along one or multiple predetermined paths and takes the shortest-time schedule as its plan. The pre-determined path(s) for each agent can be the shortest-length path (Hatzack and Nebel 2014), k-shortest-length paths (Lee, Lee, and Choi 1998) or k-disjoint paths (Ter Mors et al. 2012).

Ter Mors, Zutt, and Witteveen (2007) proposed a state-of-the-art prioritized planning approach called context-aware routing. In context-aware routing, an agent finds its optimal plan that does not create a conflict with existing plans of other agents on a free time window graph using an *A**-like algorithm. A free time window on a location is the maximal time interval that a new agent can make a reservation for traversing the location without making any conflict with the existing reservations of other agents. Ter Mors et al. (2007; 2012) show that context-aware routing is better in terms of travel time than all variants of FPS.

All the work we have discussed so far only considers the single-stage routing problem. To the best of our knowledge, the paper by Ter Mors, Van Belle, and Witteveen (2009) is the only work in the literature that deals with the multi-stage routing problem. Extended from their single-stage routing algorithm, Ter Mors, Van Belle, and Witteveen proposed the context-aware multi-stage routing algorithm that is also a prioritized planning approach.

To deal with incidents that delay agents, different approaches (Maza and Castagna 2005; Ter Mors and Witteveen 2009; Ter Mors 2011) have been developed to integrate with context-aware routing. In those approaches, after all agents make their plans, the schedule at each location is converted to a visiting order (or priorities) of agents. Maza and Castagna (2005) show that if agents maintain their priorities at each location, no conflict occurs even if the arrival times of agents are not guaranteed. Hence, an approach to avoid conflict is to let each agent respect its priority at each location. This approach requires non-delayed agents to wait for delayed agents. To achieve better solution, the work in (Maza and Castagna 2005; Ter Mors and Witteveen 2009) increases the priorities of non-delayed agents over delayed ones. In (2011), Ter Mors proposes another approach where agents re-plan routes every time an incident occurs. According to the comparisons in (Ter Mors and Witteveen 2009; Ter Mors 2011), the increasing-priority approach of Ter Mors and Witteveen (2009) achieves the best solution quality.

3 Problem Formulation

In this section we present the formal model used throughout the paper. We adopt the model introduced by Ter Mors, Van Belle, and Witteveen (2009) with several modifications to make the model more realistic. In (2009), Ter Mors, Van Belle, and Witteveen assume that agents have zero length. Such assumption is unrealistic. In our model, we assume that agents occupy physical space. This assumption leads to the differences in the resource capacity constraints and the agent plan constraints between our model and Ter Mors's model.

MARP consists of a set A of agents operating in an infrastructure. The infrastructure is a bidirectional graph $G = (V, E)$, where V is the set of vertices representing locations that agents can visit such as intersections or pick up and delivery locations, and E is the set of edges representing lanes connecting locations. The set of resources is $R = V \cup E$. Each resource $r \in R$ has a capacity $C(r)$. Each vertex has unit capacity. The capacity of each edge e is:

$$C(e) = \lfloor length(e) / \big((1 + \Delta) \times length(a)\big) \rfloor$$

where $length(e)$ is the length of the edge, $length(a)$ is the

length of an agent and Δ is the minimum separation between two agents. We assume homogeneous agents. If multiple agents occupy an edge at the same time, they must all travel in the same direction and must not overtake each other. An agent cannot change its direction when it travels along an edge. In Figure 1, agent 2 is not allowed to overtake and cannot exit from the edge before agent 3.

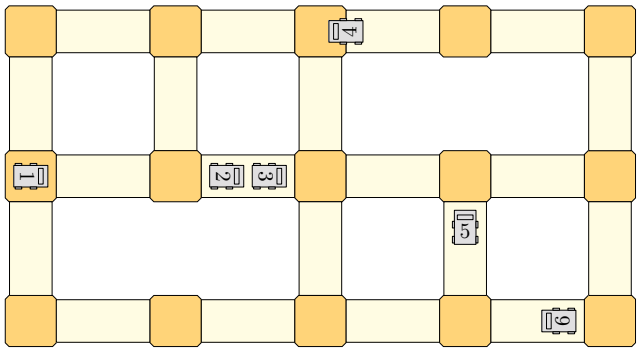


Figure 1: Agents in an infrastructure graph. Dark yellow squares represent vertices (intersections and locations). Light yellow rectangles represent edges (lanes).

In single-stage routing, an agent $a \in A$ has one start location $s \in V$ and one destination location $d \in V$. In multi-stage routing, an agent a has a tuple of destinations $D = \{\langle d_1, \ldots, d_m \rangle | d_i \in V\}$, where m is the number of destinations. The route plan of an agent is a sequence:

$$(\langle r_1, [t_1, t'_1) \rangle, \ldots, \langle r_n, [t_n, t'_n) \rangle)$$

of n plan steps. A plan step $\langle r_i, [t_i, t'_i) \rangle$ consists of a resource r_i, the entry time t_i and the exit time t'_i of agent a on r_i. In single-stage routing, the last plan step of an agent must be in the destination resource, that is, $r_n = d$. In multi-stage routing, the plan of an agent must include all the destinations in a given order and the last plan step must be in the last destination resource, that is, $r_n = d_m$. Two resources r_i and r_{i+1} of two consecutive plan steps must be adjacent in G. In the model of Ter Mors, Van Belle, and Witteveen (2009), because they assume that agents have zero length, each agent only occupies one resource at the same time. Therefore, the exit time and the entry time of two consecutive plan steps must meet each other, that is, $t'_i = t_{t+1}$. In our model, because each agent has a length greater than zero, an agent can occupy two adjacent resources simultaneously. For example, in Figure 1, agent 4 is occupying two resources. Thus, in our model, two intervals of two consecutive plan steps $[t_i, t'_i)$ and $[t_{i+1}, t'_{i+1})$ must overlap, that is, $t'_i > t_{t+1}$. The duration $\delta_i = t'_i - t_i$ must be sufficient for the agent to traverse through the resource r_i. The schedule of a resource consists of a set of plan steps. A resource has a consistent schedule if the load of the resource never exceeds the resource's capacity.

Unexpected incidents that delay agents may occur. We model incidents as events that make agents temporarily inactive. Each incident has a start time t and a duration δ_t. While suffering an incident, an agent cannot move. Agents do not have prior knowledge of incidents, that is, they do not know when incidents happen nor their duration.

4 MARP Using Delegate MAS

In this section we propose the delegate MAS approach for MARP. Delegate MAS consists of a number of autonomous agents situated in a shared environment. Agents coordinate in a decentralized way. The shared environment enables indirect communication between agents. An agent drops information of its plan to the relevant parts of the environment. Other agents can later use the information to create their plans. Such communication somewhat resembles the foraging behavior of ants, where an ant continuously drops pheromones on the environment and scents pheromones of other ants. Agents only collect directly relevant information that is distributed throughout the environment for making decisions. Delegate MAS self-organizes by continuously removing invalid information in the environment.

4.1 Multi-Agent Based Routing

Delegate MAS consists of two types of primary agents, *resource agents* and *vehicle agents* (Weyns, Holvoet, and Helleboogh 2007).

Each resource agent represents a resource of the infrastructure. A resource agent can observe changes such as unexpected obstacles at its resource. The main task of a resource agent is to manage a schedule on its resource and to provide free time windows according to the current schedule. A resource agent only allows a vehicle to enter the resource if it is consistent with the schedule. Also, a resource agent only accepts reservations that are consistent with its existing schedule. A reservation has a time-to-live. If a reservation is not confirmed regularly, the resource agent removes it. A resource agent can communicate with its neighboring resource agents. The network of resource agents establishes a *virtual environment*, that is, a software representation of the physical infrastructure graph.

Each vehicle agent represents an operating vehicle in the infrastructure. When a new vehicle enters the infrastructure, a corresponding vehicle agent is created and assigned to the vehicle. We assume that each vehicle agent knows the static graph structure of the infrastructure, but not the schedules on resources. A vehicle agent is responsible for planning routes and controlling its vehicle towards destinations. A vehicle agent continually explores alternative routes and reserves its intended route. The behavior of a vehicle agent is described in Algorithm 1. To explore routes, first, in line 2, a vehicle agent generates a set of feasible paths[1] from its current location to its destination(s) using the static infrastructure graph (Section 4.4). In line 3, it evaluates the quality of each path by asking relevant resource agents about the existing schedules (Section 4.2). Then, in line 4, it selects the most preferable one among the assessed routes. In line 5, the vehicle agent decides whether to deviate from its current plan to the new route. After that, in line 8, the vehicle agent makes reservations for its intended route and regularly refreshes the

[1] A route is a path with a schedule.

reservations via resource agents (Section 4.2). When planning routes, a vehicle agent must respect the existing reservations of other vehicle agents.

Algorithm 1 Behavior of a vehicle agent

Require: The infrastructure graph $G = (V, E)$, current location $start$, destination(s) $dest$

```
1:  loop
2:      paths ← getAlternativePaths(G, start, dest)
3:      routes ← evaluate(paths)
4:      preferredRoute ← select(routes)
5:      if revise(intendedRoute, preferredRoute) then
6:          intendedRoute ← preferredRoute
7:      end if
8:      if ¬makeReservation(intendedRoute) then
9:          goto 2
10:     end if
11: end loop
```

4.2 Agent Coordination

To achieve coordination, vehicle agents and resource agents communicate indirectly through the virtual environment using lightweight agents, called "ants". Basically, ants are smart messages that can autonomously move in the virtual environment and interact with resource agents (Holvoet, Weyns, and Valckenaers 2009). We use two types of ants: *exploration ants* and *intention ants*.

1) Exploration ants. Recall Algorithm 1. In line 3, a vehicle agent evaluates candidate paths by sending out exploration ants. Each exploration ant follows a candidate path through the virtual environment. For each resource agent that it travels through, the exploration ant queries for the existing schedule. After reaching the destination, based on all relevant reservations, the exploration ant calculates the optimal schedule along its path using the context-aware routing algorithm proposed in (Ter Mors, Zutt, and Witteveen 2007). Note that we only use context-aware routing to search for the optimal schedule on a given path, but not on the entire infrastructure graph as in (Ter Mors, Zutt, and Witteveen 2007). The exploration ant then informs the vehicle agent of its optimal schedule. In line 4, the vehicle agent compares all schedules reported by exploration ants and selects the best one. The criteria for the selection depend on concrete objectives. In this paper, a vehicle agent selects the schedule with the earliest arrival time at the destination.

2) Intention ants. In line 8 of Algorithm 1, a vehicle agent informs relevant resource agents of its plan by sending out an intention ant. The intention ant follows the intended path in the virtual environment and makes reservation with each resource agent along its path. Then, it reports to the vehicle agent whether it successfully made reservations on all resources. If the intention ant cannot reserve all the resources, the vehicle agent explores again (line 9). The reservations evaporate after a while if intention ants do not renew them. The vehicle agent therefore sends out intention ants regularly to refresh its reservations. A vehicle agent can freely change its intention. In line 5, if it finds a better route and decides to change the current plan, it stops sending intention ants on the current route and the incorrect reservations will disappear after a while. This is an example of the self-organizing property of delegate MAS. A vehicle agent repeats the exploration and reservation processes regularly, thus gradually improves the quality of its plan.

Different from context-aware routing that only allows agents to plan sequentially, delegate MAS allows agents to plan concurrently and can resolve conflicts caused by simultaneous exploring and reserving. If two intention ants arrive and request to make reservations at a resource at the same time, the resource agent randomly selects an order to process the requests. For example, in Figure 2, vehicle agent 1 and vehicle agent 2 explore routes at the same time. At that moment, no vehicle agent already placed reservations. Hence, after exploring, vehicle agent 1 selects the solid blue route and vehicle agent 2 chooses the dashed red route. They plan to travel with the same speed and therefore may collide at the resource marked by a star. They then send intention ants to make reservations. At the star resource, assuming that intention ant 1 arrives earlier, or at the same time with intention ant 2 but is selected by the resource agent to process request first. The resource agent accepts the reservation of intention ant 1 and then rejects the reservation of intention ant 2 because of schedule inconsistency. Intention ant 1 then continues making reservations while intention ant 2 stops and reports to vehicle agent 2. Vehicle agent 2 then explores new routes again. The reservations made by intention ant 2 in other resources are not refreshed and evaporate after a while thanks to the self-organizing effect.

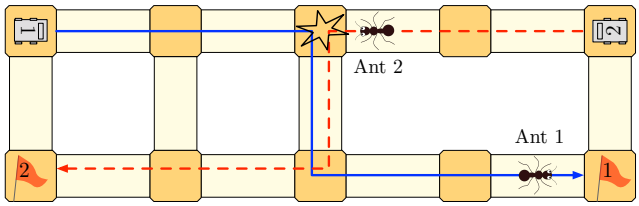


Figure 2: Two vehicle agents explore simultaneously and try to reserve resources for two conflicted plans. The resource agent at the first conflicted resource processes requests sequentially. Thus it only allows the reservation from intention ant 1 and rejects the reservation from intention ant 2.

4.3 Delay Propagation

If an incident occurs, the delayed vehicle agent estimates the delay duration δ. It then updates its current plan:

$$(\langle r1, [t1_s, t1_e)\rangle, \langle r2, [t2_s, t2_e)\rangle, \ldots, \langle rn, [tn_s, tn_e)\rangle)$$

to a new plan:

$$(\langle r1, [t1_s, t1_e + \delta)\rangle, \langle r2, [t2_s + \delta, t2_e + \delta)\rangle, \ldots, \langle rn, [tn_s + \delta, tn_e + \delta)\rangle)$$

where $r1$ is the resource that the vehicle is occupying. After that, the vehicle agent sends an intention ant to request relevant resource agents to update its reservations.

The updated reservations of a delayed vehicle may be inconsistent with the existing reservations of other vehicles. To resolve inconsistencies, a resource agent also delays the reservations of vehicles that enter the resource after the delayed one (see Figure 3a). The resource agent notifies its neighboring resource agents who apply the delay to all affected and succeeding vehicles recursively (see Figure 3b). Consequently, the changes propagate through the entire virtual environment. This delay propagation process guarantees that all updated plans are consistent and deadlock-free if the initial plans are consistent and deadlock-free. A vehicle agent updates its plan using the information reported by intention ants. After the delay is propagated, some non-delayed vehicles have to wait for the delayed one. Because the vehicle agents send out exploration ants regularly, alternative routes can be found around the delayed vehicle.

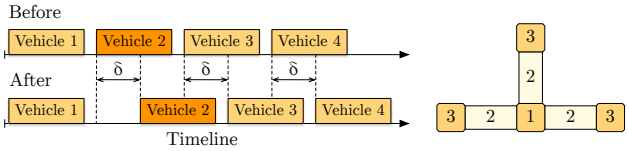


Figure 3: (a) The schedule of a resource agent before and after being requested to update the reservation of vehicle 2. The reservation of vehicle 2 and all succeeding reservations are delayed by $\boldsymbol{\delta}$. (b) Delay propagation process. Resource agent 1 receives a notification of delay. It recursively propagates the delay to its neighbors (marked by 2 and 3).

The estimated delay duration δ does not need to be the exact actual duration. However, δ should be as close to the actual delay duration as possible. Vehicle agents do not need to know the actual duration of an incident in advance, which is realistic in real-world applications. If the actual delay duration is shorter than δ, the vehicle agent explores new routes after the incident is over. If the actual incident duration is longer than δ, the vehicle agent, after estimation δ expires, estimates a new delay duration δ' and propagates it again. The process repeats until the incident is over. The more accurate δ is, the less disturbance in the plan of other vehicle agents. If an intention ant is making new reservations during the delay propagation process, the reservations may conflict with updated resource schedules. Resource agents then reject the inconsistent reservations of the intention ant. The intention ant reports the reservation failure to the vehicle agent and the vehicle agent explores again.

4.4 Alternative Path Finding

Generating possible candidate routes using the static infrastructure graph (line 2 of Algorithm 1) is an important step in exploration. Poor candidate paths lead to poor plans. Finding all paths between two locations is impractical. If we only select the shortest paths as candidate paths, congestion may occur in some central resources that have many shortest paths passing through (Ter Mors et al. 2012). However, long paths result in long travel time. Thus, it is necessary to have a diverse set of candidate paths. Moreover, because a vehicle agent explores regularly, the set of candidate paths should be different from time to time in order to increase the chance of finding a good plan.

In the literature, popular approaches to generate a set of feasible paths include k-shortest paths (Yen 1971), k-disjoint paths (Suurballe and Tarjan 1984), Pareto (Delling and Wagner 2009), Plateau (Bader et al. 2011) and Penalty (Chen, Bell, and Bogenberger 2007). For the details of these approaches, we refer readers to the review in (Bader et al. 2011). In lattice-like structures that are popular in warehouses or harbors, k-shortest paths are often similar. In k-disjoint paths, the set of paths depends much on the shortest path that is also the first path in the set. Bader et al. (2011) show that Pareto leads to solutions of low quality while Plateau and Penalty give comparable good results. We select the Penalty approach with some modifications because it is simpler than the Plateau approach.

The Penalty approach iteratively calculates the shortest path using the A^* algorithm. In each iteration, it adds the shortest path to the result set and penalizes the path by increasing edge weights. The approach terminates after receiving a sufficient number of alternative paths. In (2007), Chen, Bell, and Bogenberger, suggests increasing all edges in each found path by a relative penalty. We adopted their suggestion but preliminary experiments provide poor results. Therefore, we adapt the Penalty approach by adjusting the weight using a more probabilistic way. Algorithm 2 describes our alternative path finding algorithm.

Algorithm 2 Alternative Path Finding

Require: Infrastructure graph $G = (V, E)$, number of alternative paths N, current location $start$, destination(s) $dest$

1: $S \leftarrow \emptyset$
2: $numFails \leftarrow 0$
3: $maxFails \leftarrow 3$
4: $\alpha \leftarrow$ uniformly random value between 0 and 1
5: **while** sizeOf(S) $< N \wedge numFails < maxFails$ **do**
6: $p \leftarrow$ getShortestPath($G, start, dest$)
7: **if** $p \in S$ **then**
8: $numFails \leftarrow overlap + 1$
9: **else**
10: $S \leftarrow S \cup p$
11: $numFails \leftarrow 0$
12: **end if**
13: **for all** vertex $V_i \in p$ **do**
14: $\beta \leftarrow$ uniformly random value between 0 and 1
15: **if** $\beta < \alpha$ **then**
16: **for all** edge E_i directly connected to V_i **do**
17: weight(E_i) $\leftarrow penalty$
18: **end for**
19: **end if**
20: **end for**
21: **end while**
22: **return** S

In line 1, we initialize the solution set S. The variable $numFails$ in line 2 counts the current number of consecu-

tive times that the algorithm fails to find a new path. In line 4, we generate a uniformly random number α between 0 and 1. The while loop of line 5 iterates until we find a sufficient number of alternative paths or the algorithm fails to find a new path $maxFails$ times consecutively[2]. First, in line 6, we calculate the shortest path p using the $A*$ algorithm. Then, we check whether path p is already in the solution set S in line 7. If it is the case, we increase $numFails$ by one. If p is not in S yet, we add p to S and reset $numFails$ to zero. To penalize each found path p, the for loop in line 13 iterates over all the vertices belonged to p. For each vertex V_i, with probability α, we set the weight of all edges directly connected to V_i to $penalty$, a large value in comparison with the initial weight. If α is large, resulting paths are different from each other. If α is small, they tend to be similar. We generate different α value for each exploration process.

In single-stage routing, a vehicle agent calculates the shortest path between its current position and its only destination using the $A*$ algorithm (line 6 of Algorithm 1). In multi-stage routing, because a vehicle agent has multiple destinations $d_1, d_2, \ldots, d_n$, it calculates the shortest path p by concatenating the shortest paths between each pair of destinations. Using the $A*$ algorithm, the vehicle agent calculates n shortest paths: path p_1 between the origin and d_1, path p_2 between d_1 and d_2, ..., path p_n between d_{n-1} and d_n. Then, the candidate path p is the concatenation of all the shortest paths $p_1, p_2, \ldots, p_n$.

5 Experiments

In this section, we compare delegate MAS with context aware routing in single-stage and multi-stage routing scenarios, in static and dynamic scenarios, and in scenarios where vehicles receive new destinations continually. We implemented all algorithms in RinSim (Van Lon and Holvoet 2012), version 4.00 (Van Lon 2015), an open-source discrete-time simulator for logistics that supports MARP. We conducted experiments on a machine with four Intel Xeon X5677 3.47GHZ processors and 12GB RAM.

5.1 Experiment Setup

We evaluate the two approaches on a 30×30 lattice infrastructure with 900 vertices and 1740 edges. Each edge has capacity of two. In total, there are 2640 resources. Each vehicle has a speed of one meter per second.

In single-stage routing, we assume that at the beginning, all vehicles did not enter the infrastructure yet. A vehicle can decide when to enter the infrastructure at its origin. After reaching its destination, a vehicle leaves the infrastructure. In multi-stage routing, we assume that each vehicle has its own parking place. A parking place is a terminal vertex connected to the infrastructure and has unit capacity. A parking place of a vehicle cannot be the destination of other vehicles. Also, a vehicle cannot visit the parking places of other vehicles. The multi-stage routing scenarios reflect the AGV routing problem, where a vehicle cannot leave or enter the infrastructure at every resource. A vehicle always includes its parking place as the last destination in its plan. Initially, each vehicle stays at its parking place. We assign three different destinations for each vehicle. After receiving destinations, a vehicle starts from its parking place, visits all three destinations in the given order and then comes back to the parking place. Therefore, a vehicle has to plan for visiting four destinations (the last destination is its parking place). Figure 4 illustrates a small lattice infrastructure with four vehicles at their parking places.

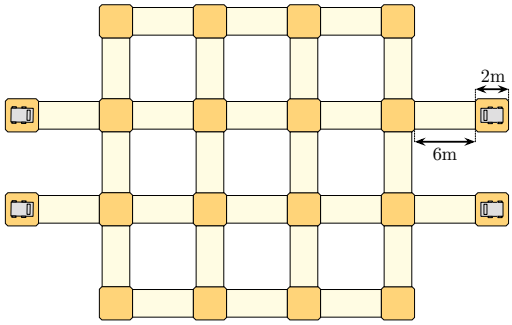


Figure 4: A $\mathbf{4} \times \mathbf{4}$ lattice graph structure with four vehicles at their parking places.

In static scenarios, we compare delegate MAS with context-aware single-stage routing (CA) (Ter Mors, Zutt, and Witteveen 2007) and context-aware multi-stage routing (CA) (Ter Mors, Van Belle, and Witteveen 2009). In such scenarios, there is no disturbance that makes the operation of a vehicle deviate from its plan. Hence, the traveling time of a vehicle is always consistent with its reservations. In dynamic scenarios, we compare delegate MAS with (1) the baseline approach (CA-Baseline) where the priorities of vehicles on a resource never change and with (2) the increasing-priority approach (CA-IP) (Ter Mors and Witteveen 2009), which is the best strategy that vehicles can employ to deal with incidents according to the studies in (Ter Mors and Witteveen 2009; Ter Mors 2011) (see Section 2). We use a homogeneous Poisson process to generate 20 incidents per 10000 seconds for each vehicle. The duration of an incident is a uniformly random value between one and 100 seconds. If two incidents overlap, they form a longer incident. If a vehicle suffers from an incident, it cannot move until the incident is over.

For each setting combination (static / dynamic, single / multi-stage), we vary the number of vehicles from 10 to 100 with steps of 10 and generate 10 problem instances per number of vehicles. A problem instance consists of a start location, a destination (single-stage routing) or a sequence of destinations (multi-stage routing), and a list of incidents (the list is empty in static scenarios) for each vehicle. A vehicle knows its destination(s) at the beginning. We stop a simulation after all vehicles reached their destinations and measure the *average travel time*:

$$\overline{\tau} = \frac{\sum_{i=1}^{|A|} \tau_i}{|A|}$$

where $|A|$ is the number of vehicles and τ_i is the travel time of vehicle i. In single-stage routing, τ_i is the time when ve-

[2]It can be the case that the graph does not have enough alternative paths.

hicle i first reaches its destination. In multi-stage routing, τ_i is the time when vehicle i arrives at its parking place after reaching all of its destinations in a given order. In the most realistic setting, multi-stage routing in dynamic environments, we also measure the *running time* of each delegate MAS simulation and CA-IP simulation.

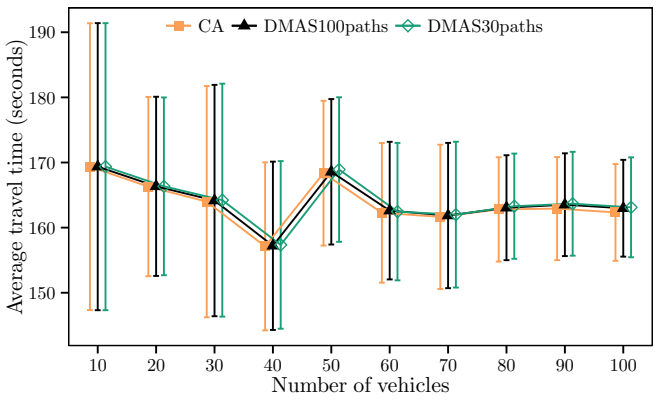
To evaluate our approach on an AGV routing scenario, where vehicles receive new destinations regularly in a dynamic environment, we compare delegate MAS to CA-IP. In the AGV routing problem, new transportation tasks may appear continually during runtime. Hence, in this scenario, at the beginning, we assign three different destinations for each vehicle. After a vehicle reached all the destinations in a given order and came back to its parking place, we assign it three new destinations. A simulation stops after a specified period (10000 seconds). We measure the *throughput*, that is, the total number of reached destinations by all vehicles after the simulation stops. In reality, the throughput represents the number of tasks completed by the system.

In each problem instance, we execute delegate MAS with two different settings where a vehicle agent generates 30 and 100 alternative paths each time it explores new routes. A vehicle agent explores new routes and refreshes its reservations with a period of eight seconds.

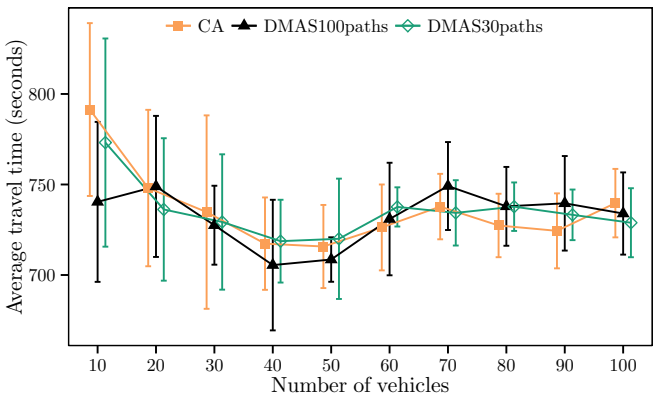
5.2 Experiment Results

Figure 5 shows the *average travel time* of delegate MAS and context-aware routing. In static scenarios, delegate MAS and CA provide comparable results. In static single-stage routing (Figure 5a), CA and delegate MAS achieve similar average travel time. We observed that there are only few interference interactions among vehicles in this scenario. Therefore, the shortest-time paths are often also the shortest-length paths. In delegate MAS, when exploring new routes, a vehicle agent always includes the shortest-length path in the set of alternative paths. Hence, delegate MAS achieves similar performance to context-aware routing. In static multi-stage routing (Figure 5b) where there are more interactions among vehicles, no approach is consistently superior to the other. The reasons that delegate MAS and CA have comparable results while CA computes single-agent optimal route and delegate MAS only samples several routes in the environment can be explained as follows. CA computes single-agent optimal routes sequentially leading to a global Pareto-optimal solution and there is no guarantee about global optimality. Delegate MAS regularly samples the environment and its solution also gradually converges to a Pareto-optimum.

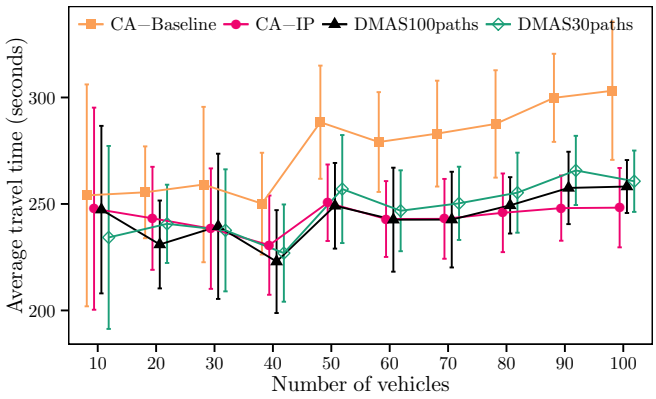
In dynamic scenarios (Figure 5c and 5d), both delegate MAS and CA-IP achieve lower average travel time than that



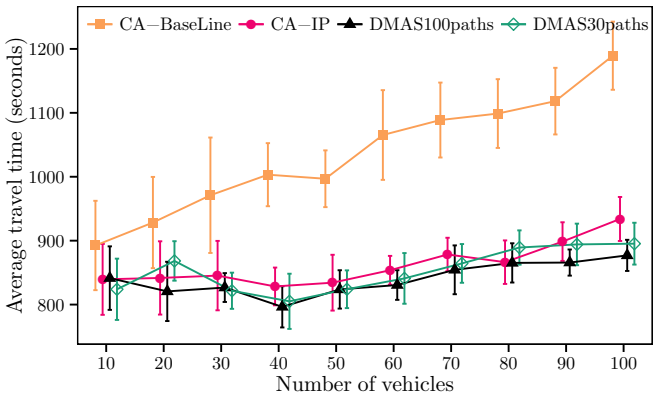
(a) Static single-stage routing



(b) Static multi-stage routing



(c) Dynamic single-stage routing



(d) Dynamic multi-stage routing

Figure 5: Average travel time per vehicle of each approach in different scenarios. Each data point is the average of results from 10 independent simulations.

of CA-Baseline. As the number of vehicles increases, the average travel time of CA-Baseline goes up while those of delegate MAS and CA-IP remain stable. The trend is more obvious in multi-stage routing in comparison with single-stage routing. It can be explained that CA-Baseline requires non-delayed vehicles to wait for delayed vehicles. The more incidents occur, the more delay that vehicles suffer from. As the number of vehicles increases, more incidents happen. Also, there are more incidents if vehicles travel in longer routes. Delegate MAS and CA-IP have comparable results. In single-stage routing, the plan cost of CA-IP tends to be lower than that of delegate MAS as the number of vehicles increases. In multi-stage routing, delegate MAS with 100 alternative paths consistently achieves lower mean than CA-IP, except for the 10-vehicle setting. However, the difference is not significant.

Figure 6 shows that in dynamic multi-stage routing, as the number of vehicles increases, the *running time* of CA-IP increases super-linearly while that of delegate MAS only increases linearly. Moreover, delegate MAS offers a trade-off between running time and solution quality. Decreasing the number of alternative paths for exploration yields a faster running time at the cost of plan quality. From these experiments, we conclude that delegate MAS is more scalable while providing comparable results with CA-IP.

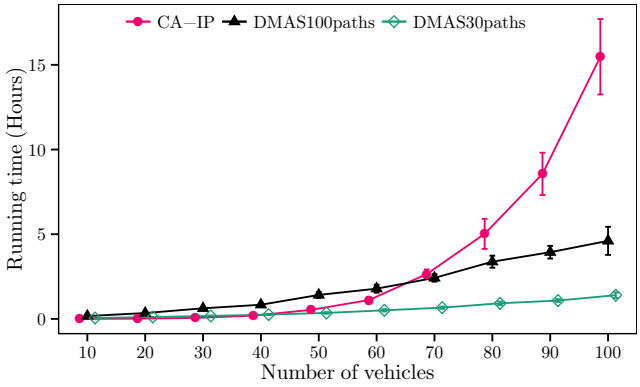


Figure 6: Simulation running time of the dynamic multi-stage routing scenario.

In the last experiment where vehicles receive new destinations continually, the result in Figure 7 shows that delegate MAS achieves higher *throughput* than CA-IP. It can be explained that CA-IP only considers the priorities of vehicles on each resource. After changing the visiting order of vehicles, the existing reservations are invalid. Therefore, a vehicle cannot plan a new route or a new vehicle cannot make a plan when other vehicles are still operating. After a vehicle finishes its tasks and comes back to its parking place, it has to wait until all other vehicles complete their tasks and arrive at their parking places so that all vehicles can start making new plans together. In delegate MAS, the self-organizing capability guarantees the validity of reservations. Therefore, a vehicle can plan a new route at any time. After being assigned new destinations, a vehicle can immediately make a plan without having to wait for other vehicles. Thus, delegate MAS can deal with application domains where vehicles have to plan new routes, new vehicles enter the infrastructure or vehicles have to change destinations during execution.

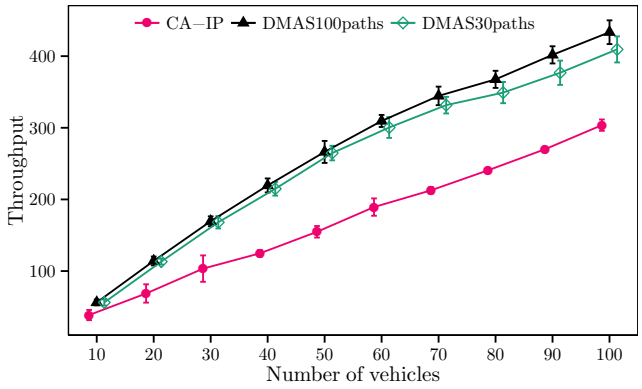


Figure 7: Throughput in the dynamic multi-stage routing scenario after 10000 seconds.

6 Conclusion

In this paper we present the delegate MAS approach for MARP. Delegate MAS can solve single-stage as well as multi-stage routing, continual routing and routing in a dynamic environment. In delegate MAS, the control is decentralized and the system self-organizes to adapt to changes in the environment, thus allowing concurrent activities of different agents. Moreover, it does not require global knowledge about the environment for each agent. Therefore, delegate MAS can be physically deployed and operated as a distributed software system.

Our evaluation is more realistic than previous work. In comparison with a state-of-the-art centralized decoupled approach, delegate MAS provides comparable solution quality while it offers better scalability in a dynamic environment. Moreover, delegate MAS achieves higher throughput when vehicles are required to plan new routes continually. Delegate MAS also offers a trade-off between computational complexity and solution quality.

Our future work will focus on tuning parameters for the exploration process, especially for the alternative path finding algorithm. We also plan to incorporate negotiation mechanisms in our approach. For example, when exploring routes, a vehicle agent may negotiate with other vehicle agents to achieve lower global cost. We will extend delegate MAS so that it can cope with both routing and charging tasks in the AGV transportation problem.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven. We thank Adriaan ter Mors for sharing his implementation of the context-aware routing algorithms.

References

Bader, R.; Dees, J.; Geisberger, R.; and Sanders, P. 2011. Alternative Route Graphs in Road Networks. In *Theory and Practice of Algorithms in (Computer) Systems*. 21–32.

Chen, Y.; Bell, M.; and Bogenberger, K. 2007. Reliable Pretrip Multipath Planning and Dynamic Adaptation for a Centralized Road Navigation System. *IEEE Transactions on Intelligent Transportation Systems* 8:14–20.

Claes, R.; Holvoet, T.; and Weyns, D. 2011. A Decentralized Approach for Anticipatory Vehicle Routing Using Delegate Multiagent Systems. *IEEE Transactions on Intelligent Transportation Systems* 12:364–373.

De Wilde, B.; Ter Mors, A. W.; and Witteveen, C. 2013. Push and rotate: cooperative multi-agent path planning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 87–94.

Delling, D., and Wagner, D. 2009. Pareto paths with SHARC. In *Experimental Algorithms*. 125–136.

Hanif, S.; van Lon, R. R. S.; Gui, N.; and Holvoet, T. 2011. Delegate MAS for Large Scale and Dynamic PDP: A Case Study. In *Intelligent Distributed Computing V*. 23–33.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4:100–107.

Hatzack, W., and Nebel, B. 2014. The operational traffic control problem: Computational complexity and solutions. In *Sixth European Conference on Planning*.

Holvoet, T., and Valckenaers, P. 2007. Exploiting the Environment for Coordinating Agent Intentions. In *Environments for Multi-Agent Systems III*. Springer Berlin Heidelberg. 51–66.

Holvoet, T.; Weyns, D.; and Valckenaers, P. 2009. Patterns of delegate mas. In *Self-Adaptive and Self-Organizing Systems, 2009. SASO'09. Third IEEE International Conference on*, 1–9. IEEE.

Lee, J. H.; Lee, B. H.; and Choi, M. H. 1998. A real-time traffic control scheme of multiple AGV systems for collision free minimum time motion: a routing table approach. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 28:347–358.

Maza, S., and Castagna, P. 2005. A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles. *Computers in Industry* 56:719–733.

Ryan, M. R. K. 2008. Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research* 497–542.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence* 195:470–495.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Silver, D. 2005. Cooperative Pathfinding. In *AIIDE*, 117–122.

Standley, T., and Korf, R. 2011. Complete Algorithms for Cooperative Pathfinding Problems. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*.

Suurballe, J. W., and Tarjan, R. E. 1984. A quick method for finding shortest pairs of disjoint paths. *Networks* 14:325–336.

ter Mors, A., and Witteveen, C. 2009. Plan Repair in Conflict-Free Routing. In *Next-Generation Applied Intelligence*. 46–55.

ter Mors, A.; Witteveen, C.; Ipema, C.; de Nijs, F.; and Tsiourakis, T. 2012. Empirical Evaluation of Multi-Agent Routing Approaches. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 2, 305–309.

ter Mors, A.; Van Belle, J.; and Witteveen, C. 2009. Context-aware multi-stage routing. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 49–56.

ter Mors, A.; Zutt, J.; and Witteveen, C. 2007. Context-Aware Logistic Routing and Scheduling. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 328–335.

ter Mors, A. W. 2010. *The world according to MARP: Multi-Agent Route Planning*. Ph.D. diss. Delft: Technische Universiteit Delft.

ter Mors, A. 2011. Conflict-free route planning in dynamic environments. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2166–2171.

Ullrich, G. 2015. Modern Areas of Application. In *Automated Guided Vehicle Systems*. 15–96.

van Lon, R. R. S., and Holvoet, T. 2012. RinSim: a simulator for collective adaptive systems in transportation and logistics. In *Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on*, 231–232.

van Lon, R. R. S. 2015. RinSim: v4.0.0. doi:10.5281/zenodo.27360.

Vis, I. F. 2006. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research* 170:677–709.

Wang, K. H. C., and Botea, A. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding. In *ICAPS*, 380–387.

Wang, W., and Goh, W. B. 2011. Spatio-temporal A* algorithms for offline multiple mobile robot path planning. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, 1091–1092.

Wang, W., and Goh, W. B. 2013. Time optimized multi-agent path planning using guided iterative prioritized planning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 1183–1184.

Weyns, D.; Holvoet, T.; and Helleboogh, A. 2007. Anticipatory Vehicle Routing using Delegate Multi-Agent Systems. In *IEEE Intelligent Transportation Systems Conference, 2007. ITSC 2007*, 87–93.

Yen, J. Y. 1971. Finding the K Shortest Loopless Paths in a Network. *Management Science* 17:712–716.

A Game Theoretical Formulation of a Decentralized Cooperative Multi-Agent Surveillance Mission

Paulo E. U. de Souza, Caroline P. C. Chanel
Univesité de Toulouse – ISAE-SUPAERO
Institut Supérieur de l'Aéronautique et de l'Espace
31055 Toulouse Cedex 4, FRANCE
name.surname@isae.fr

Sidney Givigi
Royal Military College of Canada
PO Box 17000, Station Forces
Kingston, Ontario, CANADA
sidney.givigi@rmc.ca

Abstract

This paper presents a multi-aerial-robot coordination game theoretical approach to perform a surveillance mission in a well-structured environment. Such a mission consists in constantly visiting a set of points of interest while minimizing the time interval between successive visits (idleness). The proposed approach optimizes the agents' action selection based on an N-player (cooperative) game framework. The main contributions are: (i) the formulation of an original player's utility function composed of parameters that are independent from the action choices of the others players; (ii) the demonstration that the game solution is the Nash equilibrium, and this equilibrium can be obtained by optimizing separately/individually the single player's action choice; (iii) the proposal of a decentralized algorithm used to conduct the mission, which works considering minimum communication among players. Simulations evaluate the different policies obtained, which are compared using as metric the average idleness of all points of interest. The proposed framework allows for the decrease of the idleness of watched points compared to random action selection, while keeping some kind of randomness of motion (measured by a predictability metric), which can likely be desired to curb the prediction of the team surveillance strategy by an intruder.

Introduction

The recent advancement in decision making techniques for aerial robots, also known as drones, has significantly increased the number of applications for a team of autonomous agents. In certain scenarios, multi-robot systems are more desirable than a single robot due to their robustness, stability, adaptability, and scalability (Meng 2008). For instance, search and rescue missions (Murphy et al. 2008; Suarez and Murphy 2011; Xue, Zeng, and Zhang 2011), autonomous infrastructure inspection (Scaramuzza et al. 2014), or autonomous patrolling systems (Amigoni, Basilico, and Gatti 2009; Portugal and Rocha 2011; Hernández et al. 2013).

In particular, the multi-aerial-robot autonomous patrolling or surveillance problem is very challenging: the robots must navigate through the environment so different locations that are scattered in the operational space, and they have to coordinate their actions in order to optimize the time spent to cover all the desired points of interest (Portugal and Rocha 2013a). One of the key issues of a surveillance mission for a multi-robot system is how to coordinate their behaviors in order to optimize the global performance (Meng 2008). For example, monitoring an area of interest requires that the robots move repeatedly through the environment, and the difficulty is to decide on the paths while optimizing some performance criteria (Pasqualetti, Franchi, and Bullo 2010). Moreover, since surveillance implies the maximization of the number of visits to each node in a given environment, a good surveillance strategy must reduce the time interval between visits to the same location (Chevaleyre 2004).

With the aim of evaluating surveillance stategies, a comparative study using distinct topological environments and different team sizes is presented in (Portugal and Rocha 2013b). This work analyzes the performance and scalability of each patrolling approach. For that, (Portugal and Rocha 2013b) proposed as an evaluation metric the *average idleness of the graph* (Idl_G). In the same point of view, (Chevaleyre 2004) demonstrates that minimizing *worst idleness* will also lead to a smaller average idleness. In any case, the smallest the idleness, the better is the performance.

Another key point argued by some authors is that, for security reasons, it should be suitable to consider irregular time intervals to perform visits on desired locations while optimizing the strategy, in order to avoid that a potential intruder could observe the movement of the patrol members for some time and derive an accurate belief of their strategies (Hernández et al. 2013; Amigoni et al. 2010). The key idea is to make it more difficult for an intruder to predict the motion strategy of the team members.

In this kind of surveillance application, it is well known that the minimal refresh time patrolling problem is *NP-hard* (Pasqualetti, Franchi, and Bullo 2010; Portugal and Rocha 2011; Zhang and Kingston 2015). This means that to update the state of each position at each time step would be computational and mem-

ory expensive and impractical in real-world scenarios (Meng 2008; Portugal and Rocha 2011). This is so because in order to improve the efficiency of the collective searching strategy, the action of each robot does not only depend on its own situation, but also on other robots decisions.

In this sense, recent papers have based their approaches on *Game Theory* (Amigoni et al. 2010; Hernández et al. 2013; Meng 2008; Peshkin et al. 2000; An et al. 2012; Khan 2007), which is an elegant way to model an agent's decision making process based on the others agents decisions in a decentralized and distributed way. An example of such a Game Theory application is presented in (Hernández et al. 2013), where Game Theory models of the multi-robot patrolling problem are solved with the use of dynamic and decentralized collaborative approach. Another interesting solution is proposed by (Amigoni et al. 2010) based on Game Theory, which develops a surveillance strategy to drive mobile robots around a known environment in order to avoid intrusions while implementing a non deterministic strategy for their movements in order to make more difficult the task of intruders for they do not know a priori the stochastic distribution of such motions. Others examples can be found in: (Meng 2008), which proposed an N-agent cooperative nonzero-sum game to achieve an optimal overall robots behaviors; (Peshkin et al. 2000) described a gradient-descent policy-search algorithm for cooperative multi-agent domains, where they all share a common payoff; and, (An et al. 2012) that investigated the use of zero-sum games for the protection of critical infrastructures.

For the purpose of a cooperative surveillance mission based on Game Theory, this work addresses the problem of monitoring a closed area by a team of drones minimizing the time to revisit the points of interest (idleness) while keeping some kind of randomness of motion in order to render movements less predictable. Note that, this is neither a coverage problem nor a adversarial problem, but a mix of them. The issue is the development of a dynamic and decentralized approach to multi-aerial-robot cooperation in order to solve the patrolling problem by implementing game theoretical models. In this sense, the main contributions of this work are:

- the formulation of an original player's utility function composed by three parameters that are independent from the action choices of the others players;
- the demonstration that the game solution is a Nash equilibrium, and that this equilibrium can be obtained by optimizing separately and individually the single player's action choice;
- the proposal of a decentralized algorithm used to conduct the mission, which works considering minimum communication among players.

In other words, an original heuristic utility function is presented, where not only the path travel cost is considered, but also the current positions of the other players and the last time since each point of interest was visited. And, based on this utility function, a coordinated game is generated, where the Nash Equilibrium solution guides the player's behavior toward the team goal. In order to reduce the computational complexity the following approach for the solving algorithm is proposed: (1) a fixed path between nodes in the graph and its cost are generated off-line considering the graph does not change during the mission; and (2) the communication between agents and a new game occur only when the destination of each drone has been achieved, instead of at every time step (i.e. the communication is asynchronous).

This work is organized as follows: the considered surveillance problem, its game formulation and the decentralized algorithm proposed to solve this game is presented in Section . Simulation experiments results are shown in Section to evaluate the parameters of the single player's utility function, and their different policies are compared using as metric the average idleness of all points of interest and the overall randomness of the aerial robots' movements. Finally conclusions and future work are discussed in Section .

Problem formulation

This mission can be defined as a frequent visitation problem of all preset points for an aerial robot (here also called drone) team in the lowest possible time interval without having a cycling behavior in order to make the motion model less predictable.

The idea of this paper is to present a method of coordination between drones, based on Game Theory, that is capable of carrying out a monitoring mission on a known *topological model* represented as a graph $\mathbb{G} = (\mathbb{S}, \mathbb{E})$. In this graph $\mathbb{G}$, $\mathbb{S}$ is the set of nodes representing the points of interest in the environment (i.e. positions), and where the edges $\mathbb{E} \subseteq \mathbb{S} \times \mathbb{S}$ define adjacency relationships between the nodes $\mathbb{S}$, i.e., the possible paths between positions or points of interest. Each edge has a cost that represents the time required to move from one node to another. These costs are fixed.

To define the game problem, some assumptions were taken:

- For simplicity, time was discretized in turns;
- Each node can be considered as a point of interest that should be observed, i.e. looking for an intruder;
- Each destination node is a point where the communication among the drones team arises.
- Each drone will select, only once it reaches its destination point, the next point to visit, based on the available information of the others. This means that a new action selection problem will be considered by a drone only when this one has reached the destination point, instead of each time step;
- The drones are defined as "Conscientious Cognitive" agents (Portugal and Rocha 2011), i.e., they choose the next point to visit in the global graph, instead of

in their neighborhood. So, at each time interval, each drone can move from one node to another adjacent, without necessarily selecting a new point of interest;

- All drones have perfect knowledge of the graph model, of their own positions in the graph, the last position informed by the others and their destinations in the graph;
- We assume that each drone can avoid obstacles and collisions;
- The horizon of the mission is considered as infinite.

Therefore, under these assumptions, a Game Theory formulation of the problem is proposed.

Game theory problem formulation

The surveillance mission is defined as a *dynamic game*, where the costs at each time step depend on: the minimal distance between points of interest represented as nodes in a graph, the actual position (node) of the robots and the last time since points of interest (nodes) were visited. Formally, it can be defined as a N-player finite game $\Gamma = (N, \mathbb{A}, u)$, where:

- $N = \{1, \cdots, n\}$ is the finite set of n players, indexed by i;
- $\mathbb{A} = A_1 \times A_2 \times \cdots \times A_n$ represents all possible actions to be taken by all drones;
- $u = h(u_1, \cdots, u_i, \cdots, u_n)$ is the payoff function which is function of the payoff of each single player, with $u : \mathbb{A} \rightarrow \mathbb{R}$, and $u_i : A_i \rightarrow \mathbb{R}$ for each player i.

Players' actions. In conformity with the Game Theory formulation, $\bar{a} = [a_1, a_2, \cdots, a_n]$ is defined as the vector of actions for all $n \in N$ drones and $A_i = \{a_i^1, a_i^2, \cdots, a_i^q\}$, where q is the number of actions at the disposal of the i^{th} drone. Observe that the sets of actions A_i do not need to be equal for all drones; however, in the scenario we are modeling, we will consider the possible actions to be all equal. Then, one may conclude that $\mathbb{A} = A^n$ and the cardinality $|A| = q$. Another point is that, the set A is equal to the subset of states $S = R \subset \mathbb{S}$ meaning that the drone can choose as an action any node $s_k \in S$, then $A = S$.

Payoff function. According to the positions/destinations of drones at time step t, the utility function μ^t can be calculated. The utility is defined as the summation of the utilities of all players involved in the game, i.e.,

$$\mu^t(\bar{a}, \bar{s}^t) = \sum_{\bar{a} \in \mathbb{A}} \mu_i^t(\bar{a}, \bar{s}^t) \tag{1}$$

where $\bar{a}$ is the vector of actions and $\bar{s}^t$ is the state of the drones at time t.

The utility functions for each drone $i \in N$ at time t is defined as

$$\mu_i^t(\bar{a}, \bar{s}^t) = \delta_i(a_i, s_i^t) + \lambda_{-i}(a_i, \bar{s}_{-i}^t) - \rho_i^t(a_i) \tag{2}$$

where:

- $\delta_i(\cdot)$ is the *cost to go* for the drone i from the current position, i.e., the distance for the drone to move from its current position s_i^t to all its possible future locations $a_i^k \in A_i$, with $k \in \{1, \cdots, q\}$. Therefore, considering that $f^*(s_i^t, a_i^k)$ is the optimal distance cost that refers to the optimal (or sub-optimal, when the optimal cannot be calculated) path from node s_i^t to a_i^k, one gets:

$$\delta_i(a_i^k, s_i^t) = f^*(s_i^t, a_i^k) \tag{3}$$

- $\lambda_{-i}(\cdot)$ is the weighted sum of all other drones *inverted distance* ($\Psi(\cdot)$), where *inverted distance* is defined as a value that is equal to the maximum distance for the nearest point and decreases with the distance. Therefore, for the chosen action $a_i^k \in A_i$ we have:

$$\Psi_j(s_j^t, a_i^k) = \max_{a_j^p \in A_j} (\delta_j(a_j^p, s_j^t)) \quad + \quad \min_{a_j^r \in A_j} (\delta_j(a_j^r, s_j^t)) \quad - \quad \delta_j(a_i^k, s_j^t). \tag{4}$$

  The idea here is to make the points more distant from the other drones more attractive for drone i, then, $\lambda_{-i}(\cdot)$ for a determined action $a_i^k \in A_i$ is given by:

$$\lambda_{-i}(a_i^k, \bar{s}_{-i}^t) = \frac{\sum_{j=1}^{n-1} \{\Psi_j(s_{-i,j}^t, a_i^k)\}}{n-1} \mid j \neq i. \tag{5}$$

- Finally, $\rho_i^t(a_i^k)$ is the *expected reward* to reach the node a_i^k. These values are collected (turn into zero) when a drone passes over the position and increase by a factor γn each time step that they are not visited, where γ is a normalizer constant and n is the number of drones:

$$\rho_i^{t+1}(a_i^k) = \rho_i^t(a_i^k) + (\gamma n) \mid \gamma \in [0, 1] \tag{6}$$

  Note that since all action sets A_i are equal to A, the *expected reward* is equal for all drones.

Based on the definition of the utilities, the minimal global cost for this game would be:

$$\mu^{t*}(\bar{a}, \bar{s}^t) = \min_{\bar{a} \in \mathbb{A}} \mu^t(\bar{a}, \bar{s}^t) \tag{7}$$

Notice that the utilities for each drone i, $\mu_i^t(\cdot)$, is only directly dependent on a_i and only indirectly takes into consideration the actions of all other drones (through $\lambda_{-i}(\cdot)$). So, individual's utility functions are composed by three parameters that are, by definition, independent from the action choices of the others players. In this sense, (7) may be rewritten as:

$$\mu^{t*}(\bar{a}, \bar{s}^t) = \min_{a_1 \in A_1} \mu_1^t(a_1, \bar{s}^t) + \cdots + \min_{a_n \in A_n} \mu_n^t(a_n, \bar{s}^t) \tag{8}$$

Therefore, the minimal global cost strategy solution for drone i, σ_i^{t*}, is adopted for the decoupled game as described in:

$$\sigma_i^{t*} = \operatorname*{argmin}_{a_i \in A_i} \mu_i^t(a_i, \bar{s}^t) \tag{9}$$

It means that for this formulation the action choice for drone i is independent from the action choices of the others drones. We are now ready to enunciate and prove the following theorem:

This result is summarized in the following theorem.

Theorem 1. *The N-player finite game* $\Gamma = (N, \mathbb{A}, u)$ *with utility functions defined in* (1) *and* (2) *possess a pure-strategy equilibrium.*

Proof. Let us consider a *Wonderful Life Utility* for drone i.

$$WLU_i = \phi(z) - \phi(z_{-i})$$

where z is the collection of all players and z_{-i} is the collection of all players except player i. It is clear, that if one considers $\phi = \mu^t(\cdot)$, then

$$WLU_i = \mu_i(\cdot)$$

Therefore, the game Γ becomes a *Potential Game*, i.e., the drones' utilities $\mu_i(\cdot)$ are aligned to the global utility $\mu(\cdot)$. Therefore, it is guaranteed to have a pure-strategy equilibrium according to *Corollary 2.2* of (Monderer and Shapley 1996). □

Moreover, it may be verified that this pure-strategy equilibrium is indeed the Nash equilibrium of the game (Philip, Givigi Jr, and Schwartz 2014), for:

$$\mu^t(\bar{a}^*, \bar{s}^t) \leq \mu^t([a_1^*, \cdots, a_{j-1}^*, a_j, a_{j+1}^*, \cdots, a_n^*], \bar{s}^t),\ \forall j \in N.$$

Finally, notice that this decentralized approach, where the action selection is formalized as a potential game, allows to drones to take decision in an asynchronous way, as each drone selects the next action only once it reaches the destination point based only on available (last) information.

Algorithm for coordination

Algorithm 1 presents the process inside each drone. To better explain this algorithm we introduce two execution status on which drones' action selection relies. Before a drone starts to move it changes its status to *Busy* and when it arrives at the destination point it changes to *NotBusy*.

When one of then is *NotBusy*, i.e when it reaches a destination point, (line 2 of Alg. 1), it sends a message of its position and updates its knowledge of the position of the *NotBusy* drones and the destination position for the *Busy* ones (lines 3 and 4). Then, the drone changes the cost value of its current position s_i^t to ∞ which forces it to move to somewhere else (line 5). After, it proceeds all calculations for compute the cost vector μ_i^t, and it selects the minimal cost strategy (lines 6 and 7) applying the proposed approach. The concerned drone computes the global *minimal cost* knowing that the others will do the same. In this way, using game theory to predict what others will do, coordination arises. Finally, it informs its next destination to the others, change its status, and starts to navigate again (lines 8-10).

Algorithm 1 Patrol mission for $Drone_i$

```
1: while True do
2:   if status == NotBusy then
3:     report current position
4:     read messages
5:     assign infinity to current position cost - f*(s_i^t, s_i^t) = ∞
6:     compute the cost vector μ_i^t (Eq. (2))
7:     find and select the minimal cost strategy (Eq. (9))
8:     report destination
9:     assign Busy to its current status
10:    start navigation
11:  else
12:    if position == destination then
13:      assign NotBusy to its current status
14:    else
15:      continue navigation
16:    end if
17:  end if
18: end while
```

When the drone is *Busy*, it only continuously verifies if the destination point is reached, if is the case, it changes its status to *NotBusy*, if not, it continues to navigate (lines 11-15).

To evaluate the proposed approach an application case is presented next.

Simulation Experiments

Setup

The topological model considered for experiments is shown in Figure 1. This topological model is represented by the graph $\mathbb{G} = (\mathbb{S}, \mathbb{E})$ in that the nodes $\mathbb{S} = R \cup D$ represent some positions in the environment, with $R = \{r_1, r_2, ..., r_q\}$ the set of positions inside the rooms and corridors (points of interest) and D the set of doors. The edges $\mathbb{E} \subseteq \mathbb{S} \times \mathbb{S}$ define adjacency relationships between the nodes $\mathbb{S}$, i.e., the possible paths. Each edge has a fixed cost associated with, here, the time required to move from one node to another.

To evaluate the approach, a patrol simulator has been developed in Python 2.7.8. In this simulation model there are 25 *points of interest* (R), the 7 doors are considered as connection points (D) and 60 edges, i.e. $|R| = 25, |D| = 7, |\mathbb{E}| = 60$ respectively, as shown in Figure 1. Please note that, the set S of possible locations is equal to R (we do not consider doors - these specific connection points - as points of interest), and the set of actions A_i of each drone is equal to S.

Figure 2 shows a moment during the mission with three drones. In this simulation, the color of the floor is related to the *idleness* of the point, the blue areas are associated with greater rewards ρ^t.

We note that, as commented before, the approach presented in this paper is neither a coverage problem nor an adversarial problem, but a mix of them. The

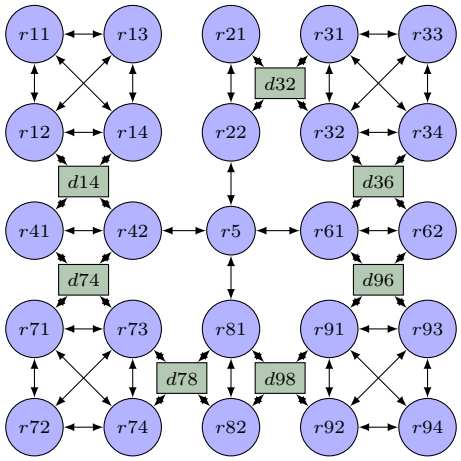


Figure 1: Topological model with the points of interest and all possible transitions.

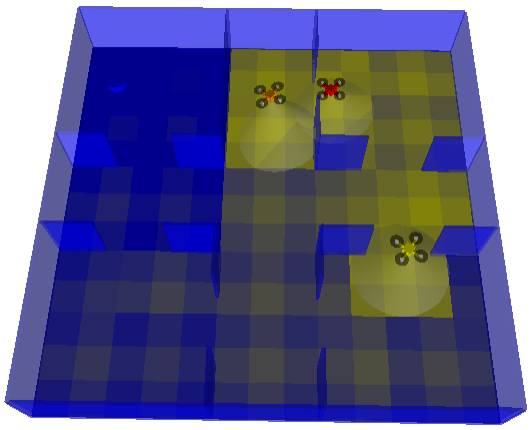


Figure 2: Geometric model.

mixed problem proposed, as far as we know, is for the first time studied, and for this reason a comparison to previous approaches is not straightforward possible.

In this context and in order to verify the performance of the patrolling algorithm considering different numbers of drones in the team and the influence of each component of the utility function (δ, λ and ρ from eq. 2), five scenarios were designed:

- *complete Utility* - where all components of the utility function were used;
- *no Reward* - where ρ was removed from the utility function;
- *no Inverse* - where λ was removed from the utility function;
- *only Distance* - where ρ and λ were removed from the utility function;
- *random* - where the drones select their destinies randomly.

1000 missions for each scenario were played and each patrol mission ran until each point of interest was visited at least fifty times.

Metrics

This study has been focused on (1) the interval between visits (*idleness*) and (2) the difficulty of prediction of the next position of the patrols (*predictability*). For the first one the *average idleness of the graph* Idl_G (Portugal and Rocha 2013b) was used as a metric, and for the second one, the Ljung-Box test (Box, Jenkins, and Reinsel 2008) results were considered.

The *average idleness of the graph* (Idl_G) proposed by (Portugal and Rocha 2013b) is defined as:

Starting with the *instantaneous idleness* (Idl_{t_k}) of a position $s_i \in S$ in the time step t_k:

$$Idl_{t_k}(s_i) = t_k - t_{last_{visit}} \tag{10}$$

where $t_{last_{visit}}$ corresponds to the last time step when that point s_i was visited by a drone. Consequently, the *average idleness* (Idl_m) of a point s_i in a total time T is defined as:

$$Idl_m(s_i) = \frac{\sum_{k=0}^{T} Idl_{t_k}(s_i)}{T} \tag{11}$$

And, finally, the *average idleness of the graph* (Idl_G) is defined as:

$$Idl_G = \frac{\sum_{i=0}^{|S|} Idl_m(s_i)}{\mid S \mid} \tag{12}$$

where $\mid S \mid$ represents the cardinality of the set S.

On the other hand, to evaluate how "unpredictable" the drone paths were, the *Ljung-Box test* was used. This statistical test allows the measurement of the "overall randomness" based on a number of lags of a time series by means of a single value Q:

$$Q = p(p+2) \sum_{l=1}^{m} \frac{\hat{\rho}_l^2}{p-l} \tag{13}$$

and:

$$\hat{\rho}_l = \frac{\sum_{k=1}^{p-l} (Y_i - \bar{Y})(Y_{i+l} - \bar{Y})}{\sum_{k=1}^{p} (Y_i - \bar{Y})^2} \tag{14}$$

where p is the sample size, m is the number of lags being tested, $\hat{\rho}_l$ is the autocorrelation function (ACF) at lag l and $Y = (Y_1, \cdots, Y_p)$ are the measurements. For a significance level α, the critical region for rejection of the hypothesis of randomness is given by the percentile $(1-\alpha)$ of the chi-squared distribution with m degrees of freedom:

$$Q > \chi^2_{1-\alpha,m} \tag{15}$$

Thus, if Eq. 15 is *TRUE* it is possible to say that exists a linear correlation, in other words, the information of past positions allows an inference of future positions. Moreover, Q weights the correlation process, i.e., the higher the value the greater the correlation.

Obviously, all tested scenarios have a high degree of autocorrelation between adjacent and near-adjacent positions, due to the movement model of the drones. Even though, Q can identify an appropriate time series model even when the data are not random.

In the end, in order to use this values as a metric of predictability (π) in the present work, Q for each scenario c was normalized by the worst value (per number of drones n):

$$\pi_c^n = \frac{Q_c^n}{max(Q^n)} \tag{16}$$

In this work, for a specific number of drones, the degrees of freedom m were selected among all scenarios as the smallest median number of steps necessary to complete a cycle (i.e., to visit all positions at least once) with an $\alpha = 0.05$.

Results

Figure 3 shows that increasing the number of drones implies the convergence of idleness, which will be zero when the number of drones reaches the number of points of interest. Nevertheless, looking to these charts it is possible to infer the minimal number of drones to achieve the goal of the mission in an efficient way, defined as the ratio $\frac{|N|}{Idl_G}$ (best cost-benefit ratio). Interestingly, in the *no Inverse* scenario, differently from the others, the idleness seems to be almost steady with two drones or more. The reason for that must be interpreted with caution, but it seems that when they do not need to coordinate their moves (and that is in essence what λ do), they can reach a local optimum very fast; however, these values will eventually decrease to zero. Also, it can be seen that the variance decreases with the number of drones, except for the *no Inverse* scenario. Together these results provide important insights into the approach presented. It is easy to observe the importance of each cost variable and their contribution for idleness.

The increase of the mission performance with the rise in the number of drones in all scenarios for both metrics, *idleness* and *predictability*, is shown in Figure 4. The results also indicate that when ρ was not used (*no Reward*) the paths became more predictable (greater values of π). With a single drone the scenarios *no Reward* and *only Distance* achieved the same value, as expected, since, in this case, they have the same utility function.

On the other hand, still looking to the single drone case, a very predictable path can be identified for *no Inverse* and *complete Utility* scenarios. A possible explanation for these results may be that they tried to maximize the reward earned at each iteration. Interestingly, for more than one drone, the *no Reward* scenario appears to maintain predictable paths. Overall, these charts indicate that the best scenario is the *complete Utility*.

The charts in Figure 5 present a slice of the surveillance mission for three drones with 100 arbitrarily collected steps from all scenarios, where each line represents the path of a drone. What is interesting here is that in *complete Utility*, *no Reward* and *only Distance* the drones tend to maintain themselves in a separated sector from the others. The *Random* scenario presented, as expected, the worst results as the drones moved randomly around the environment. In *complete Utility* and *no Inverse*, the path were longer than the others and with almost no local cycles, indicating global movement in contrast with some "sawtooth" path in the others charts. Another interesting behavior is observed in the *no Inverse* scenario where it seems like that the drones are following each other, maintaining almost the same path. The most striking observation to emerge from the data comparison is that the *complete Utility* generated longer and clearer paths, maintaining the drones separated for almost all time and changing the patrol sectors once in a while.

Conclusion and future work

A multi-aerial-robot game theoretical surveillance approach is proposed in this paper. The main contributions are the development of a dynamic and decentralized approach to cooperation in order to solve the patrol problem by implementing game theoretical models. In this way, a heuristic utility function is presented, where not only the travel cost is considered, but also the current positions of the other team members as well as the last time each point of interest was visited. Based on this utility function, an *N-player game* is played inside each drone, wherein the *Nash Equilibrium* was applied to the drones in order to make their decisions. To improve the real-time performance, the game is played only in the destination points of each drone. Five scenarios and two metrics were designed and used to evaluate the proposed model. Overall, the results indicate that the proposed *utility function* can minimize the *idleness* while also minimizing the patrol *predictability*.

There is abundant room for further progress in this proposed model. Future studies should consider:

- an unreliable human operator in the control loop;
- an intruder and different payoff values for drones and positions;
- leader-follower equilibria;
- uncertainty in the movements and in the detection of the evader;
- imperfect and not cost-free communication.

As it is known, depending on the type of game used, the computational complexity would become intractable with a large-scale team. This was the reason why a potential game was proposed. Furthermore, in the near future we intend to investigate the scalability of this approach. Also, new models for the utilities of the drones will be the focus of future investigations.

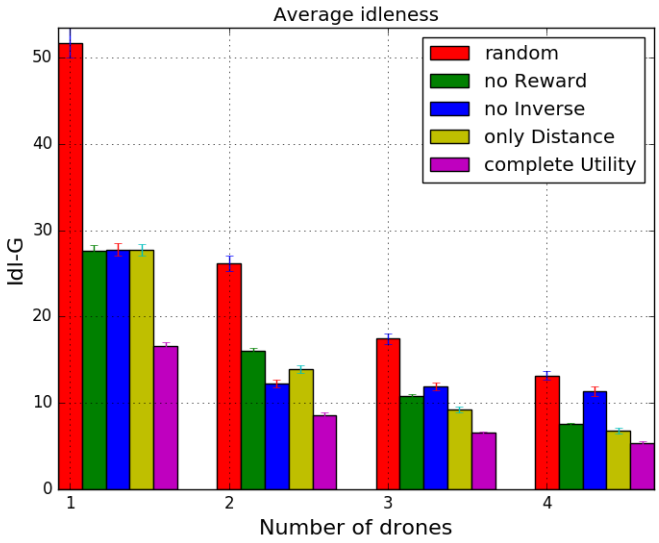


Figure 3: Average idleness of the graph per number of drones.

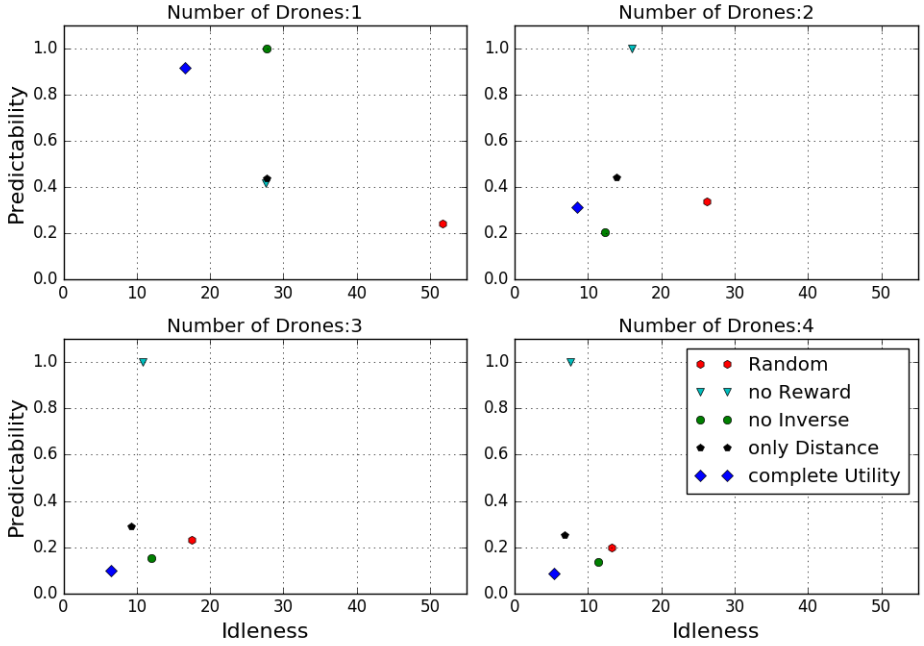


Figure 4: Predictability versus idleness.

References

Amigoni, F.; Basilico, N.; and Gatti, N. 2009. Finding the optimal strategies for robotic patrolling with adversaries in topologically-represented environments. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 819–824. IEEE.

Amigoni, F.; Basilico, N.; Gatti, N.; Saporiti, A.; and Troiani, S. 2010. Moving game theoretical patrolling strategies from theory to practice: An usarsim simulation. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 426–431. IEEE.

An, B.; Kempe, D.; Kiekintveld, C.; Shieh, E.; Singh, S.; Tambe, M.; and Vorobeychik, Y. 2012. Security games with limited surveillance. *Ann Arbor* 1001:48109.

Box, G.; Jenkins, G.; and Reinsel, G. 2008. *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley.

Chevaleyre, Y. 2004. Theoretical analysis of the multi-agent patrolling problem. In *Intelligent Agent Technology, 2004. (IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*, 302–308.

Hernández, E.; Cerro, J. d.; Barrientos, A.; et al. 2013. Game theory models for multi-robot patrolling of infrastructures. *International Journal of Advanced*

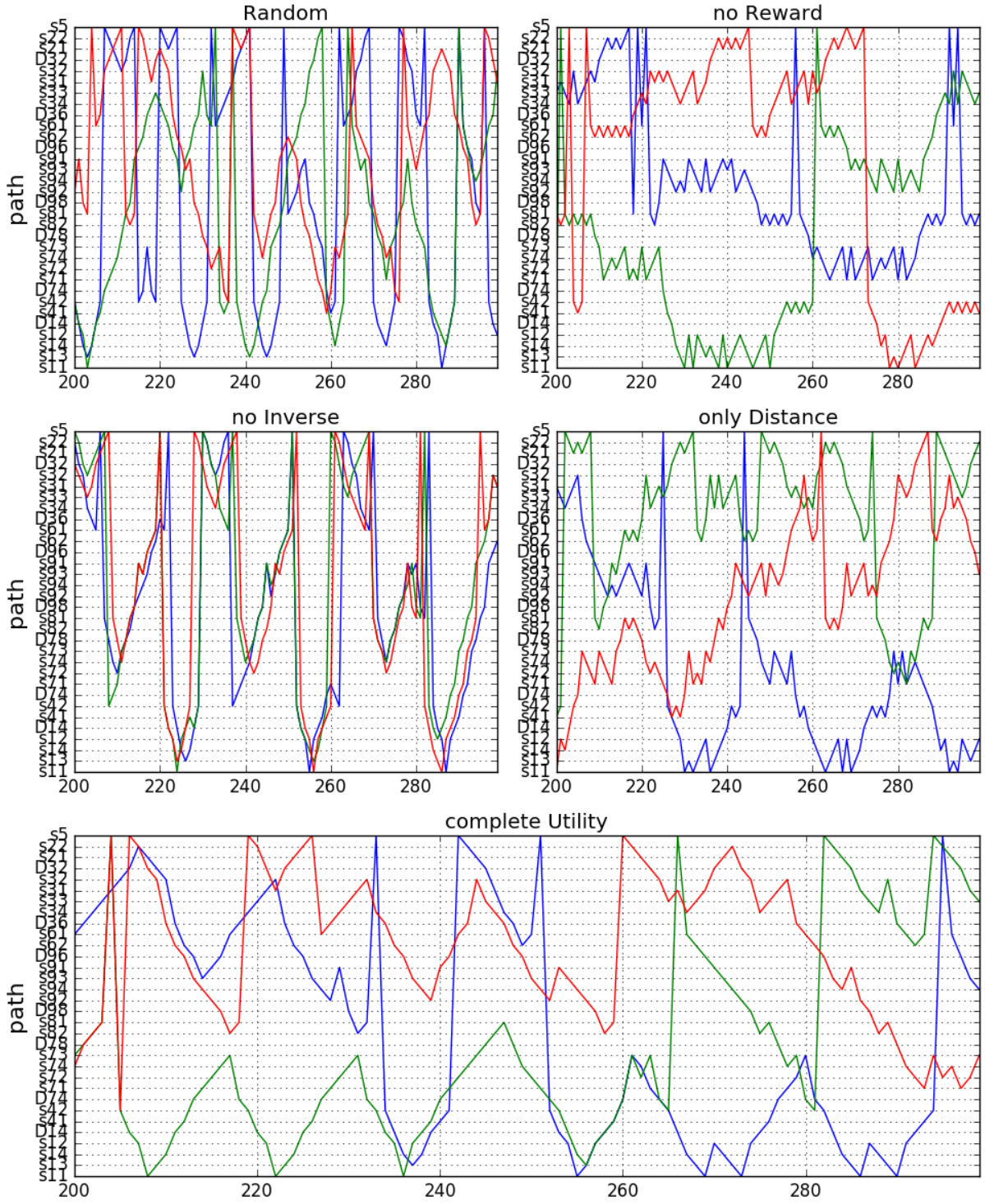


Figure 5: Paths generated with three drones. The "path" axis contains the nodes of the graph.

Robotic Systems 10(181).

Khan, M. E. 2007. Game theory models for pursuit evasion games. Technical report, Technical report, University of British Columbia, Vancouver.

Meng, Y. 2008. Multi-robot searching using game-theory based approach. *International Journal of Advanced Robotic Systems* 5(4):341–350.

Monderer, D., and Shapley, L. S. 1996. Potential games. *Games and Economic Behavior* 14(1):124 – 143.

Murphy, R. R.; Tadokoro, S.; Nardi, D.; Jacoff, A.; Fior-

ini, P.; Choset, H.; and Erkmen, A. M. 2008. Search and rescue robotics. In *Springer Handbook of Robotics*. Springer. 1151–1173.

Pasqualetti, F.; Franchi, A.; and Bullo, F. 2010. On optimal cooperative patrolling. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, 7153–7158. IEEE.

Peshkin, L.; Kim, K.-E.; Meuleau, N.; and Kaelbling, L. P. 2000. Learning to cooperate via policy search. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, 489–496. Morgan Kaufmann Publishers Inc.

Philip, G.; Givigi Jr, S. N.; and Schwartz, H. M. 2014. Cooperative navigation of unknown environments using potential games. *International Journal of Mechatronics and Automation* 4(3):173–187.

Portugal, D., and Rocha, R. 2011. A survey on multi-robot patrolling algorithms. In *Technological Innovation for Sustainability*. Springer. 139–146.

Portugal, D., and Rocha, R. P. 2013a. Distributed multi-robot patrol: A scalable and fault-tolerant framework. *Robotics and Autonomous Systems* 61(12):1572–1587.

Portugal, D., and Rocha, R. P. 2013b. Multi-robot patrolling algorithms: examining performance and scalability. *Advanced Robotics* 27(5):325–336.

Scaramuzza, D.; Achtelik, M. C.; Doitsidis, L.; Friedrich, F.; Kosmatopoulos, E.; Martinelli, A.; Achtelik, M. W.; Chli, M.; Chatzichristofis, S.; Kneip, L.; et al. 2014. Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in gps-denied environments. *Robotics & Automation Magazine, IEEE* 21(3):26–40.

Suarez, J., and Murphy, R. 2011. A survey of animal foraging for directed, persistent search by rescue robotics. In *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, 314–320. IEEE.

Xue, S.; Zeng, J.; and Zhang, G. 2011. A review of autonomous robotic search. In *Electrical and Control Engineering (ICECE), 2011 International Conference on*, 3792–3795. IEEE.

Zhang, M., and Kingston, D. 2015. Time-space network based exact models for periodical monitoring routing problem. In *American Control Conference (ACC), 2015*, 5264–5269. IEEE.

Better Eager Than Lazy?
How Agent Types Impact the Successfulness of Implicit Coordination

Thomas Bolander
DTU Compute
Technical University of Denmark
Copenhagen, Denmark
tobo@dtu.dk

Thorsten Engesser
University of Freiburg
Freiburg, Germany
engesser@cs.uni-freiburg.de

Robert Mattmüller
University of Freiburg
Freiburg, Germany
mattmuel@cs.uni-freiburg.de

Bernhard Nebel
University of Freiburg
Freiburg, Germany
nebel@cs.uni-freiburg.de

Abstract

Epistemic planning can be used for decision making in multi-agent situations with distributed knowledge and capabilities. In recent work, we proposed a new notion of strong policies with implicit coordination. With this it is possible to solve planning tasks with joint goals from a single-agent perspective without the agents having to negotiate about and commit to a joint policy at plan time. We study how and under which circumstances the decentralized application of those policies leads to the desired outcome.

1 Introduction

One important task in multi-agent systems is to collaboratively reach a joint goal with multiple autonomous agents (e.g. robots and humans). For instance, if there is a group of robots that are supposed to reach target locations, they have to develop a plan that enables each robot to accomplish its goal. Taking, for instance the situation in Figure 1, where the circular robot C wants to go to the cell marked by the solid circle and the square robot S wants to reach the place with the solid square (the empty circle and square will only become important later). One could come up with the following plan: (i) C moves to 2 and then to 4, (ii) S moves to 2 and then to target location 3, and (iii) C finally moves to target location 2.

This plan could be generated *centrally* by an external observer and then communicated to the two agents, which will execute it. We will assume, however, that all plans are developed by the agents in a *distributed* fashion. Assuming that the two agents can observe everything in the world, have full knowledge of their goals, and execution is deterministic, they both can come up with the same plan as above and execute this plan in a distributed way. If they came up with different plans but have anticipated that the other agents might deviate, then the joint execution might still be successful. We have to make strong assumptions about the *planning agent types*, though, as we will demonstrate.

The problem of planning and executing in a distributed fashion becomes significantly more difficult if we drop the assumption about full observability. In order to illustrate this point, let us again consider the situation in Figure 1, but unlike before, let us assume that each robot knows about their own target positions with certainty (the solid circle and square), but there is uncertainty about the target position of the other robot (the empty circle and square are considered as possible target positions for C and S, respectively). This means that we still assume a common goal, namely each robot wants that in the end all the robots have reached their target positions. However, these target positions are not common knowledge. In such a situation, we will consider *policies* instead of plans, which can branch on observations and sensing actions. As it turns out, an agent can still come up with a successful policy which is *implicitly coordinated*, i.e., contains only steps such that the acting agent knows that her step contributes to reaching the goal. The key for generating such policies is to take *perspective shifts*, i.e., picturing oneself in the shoes of the other agent. Giving general success guarantees for the joint execution of *policy profiles* in a partially observable setting appears to be much more difficult than in the former case of full observability, though.

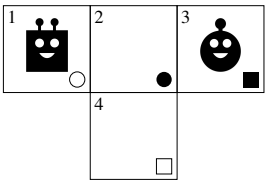


Figure 1: Multi-robot coordination example

Although taking into account the plans of other agents to achieve cooperation has been identified as an interesting topic of artificial intelligence research for a considerable amount of time (Konolige and Nilsson 1980), the application of implicit coordination has been limited almost exclusively to the fields of probabilistic robotics (Stulp *et al.* 2006; Anderson and Papanikolopoulos 2008; Hollinger *et al.* 2009) and Dec-POMDPs (Spaan *et al.* 2006). While existing classical planning approaches rely on continual (re-)planning (Brenner and Nebel 2009), the work we present in this paper is situated in the context of (multi-agent) epistemic planning, which can be approached algorithmically either by compilation to classical planning (Albore *et al.* 2009; Kominis and Geffner 2015; Muise *et al.* 2015) or by search in the space of "nested" (Bolander and Andersen 2011; Engesser *et al.* 2015) or "shallow" knowledge states (Petrick and Bacchus 2002; 2004; Petrick and Foster 2013).

In Section 2, we describe the formal framework for rep-

resenting states as the one in Figure 1, and how actions can change these states. Section 3 formalizes the notions of policies, policy profiles and their executions. In Section 4, we analyze the conditions under which the execution of policy profiles can be successful.

2 Theoretical Background: DEL

2.1 Epistemic States and Perspective Shifts

To represent planning problems as the one described above we need a formal framework where: (1) agents can reason about the first- and higher-order knowledge and ignorance of other agents; (2) both fully and partially observable actions can be described in a compact way. Dynamic Epistemic Logic (DEL) satisfies these conditions. We first very briefly recapitulate the foundations of DEL, following the conventions of Bolander and Andersen (2011).

In the following we will define epistemic languages, epistemic states and epistemic actions. All of these are defined relative to a given finite set of *agent names* (or simply *agents*) $\mathcal{A}$ and a given finite set of *atomic propositions* P. To keep the exposition simple, we will not mention the dependency on $\mathcal{A}$ and P in the following. The *epistemic language* $\mathcal{L}_{\mathrm{K}}$ is

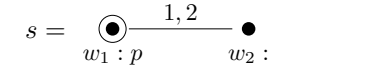
$$\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi\wedge\varphi \mid K_i\varphi, \text{ where } p \in P \text{ and } i \in \mathcal{A}.$$

As usual, we read $K_i\varphi$ as "agent i knows φ". Formulas are evaluated in *epistemic models* $\mathcal{M} = \langle W, (\sim_i)_{i\in\mathcal{A}}, V\rangle$ where the *domain* W is a non-empty finite set of *worlds*; $\sim_i \subseteq W^2$ is an equivalence relation called the *indistinguishability relation* for agent i; and $V : P \to \mathcal{P}(W)$ assigns a *valuation* to each atomic proposition. For $W_d \subseteq W$, the pair $(\mathcal{M}, W_d)$ is called an *epistemic state* (or simply a *state*), and the worlds of W_d are called the *designated worlds*. A state is called *global* if $W_d = \{w\}$ for some world w (called the *actual world*), and we then often write $(\mathcal{M}, w)$ instead of $(\mathcal{M}, \{w\})$. We use S^{gl} to denote the set of global states. For any state $s = (\mathcal{M}, W_d)$ we let $\mathit{Globals}(s) = \{(\mathcal{M}, w) \mid w \in W_d\}$. We define truth in states as follows, where the propositional cases are standard and hence left out:

$$\begin{array}{lll}(\mathcal{M}, W_d) \models \varphi & \text{iff} & (\mathcal{M}, w) \models \varphi \text{ for all } w \in W_d\\ (\mathcal{M}, w) \models K_i\varphi & \text{iff} & (\mathcal{M}, w') \models \varphi \text{ for all } w' \sim_i w\end{array}$$

A state $(\mathcal{M}, W_d)$ is called a *local state* for agent i if W_d is closed under $\sim_i$. Given a state $s = (\mathcal{M}, W_d)$, the *associated local state* of agent i, denoted s^i, is $(\mathcal{M}, \{v \mid v \sim_i w \text{ and } w \in W_d\})$. Going from s to s^i amounts to a *perspective shift* to the local perspective of agent i.

Example 1. Consider the global state $s = (\mathcal{M}, w_1)$ given as follows, where the nodes represent worlds, the edges represent the indistinguishability relations (reflexive edges left out), and ⦿ is used for designated worlds:



Each node is labeled with the name of the world, and the list of atomic propositions true at the world. In the state s, the proposition p is true but agent 1 does not know this: $s \models p \wedge \neg K_1 p$. Hence from the local perspective of agent 1, p cannot be verified, and we correspondingly have $s^1 \not\models p$ and $s^1 \not\models \neg p$.

2.2 Epistemic Actions and Product Update

To model actions, we use the *event models* of DEL. An *event model* is $\mathcal{E} = \langle E, (\sim_i)_{i\in\mathcal{A}}, \mathrm{pre}, \mathrm{post}\rangle$ where the *domain* E is a non-empty finite set of *events*; $\sim_i \subseteq E^2$ is an equivalence relation called the *indistinguishability relation* for agent i; $\mathrm{pre} : E \to \mathcal{L}_{\mathrm{K}}$ assigns a *precondition* to each event; and $\mathrm{post} : E \to \mathcal{L}_{\mathrm{K}}$ assigns a *postcondition* to each event. For all $e \in E$, $\mathit{post}(e)$ is a conjunction of literals (atomic propositions and their negations, including $\top$ and $\bot$). For $E_d \subseteq E$, the pair $(\mathcal{E}, E_d)$ is called an *epistemic action* (or simply *action*), and the events in E_d are called the *designated events*. Similar to states, $(\mathcal{E}, E_d)$ is called a *local action* for agent i when E_d is closed under $\sim_i$.

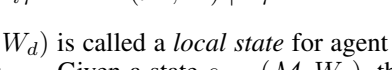
Each event of an action represents a different possible outcome. By using multiple events $e, e' \in E$ that are indistinguishable (i.e. $e \sim_i e'$), it is possible to obfuscate the outcomes for some agent $i \in \mathcal{A}$, i.e. modeling partially observable actions. Using event models with $|E_d| > 1$, it is also possible to model sensing actions and nondeterministic actions (Bolander and Andersen 2011).

The *product update* is used to specify the successor state resulting from the application of an action in a state. Let a state $s = (\mathcal{M}, W_d)$ and an action $a = (\mathcal{E}, E_d)$ be given with $\mathcal{M} = \langle W, (\sim_i)_{i\in\mathcal{A}}, V\rangle$ and $\mathcal{E} = \langle E, (\sim_i)_{i\in\mathcal{A}}, \mathrm{pre}, \mathrm{post}\rangle$. Then the *product update* of s with a is defined as $s \otimes a = (\langle W', (\sim'_i)_{i\in\mathcal{A}}, V'\rangle, W'_d)$ where

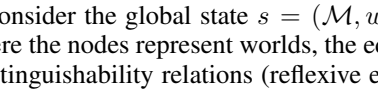
- $W' = \{(w, e) \in W \times E \mid \mathcal{M}, w \models \mathrm{pre}(e)\}$;
- $\sim'_i = \{((w, e), (w', e')) \in (W')^2 \mid w \sim_i w' \;\&\; e \sim_i e'\}$;
- $V'(p) = \{(w, e) \in W' \mid \mathrm{post}(e) \models p \text{ or } (\mathcal{M}, w \models p \text{ and } \mathrm{post}(e) \not\models \neg p)\}$;
- $W'_d = \{(w, e) \in W' \mid w \in W_d \text{ and } e \in E_d\}$.

We say that $a = (\mathcal{E}, E_d)$ is *applicable* in $s = (\mathcal{M}, W_d)$ if for all $w \in W_d$ there is an event $e \in E_d$ s.t. $(\mathcal{M}, w) \models \mathrm{pre}(e)$.

Example 2. Consider the following epistemic action $a = (\mathcal{E}, \{e_1, e_2\})$, using the same conventions as for epistemic states, except each event is labeled with $\langle \mathrm{pre}(e), \mathrm{post}(e)\rangle$:



It is a private sensing action for agent 1, where agent 1 privately gets to know the truth value of p, since e_1 and e_2 are distinguishable to agent 1 (and *in*distinguishable to agent 2). Letting s be the state from Example 1, we get:



After the private sensing of p by agent 1, agent 1 will know that p is true, but agent 2 will still not: $s \otimes a \models K_1 p \wedge \neg K_2 p$.

Isomorphic states and actions will be identified.

3 Planning Tasks, Policies and Executions

In this paper we consider *cooperative* planning tasks, that is, planning tasks in which the agents plan towards a joint goal

(Engesser *et al.* 2015). Each action in a planning task is assumed to be executable by a unique agent, called the *owner* of the action. More precisely, given a set of actions A, an *owner function* is a mapping $\omega : A \to \mathcal{A}$ from actions to their owners. Mapping each action to a unique agent can be done without loss of generality, since semantically equivalent duplicates can always be added to the action set.

Definition 1. A *planning task* $\Pi = \langle s_0, A, \omega, \gamma \rangle$ consists of a global state s_0 called the *initial state*; a finite set of actions A; an owner function $\omega : A \to \mathcal{A}$; and a *goal formula* $\gamma \in \mathcal{L}_{\mathrm{K}}$. We require that each $a \in A$ is local for $\omega(a)$.

Example 3. Consider the planning task $\langle s_0, \{a_1, a_2\}, \omega, p \rangle$ with initial state $s_0 = \circledbullet$ and two semantically equivalent actions $a_1 = \circledbullet\, e_1 : \langle \top, p \rangle$ and $a_2 = \circledbullet\, e_1' : \langle \top, p \rangle$ for the owners $\omega(a_1) = 1$ and $\omega(a_2) = 2$ (both actions making the goal p true unconditionally). Both the initial state s_0 and the effects of the actions a_1 and a_2 are fully observable for both agents. Intuitively, a solution should prescribe the action a_1 for agent 1 or the action a_2 for agent 2.

3.1 Policies and Executions

Instead of working with sequential plans, our plans are going to be policies, representing instructions that can be individually followed by each of the agents. We impose some minimal requirements on these policies to be reasonable. First, we require *knowledge of preconditions* (KOP), i.e., in each state s, the agent i supposed to perform a particular action a according to policy π must *know* that a is applicable in s. Moreover, we require π to be *unambiguous* for all agents in the sense that in each state s where an agent i is supposed to act according to π, π is *deterministic* for agent i (DET); finally, we require *uniformity*, i.e., if the policy π prescribes some action a to agent i in state s and agent i cannot distinguish s from some other state t, then π has to prescribe the same action a for i in t as well (UNIF). More formally:

Definition 2. Let $\Pi = \langle s_0, A, \omega, \gamma \rangle$ be a planning task. Then a *policy* π for Π is a partial mapping $\pi : S^{\mathrm{gl}} \hookrightarrow \mathcal{P}(A)$, s. t.

(KOP) f. a. $s \in S^{\mathrm{gl}}$, $a \in \pi(s)$: a is applicable in $s^{\omega(a)}$,

(DET) f. a. $s \in S^{\mathrm{gl}}, a, a' \in \pi(s)$ s. t. $\omega(a) = \omega(a')$: $a = a'$,

(UNIF) f. a. $s, t \in S^{\mathrm{gl}}$ s. t. $s^{\omega(a)} = t^{\omega(a)}$, $a \in \pi(s)$: $a \in \pi(t)$.

To characterize the different outcomes of agents acting according to a common policy, we define the notion of policy executions. As in more classical non-epistemic settings, relevant questions are whether the execution process terminates or not, and if it does, whether a goal state is reached.

Definition 3. An *execution* of a policy π from a global state s_0 is a maximal (finite or infinite) sequence of alternating global states and actions $(s_0, a_1, s_1, a_2, s_2, \ldots)$, such that for all $m \geq 0$,

(1) $a_{m+1} \in \pi(s_m)$, and

(2) $s_{m+1} \in \mathit{Globals}(s_m \otimes a_{m+1})$.

An execution is called *successful* for a planning task $\Pi = \langle s_0, A, \omega, \gamma \rangle$, if it is a finite execution $(s_0, a_1, s_1, \ldots, a_n, s_n)$ such that $s_n \models \gamma$.

In the following, we will restrict our attention to policies that are guaranteed to achieve the goal after a finite number of steps. More formally, this means that all of their executions must be successful. As in nondeterministic planning, we call such policies *strong* (Cimatti *et al.* 2003).

Definition 4. A policy π for a planning task $\Pi = \langle s_0, A, \omega, \gamma \rangle$ is called *strong* if $s_0 \in \mathrm{Dom}(\pi)$ and for each $s \in \mathrm{Dom}(\pi)$, any execution of π from s is successful for Π. A planning task Π is called *solvable* if a strong policy for Π exists. For $i \in \mathcal{A}$, we call a policy π *i-strong* if it is strong and $\mathit{Globals}(s_0^i) \subseteq \mathrm{Dom}(\pi)$.

When a policy is i-strong it means that the policy is strong and defined on all the global states that agent i cannot initially distinguish between. It follows directly from the definition that any execution of an i-strong policy from any of those initially indistinguishable states will be successful. So if agent i comes up with an i-strong policy, it means that agent i knows the policy to be successful.

We introduce the notion of reachability to talk about states that can occur during executions.

Definition 5. Given global states s_0 and s, we call s *reachable* from s_0 if there are sequences of actions $a_1, \ldots, a_n$ and states $s_1, \ldots, s_n = s$ such that a_{m+1} is applicable in s_m and $s_{m+1} \in \mathit{Globals}(s_m \otimes a_{m+1})$ for all $m = 0, \ldots, n-1$. We call s reachable from s_0 *by following* a policy π if it is part of an execution $(s_0, a_1, \ldots, s, \ldots)$ of π.

A strong policy π is *implicitly coordinated* in the sense that at any point during its execution, at least one agent knows that it can execute a particular action as part of the strong policy π. This is formalized by the following proposition, that follows straightforwardly from Def. 4 and the uniformity condition in Def. 2.

Proposition 1. *Let π be a strong policy for $\Pi = \langle s_0, A, \omega, \gamma \rangle$ and let s be a non-goal state reachable from s_0 by following π. Then for some $i \in \mathcal{A}$: $\pi(s) \cap \{a \mid \omega(a) = i\} \neq \emptyset$ and π is an i-strong policy for $\langle s, A, \omega, \gamma \rangle$.*

In this paper, our agents will most often try to plan for all contingencies (as seen from their local perspective), so that it never becomes necessary to change policy due to unexpected actions by other agents. The relevant notion of "planning for all contingencies" in this setting is captured by what we call *maximality* of strong policies.

Definition 6. We call a strong policy π a *maximal strong policy* for agent i and planning task $\Pi = \langle s_0, A, \omega, \gamma \rangle$ if $s \in \mathrm{Dom}(\pi)$ for all states s such that: (1) s is reachable from some $s_0' \in \mathit{Globals}(s_0^i)$, and (2) $\langle s, A, \omega, \varphi \rangle$ is solvable.

3.2 Policy Profiles

Besides the centralized scenario in which one agent plans centrally for all agents, or equivalently, in which the involved agents can already coordinate on a common plan at plan time, we especially want to study the scenario in which the agents *cannot* coordinate their plans, but rather have to come up with plans individually. Those plans can easily differ, not only because of different reasoning capabilities of

the different agents, but also because of their non-uniform knowledge of the initial state and of action outcomes. For our formal analysis, we define a *policy profile* for a planning task Π to be a family $(\pi_i)_{i\in\mathcal{A}}$, where each π_i is a policy for Π. Executions can be generalized to policy profiles as follows.

Definition 7. An *execution* of a policy profile $(\pi_i)_{i\in\mathcal{A}}$ is a maximal (finite or infinite) sequence of alternating global states and actions $(s_0, a_1, s_1, \ldots)$, such that for all $m \geq 0$,

(1) $a_{m+1} \in \pi_i(s_m)$ where $i = \omega(a_{m+1})$, and

(2) $s_{m+1} \in \mathit{Globals}(s_m \otimes a_{m+1})$.

We call such an execution successful if it is a finite execution $(s_0, a_1, s_1, \ldots, a_n, s_n)$ such that $s_n \models \gamma$.

Note that there are two sources of nondeterminism for executions. One as a result from the possibility of multiple policies prescribing actions for their respective agents (item 1 in Def. 7). The other one results from the possibility of nondeterministic action outcomes (item 2 in Def. 7). Definition 4 implies that strong policies are *closed* in the usual sense that following a strong policy cannot lead to an "off-policy" non-goal state where the policy is undefined (Cimatti *et al.* 2003). As a first positive result, we can now show that *policy profiles* consisting of maximal strong policies are also *closed* in the sense that they do not produce dead-end executions, i. e. executions ending in a non-goal state where some of the policies in the profile are undefined. By inductive application of Proposition 1, we can show that in each execution step from s via $a = \pi_i(s)$ to s', the policy π_i must be defined for s', and by maximality of the other policies π_j, $j \neq i$, the policies of all other agents have to be defined in s' as well. Hence:

Proposition 2. *Let $(\pi_i)_{i\in\mathcal{A}}$ be a policy profile where each π_i is a maximal strong policy for agent i and task Π. Then $s \in \mathrm{Dom}(\pi_i)$ for all agents $i \in \mathcal{A}$ and states $s \in S^{gl}$ occurring in arbitrary executions $(s_0, a_1, \ldots, s, \ldots)$ of $(\pi_i)_{i\in\mathcal{A}}$.*

If all agents have *one strong policy in common* which all of them follow, then at execution time, the goal is guaranteed to be eventually reached. If, however, each agent acts on *its individual strong policy*, then the incompatibility of the individual policies may prevent the agents from reaching the goal, even though each individual policy is strong. The following example illustrates what may go wrong.

Example 4. Let $\Pi = \langle s_0, \{a_1, a_2\}, \omega, p\rangle$ be the planning task described in Example 3, and let (π_1, π_2) be the policy profile consisting of the two maximal strong policies π_1 assigning only a_2 to s_0 and π_2 assigning only a_1 to s_0. Here each agent expects the other agent to do the work, since the policy π_1 for agent 1 specifies the action a_2 belonging to agent 2 and vice versa. This profile has only one execution, the empty one, which is unsuccessful.

This shows that agents following maximal strong policies may still not reach the goal. The issue we see here is that the agents run into a "deadlock", and the underlying reason is that both agents are "lazy", expecting the other agent to act. In the following, we will discuss for which types of agents (lazy, eager, ...) and for which combinations of them success can or cannot be guaranteed.

4 Agent Types

We distinguish between different agent types by distinguishing between the types of policies they produce. To that end, we identify agents with mappings from planning tasks to policies. Additionally, the agent mapping must be associated with the part the agent plays in the planning task, since a policy that is lazy from one agent's perspective may be eager from another agent's perspective; e.g. the policy π_1 in Example 4 is lazy for agent 1, but eager for agent 2.

Definition 8. A *planning agent* (or simply *agent*) is a pair (i, T), where i is an agent name and T is a mapping from planning tasks to policies, such that $T(\Pi)$ is an i-strong policy for Π, whenever such a policy exists.

The requirement of $T(\Pi)$ being i-strong, whenever such a policy exists, comes from the fact that each agent should produce a policy that it knows will be successful, whenever it is possible to form such a policy. We can now extend the definition of executions to *groups* of agents $(i, T_i)_{i\in\mathcal{A}}$.

Definition 9. Let $(i, T_i)_{i\in\mathcal{A}}$ be a group of agents and let Π be a planning task. Then the *executions* by $(i, T_i)_{i\in\mathcal{A}}$ of Π are the executions of the policy profile $(T_i(\Pi))_{i\in\mathcal{A}}$.

4.1 Lazy and Naively Eager Agents

We already saw that agents being lazy can be problematic. To formally capture laziness (and its dual, eagerness), we note that laziness essentially means having a preference against using one's own actions, and planning with someone else's actions instead. Similarly, eagerness means preferring one's own actions over someone else's. Intuitively, an agent has a preference for (or against) a set of actions if whenever a policy produced by that agent is defined, it prescribes at least one preferred action (or no unpreferred action), unless violating the preference is unavoidable in that state.

Definition 10. For a state s, a policy π, and a set of actions A', we say that π *uses* A' in s if $\pi(s) \cap A' \neq \emptyset$. Then we say that agent (i, T) *has preference*

(1) *for* (the actions in) A' if for all Π and all $s \in \mathrm{Dom}(T(\Pi))$, policy $T(\Pi)$ uses A' in s unless no i-strong policy for Π uses A' in s, and

(2) *against* (the actions in) A' if for all Π and all $s \in \mathrm{Dom}(T(\Pi))$, policy $T(\Pi)$ does not use A' in s unless every i-strong policy for Π uses A' in s.

Unfortunately, *preference against* a set of actions is not the same as *preference for* its complement, which is why we need both notions. We can now define laziness as preference against one's own actions, that is, we call an agent (i, T) *lazy* if it has preference against the actions in $\{a \in A \mid \omega(a) = i\}$.

To formalize in which sense Example 4 is problematic, we still have to define deadlocks, which intuitively are states where (1) something still *needs to be done*, where (2) it is known that something *can be done*, but where (3) nothing *will be done* because of incompatible individual policies.

Definition 11. A *deadlock* for a policy profile $(\pi_i)_{i\in\mathcal{A}}$ is a global state s such that (1) s is not a goal state, (2) $s \in \mathrm{Dom}(\pi_i)$ for some $i \in \mathcal{A}$, and (3) $\omega(a) \neq i$ for all $i \in \mathcal{A}$ and $a \in \pi_i(s)$.

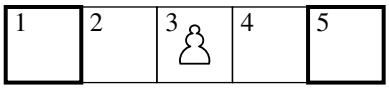


Figure 2: Planning task—move chess piece left or right.

Requirement (2) is included to distinguish deadlocks from *dead ends*, where none of the agents' policies prescribe an action, not even for another agent. From the above definitions and Example 4 we immediately get the following result.

Proposition 3. *There are solvable planning tasks for which all executions by lazy agents result in a deadlock.*

To avoid deadlocks, we define (naively) eager agents as agents who have a preference *for* their own actions. That is, we call an agent (i, T) *naively eager* if it has a preference for the actions in $\{a \in A \mid \omega(a) = i\}$. They are called *naively* eager since it will turn out that in their eagerness they can *interfere* with other agents' plans and executions in a harmful way. But still, we first get the positive result that eagerness prevents the deadlocks we observed for lazy agents.

Proposition 4. *Let Π be a planning task and $(i, T_i)_{i\in\mathcal{A}}$ be a group of naively eager agents. If $\pi_i = T_i(\Pi)$ is a maximal strong policy for each $i \in \mathcal{A}$, then all executions of $(\pi_i)_{i\in\mathcal{A}}$ are deadlock-free.*

Proof sketch. Assume for contradiction that s is a deadlock for $(\pi_i)_{i\in\mathcal{A}}$. Then there has to exist an agent name $i \in \mathcal{A}$ and an action a such that $a \in \pi_i(s)$ with $\omega(a) = j$ and $j \neq i$. Because π_j is a maximal strong policy, we have $s \in \mathrm{Dom}(\pi_j)$. Then there also has to exist an $a' \in \pi_j(s)$ with $\omega(a') = j$, since (j, T_j) is naively eager and has preference for its own actions. This contradicts item (3) of Definition 11. □

Example 5. Consider the scenario in Fig. 2. The chess piece can be moved left by agent 1 and right by agent 2 (one cell at a time). Everything is fully observable. The goal is to move the piece to one of the highlighted target cells. Every possible naively eager policy π_1 of agent 1 must assign action *left* to every non-goal state, and similarly, every naively eager policy π_2 of agent 2 must assign *right* to every non-goal state. Using s_i to denote that the piece is in cell i, one possible execution of any such policy profile (π_1, π_2) is the infinite sequence $(s_3, \mathit{left}, s_2, \mathit{right}, s_3, \mathit{left}, \ldots)$, which is clearly not successful.

This shows that naively eager agents may also not reach the goal since they can potentially produce infinite executions.

Proposition 5. *There are solvable planning tasks for which some executions by naively eager agents are infinite.*

4.2 Optimally Eager Agents

In order to address the stated problem, we will now consider agents who always try to simplify the problem by reaching states closer to the goal. This means that the agents should come up with *optimal policies*, policies that reach the goal in the fewest number of steps. In order to formally define optimal policies, we need the notion of *cost*. The cost of a policy can be defined as its worst-case execution length, that is, the number of actions in its longest possible execution. An optimal policy is then one of minimal cost. However, due to partial observability, different agents might assign different costs to the same policy, and hence disagree on which policies have minimal cost.

For instance, in a variant of Example 5, agent 1 might not know whether the chess piece is initially in cell 3 or 4, and agent 2 might not know whether it is initially in cell 2 or 3. Then agent 1 would assign cost 2 to the "go right" strategy, but cost 3 to the "go left" strategy (according to the knowledge of agent 1, the chess piece might initially be in cell 4, and hence 3 cells away from cell 1). Conversely, agent 2 would assign cost 2 to the "go left" strategy and cost 3 to the "go right" strategy. If both agents choose strategies of minimal cost, they would choose opposing strategies: agent 1 would want agent 2 to go right, and agent 2 would want agent 1 to go left. Clearly this will result in a deadlock.

To remedy this, we need the agents to measure cost in a "perspective-sensitive" way: the assigned cost takes the different perspectives of the involved agents into account.

Definition 12. Let π be a strong policy for a planning task Π. The *perspective-sensitive cost* (or simply *cost*) of π from a state $s \in \mathrm{Dom}(\pi)$, denoted $\kappa_\pi(s)$, is defined as:

$$\kappa_\pi(s) = \begin{cases} 0 & \text{if there exists no } a \in \pi(s) \\ 1 + \max_{a\in\pi(s), s'\in \mathit{Globals}(s^{\omega(a)}\otimes a)} \kappa_\pi(s') & \text{else.} \end{cases}$$

We extend this to local states s with $\mathit{Globals}(s) \subseteq \mathrm{Dom}(\pi)$ by letting $\kappa_\pi(s) := \max_{s'\in \mathit{Globals}(s)} \kappa_\pi(s')$.

The following proposition captures the intuition that perspective-sensitive costs can only increase with additional uncertainty (by shifting perspective), and that in each global state s with $\pi(s) \neq \emptyset$, one or more actions can be identified as the ones maximizing the perspective-sensitive cost for the successor state and thus defining the value of $\kappa_\pi(s)$. We will need this to prove deadlock-freedom in Proposition 7.

Proposition 6. *For any policy π, epistemic state s and agent $i \in \mathcal{A}$, it holds that $\kappa_\pi(s) \leq \kappa_\pi(s^i)$. Moreover, if $\kappa_\pi(s) > 0$, then there is an action $a \in \pi(s)$ such that $\kappa_\pi(s) = \kappa_\pi(s^{\omega(a)})$.*

It can be verified that in the variant of Example 5 with partial observability about the initial state of the chess piece, both the "go left" and the "go right" strategy will have the same (perspective-sensitive) cost 3. The point is that the cost assigned to the "go left" strategy will be measured from the local state of the owner of the "go left" action, and similarly for "right", as seen from the Def. 12.

Definition 13. A policy π for a planning task $\Pi = \langle s_0, A, \omega, \gamma\rangle$ is called *subjectively optimal* if for all $s \in \mathrm{Dom}(\pi)$, all $a \in \pi(s)$ and all $\omega(a)$-strong policies π' for $\langle s, A, \omega, \gamma\rangle$ we have $\kappa_{\pi'}(s^{\omega(a)}) \geq \kappa_\pi(s^{\omega(a)})$.

Definition 14. Given a set of actions A', we say that agent (i, T) is *subjectively optimal with preference for the actions in A'*, if for all Π: (1) $T(\Pi)$ is an i-strong subjectively optimal policy if such a policy exists, and (2) $T(\Pi)$ uses A' in

each $s \in \mathrm{Dom}(\pi)$ unless no i-strong subjectively optimal policy for Π uses A' in s.

We call an agent that is subjectively optimal with preference for its own actions *optimally eager*. That is, a planning agent (i, T) is called *optimally eager* if it is subjectively optimal with preference for the actions in $\{a \in A \mid \omega(a) = i\}$.

In the variant of Example 5 with partial observability about the initial state, optimally eager agents will always be successful. They assign the same cost to both the "go left" and "go right" strategies, but are eager, and will hence prefer the policy where they act themselves. So initially they specify conflicting actions. Assume agent 1 gets to act first and moves one cell left. In the resulting state, agent 2 assigns cost 3 to the "move right" strategy and only cost 2 to the "move left" strategy. Hence agent 2 will not try to prevent agent 1 from moving the chess piece to the far left.

On the other hand, an *optimally lazy* agent (which we could define analogously, by first defining *subjective optimality with preference against* own actions) would exhibit the same deadlock potential as naively lazy agents. We can also see this in Example 4, where both policies are in fact subjectively optimal ones. Our focus will thus be on optimally eager agents. We can indeed show that optimally eager agents do not produce deadlocks.

Proposition 7. *Let Π be a planning task and $(i, T_i)_{i\in\mathcal{A}}$ be a group of optimally eager agents. If $\pi_i = T_i(\Pi)$ is a maximal strong policy for each $i \in \mathcal{A}$, then all executions of $(\pi_i)_{i\in\mathcal{A}}$ are deadlock-free.*

Proof sketch. Let s be a reachable non-goal state. We analyze *waiting chains*, i. e., sequences of agents $i^1, \ldots, i^{n+1}$, such that (abbreviating π_{i^j} as π^j, and κ_{π^j} as κ^j), for all $j = 1, \ldots, n$, (1) there is no $a \in \pi^j(s)$ with $\omega(a) = i^j$, and (2) there is an $a \in \pi^j(s)$ with $\kappa^j(s^{\omega(a)}) = \kappa^j(s)$ and $\omega(a) = i^{j+1}$. By Proposition 6 and the definition of subjective optimality, we have $\kappa^{j+1}(s) \leq \kappa^{j+1}(s^{\omega(a)}) \leq \kappa^j(s^{\omega(a)}) = \kappa^j(s)$ for all $j = 1, \ldots, n$. This implies that no agent can occur more than once in a waiting chain, since that would directly contradict its eagerness. Thus, if $s \in \mathrm{Dom}(\pi^1)$ and $\pi^1(s) \neq \emptyset$ for some agent $i^1 \in \mathcal{A}$, then there has to exist a maximal waiting chain $i^1, \ldots, i^n$, where the last agent i^n has an action $a \in \pi^n(s)$ such that $\omega(a) = i^n$. □

We can also show that all agents being optimally eager prevents infinite executions in the simple setting with uniform observability. We call a planning task $\langle s_0, A, \omega, \gamma\rangle$ *uniformly observable* if all agents share the same indistinguishability relations, both in the initial state s_0 and in all actions $a \in A$, which is tantamount to assuming that there is a *single* agent planning in the *belief space* of a partially observable nondeterministic (POND) problem (Bonet and Geffner 2000).

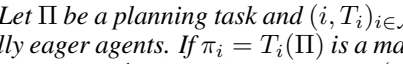
Proposition 8. *Let Π be a uniformly observable and solvable planning task and let $(i, T_i)_{i\in\mathcal{A}}$ be a group of optimally eager agents. Then all executions by $(i, T_i)_{i\in\mathcal{A}}$ of Π are finite.*

Proof sketch. Let $\pi_i = T_i(\Pi)$ for each agent $i \in \mathcal{A}$. Then for any transition $(\ldots, s, a, s', \ldots)$ occurring in an execution, we have $\kappa_{\pi_i}(s') \leq \kappa_{\pi_i}(s) - 1$ for the acting agent $i = \omega(a)$. Due to uniform observability and optimality, $\kappa_{\pi_i}(s') = \kappa_{\pi_j}(s')$ for any $j \in \mathcal{A}$ with $s' \in \mathrm{Dom}(\pi_j)$. Thus, by monotonicity, the execution ends after at most $\kappa_{\pi_i}(s')$ more actions. □

This means that in the uniformly observable setting, we can guarantee each execution to be successful, given all agents are optimally eager and act with respect to a maximal strong policy. Our result follows directly from Propositions 7 and 8.

Proposition 9. *Let Π be a uniformly observable planning task and $(i, T_i)_{i\in\mathcal{A}}$ be a group of optimally eager agents. If $\pi_i = T_i(\Pi)$ is a maximal strong policy for each $i \in \mathcal{A}$, then all executions of $(\pi_i)_{i\in\mathcal{A}}$ are successful.*

Unfortunately, if there is non-uniform observability, optimally eager agents cannot always prevent infinite executions, as we see in the following example.

Example 6. Consider another variant of Example 5, where the initial position of the chess piece is again fully observable, but where the information about possible target cells is non-uniformly distributed. The initial state is given as $s_3 =$ 

with at_i meaning that the piece is in cell i, and t_i meaning that cell i is a target position. The joint goal is $\gamma = (t_1 \to at_1) \wedge (t_5 \to at_5)$. Since agent 1 only knows that cell 1 is a target while agent 2 only knows that cell 5 is one, optimally eager agents would produce policies where they move the piece always in their own direction. Similar to Example 5, an infinite execution would then be $(s_3, \mathit{left}, s_2, \mathit{right}, s_3, \mathit{left}, \ldots)$.

We can see from Example 6 that it is generally not possible to solve the problem of infinite executions just by imposing restrictions on the types of agents. Since, in this example, for each state s and agent i there is only one possible choice of action as part of an i-strong policy (*left* for agent 1, *right* for agent 2), every conceivable combination of planning agents produces infinite executions. Hence we get the following:

Proposition 10. *For every group of at least two agents $(i, T_i)_{i\in A}$ there exists a partially observable and solvable planning task Π that has unsuccessful executions by $(i, T_i)_{i\in A}$ of Π.*

It is important to note that planning tasks with non-uniform knowledge do exist in which implicit coordination by optimally eager agents is guaranteed to be successful, i.e., without the potential occurrence of infinite executions. In particular, by allowing communication between the agents to be modelled directly as part of the planning task (using announcement actions), it is possible to solve more problems. One example in this class of planning tasks is the robots example from the introduction. To guarantee the existence of strong policies, we enable a robot that has reached its target position to publicly announce that fact as its final action (e.g., by visibly powering down).

A subjectively optimal policy for the square robot (that can be easily extended to a maximal, optimally eager one) is depicted in Figure 3. Solid edges denote actions and dashed edges denote indistinguishability. For clarity, only such indistinguishability edges are shown that talk about the agent designated to act and that, via uniformity, enforce inclusion of some action in the policy. Here, the square robot starts by moving out of the way of the circular robot, in order to allow the circular robot to move to the leftmost cell. This is because only from this position, the circular robot can make sure that the square robot will be able to reach its goal cell. Independently of the actual goal cell of the square robot, the square robot will then be able to move there and power down, after which the circular robot can finish the task. Note that this strategy will succeed for the given global initial state no matter which strong policy the circular robot chooses, just provided it is subjectively optimal. If the actual goal cell for the circular robot was the leftmost one, an optimally eager circular robot would already try to announce and power down earlier when having reached its destination. This contingency is covered by the maximal version of the policy (or with re-planning).

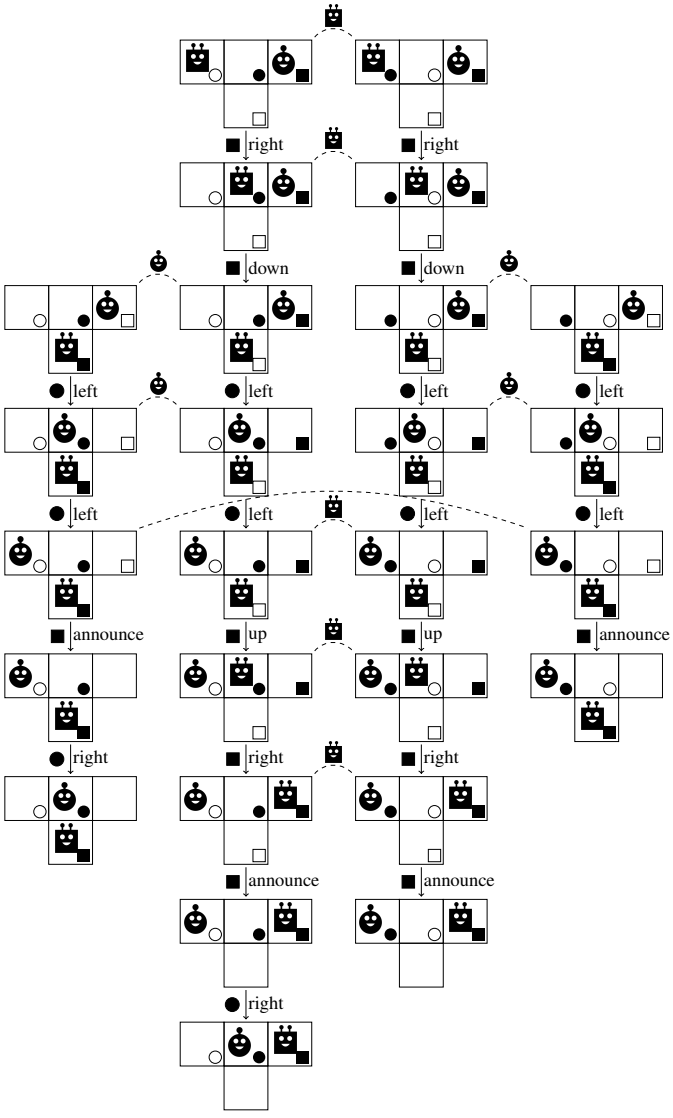


Figure 3: Depiction of a strong policy for the robots example

5 Conclusion and Discussion

We investigated how agent types impact the successfulness of implicit coordination in cooperative multi-agent planning with distributed knowledge and capabilities. We distinguished between *lazy* and *eager* agents and saw that lazy agents may produce *deadlocks* (waiting for one another to move), a problem that does not show up with eager agents. However, it turned out that over-eager agents can produce *infinite executions* instead (unintentionally working against each other), which can only be avoided under rather strong assumptions, namely if the agents optimize what we termed perspective-sensitive costs and if they have uniform observability. Under non-uniform observability, even optimally eager agents may unintentionally sabotage each other.

This means that there is no general positive result for non-uniformly observable settings such as the motivating multi-robot coordination example with uncertain target positions (Fig. 1). Still, in that particular example, implicit coordination does work and we can guarantee a successful execution taking both robots to their targets, if we allow the first robot that reaches its target to publicly announce that fact. However, to ensure successful implicit coordination, the square robot has to move first, and the total number of moves (excluding the announcement) will be 7 instead of 5 as in the full observability case.

For future work, we plan to investigate under which additional assumptions unintentional sabotage can be avoided. While, using our current solution concept, infinite executions often cannot be prevented, some improvements certainly can be made by increasing the agents' reasoning capacity. Currently, our only assumption is that by performing perspective shifts, agents can ensure other agents to be able to find the relevant subpolicies in the future. By making an even stronger assumption, namely that it is common knowledge that each observed state change has to be caused by the action of a rational agent of a certain type (e.g., an optimally eager one), it would be possible for agents to infer additional useful information. This way, e.g., in Example 6, the move of an agent would already signal the existence of the unknown target cell to the other agent. At least after one action from both agents, the remaining task would be fully observable and thus without potential for infinite executions. Similarly, in the robots example, it should be possible for the square robot to signal being at the goal position to the circle agent just by waiting, effectively rendering the additional announcement action unnecessary. We believe that by improving our solution concepts to enable this kind of reasoning, and by imposing sufficient conditions on the actions that are available to the agents, it will be possible to solve a wide range of cooperative tasks using implicit coordination.

References

Alexandre Albore, Hector Palacios, and Hector Geffner. A translation-based approach to contingent planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1623–1628, 2009.

Monica Anderson and Nikolaos Papanikolopoulos. Implicit cooperation strategies for multi-robot search of unknown areas. *Journal of Intelligent and Robotic Systems*, 53(4):381–397, 2008.

Thomas Bolander and Mikkel Birkegaard Andersen. Epistemic planning for single and multi-agent systems. *Journal of Applied Non-Classical Logics*, 21(1):9–34, 2011.

Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS 2000)*, pages 52–61, 2000.

Michael Brenner and Bernhard Nebel. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331, 2009.

Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1–2):35–84, 2003.

Thorsten Engesser, Thomas Bolander, Robert Mattmüller, and Bernhard Nebel. Cooperative epistemic multi-agent planning with implicit coordination. In *Proceedings of the 3rd Workshop on Distributed and Multi-Agent Planning (DMAP 2015)*, pages 68–75, 2015.

Geoffrey Hollinger, Sanjiv Singh, Joseph Djugash, and Athanasios Kehagias. Efficient multi-robot search for a moving target. *The International Journal of Robotics Research*, 28(2):201–219, 2009.

Filippos Kominis and Hector Geffner. Beliefs in multiagent planning: From one agent to many. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015)*, pages 147–155, 2015.

Kurt Konolige and Nils J. Nilsson. Multiple-agent planning systems. In *Proceedings of the 1st Annual National Conference on Artificial Intelligence (AAAI 1980)*, pages 138–142, 1980.

Christian Muise, Vaishak Belle, Paolo Felli, Sheila McIlraith, Tim Miller, Adrian R. Pearce, and Liz Sonenberg. Planning over multi-agent epistemic states: A classical planning approach. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015)*, pages 3327–3334, 2015.

Ronald P. A. Petrick and Fahiem Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS 2002)*, pages 212–222, 2002.

Ronald P. A. Petrick and Fahiem Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 2–11, 2004.

Ronald P. A. Petrick and Mary Ellen Foster. Planning for social interaction in a robot bartender domain. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*, pages 389–397, 2013.

Matthijs T.J. Spaan, Geoffrey J. Gordon, and Nikos Vlassis. Decentralized planning under uncertainty for teams of communicating agents. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 249–256, 2006.

Freek Stulp, Michael Isik, and Michael Beetz. Implicit coordination in robotic teams using learned prediction models. In *Proceedings 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*, pages 1330–1335, 2006.

Trial-based Heuristic Tree-search for Distributed Multi-Agent Planning

Tim Schulte
Institut für Informatik
Albert-Ludwigs-Universität
Freiburg, Germany
schultet@cs.uni-freiburg.de

Bernhard Nebel
Institut für Informatik
Albert-Ludwigs-Universität
Freiburg, Germany
nebel@cs.uni-freiburg.de

Abstract

We present a novel search scheme for privacy-preserving multi-agent planning. Inspired by UCT search, the scheme is based on growing an asynchronous search tree by running repeated trials through the tree. We describe key differences to classical multi-agent forward search, discuss theoretical properties of the presented approach, and evaluate it based on benchmarks from the *CoDMAP* competition.

Introduction

In multi-agent planning multiple agents attempt to satisfy a given objective by interacting appropriately. Many tasks require collaboration among agents, either because they cannot solve the problem on their own, or because they cannot do so in a cost effective way. Planning algorithms generating solutions to such problems can, in principle, implement one of two different concepts. First, *centralized* multi-agent planning algorithms grant a single agent access to the full description of the planning task. This agent then devises plans for the coordinated execution of all agents. Therefore, centralized multi-agent planning can be described as single-agent planning for multiple agents. Second, *distributed* multi-agent planning (DMAP) algorithms implement local planning by each of the agents. In contrast to centralized multi-agent planning, no trusted center is required. Each agent utilizes its own planning system that needs to exploit only those parts of the search space which are relevant to it. The agents inform each other about world states relevant to one another, therefore communication and coordination during the planning process are essential.

In this work, we consider a form of distributed multi-agent planning where agents cooperate with one another while keeping various information private. *MA-STRIPS* (Brafman and Domshlak 2013) is one of the most basic formalisms for this type of cooperative multi-agent planning, and several planning techniques have since been proposed to solve respective tasks (Nissim and Brafman 2013; 2014; Torreño, Onaindia, and Sapena 2014). The recent emergence of a dedicated competition on distributed and multi-agent planning (CoDMAP) (Štolba, Komenda, and Kovacs 2015) emphasizes the raising interest in this field.

In this paper, we introduce a novel search technique for privacy preserving distributed multi-agent planning. The approach is based on *trial-based heuristic tree-search* (THTS) (Keller and Helmert 2013); a general scalable framework for solving different types of planning tasks. Though originating from the field of probabilistic planning, THTS has recently been applied to classical planning (Schulte and Keller 2014). If we want to integrate THTS in a multi-agent planning context, the challenging part is to incorporate communication between the agents in such a way that the resulting algorithm preserves privacy and completeness. To achieve this, we define a suitable message passing scheme and explain how the agents can integrate states from other agents into their local search tree. Our main contribution is the definition of the resulting search framework, which we call *distributed multi-agent trial-based heuristic tree-search* (DMT). This framework extends the way of how distributed plans can be generated and so might be useful for portfolio approaches to multi-agent planning. We exemplify two DMT algorithms. The first approach resembles best-first search, comparable to MAFS, the second balances exploitation and exploration similar to UCT (Kocsis and Szepesvári 2006). We show that both algorithms are sound and complete, and evaluate them on a set of benchmark problems from the CoDMAP competition.

Background

We consider the problem of classical planning for multiple cooperative agents that maintain private information on their capabilities and internal states. The following definitions are based on MA-STRIPS (Brafman and Domshlak 2013) but use a multi-valued variable representation. Furthermore, privacy is not implied by the definition of the agents actions as in MA-STRIPS, but declared explicitly.

Privacy-Preserving Multi-Agent Planning

Definition 1. *A multi-agent multi-valued planning task (MMPT) is a tuple* $\Pi = \langle N, V, s_0, s_\star, \{A_i\}_{i \in N} \rangle$ *where*

- N *is a finite set of agents* φ_i*, indexed* $1, \ldots, |N|$,
- V *is a finite set of* finite-domain state variables*. Each* $v \in V$ *is associated with a domain* D_v*. A* partial variable assignment *over* V *is a function* s *on some subset of* V *such that* $s(v) \in D_v$ *wherever* $s(v)$ *is defined. A* partial variable assignment *defined for all variables in* V *is called* state*.*

- s_0 *is the* initial state.
- $s_\star$ *is a partial variable assignment over* V *called the goal.*
- A_i *is a finite set of actions available to agent* φ_i*. Each* action $a = \langle \mathit{pre}(a), \mathit{eff}(a), c(a) \rangle \in A_i$ *consists of two partial variable assignments over* V *called* precondition *and* effect*; and a* cost $c(a) \in \mathbb{R}_0^+$.

An action a is *applicable* in state s if its precondition *holds* in that state, i.e. s is identical to $\mathit{pre}(a)$ wherever $\mathit{pre}(a)$ is defined. Application of action a in state s, denoted by $a(s)$, yields *successor state* s' which is identical to $\mathit{eff}(a)$ where $\mathit{eff}(a)$ is defined, and identical to s, elsewhere. The solution to a MMPT is a sequence of actions $\pi = (a_1, \ldots, a_k)$ such that a_1 is applicable in s_0, every subsequent action is applicable in the state generated by its preceding action, and the goal holds in $a_k(\ldots(a_1(s_0))\ldots)$. Such a sequence is called *plan*. A plan is *optimal* if its incurred cost $\sum_{i=1}^{k} c(a_i)$ is minimal among all plans.

In privacy preserving domains, the set of variables V is partitioned into sets of private variables V_i^{int} containing those variables proprietary to agent φ_i, and a set of public variables V^{pub} containing the remaining variables which are common to all agents. Private variables can only be observed and be affected by actions of the agent to which the variables are private. The agents are mutually unaware of variables private to another agent. In principle, it is possible to define goals on public and private variables. For a simpler exposition of the algorithms presented below, we assume that goals are only defined for public variables $v \in V^{pub}$. In the same manner as the set of variables is partitioned into sets of private and public variables, each agents' set of actions is partitioned into a set of private actions A_i^{int} and a set of public actions A_i^{pub}. Private actions are only known to the agent to which they are private and only depend on and affect its private variables. Public actions can affect or depend on both public and private variables of the agent. During planning, the agents use both their private and public variables and actions, but restrict information exchange to the set of public variables and actions. To hide private preconditions or effects of public actions, the agents create and solely exchange public projections of their actions.

Definition 2. *A* public projection $a|_{pub}$ *of an action* a *of agent* φ_i *consists of the actions' precondition and effect restricted to the set of public variables* V_i^{pub}*:*

$$a|_{pub} = \langle \mathit{pre}(a)|_{pub}, \mathit{eff}(a)|_{pub}, c(a) \rangle$$

where $\mathit{pre}(a)|_{pub}$ *and* $\mathit{eff}(a)|_{pub}$ *are partial variable assignments over* V^{pub}*, such that* $\mathit{pre}(a)|_{pub} = \mathit{pre}(a)$ *for all variables* $v \in V^{pub}$ *for which* $\mathit{pre}(a)$ *is defined and* $\mathit{eff}(a)|_{pub} = \mathit{eff}(a)$ *for all variables* $v \in V^{pub}$ *for which* $\mathit{eff}(a)$ *is defined.*

The set of public projections of φ_i's public actions is $A_i|_{pub}$, the set of all agents public projections is $A|_{pub} = \bigcup_{i \in N} A_i|_{pub}$. Note that an MMPT planning task is a MA-STRIPS task when (1) the domain of each state variable is binary, (2) variables only affected or required by agent φ_i's actions are private to φ_i, and (3) actions that solely affect or depend on variables private φ_i are private to φ_i. In other words, MMPT is a generalization of MA-STRIPS.

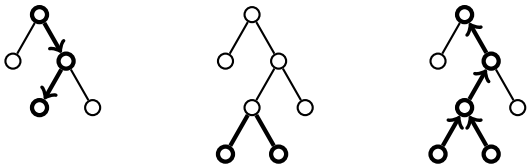


Figure 1: Phases of THTS.

Multi-Agent Forward Search

Multi-Agent Forward Search (MAFS) (Nissim and Brafman 2014) is a general search scheme for privacy preserving multi-agent planning. Each agent conducts a best-first search, maintaining its own *open* and *closed* list. Successors of expanded states are generated by using the agents own actions only. Whenever a state is generated for which another agent has an applicable public action, a message is sent to that agent. The message contains the full state, heuristic score and g-value of the sending agent. Private fluents of the state are encrypted such that only the relevant agents can decrypt it. When agent φ_i receives a message $m = \langle s, h_j(s), g_j(s) \rangle$ of some other agent φ_j, it checks whether s is already in its open or closed list. If this is not the case, φ_i puts s on its open list. If φ_i generated state s previously with higher cost, it puts s on its open list again and assigns new costs $g_j(s)$ to it. When an agent generates a goal state, it initiates a distributed plan extraction procedure by broadcasting the goal state in a message to all agents.

Trial-based Heuristic Tree-search

In the same way as MAFS is locally based on best-first search, DMT is based on trial-based heuristic tree-search. *Trial-based Heuristic Tree-Search* (Keller and Helmert 2013) is a generic search framework for probabilistic planning that was recently applied to classical planning (Schulte and Keller 2014). THTS algorithms repeatedly execute three phases. Each of these phases corresponds to a search component that must be specified in order to derive a concrete algorithm. In contrast to best-first search (BFS) approaches which expand nodes from an *open* list that is sorted by priority, THTS algorithms maintain a tree of nodes and select one of its leaf nodes for expansion in each search step. We will briefly sketch the three phases of THTS using the examples displayed in Figure 1.

1. *Selection* is the first phase of the algorithm with the objective to select one of the leaf nodes for expansion. Beginning from the root, a selection strategy recursively selects a child, until a leaf node is reached.
2. In the *initialization* phase, the previously selected leaf node is initialized. Successor Nodes are generated and integrated into the tree.
3. During *backpropagation* (or *backup*) phase new information, like value estimates or the number of times a node has been visited during selection, is propagated through the tree.

After the backpropagation phase, the algorithm starts again with the first phase. This process is repeated until a goal state is generated, or some limit is reached.

Distributed Multi-Agent THTS

We now present a complete and privacy preserving scheme for the distributed application of trial-based heuristic tree-search. The concept is similar to MAFS, where forward-search is concurrently executed while state information is exchanged between the planning agents according to a specific message passing scheme. Each agent performs THTS locally, using its own actions only. Whenever agent φ_i expands a state s in which a public projection of an action of φ_j is applicable, φ_i will send a message to φ_j containing s. φ_j then integrates s into its search tree, such that it can prospectively select s for expansion. To accomplish this, φ_j identifies a suitable parent and adds s as a child to it. In principle, any node can be used as a parent without soundness or completeness being compromised. However, since the tree structure is crucial to the success of THTS algorithms, it is important where new states are integrated. Let s be the result of applying the sequence of actions $(a_1, \ldots, a_k)$ in the initial state, i.e. $a_k(...(a_1(s_0))...) = s$, and let a_j be the last action of φ_j in that sequence. If a_j exists, φ_j adds s as a child to $s' = a_j(...(a_1(s_0))...)$. Otherwise, φ_j adds s as a child to the root. Note that φ_j is not aware of all actions in the sequence leading to s and hence cannot compute s'. We enable φ_j to identify s' by using a special message type.

Definition 3 (State message)**.** *A* state message *from φ_i to φ_j for state s is a tuple $m = \langle s, h_i, g_i, T\rangle$, where*

- *s is a state; private components are encrypted, such that each agent can only decrypt its own private components.*
- *h_i is a value estimate of φ_i for state s,*
- *g_i is the cost of φ_i to establish state s,*
- *T is a set of* state tokens*.*

Each *state token* belongs to an agent φ_k and contains a state identification number. This number references a node in the local search space of φ_k and is meaningless to all other agents. Figure 2 illustrates how tokens are used to integrate states. Here, two agents φ_i and φ_j are planning concurrently. Numbers next to nodes depict state IDs that correspond to the local state represented by the node. Nodes associated with states for which the other agent has an applicable public projection are rendered in bold. When φ_j initializes the node with state ID 3, it transmits message $m_1 = \langle s, 7, 2, \{\varphi_j \mapsto 3\}\rangle$ to φ_i. m_1 contains a token that enables φ_j to identify the node labelled with 3. When φ_i receives m_1, it creates a new search node for s. Because m_1 contains no token for φ_i, the new node is attached as a child to the root. Later on, φ_i initializes the node with state ID 5, for which φ_j has an applicable public projection. The message $m_2 = \langle s', 5, 2, \{\varphi_j \mapsto 3, \varphi_i \mapsto 5\}\rangle$ is sent back, from φ_i to φ_j. Because state 5 was generated in a branch to which φ_j contributed an *ancestor* state, the token $\varphi_j \mapsto 3$ is attached to the message, along with the new token $\varphi_i \mapsto 5$ of φ_i. The latter token enables φ_i to identify the state corresponding to state ID 5. When φ_j receives m_2 it looks up its

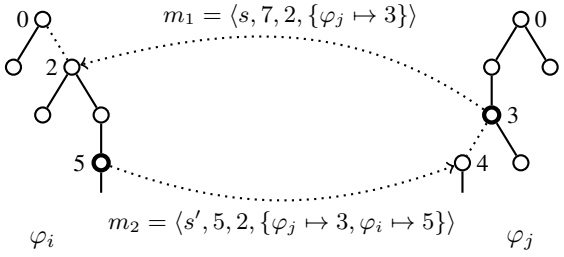


Figure 2: State integration.

Algorithm 1: DMT for φ_i

Data: $\langle N, V_i^{int}, V^{pub}, s_0, s_\star, A_i^{int}, A_i^{pub}, A|_{pub}\rangle$
Result: plan $\pi = \langle a_k \in A_i\rangle_{k=1}^{K}$

```
root ← new tree from s_0
while within computational budget do
    σ ← root
    if ¬l(σ) then
        while children(σ) ≠ ∅ do
            σ ← select(children(σ))
        initialize(σ)                    // memorizes plans
        send-messages(σ, N)              // distribution
        mark σ for backup
    process-messages()                   // integration
    backup()
return best memorized plan
```

token $\varphi_j \mapsto 3$, creates a new node for state s', and attaches it as a child to the node with state ID 3.

An overview of the resulting search scheme is depicted in Figure 3. The algorithms main routine is defined in Algorithm 1. Methods *process-messages, select, initialize, send-messages* and *backup* correspond to *integration-, selection-, initialization-, distribution-* and *backup-phase* respectively. These components are described in detail below. For ease of exposition we define the following functions to access information stored with each search node σ:

- *state*(σ): associated search state
- *par*(σ): parent of σ
- *children*(σ): set of children of σ
- *action*(σ): action leading from *state*(*par*(σ)) to *state*(σ)
- $h(\sigma)$: value estimate for σ

We refer to a search node σ and its associated state s = *state*(σ) interchangeably where convenient.

Selection A selection strategy is a function that maps from a set of search nodes Σ to a single node $\sigma \in \Sigma$. To ensure that the node selected last in the selection phase is an uninitialized leaf node, a special *locking mechanism* is used. The idea is to mark initialized nodes from which no uninitialized leaf node is reachable as *locked* and to ignore such nodes in the selection phase. Each initialized node σ^* without any non-locked children is locked in the backup phase by setting $l(\sigma^*) = \mathit{true}$. New nodes created in the initialization phase are non-locked by default. We use the following two

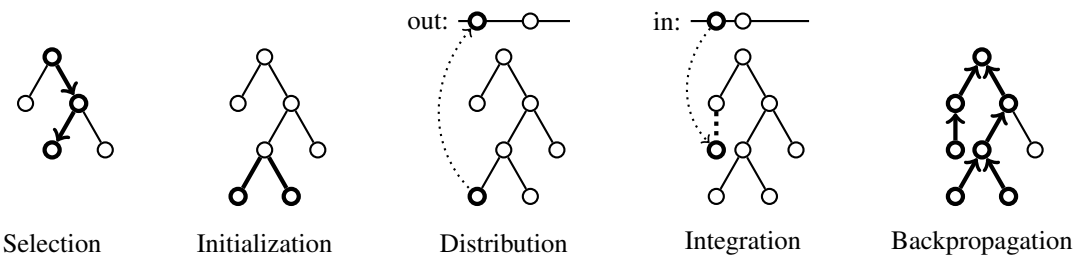


Figure 3: Phases of DMT.

selection strategies.

$$\text{gbfs}(\Sigma) = \underset{\sigma \in \Sigma, \neg l(\sigma)}{\arg\min}\ h(\sigma)$$

$$\text{ucb}(\Sigma) = \underset{\sigma \in \Sigma, \neg l(\sigma)}{\arg\min}\ \overline{h}(\sigma) - c \cdot \sqrt{\frac{\ln v(par(\sigma))}{v(\sigma)}}$$

gbfs constitutes a greedy best-first search variant, selecting the successor node σ with the best (minimum) value estimate $h(\sigma)$.

ucb aims to balance exploration and exploitation by using a selection formula similar to UCB1 (Auer, Cesa-Bianchi, and Fischer 2002) found in UCT algorithms (Kocsis and Szepesvári 2006). Here, $\overline{h}(\sigma) \in [0, 1]$ is the normalized value estimate of σ, such that $\overline{h}(\sigma^\star) = 0$ for the node $\sigma^\star$ with the best (minimum) value estimate from Σ and $\overline{h}(\sigma^-) = 1$ for the node σ^- with the worst (maximum) value estimate from Σ. All other nodes $\sigma' \in \Sigma$ are interpolated accordingly. The number of times a node has been selected during selection phase is denoted by $v(\sigma)$ (visits). *ucb* selection favours nodes with fewer visits. Coefficient c is a weight bias to increase or decrease the desired amount of exploration. The higher c the higher the bias towards exploration. *gbfs* and *ucb* are just two examples of selection strategies that can be used in line 6 of Algorithm 1.

Initialization Algorithm 2 specifies how a node σ is initialized by an agent φ_i. First, a heuristic value for *state*(σ) is computed and $h(\sigma)$ is set to that value. Then, all successor states s' are generated. For each successor state s' that is not already in the tree a new node σ' is created and added to *children*(σ); its values are set accordingly (Algorithm 2, line 9-11). If a successor state s' is already in the tree, the respective search node σ' with *state*$(\sigma') = s'$ is determined. If the new path to s' induces lower costs than the existing path, the subtree rooted at σ' is moved to *children*(σ) by adapting parent and child pointers of the involved nodes (Algorithm 2, line 16-18). Since the former parent of σ' lost a child, the value estimates of all nodes along the path from the former parent to the root are deprecated. Therefore, before σ' is moved to its new parent σ, *par*(σ') is marked to get updated in the next backup phase (line 15).

Distribution Let σ be the node φ_i initialized last. In the *distribution* phase φ_i creates a *state message* $m = \langle \textit{state}(\sigma), g(\sigma), h(\sigma), T \rangle$, such that T contains a token of φ_i to identify σ. For each other agent the first token traceable on the path from σ to the root is attached to T. Then, φ_i sends m to all agents that have a public action projection applicable in s.

Algorithm 2: Initialization for φ_i

```
Data: σ, A_i = A_i^int ∪ A_i^pub
Result: modified tree node σ
1  s ← state(σ)
2  h(σ) ← evaluate heuristic function for s
3  foreach action a ∈ A_i applicable in s do
4      s' ← a(s)
5      if s' is a goal state then
6          extract and memorize plan
7      if s' is not in the tree then
8          σ' ← new node
9          par(σ'), action(σ'), h(σ') ← σ, a, h(σ)
10         state(σ'), v(σ'), l(σ') ← s', 0, false
11         children(σ) ← children(σ) ∪ {σ'}
12     else
13         lookup σ' where state(σ') = s'
14         if g(σ) + c(a) < g(σ') and ¬l(σ') then
15             mark par(σ') for backup
16             remove σ' from children(par(σ'))
17             par(σ'), action(σ'), h(σ') ← σ, a, h(σ)
18             children(σ) ← children(σ) ∪ {σ'}
```

Integration Following the distribution phase φ_i integrates each state s received in a message $m = \langle s, h_j, g_j, T \rangle$ into its local search tree. First, φ_i identifies the new parent σ^* for s by looking up its token from T. If T contains no token for φ_i, then σ^* is set to the tree's root node. If s is new to φ_i, a new search node σ is created and added to *children*(σ^*). If some node σ' representing s is already in the tree, it is moved to *children*(σ^*) in case s is reachable with lower cost that way. As in the initialization phase, when σ' is moved, its old parent is marked for backup.

Backpropagation The backup function starts at the node σ that was initialized last and updates its values. The nodes visits are increased by one, its value estimate is set to the minimum among its non-locked children, and the locked flag

is set if the node itself has no non-locked child:

$$v(\sigma) = v(\sigma) + 1$$
$$h(\sigma) = \min_{\sigma' \in children(\sigma), \neg l(\sigma)} h(\sigma')$$
$$l(\sigma) = \bigwedge_{\sigma' \in children(\sigma)} l(\sigma')$$

Then backup continues with the nodes parent $par(\sigma)$ and updates it accordingly. This process is repeated until the root node is reached. In case other nodes have been marked for backup, during initialization or integration, the process is repeated for each marked node. This may lead to the same node getting updated multiple times, but can easily be avoided by using a backup queue.

Trial Length When a node σ is initialized, all its successors are generated and associated state messages are sent. Before the agent continues with the integration phase, it can select one of the newly generated nodes and initialize it as well. By alternatingly executing selection-, initialization- and distribution phase, multiple nodes can be initialized in each search step. The number of nodes to get initialized in a single search step is denoted as *trial length*. For simplicity we did not include it in Algorithm 1. It can easily be implemented by looping around lines 5-10.

Plan Extraction

If an agent φ_i generates a state that satisfies the goal a valid plan can be extracted. φ_i informs all other agents about the goal state and initiates a distributed plan extraction process. First, it traces back all states of its local plan, until a state s^* is reached that was received in a state message from another agent φ_j. Then, φ_i sends a plan extraction request to φ_j, including s^*. φ_j then continues to trace back its local plan, beginning from the state received in the state message. This process is repeated until some agent reaches the initial state, at which point plan extraction ends. The solution plan is sequential but can often be parallelized.

The first solution found is not necessarily the optimal solution. Therefore, if more planning time is available, DMT search can easily be extended to progressively search for better solutions. When a plan is extracted, its cost is computed and the plan with the best cost found so far is memorized as π. From then on each agent marks search nodes with a higher g-value than π as locked. Each time a new goal state is reached, its g-value is computed, and, if it is an improvement, π is updated. Once each agents root is locked, π is the optimal solution. If the time limit is exceeded earlier, π is returned.

Soundness and Completeness

Lemma 1. *Each state s in the search tree of an agent φ_i is reachable.*

Proof sketch. The first state generated by DMT is the initial state. Each subsequently generated state is reached by an action applied in a previously generated state. Therefore, every state s in the search tree represents a valid sequence of actions that is applicable starting with the initial state, and that results in state s. Hence, if a state satisfies the goal, a valid plan can be extracted. □

Lemma 2. *If a goal is reachable by some sequence of actions then some agent will generate a goal.*

Proof sketch. We will only consider sequences in which a private action of an agent is followed by another action of that agent. In (Nissim and Brafman 2014) it was shown that it suffices to consider such sequences for any goal that involves public variables only. Completeness must be decided individually for each concrete DMT algorithm, because it depends on the components used. In the following we argue that the presented two selection functions (*gbfs* and *ucb*), in combination with the other components presented, yield complete algorithms.

In every search step, each agent initializes a new leaf node and generates all its successors. Nodes without children are locked, either because they are dead-ends or because all of their successor states can be reached on shorter paths and have been moved to other states in the tree. Therefore, all paths that do not lead to a solution will eventually be locked. Both selection functions solely select non-locked nodes and will eventually, for the lack of an alternative, select a node along a path that leads to a goal. Given sufficient time, all nodes along such a path will be selected until the goal is reached. If no such path exists in an agents local search space, the agent exhaustively generates all possible states, until its root node is locked.

We now regard sequences that involve actions of different agents and that lead to a goal state. It is easy to see that each agent transmits the last state s, established by a subsequence of its own actions, to the agent owning the next action in the sequence. If the next action is private, it is always followed by another action of the same agent, until one action is public. This actions public projection is applicable in state s, and hence sent to the agent in a state message. □

Relation to MAFS

MAFS and DMT are both schemes for distributing search algorithms, such that completeness and privacy is preserved. They differ in the types of algorithms that they support. MAFS supports forward search algorithms where nodes are expanded from an open list, while DMT supports THTS algorithms that use a search tree instead. In MAFS, states are inserted into an open list together with a static value estimate computed prior insertion. The value estimates of states in the open list never changes, hence, their relative order remains unchanged. DMT algorithms, by way of contrast, insert states into a tree together with value estimates that are continuously subject to change. Therefore, algorithms that depend on a dynamic node ordering, like UCT (Kocsis and Szepesvári 2006), can easily be expressed as DMT algorithms by defining appropriate selection, backup and initialization functions. It is not possible to implement these algorithms competitively with an open list, especially when a large number of nodes change their relative position in each search step.

Another major difference between the two approaches concerns the *reopening* of closed states. In MAFS, a newly

	$t = 1$			$t = 100$		
Domain	*mafs*	*dmt-bfs*	*dmt-gus*	*mafs*	*dmt-bfs*	*dmt-gus*
blocksworld	-	-	-	3	-	2
depot	1	1	-	2	-	4
driverlog	16	16	15	17	16	16
elevators	-	-	-	1	-	-
logistics	8	5	1	9	5	2
rovers	10	6	1	19	19	18
satellites	3	2	3	6	11	9
sokoban	4	8	8	3	9	8
taxi	17	14	11	10	14	6
wireless	2	2	-	1	1	-
woodworking	6	3	1	5	4	6
zenotravel	13	12	12	13	13	13
Total (240)	80	69	52	89	92	84

Table 1: Coverage

generated state s is put on the open list, only, if it is not already on the closed list or if its new g-value is smaller than the registered g-value. In the latter case, states previously generated as successors to s will potentially be reopened in future search steps as well. In DMT, if s is already in the tree and its new g-value is smaller than the current g-value, the subtree of the existing node is moved to the node that is currently initialized. This is achieved by adapting parent and child pointers of the involved nodes (Algorithm 2, line 15-18). Successor states must not be generated all over again.

Evaluation

The presented DMAP algorithms were implemented in a distributed multi-agent planning system written in Go. Experiments were run on a PC with an Intel 3.2 Ghz quad-core CPU and 4 GB of RAM. The four cores were shared among all agents; assignment of processor time was left to the Linux process scheduler. For communication between processes a TCP connection was used. We experimented with the set of benchmarks from the CoDMAP competition (Štolba, Komenda, and Kovacs 2015) consisting of 12 domains with 20 problem instances each. Planning time was limited to two minutes per planning task. Table 1 shows coverage results for the tested configurations: Multi-agent forward search (*mafs*), DMT with greedy selection (*dmt-bfs*) and DMT with ucb selection (*dmt-ucb*). The configurations were tested with a trial length of either 1 or 100. In all cases FF heuristic (Hoffmann and Nebel 2001) was used to compute state value estimates. The heuristic function was applied to the agents local problem projection, containing the agents private and public variables and actions together with the other agents public actions projections.

Regarding configurations with a trial length of 1 ($t = 1$), the numbers reflect that *mafs* performs best, solving 11 instances more than *dmt-bfs*, and 28 instances more than *dmt-gus*. The only domain in which *dmt-bfs* and *dmt-gus* solve more instances than *mafs* is *sokoban*. We expected *dmt-bfs* to perform slightly worse than *mafs*, because both approaches search the state space in a greedy manner, but the DMT approach is computationally more expensive. Due to the brief time limit of 2 minutes, this also affects coverage. When the trial length is set to 100 ($t = 100$) all configurations improve in coverage. *dmt-gus* records the biggest gain solving 32 additional instances, followed by *dmt-bfs* with 23 and *mafs* with 9 additional instances solved. The increase in coverage is most noticable in the *rovers* domain where *mafs*, *dmt-bfs* and *dmt-gus* increase their coverage by factor 1.9, 3.17 and 18.0 respectively. Increasing the trial length causes regular search to perform additional exploration and encourages faster escape from local minima. This is most beneficial in domains where many solution paths exist but search is misguided into local minima by inaccurate heuristic values. When combining the solutions solved between configurations, we find that the two MAFS configurations solve 102 problems combined, while the DMT configurations solve 110 problems combined. A portfolio planner running *dmt-gus*, *dmt-bfs* and *mafs* with $t = 100$ for 2 minutes each solves 117 instances, which shows that MAFS and DMT complement each other well.

Conclusion

In this paper we presented DMT, a novel and privacy preserving scheme for distributing THTS algorithms. Based on DMT, we derived two concrete algorithms and showed them to be sound and complete. The algorithms were evaluated on a set of benchmark instances from the CoDMAP competition and compared to classical multi-agent forward search. Overall, DMT and MAFS approaches performed equally well, complementing each other in a promising way. In future work we will create and analyze new DMT algorithms to further exploit such complementary strengths. Additionally, we would like to use DMT in settings where goals are also defined for private variables.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2-3):235–256.

Brafman, R. I., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence* 198:52–71.

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

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Proceedings of the Seventeenth European Conference on Machine Learning (ECML 2006)*, 282–293.

Nissim, R., and Brafman, R. I. 2013. Cost-optimal planning by self-interested agents. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*.

Nissim, R., and Brafman, R. I. 2014. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research (JAIR 2014)* 51:293–332.

Schulte, T., and Keller, T. 2014. Balancing exploration and exploitation in classical planning. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*.

Štolba, M.; Komenda, A.; and Kovacs, D. L. 2015. Competition of distributed and multiagent planners (CoDMAP). In *The International Planning Competition (WIPC 2015)*, 24–28.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014. FMAP: distributed cooperative multi-agent planning. *Applied Intelligence* 41(2):606–626.

Hierarchical Planning with Traffic Zones for a Team of Industrial Transport Robots

Stefan Imlauer and **Clemens Mühlbacher** and **Gerald Steinbauer**
Institute for Software Technology, Graz University of Technology, Austria
{simlauer,cmuehlbacher,steinbauer}@ist.tugraz.at

Michael Reip and **Stephan Gspandl**
incubed IT, Hart bei Graz, Austria
{m.reip,s.gspandl}@incubedit.com

Abstract

Fleets of transport robots in industrial settings can gain performance in relation to safety and throughput by using zones with traffic constraints in the environment. In this paper we present a hierarchical navigation system for a fleet of robots that is able to consider such zones in planning. The performance of the proposed planning system is based on an enriched roadmap representation, a central zone reservation and a search heuristic that is able to cope with waiting times for temporary unavailable zones. The proposed system was implemented on top of existing industrial transport robots and evaluated in an industrial use case.

1 Introduction

Nowadays it becomes increasingly common to use fleets of autonomous transport robots to carry out transport tasks. Prominent examples are the replacement of traditional conveyors in warehouse automation and the automated delivery of parts in production settings. The major advantages of transport robots in comparison to traditional installations are: (1) the installation costs are much lower than static conveyors, (2) the robots are much more flexible because they can be simply rerouted if setups change and (3) the throughput of a robot team scales quite well with its size. An example of an industry-grade robot is depicted in Figure 1.

Figure 1: Robots of the *incubed IT* system. © incubed IT

In order to realize a fleet of automated transport robots several challenges have to be tackled. The basic challenge is to realize safe and reliable navigation of individuals between locations to perform individual transport tasks. Safety plays a major role here because the robots share their environment with humans. Research in robotics provided almost perfect solution to this challenge using various planning methods to find a route. On the fleet level the allocation of transport tasks to the robots and the coordination of the navigation of the individual robots are challenges. The former is usually solved as a centralized scheduling problem while the latter can be solved in a centralized, distributed or hybrid way.

In this paper we focus on the planning and coordination of the navigation of the robots in industrial settings. Although robots are able to navigate freely in the environment making use of alternative routes for safety and operational reasons and to optimize the throughput the navigation of the robots might be restricted. This is realized by introducing traffic zones into the environment. Similar to certain traffic regulations in public streets such zones impose constraints on robots navigating in it. For example in a traffic zone one can restrict that only one robot is allowed to enter this zone. Given the possibility of such traffic zones a warehouse manager can optimize safety and throughput by installing a one-way zone in a narrow corridor or a single robot zone near an emergency exit.

In order to integrate the concept of traffic zones into the multi-robot system we propose a hierarchical planning approach. Traffic zones are treated like common resources where their use is restricted by their constraints. Robots can reserve zones for a (future) time period on a central server. Each robot plans its route locally based on a roadmap and the current reservations. If successful the robot comes up with a route and a consistent reservation of zones along this route. The roadmap is a graph-based representation of the environment enriched with information about the constraints imposed by the traffic zones. The planning algorithm of the individual robots is an extension of the well-known A* algorithm (Hart, Nilsson, and Raphael 1968). The key contribution here is the development of a heuristics that estimates the time to the goal taking into consideration the traffic constraints such as a waiting time to enter a restricted zone. Such an intelligent heuristics allows the planner to prefer a detour if a zone along the direct path is not available for some time. The actual navigation is done using a more fine-grained path obtained using a gridmap of the reserved zones.

The remainder of the paper is organized as follows. In the next section we will briefly review some related research. In Section 3 we will state the problem of multi-robot navigating in an environment with traffic zones more formally. In the proceeding section we describe the system architecture. In Section 5 we discuss the planning algorithm in detail. This section is followed by an experimental evaluation. Finally we draw some conclusions and point out future work.

2 Related Research

We start our brief review of related research with a method which uses a central server for coordination. In (Kleiner, Sun, and Meyer-Delius 2011) the authors describe a method which creates a roadmap based on an adaptive gridmap. The gridmap is initially created through an simultaneous localization and mapping (SLAM) algorithm. During the life time of the system every robot reports inconsistencies of this map to the central server. The central server uses a hidden Markov Model (HMM) combined with the gridmap to update the gridmap according to the reported inconsistencies. To perform this update for every gridmap cell a Bayes filter is used to keep track of the reported inconsistencies. If a certain amount of cells are changed in the gridmap the roadmap is recalculated. With the help of this roadmap and a linear programming method the ideal routing of the deliver tasks is calculated taking into account capacities as well as flow directions. Finally the single robot queries the routing for its task. The main difference of this approach to the approach presented within this paper is that the central server is used within our approach for bookkeeping about zones only. The planning is done on the robots allowing good scalability.

Another approach which uses a central server was described in (Ryan 2010). The method poses the planning problem of a fleet of robots as a constrained satisfaction problem (CSP). The method composes a graph to represent the different robots and its environment. In order to solve the problem the graph is split into subgraphs which represent special structures like cliques and halls. Each subgraph is encoded as CSP. Again in contrast to the approach presented in this paper the authors use a central planning instance. Which always poses the risk to scale badly with a fleet.

Many other approaches of multi-robot navigation don't rely on an central coordination. This allows the approach to be more scalable than a complete centralized one. These concepts follow the model of the distributed robot architecture (DRA) (Siciliano and Khatib 2008, chap. 40.2.3). We will discuss two different approaches following this idea.

A method we want to discuss as an example for a decentralized method was presented in (Kleiner, Nebel, and others 2014). Each robot plans its path without considering the other robots. Then the robot starts to navigate along the calculated path. If two robots meet each other simple behaviors are applied. These behaviors are designed to handle crossings, congestions and more. To coordinate these behaviors the robots communicate locally to their neighbors. The authors showed that through this method other multi-robot navigation methods are outperformed. Especially when a fleet becomes large. In contrast to our approach no central component is used at all. Thus there is no possibility to deal with resources in a global way. Considering a long corridor which can only be traversed by one robot. As there is no global coordination both robots would enter the corridor and would block each other. Such an effect would be avoided through our central server.

The second method which uses a decentralized method was presented in (Wang and Premvuti 1995). All robots use a network of traffic segments. Each traffic segment has only a finite capacity and thus needs to be handled properly. If a robot leaves or enters a traffic segment the robots communicate with each other to negotiate which robot can enter the traffic segment. As this negotiations are only performed locally the path a robot is traveling might not be optimal in a global sense. This is caused as no information about future needs are presented in the system and thus a robot can not plan ahead to avoid traffic segments.

Beside the research on multi-robot navigation the management of resource allocation with multiple robots is a close related field. The different traffic zones in the navigation scenario are the resources the robots need to shared. In the remainder of this section we will discuss approaches of resource management and the impact on the work presented in this paper.

In (Alami et al. 1995) the authors describe the so called 'Plan-Merging Paradigm'. The idea is that each robot creates a plan satisfying its goal. Before the plan can be executed the robot has to perform plan merging operations. These operations are communicated through a shared resource or by direct communication to all robots. By using these operations conflicting plan steps are detected and the robot can repair its plan accordingly. This concept is used within the method presented in this paper as every robot plan its path and afterwards it coordinates the plan the other robots through a central server.

3 Problem Definition

As motivated in the introduction the problem we consider is to navigate a fleet of robots within an environment containing traffic zones. Thus we consider a fleet of n robots $\mathcal{R} := \{R_1, R_2, \ldots, R_n\}$. Where each robot has for any time $t \in \mathbb{R}^+$ a position given through the function $\mathbf{pos}: \mathcal{R} \times t \to \mathbb{R}^2$. Additionally the direction for each robot at a given time is defined through a direction vector represented by the function $\mathbf{dir}: \mathcal{R} \times t \to \mathbb{R}^2$.

All robots navigate in an environment $\mathcal{E}$. To represent this environment we use convex polygons where each convex polygon $\mathbb{P}$ is defined through a list of points $\mathbb{P} := (P_1, P_2, \ldots, P_l), P_i \in \mathbb{R}^2, 1 \leq i \leq l$. Furthermore we use an inclusion operator $\overline{\in}$ which is defined as follows:

$$\overline{\in}: \mathbb{R}^2 \times \mathbb{P} \to \begin{cases} 1, & \text{if position } p \text{ within the area of } \mathbb{P} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The environment is now characterized through $\mathcal{E} := \{x \in \mathbb{R}^2 \mid x \overline{\in} \mathbb{P}\}$. As the robots are only allowed to drive in the environment it holds that $\forall r \in \mathcal{R}, t \in \mathbb{R}^+ . pos(r,t) \overline{\in} \mathcal{E} = 1$.

Beside the robots the environment contains static stations $\mathcal{L} := \{L_1, L_2, \ldots, L_k\}, L_i \in \mathcal{E}, 1 \leq i \leq k$. These static

stations are the start respectively the end of a navigation task $T = \langle s, g \rangle$, where $s \in \mathcal{L}$ and $g \in \mathcal{L}$. Additionally such a navigation task is assigned to exactly one robot. As the problem considers the navigation of the complete fleet we define the set $\mathcal{T}$ to contain all navigation tasks. As we consider the navigation of the fleet we will not discuss the assignment of the tasks which itself is a complex planning problem.

To define the traffic zones within the environment we use a set of m areas $\mathcal{A} := \{A_1, A_2, \ldots, A_m\}$. Each area A_i is defined as a tuple $\langle \mathbb{P}_i, \mathcal{C}_i \rangle$, where $\mathbb{P}_i$ is the polygon of the area and $\mathcal{C}_i \subseteq \mathcal{C}$ denotes a set of constraints imposed on the area.

We consider the following constrains $\mathcal{C}$ for an area:

- **N-Robots** c_N
  A maximum number of robots are permitted in an area at any time. N_{A_i} is defined as the maximum number of allowed robots in zone A_i:

$$\forall t \in \mathbb{R}^+ . \left(\sum_{r \in \mathcal{R}} \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \right) \leq N_{A_i} \tag{2}$$

- **Single Robot** c_S
  Only one robot is permitted in an area at any time:

$$\forall t \in \mathbb{R}^+ . \left(\sum_{r \in \mathcal{R}} \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \right) \leq 1 \tag{3}$$

- **Forbidden** c_F
  No robot is allowed to traverse the area:

$$\forall t \in \mathbb{R}^+ . \left(\sum_{r \in \mathcal{R}} \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \right) = 0 \tag{4}$$

- **One Way** c_O
  This zone is only traversable in a specific direction: The direction of this zone is defined by $\mathbf{dir}_{A_i}$.

$$\forall t \in \mathbb{R}^+, r \in R \; . \; \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \rightarrow \mathbf{dir}(r,t) = \mathbf{dir}_{A_i} \tag{5}$$

- **Velocity** c_V
  The maximal allowed velocity of a zone is restricted to $\mathbf{vel}_{A_i}$:

$$\forall t \in \mathbb{R}^+, r \in R \; . \; \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \rightarrow \|\mathbf{dir}(r,t)\| \leq \mathbf{vel}_{A_i} \tag{6}$$

By combining these simple traffic constraints and areas also more complex traffic zones can be described. We will give an example of a more complex traffic zone consisting of several simpler traffic zones.

- **Right Hand Traffic** c_R
  Robots in this region are forced to drive on the right hand side. This constraint is built with two c_O constraints. The areas corresponding to the constraints have to be adjoined and face in opposite directions (Figure 2).

To state the planning problem we define the set of all constraints imposed by $\mathcal{A}$ as $\mathbb{C} = \bigcup_{A_i \in \mathcal{A}} \mathcal{C}_i$. Additionally we define the path of a robot r, following (Choset 2005, chap. 3), as a continuous mapping $\pi^{(r)} \colon [t_r, t_r + \Delta t_r] \rightarrow \mathcal{E}$. Given a transport order for a robot r from location s_r to a location g_r the start of the path is constraint to $\pi^{(r)}(t_r) = s_r$ and the goal of the path is constraint to $\pi^{(r)}(t_r + \Delta t_r) = g_r$. t_r represents the starting time of the task while Δt_r represents its duration. Furthermore the velocity on the path of robot r at time t is computed in Equation 7.

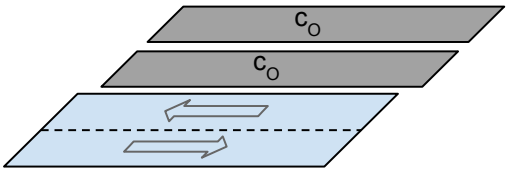


Figure 2: Constraints of Right Hand Traffic. Illustration of the constraint decomposition. (c_O) shows the one way constraints.

$$\mathbf{dir}(r,t) = \frac{d\pi^{(r)}(t)}{dt} = \begin{bmatrix} \frac{d\pi_x^{(r)}(t)}{dt} \\ \frac{d\pi_y^{(r)}(t)}{dt} \end{bmatrix} \tag{7}$$

As we are concerned with all tasks the fleet executes we want to find the set of paths Π which comprises the paths of all robots and minimizes the maximal end time $t_{r_i} + \Delta t_{r_i}$ for all paths $\pi^{(r_i)} \in \Pi$ subjected to $\mathbb{C}$, or more formally stated in Equation 8.

$$\Pi^* = \underset{\Pi}{\operatorname{argmin}} \left\{ \max_{\pi^{(r_i)} \in \Pi} (t_{r_i} + \Delta t_{r_i}) \right\} s.\ t.\ \mathbb{C} \tag{8}$$

4 Realization

The optimal solution for this problem requires the complete information of every task assigned to a robot from the beginning. Since this information is simply not available in a real industrial system we are not able to generate an optimal solution for that use case. Therefore we relax the planning problem and allow the robots to look individually for their optimal solution given the already known paths of other robots. We define in Equation 9 a new planning problem which minimizes the end time of the task of a specific robot r_i subjected to $\mathbb{C}$ and a given set of paths Π from other robots.

$$\pi^{(r_i)*} = \underset{\pi^{(r_i)}}{\operatorname{argmin}} \{(t_{r_i} + \Delta t_{r_i})\} \, s.\ t.\ \mathbb{C}, \Pi \tag{9}$$

The given set Π of already known paths of other robots can be mapped to a set of n tuples $\mathcal{I}$ which represents the traversal of a robot through a zone A_j. $I_i \in \mathcal{I}$ denotes the tuple $I_i := \langle t_{s_i}, t_{e_i}, r_i, \mathbb{P}_i \rangle$ where $t_{s_i}, t_{e_i} \in \mathbb{R}^+$ are the time robot $r_i \in \mathcal{R}$ enters respectively leaves the area $\mathbb{P}_i$ of A_j.

This information formulates an additional constraint c_I:

$$\forall I_i \in \mathcal{I}, t \in [t_{s_i}, t_{e_i}].\mathbf{pos}(r_i, t) \overline{\in} \mathbb{P}_i = 1 \quad (10)$$

This constraint can be used to reformulate the problem above as follows:

$$\pi^{(R_i)*} = \underset{\pi^{(R_i)}}{\operatorname{argmin}} \{(t_{R_i} + \Delta t_{R_i})\} \, s.\, t.\, \mathbb{C} \cup \{c_I\} \quad (11)$$

Thus the planning problem is to find an optimal path for individual robots by considering all constraints of the areas and all intervals of other robots within certain areas. As not every area has constraints concerning for instance the number of robot in an area we don't consider all intervals but instead only those intervals where the related area needs a reservation.

System Overview

In order to realize the system we use a central server which keeps track of all the reservations. Each robot queries the central server for the current reservations of areas and uses this information to calculate its optimal path assuming a static environment. Then the robot tries to reserve all necessary areas along its path at the server. The server validates if the requested reservations violates any constraint imposed by the traffic zones. If this reservation succeeds the robot starts moving along the calculated path. Otherwise the robot fetches the reservations again and starts over again with the planning. This optimistic approach avoids issues due to multiple robots planning their paths at the same time while still achieving a decent throughput. In general this hybrid approach allows a high scalability of robot fleets due to balanced resource utilization and lightweight communication. Especially as each robot is already equipped with a industrial computer which is powerful enough to perform the planning procedure in a short amount of time.

Environment Representation

Besides an effective management of the reservations of zones a powerful representation of the environment and the traffic zones is needed. The starting point for the planning is a representation based on labeled polygons representing different traffic zones as well as obstacles. An example for such an environment is depicted in Figure 3(a). Such maps are created by the industrial users of the robot fleet according to their needs.

As there are basically no constraints on location, orientation and shape of traffic zones overlaps or even inclusions of traffic zones are possible. Moreover, some types of zones such as right-hand traffic are combinations of individual zones. Therefore, in a pre-processing step all intersecting areas are replaced by areas representing their partition where the new areas collect all constraints of the involved areas. For instance the two intersection one-way areas in the example are replaced by three areas where one represents the intersecting area and the others the initial areas with the intersection cut out.

Beside the traffic zones the map also contains space which is not occluded by a traffic zone. This free space can be traversed by the robot without any restrictions. The free space is not necessarily a convex polygon. In order to represent the environment (traffic zones and free space) using only convex polygons which is important for the next step the environment is triangulated into sub-areas.

In order to enable the use of search-based path planning the sub-areas are converted into a roadmap. The roadmap consists of nodes on the center of the edges of the sub-areas. Thus edges in the roadmap represent the traversal of a sub-area. Using this representation the robot needs to reserve a sub-area in order to use such an edge in the path. The edges know to which original traffic zone they belong. The sub-areas and the resulting roadmap are depicted in Figure 3(b).

5 Planning

In order to solve the route planning for an individual robot we propose a hierarchical approach. On the top we use the roadmap to find a path from the node representing the sub-area containing the start to the node representing the sub-area containing the goal. After finding a path within the roadmap the robot uses this path to reserve the areas it traverses. After the successful reservation the robot uses the nodes in the graph as way-points for its navigation. To find a smooth path between these way-points we use a gridmap representing the obstacles and the free areas and standard path planning for robotics (Marder-Eppstein et al. 2010). The planning on the gridmap is restricted to those areas which where reserved through the previous step. After this plan has been created the robot executes the plan using a local path planner which generates a motion plan to consider dynamic obstacles (Marder-Eppstein et al. 2010). Hence the system is able to avoid collisions independently from the previous planning steps. Dynamic obstacles are identified with laser scanners and represented in a local occupancy gridmap. Thus the path planning for a robot consists of a hierarchy of three planning steps allowing fast planning and robust execution of paths.

To perform this top-level planning the start and the goal nodes are marked in the roadmap which was created from information about the environment. A modified A* is used to find a path within the roadmap. The modified A* is depicted in Algorithm 1. The major modification is located at line 19 which estimates the heuristic to the goal as well as calculates a potential waiting time for entering an area. Thus during an expansion the algorithm uses as costs the time to move along the edge towards the entry as well as the time the robot needs to wait until it can enter the zone.

The calculation of the heuristic (line 25) uses the predecessor node n and the border node n' to an area. The calculation consists of calculating the shortest path through the area in the direction of the goal. This is done by expanding a copy of the roadmap towards the goal until a node is found that represents the transition to another area. The estimated time to traverse this path is later used together with the shortest line distance from the leaving node to the goal as heuristic $h(n')$. The traversal time is used to find a valid slot (given by the time the entry node is reached and the time for traversal) within the existing area reservations which allow the robot to traverse the area.

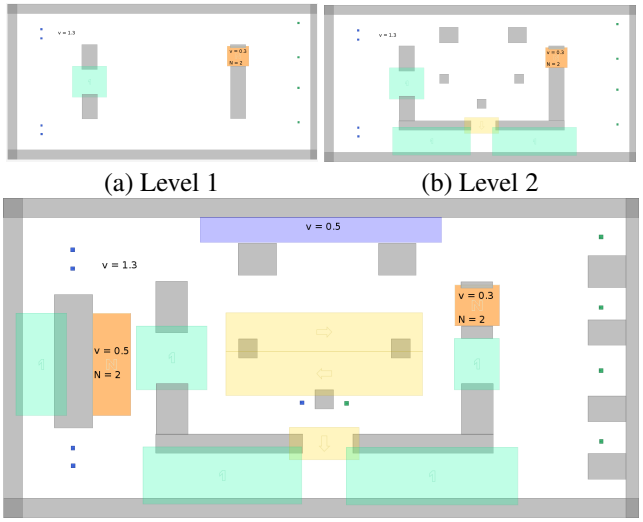
Algorithm 1: A* extensions (Adapted from (Nash et al. 2007))

Data: $G \ldots$ directed graph
Data: $n_{start} \in G \ldots$ start node of the graph
Data: $n_{goal} \in G \ldots$ goal node of the graph
begin
  $openList \longleftarrow \{\}$
  $closedList \longleftarrow \{\}$
  $g(n_{start}) \longleftarrow 0$
  $f(n_{start}) = g(n_{start}) + h_{SLT}(n_{start})$
  $openList.insert(n_{start}, f(n_{start}))$
  while $openList \neq \{\}$ **do**
    $n \longleftarrow openList.pop()$
    if $n = n_{goal}$ **then**
      return *"found path"*
    end
    $closedList \longleftarrow closedList \cup \{n\}$
    foreach $n' \in successor(n)$ **do**
      if $n' \notin closedList$ **then**
        if $n' \notin openList$ **then**
          $g(n') \longleftarrow \infty$
          $parent(n') \longleftarrow NULL$
        end
        $UpdateVertex(n, n')$
      end
    end
  end
  return *"no path found"*
end

Function $UpdateVertex\ (n, n')$
if $g(n) + w_e(n, n') < g(n')$ **then**
  if $n' \in openList$ **then**
    $openList.remove(n')$
  end
  `/* calculate heuristic and waiting time */`
  $g(n') \longleftarrow g(n) + w_e(n, n')$
  $h(n'), t_w \longleftarrow CalculateHeuristic(n')$
  $g(n') \longleftarrow g(n') + t_w$
  $f(n') = g(n') + h(n')$
  $parent(n') \longleftarrow n$
  $openList.insert(n', f(n'))$
end

If such a slot exists the robot is immediately allowed to enter the area once it arrives at its edge. Hence the waiting time $t_w = 0$. Otherwise the algorithm takes the arrival and traversing time to search for a slot that is consistent with all area constraints by simply iterating through all the conflicting intervals and looking for the earliest starting time for such a slot. The waiting time is the difference of the start of the slot and the arrival time.

The result of the top-level planning is a path consisting of way-points and the areas which are traversed by this path. Two example paths are depicted in Figure 4. The figure shows that after the top-level planning the robot could start to move along the edges of the roadmap to reach its goal. It also shows the benefit of the extended heuristics. In left example the lower single-robot zone is unreserved. Thus the planner finds its way directly trough the zone. In the right example the single robot-zone is already reserved by another robot for some time. Because the estimation which is now increased by the waiting time other nodes are expanded as well and the algorithm finds a quicker detour to the goal.



(a) Level 1 (b) Level 2

(c) Level 3

Figure 5: Evaluation scenarios with increasing complexity. The variable v describes the maximal allowed velocity in the corresponding region. The blue and green spots in the free space denotes goal stations. The maps comprises: one ways (yellow); n-robot zone where maximal two robots are allowed (orange); single Robot zone (green); simple traffic areas (blue); forbidden areas (gray).

After finding the path through the areas the robot needs to navigate in the geometric world between the way-points defining this path. To plan between the given way-points and to execute this plan we use the well-known approach presented in (Marder-Eppstein et al. 2010). As it implements a global planner for the way-point navigation as well as a local planner which allows to consider dynamic obstacles.

For a detailed description of the planning hierarchy we refer the interested reader to (Imlauer 2016).

6 Evaluation

The proposed hierarchical navigation system has been evaluated in a simulation suite representing the industrial use case. The evaluation considers two different aspects of the planning system. The first evaluation investigates the computation time of the developed planner for an individual robot. The second evaluation is intended to determine performance gains in the overall system with multiple robots. In Figure 5 we denote three scenarios with increasing complexity. In this evaluation it is assumed that the number of zone with traffic regulating constraints and their arrangement increase the complexity of an environment.

	t_{median}	t_{avg}
Level 1	8.90	9.38
Level 2	9.78	11.83
Level 3	9.24	14.92

Table 1: Planner evaluation on three different evaluation scenarios. The runtimes are given in ms.

Planner Evaluation

The planner evaluation is subjected to the performance criteria defined by the computation time needed for plan generation. With this evaluation it is intended to show that the introduced heuristic is suitable for the graph search procedure used by our algorithm. The evaluation setup consists of 1000 randomly generated pairs of start and goal positions. We sampled the positions in the most complex scenario (*Level 3*). Since all three complexity levels build on each other (see Figure 5(a-c)) we can guarantee to sample valid start and goal positions in every scenario. Furthermore we provide equal problems for every evaluation scenario which allows a fair comparison.

In Table 1 the evaluation results of the previous evaluation setup are illustrated. This table shows the median and mean of the computation time. By examining these results one can see that the planner is able to find plans in reasonable time. Another observation shows that the assumed increasing complexity of the evaluation scenarios does not affect the performance dramatically. The medians of the individual evaluations are approximately equal. Since good average performance on growing graphs is the nature of good heuristics in heuristic search the observation is intended. As a result we can claim that the chosen heuristic performs well on our problem.

However these observations raises the need of harder problems in order to quantify the worst case computation of the planner. Considering harder problems difficulties for the planning instance have to be identified. We assume that paths through zones that can be reserved and additional traffic region reservations will challenge the planner most. We can force the paths to traverse such regions more likely by increasing the Euclidean distance between starts and goals. Region reservations are modeled with randomly generated intervals represented by duration and start time. We define two reservation models: (1) fluctuating reservations, (2) long reservations. The first model creates reservations that do not reserve regions continuously. Hence, the planner has to find a path and schedule corresponding the interval set to find a valid plan. This includes the computation of waiting times. The second generates reservations which tend to start immediately and last "long" which likely prohibits a robot to find a valid time slot for the region quickly.

In Table 2 the median and average value of the computation times are visualized for four experiments in the scenario of *Level 3*. This table compares the results of the previous evaluation with the no reservations and the two new reservation models. As one can see the performance of the planner applied on the new problems is actually worse than for the primarily defined problem. Already the enlarged Euclidean distance increases the computation time due to longer paths. The evaluation trials considering reservations between start and goal are even worse. Furthermore it seems to be not relevant for the worst case computation whether the planner has to deal with (1) or (2). These observations indicate that we can generate the worst case for our planning algorithm by forcing the planner to investigate any region reservations and to detour reserved traffic regions.

Level 3	Median	Average
Prev. Evaluation	9.24	14.9
No Reservation	16.8	23.4
Fluctuating Reservation	21.21	29.2
Long Reservation	20.5	29.6

Table 2: Evaluation of a worst case scenario for the new planning instance. The runtimes are given in ms.

Level	Robots	Original System		New System	
		Median	Avg.	Median	Avg.
Level 1	2	39.9	46.6	40.55	43.6
	4	50.05	51.3	48.5	49.1
	10	53.2	56.5	53.3	55.6
Level 2	2	50.2	59.8	50.8	54.8
	4	55	61.1	56.6	60.5
	10	62.5	67.8	55.6	62.7
Level 3	2	60.8	63.5	56.85	57.8
	4	68.6	69.8	58.6	61.7
	10	67.3	73.5	61.9	69.5

Table 3: System evaluation with random goal assignment. Median and average goal execution time show a performance increase of the new system. The execution time is given in seconds.

System Evaluation

The system evaluation aims on identifying the performance increase of a multi-robot system with the new hierarchical planner. Therefore the new system proposed in this paper is compared with the original system used in the industrial use case as a baseline. In comparison to the previous evaluation we focus here on the execution time t_e of transport tasks of individual robots. This evaluation method considers fleets of different sizes (2, 4, 10 robots) performing 50 individual navigation tasks per robot. In a first method of generating navigation tasks randomly sampled start and goal positions from the scenarios are used. A navigation task is given through a combination of two sampled positions. We perform evaluations for all scenarios and the three different sizes of robot fleets with the *Original System* and the *New System*. In Table 3 the results of this evaluation are listed.

The table generally states a continuous increase of the execution times with increasing complexity of the scenarios. Furthermore the execution times rise with the number of robots in a fleet. The former observation can be motivated by the scenario topology. The more obstacles a scenario consists of the more likely longer paths which have

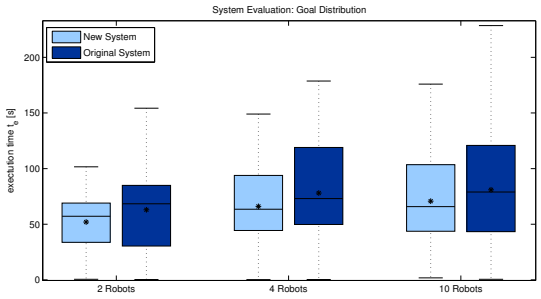


Figure 6: Visualization of a more realistic system evaluation with different number of robots. Comparison of the execution time (t_e) distributions of the New System and the Original System. The asterisk (*) denotes the average value (execution time per goal).

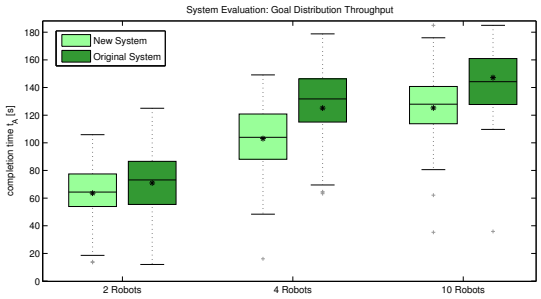


Figure 7: Visualization of the completion time t_A for the evaluation trials of the all robots in fleet. The completion time represents the execution time of the last finishing robot which is a good indicator for the throughput.

to detour obstacles are. The latter is basically due to the fact that in multi-robot scenario the probability of colliding paths increase with the number of robots in a fleet. However the amount of time loss due to the path collision is obviously depending on the evaluation setup. The data of the *Original System* show in general higher execution times than the *New System*. Hence this table states that the *New System* is actually increasing the performance, but the data shows that the gain of performance is moderate.

For this reason we tried to generate a second setup which favors the new hierarchical planner and is moreover closer to a realistic scenario. Therefore we define a list of fixed goal stations in the scenario. A navigation task is now given by a combination of two stations out of a predefined set of stations. These goals are randomly assigned to a robot. The setup consists of 50 random goals drawn from this list. We perform this evaluation only on scenario *Level 3* but again with different fleet sizes (2, 4, 10 robots). The results of this evaluation are depicted in Figure 6.

This figure shows again the execution time per goal and the distribution of the measured execution times via a box plot. In this plot the performance gain of the *New System* is even higher. This is of course the desired effect, since this more realistic scenario provokes paths through allocatable regions. These paths generate the need of reservations and a high-level plan which considers paths of other robots too.

Furthermore we investigated also the throughput of the system in this more realistic evaluation scenario. We define the throughput as the completion time t_c of the last robot of all robots executing a set of navigation tasks. The data distributions in Figure 7 show that even the throughput is significantly increased by the *New System*.

7 Conclusion and Future Work

Fleets of transport robots in industrial settings can gain performance in relation to safety and throughput by using zones with traffic constraints in the environment. In this paper we presented an hierarchical navigation system for a fleet of robots that is able to consider such zones in planning. A key contribution of this paper is a powerful graph-based representation of the enriched environment that includes information about the zones and can be automatically generated from a given map using triangulation and manipulation of polygons. Another contribution is the top-level planning system based on the A* algorithm that uses the above representation and is able to integrate information about the temporal availability of zones by using an intelligent heuristic. The proposed planning system has been implemented on top of existing industrial transport robots. In an simulated evaluation the system showed to be scalable and to provide plans in reasonable time. Moreover, in a multi-robot setting the proposed planner with a central resource management outperformed the existing system in terms of throughput. In future work we like to extend the system in the direction of an improved triangulation of the environment allowing smoother path and a more intelligent zone allocation. Currently the reservations are done on a first come first serve basis ignoring completely the spatial and temporal constraints of the transport orders.

References

Alami, R.; Robert, F.; Ingrand, F.; and Suzuki, S. 1995. Multi-robot cooperation through incremental plan-merging. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 3, 2573–2579. IEEE.

Choset, H. M. 2005. *Principles of robot motion: theory, algorithms, and implementation*. MIT press.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

Imlauer, S. 2016. A hierarchical navigation system for groups of autonomous logistics robots in industrial environments. Master's thesis, Faculty of Computer Science and Biomedical Engineering, Graz University of Technology.

Kleiner, A.; Nebel, B.; et al. 2014. Behavior-based multi-robot collision avoidance. In *IEEE International Conference on Robotics and Automation (ICRA), 2014*, 1668–1673. IEEE.

Kleiner, A.; Sun, D.; and Meyer-Delius, D. 2011. Armo: Adaptive road map optimization for large robot teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011*, 3276–3282.

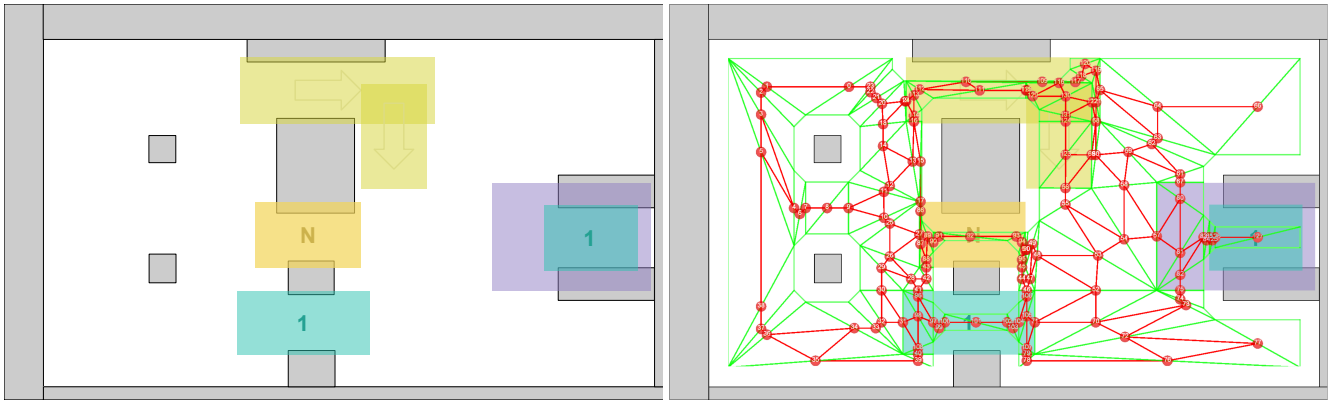
Marder-Eppstein, E.; Berger, E.; Foote, T.; Gerkey, B.; and Konolige, K. 2010. The office marathon. *IEEE International Conference on Robotics and Automation (ICRA)*.

Nash, A.; Daniel, K.; Koenig, S.; and Felner, A. 2007. Theta*: Any-Angle Path Planning on Grids. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22(2), 1177. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Ryan, M. 2010. Constraint-based multi-robot path planning. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 922–928. IEEE.

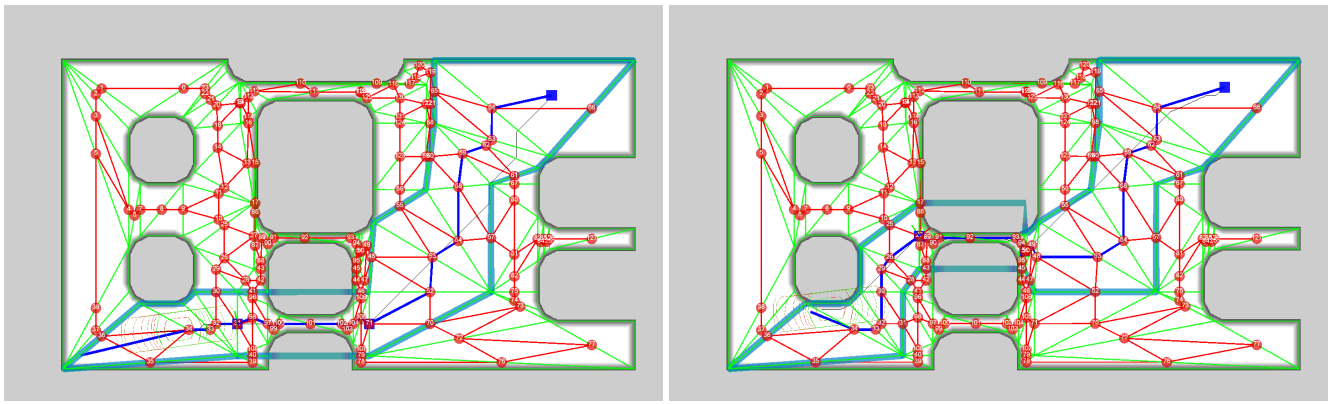
Siciliano, B., and Khatib, O. 2008. *Handbook of robotics*. Springer Science & Business Media.

Wang, J., and Premvuti, S. 1995. Distributed traffic regulation and control for multiple autonomous mobile robots operating in discrete space. In *IEEE International Conference on Robotics and Automation, 1995. Proceedings., 1995*, volume 2, 1619–1624. IEEE.



(a) Map of an example environment with traffic zones. (b) Graph of the roadmap overlaid with the initial map.

Figure 3: Illustration of the generated roadmap based on the triangulation of the environment. In green the triangulation of the environment is shown Please note the safety margin to obstacles (gray). The graph shown in red representing the resulting roadmap.



(a) Planner result trough an empty single robot zone. (b) Planner result considering a blocked single robot zone.

Figure 4: Resulting high-level plan for a start-goal combination in example environment. In (a) the high-level plan leads through the empty single robot region. In (b) the high-level plan bypasses a blocked single robot region, since waiting in front of the region would take longer than detouring the blocked region. Path connecting the waypoints (blue); corresponding local planning window (cyan).

Efficient SAT Approach to Multi-Agent Path Finding under the Sum of Costs Objective

Pavel Surynek
Charles University Prague
Malostranské náměstí 25
11800, Praha, Czech Republic
pavel.surynek@mff.cuni.cz

Ariel Felner and **Roni Stern**
Ben Gurion University
Beer-Sheva, Israel 84105
felner,sternron@bgu.ac.il

Eli Boyarski
Bar-Ilan University
Ramat-Gan, Israel
eli.boyarski@gmail.com

Abstract

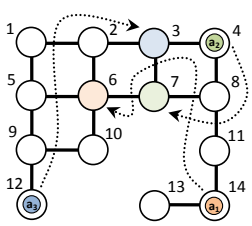
In the *multi-agent path finding* (MAPF) the task is to find non-conflicting paths for multiple agents. In this paper we present the first SAT-solver for the *sum-of-costs* variant of MAPF which was previously only solved by search-based methods. Using both a lower bound on the sum-of-costs and an upper bound on the makespan, we are able to have a reasonable number of variables in our SAT encoding. We then further improve the encoding by borrowing ideas from ICTS, a search-based solver. Experimental evaluation on several domains shown that there are many scenarios where the new SAT-based method outperforms the best variants of previous sum-of-costs search solvers - the ICTS and ICBS algorithms.

1 Introduction and Background

The *multi-agent path finding* (MAPF) problem consists a graph, $G = (V, E)$ and a set $A = \{a_1, a_2, \ldots a_m\}$ of m agents. Time is discretized into time steps. The arrangement of agents at time-step t is denoted as α_t. Each agent a_i has a start position $\alpha_0(a_i) \in V$ and a goal position $\alpha_+(a_i) \in V$. At each time step an agent can either *move* to an adjacent empty location[1] or *wait* in its current location. The task is to find a sequence of move/wait actions for each agent a_i, moving it from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ such that agents do not *conflict*, i.e., do not occupy the same location at the same time. Formally, an MAPF instance is a tuple $\Sigma = (G = (V, E), A, \alpha_0, \alpha_+)$. A *solution* for Σ is a sequence of arrangements $\mathcal{S}(\Sigma) = [\alpha_0, \alpha_1, ..., \alpha_\mu]$ such that $\alpha_\mu = \alpha_+$ where α_{t+1} results from valid movements from α_t for $t = 1, 2, ..., \mu - 1$. An example of MAPF and its solution are shown in Figure 1.

MAPF has practical applications in video games, traffic control, robotics etc. (see Sharon et al. (2015) for a survey). The scope of this paper is limited to the setting of *fully cooperative* agents that are centrally controlled. MAPF is usually solved aiming to minimize one of the two commonly-used global cumulative cost functions:

[1]Some variants of MAPF relax the empty location requirement by allowing a chain of neighboring agents to move, given that the head of the chain enters an empty locations. Most MAPF algorithms are robust (or at least easily modified) across these variants.



(Σ)	α_0	α_1	α_2	α_3	α_4	$\alpha_5 = \alpha_+$
a_1	14	11	8	7	6	6
a_2	4	4	4	4	8	7
a_3	12	9	5	1	2	3

Figure 1: Example of MAPF for agents a_1, a_2, and a_3 over a 4-connected grid (left) and its optimal solution (right)

(1) sum-of-costs (denoted ξ) is the summation, over all agents, of the number of time steps required to reach the goal location Dresner and Stone (2008); Standley (2010); Sharon et al. (2013, 2015). Formally, $\xi = \sum_{i=1}^{m} \xi(a_i)$, where $\xi(a_i)$ is an *individual path cost* of agent a_i.
(2) makespan: (denoted μ) is the total time until the last agent reaches its destination (i.e., the maximum of the individual costs) Surynek (2010, 2014a, 2015).

It is important to note that in any solution $\mathcal{S}(\Sigma)$ it holds that $\mu \leq \xi \leq m \cdot \mu$ Thus the optimal *makespan* is usually smaller than the optimal *sum-of-costs*.

Finding optimal solutions for both variants is NP-Hard Yu and LaValle (2013b); Surynek (2010). Therefore, many sub-optimal solvers were developed and are usually used when m is large Ryan (2010); Cohen, Uras, and Koenig (2015); Silver (2005); Röger and Helmert (2012); Khorshid, Holte, and Sturtevant (2011); Wang and Botea (2011)

1.1 Optimal MAPF Solvers

The focus of this paper is on optimal solvers which are divided into two main classes:
(1) Reduction-based solvers. Many recent optimal solvers reduce MAPF to known problems such as CSP Ryan (2010), SAT Surynek (2012), Inductive Logic Programming Yu and LaValle (2013a) and Answer Set Programming Erdem et al. (2013). These papers mostly prove a polynomial-time reduction from MAPF to these problems. These reductions are usually designed for the *makespan* variant of MAPF; they are not applicable for the sum-of-costsvariant.
(2) Search-based solvers. By contrast, many recent opti-

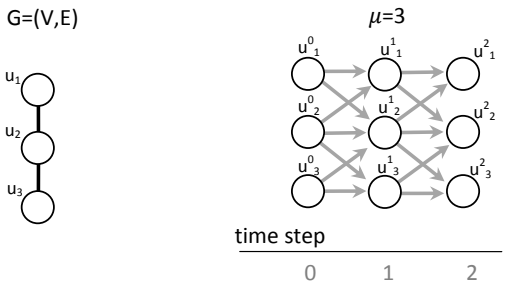


Figure 2: An example of time expansion graph.

mal MAPF solvers are search-based. Some are variants of the A* algorithm on a global *search space* – all different ways to place m agents into V vertices, one agent per vertex Standley (2010); Wagner and Choset (2015). Other employ novel search trees Sharon et al. (2013, 2015); Boyarski et al. (2015). These search-based solvers are usually designed for the *sum-of-costs* MAPF variant.

A major weaknesses is that connection/comparison between different algorithms was usually done only within a given class of algorithms and cost variant but not across these two classes.

1.2 Contributions

This paper aims to start and close the gap. Most of the search-based algorithms can be easily modified to the makespan variant by modifying the cost function and the way the state-space is represented. Some initial directions are given by Sharon et al. (2015). By contrast, the reduction-based algorithms are not trivially modified to the sum-of-costs variant and sometimes a completely new reduction is needed.

In this paper we develop the first SAT-based solvers for the sum-of-costs variant which is based on adding *cardinality constraints* Bailleux and Boufkhad (2003); Silva and Lynce (2007) for bounding the sum-of-costs. We show how to use known lower bounds on the sum-of-costs to reduce the number of variables that encode these cardinality constraints so as to be practicle for current SAT solvers. We then present an *enhanced SAT-solver* which adapts ideas from the ICTS algorithm Sharon et al. (2013) and uses *multi-value decision diagrams* (MDDs) Srinivasan et al. (1990) to further reduce the encoding. Experimental results show that our SAT solver outperforms the best existing search-based solvers for the sum-of-costs variant on many scenarios.

2 SAT Encoding for Optimal Makespan

SAT solvers encompass boolean variables and answer binary questions. The challenge is to apply SAT for MAPF where there is a cumulative cost function. This challenge is stronger for the sum-of-costs variant where each agent has its own cost. We first describe existing SAT encodings for makespan. Then, we present our SAT encoding for sum-of-costs.

A *time expansion graph* (denoted TEG) is a basic concept used in SAT solvers for makespan Surynek (2014a). We use it too in the sum-of-costs variant below. A TEG is a directed acyclic graph (DAG). First, the set of vertices of the underlaying graph G are duplicated for all time-steps from 0 up to the given bound μ. Then, possible actions (move along edges or wait) are represented as directed edges between successive time steps. Figure 2 shows a graph and its TEG for time steps 0, 1 and 2 (vertical layouts). It is important to note that in this example (1) horizonal edges in TEG correspond to *wait* actions. (2) diagonal moves in TEG correspond to real moves. Formally a TEG is defined as follows:

Definition 1. *Time expansion graph of depth μ is a digraph (V, E) where $V = \{u_j^t | t = 0, 1, ..., \mu \wedge u_j \in V\}$ and $E \subseteq \{(u_j^t, u_k^{t+1}) | t = 0, 1, ..., \mu - 1 \wedge (\{u_j, u_k\} \in E \vee j = k)\}$.*

The encoding for MAPF introduces propositional variables and constraints for a single time-step t in order to represent any possible arrangement of agents at time t. Given a desired makespan μ, the formula represents the question of whether there is a solution in the TEG of μ time steps. The search for optimal makespan is done by iteratively incrementing μ (=$0, 1, 2...$) until a satisfiable formula is obtained. This ensures optimality in case of a solvable MAPF instance. More information on SAT encoding for the makespan variant can be found, e.g. in Surynek (2014a,b,c)

3 Basic-SAT for Optimal Sum-of-costs

The general scheme described above for finding optimal makespan is to convert the optimization problem (finding minimal makespan) to a sequence of decision problems (is there a solution of a given makespan μ). We apply the same scheme for finding optimal sum-of-costs, converting it to a sequence of decision problems – is there a solution of a given sum-of-costs ξ. However, encoding this decision problem is more challenging than the makespan case, because one needs to both bound the sum-of-costs, but also to predict how many time expansions are needed. We address this challenge by using two key techniques descried next: **(1)** Cardinality constraint for bounding ξ and **(2)** Bounding the Makespan.

3.1 Cardinality Constraint for Bounding ξ

The SAT literature offers a technique for encoding a *cardinality constraint* Bailleux and Boufkhad (2003); Silva and Lynce (2007), which allows calculating and bounding a numeric cost within the formula. Formally, for a bound $\lambda \in \mathbb{N}$ and a set of propositional variables $X = \{x_1, x_2, ..., x_k\}$ the *cardinality constraint* $\leq_\lambda \{x_1, x_2, ..., x_k\}$ is satisfied iff the number of variables from the set X that are set to `TRUE` is $\leq \lambda$.

In our SAT encoding, we bound the sum-of-costs by mapping every agent's action to a propositional variable, and then encoding a cardinality constraint on these variables. Thus, one can use the general structure of the makespan SAT encoding (which iterates over possible makespans), and add such a cardinality constraint on top. Next we address the challenge of how to connect these two factors together.

3.2 Bounding the Makespan for the Sum of Costs

Next, we compute how many time expansions (μ) are needed to guarantee that if a solution with sum-of-costs ξ exists then

Algorithm 1: SAT consult

```
1  MAPF-SAT(MAPF Σ = (G = (V, E), A, α0, α+))
2      μ0 = max_{ai∈A} ξ0(ai) ;Δ ← 0
3      while Solution not found do
4          μ ← μ0 + Δ;
5          for each agent ai do
6              build TEGi(μ);
7          end
8          Solution=Consult-SAT-SOLVER(Σ, μ,Δ);
9          if Solution not found then
10             Δ++;
11         end
12     end
13     return (Solution);
14 end
```

it will be found. In other words, in our encoding, the values we give to ξ and μ must fulfill the following requirement:

R1: *all possible solutions with sum-of-costs ξ must be possible for a makespan of at most μ.*

To find a μ value that meets R1, we require the following definitions. Let $\xi_0(a_i)$ be the shortest individual path for agent a_i, and let $\xi_0 = \sum_{a_i \in A} \xi_0(a_i)$. ξ_0 was called the *sum of individual costs* (SIC) Sharon et al. (2013). ξ_0 is an admissible heuristic for optimal sum-of-costs search algorithms, since ξ_0 is a lower bound on the minimal sum-of-costs. ξ_0 is calculated by relaxing the problem by omitting the other agents. Similarly, we define $\mu_0 = \max_{a_i \in A} \xi_0(a_i)$. μ_0 is length of the *longest* of the shortest individual paths and is thus a lower bound on the minimal makespan. Finally, let Δ be the extra cost over SIC (as done in Sharon et al. (2013)). That is, let $\Delta = \xi - \xi_0$.

Proposition 1. *For makespan μ of any solution with sum-of-costs ξ, R1 holds for $\mu \leq \mu_0 + \Delta$.*

Proof outline: The worst-case scenario, in terms of makespan, is that all the Δ extra moves belong to a single agent. Given this scenario, in the worst case, Δ is assigned to the agent with the largest shortest-path. Thus, the resulting path of that agent would be $\mu_0 + \Delta$, as required. □

Using Proposition 1, we can safely encode the decision problem of whether there is a solution with sum-of-costs ξ by using $\mu = \mu_0 + \Delta$ time expansions, knowing that if a solution of cost ξ exists then it will be found within $\mu = \mu_0 + \Delta$ time expansions. Algorithm 1 summarizes our optimal sum-of-costs algorithm. In every iteration, μ is set to $\mu_0 + \Delta$ (Line 4) and the relevant TEGs (described below) for the various agents are built. Next a decision problem asking whether there is a solution with sum-of-costs ξ and makespan μ is queried (Line 8). The first iteration starts with $\Delta = 0$. If such a solution exists, it is returned. Otherwise ξ is incremented by one, Δ and consequently μ are modified accordingly and another iteration of SAT consulting is activated.

This algorithm clearly terminates for solvable MAPF instances as we start seeking a solution of $\xi = \xi_0$ ($\Delta = 0$) and increment ξ (and Δ) to all possible values. The unsolvability of an MAPF instance can be checked separately by a polynomial-time complete sub-optimal algorithm such as PUSH-AND-ROTATE de Wilde, ter Mors, and Witteveen (2014).

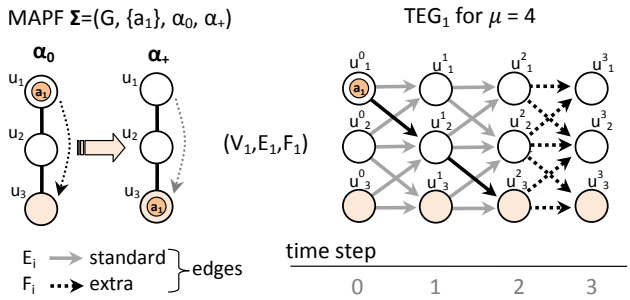


Figure 3: A TEG for an agent that need to go from u_1 to u_3.

3.3 Efficient Use of the Cardinality Constraint

The complexity of encoding a cardinality constraint depends linearly in the number of constrained variables Silva and Lynce (2007); Sinz (2005). Since each agent a_i must move at least $\xi_0(a_i)$, we can reduce the number of variables counted by the cardinality constraint by only counting the variables corresponding to extra movements over the first $\xi_0(a_i)$ movement a_i makes. We implement this by introducing a TEG for a given agent a_i (labeled TEG_i).

TEG_i differs from TEG (Definition 1) in that it distinguishes between two types of edges: E_i and F_i. E_i are (directed) edges whose destination is at time step $\leq \xi_0(a_i)$. These are called *standard edges*. F_i denoted by *extra edges* are directed edges whose destination is at time step $> \xi_0(a_i)$. Figure 3 shows an underlying graph for agent a_1 (left) and the corresponding TEG_1. Note that the optimal solution of cost 2 is denoted by the diagonal path of the TEG. Edges that belong to F_i are those that their destination is time step 3 (dotted lines). The key in this definition is that the cardinality constraint would only be applied to the extra edges, that is, we will only bound the number of extra edges (they sum up to Δ) making it more efficient.

3.4 Detailed Description of the SAT Encoding

Agent a_i must go from its initial position to its goal within TEG_i. This simulates its location in time in the underlying graph G. That is, the task is to find a path from $\alpha_0^0(a_i)$ to $\alpha_+^\mu(a_i)$ in TEG_i. The search for such a path will be encoded within the Boolean formula. Additional constraints will be added to capture all movement constraints such as *collision avoidance* etc. And, of course, we will encode the cardinality constraint that the number of extra edges must be exactly Δ.

We want to ask whether a sum-of-costs solution of ξ exist. For this we build TEG_i for each agent $a_i \in A$ of depth $\mu_0 + \Delta$. We use V_i to denote the set of vertices in TEG_i that agent a_i might occupy during the time steps. Next we introduce the Boolean encoding (denoted `BASIC-SAT`)which has the following Boolean variables:

1:) $\mathcal{X}_j^t(a_i)$ for every $t \in \{0, 1, ..., \mu\}$ and $u_j^t \in V_i$ – Boolean variable of whether agent a_i is in vertex v_j at time step t.

2:) $\mathcal{E}^t_{j,k}(a_i)$ for every $t \in \{0, 1, ..., \mu - 1\}$ and $(u^t_j, u^{t+1}_k) \in (E_i \cup_i)$ — Boolean variables that model transition of agent a_i from vertex v_j to vertex v_k through any edge (standard or extra) between time steps t and $t + 1$ respectively.

3:) $\mathcal{C}^t(a_i)$ for every $t \in \{0, 1, ..., \mu-1\}$ such that there exist $u^t_j \in V_i$ and $u^{t+1}_k \in V_i$ with $(u^t_j, u^{t+1}_k) \in F_i$ — Boolean variables that model cost of movements along **extra edges** (from F_i) between time steps t and $t + 1$.

We now introduce constraints on these variables to restrict illegal values as defined by our variant of MAPF. Other variants may use a slightly different encoding but the principle is the same. Let $T_\mu = \{0, 1, ..., \mu - 1\}$. Several groups of constraints are introduced for each agent $a_i \in A$ as follows:

C1: If an agent appears in a vertex at a given time step, then it must follow through exactly one adjacent edge into the next time step. This is encoded by the following two constraints, which are posted for every $t \in T_\mu$ and $u^t_j \in V_i$

$$\mathcal{X}^t_j(a_i) \Rightarrow \bigvee_{(u^t_j, u^{t+1}_k) \in E_i \cup F_i} \mathcal{E}^t_{j,k}(a_i) \quad (1)$$

$$\bigwedge_{(u^t_j, u^{t+1}_k), (u^t_j, u^{t+1}_l) \in E_i \cup F_i \wedge k < l} \neg\mathcal{E}^t_{j,k}(a_i) \vee \neg\mathcal{E}^t_{j,l}(a_i) \quad (2)$$

C2: Whenever an agent occupies an edge it must also enter it before and leave it at the next time-step. This is ensured by the following constraint introduced for every $t \in T_\mu$ and $(u^t_j, u^{t+1}_k) \in E_i \cup F_i$:

$$\mathcal{E}^t_{j,k}(a_i) \Rightarrow \mathcal{X}^t_j(a_i) \wedge \mathcal{X}^{t+1}_k(a_i) \quad (3)$$

C3: The target vertex of any movement except wait action must be empty. This is ensured by the following constraint introduced for every $t \in T_\mu$ and $(u^t_j, u^{t+1}_k) \in E_i \cup F_i$ such that $j \neq k$.

$$\mathcal{E}^t_{j,k}(a_i) \Rightarrow \bigwedge_{a_l \in A \wedge a_l \neq a_i \wedge u^t_j \in V_l} \neg\mathcal{X}^t_j(a_l) \quad (4)$$

C4: No two agents can appear in the same vertex at the same time step. That is the following constraint is added for every $t \in T_\mu$ and pair of of agents $a_i, a_l \in A$ such that $i \neq l$:

$$\bigwedge_{u^t_j \in V_i \cap V_l} \neg\mathcal{X}^t_j(a_i) \vee \neg\mathcal{X}^t_j(a_l) \quad (5)$$

C5: Whenever an extra edge is traversed the cost needs to be accumulated. In fact, this is the only cost that we accumulate as discussed above. This is done by the following constraint for every $t \in T_\mu$ and extra edge $(u^t_j, u^{t+1}_k) \in F_i$.

$$\mathcal{E}^t_{j,k}(a_i) \Rightarrow \mathcal{C}^t(a_i) \quad (6)$$

C6: Cardinality constraint. Finally the bound on the total cost needs to be introduced. Reaching the sum-of-costs of ξ corresponds to traversing exactly Δ extra edges from F_i. The following cardinality constrains ensures this:

$$\leq_\Delta \left\{ \begin{array}{c} \mathcal{C}^t(a_i) | i = 1, 2, ..., n \wedge t = 0, 1, ..., \mu - 1 \\ \wedge \{(u^t_j, u^{t+1}_k) \in F_i\} \neq \emptyset \end{array} \right\} \quad (7)$$

Final formula. The resulting Boolean formula that is a conjunction of $C1 \ldots C6$ will be denoted as $\mathcal{F}_{BASIC}(\Sigma, \mu, \Delta)$ and is the one that is consulted by Algorithm 1 (line 4).

The following proposition summarizes the correctness of our encoding.

Proposition 2. *MAPF $\Sigma = (G = (V, E), A, \alpha_0, \alpha_+)$ has a sum-of-costs solution of ξ if and only if $\mathcal{F}_{BASIC}(\Sigma, \mu, \Delta)$ is satisfiable. Moreover, a solution of MAPF Σ with the sum-of-costs of ξ can be extracted from the satisfying valuation of $\mathcal{F}_{BASIC}(\Sigma, \mu, \Delta)$ by reading its $\mathcal{X}^t_j(a_i)$ variables.*

Proof: The direct consequence of the above definitions is that a valid solution of a given MAPF Σ corresponds to non-conflicting paths in the TEGs of the individual agents. These non-conflicting paths further correspond to satisfying the variable assignment of $\mathcal{F}_{BASIC}(\Sigma, \mu, \Delta)$, i.e., that there are Δ extra edges in TEGs of depth $\mu = \mu_0 + \Delta$. □

Proposition 3. *Let D be the maximal degree of any vertex in G and let m be the number of agents. If $m \cdot |E| \geq \Delta$ and $m \geq D$ then the number of clauses in $\mathcal{F}_{BASIC}(\Sigma, \mu, \Delta)$ is $O(\mu \cdot m^2 \cdot |E|)$, and the number of variables is $O(\mu \cdot |E| \cdot m)$.*

Proof: The components of $\mathcal{F}_{BASIC}(\Sigma, \mu, \Delta)$ is described in equations 1– 7. Equation 1 introduces at most $O(m \cdot \mu \cdot |E|)$ clauses. Equation 2 introduces at most $O(m \cdot \mu |E| \cdot D)$ clauses. Equation 3 introduces at most $O(m \cdot \mu \cdot |E|)$ clauses. Equation 4 introduces at most $O(m^2 \cdot \mu \cdot |E|)$ Equation 5 introduces at most $O(m^2 \cdot \mu \cdot |V|)$ clauses. Equation 6 introduces at most $O(m \cdot \mu \cdot |E|)$ clauses. Equation 7 introduces at most $O(m \cdot \mu \cdot (\xi - \xi_0))$ clauses, since a cardinality constraint checking that n variables has a cardinality constraint of m requires $O(n \cdot m)$ clauses Sinz (2005). Summing all the above results in a total of $O(\mu \cdot m \cdot (|E| \cdot (D + m) + (\xi - \xi_0)))$. If we assume that $m > D$ and that $m \cdot |E| > (\xi - \xi_0)$ then the number of clauses is $O(\mu \cdot m^2 \cdot |E|)$. The number of variables is easily computed in a similar way. □

4 Improving Basic SAT by Adding MDDs

A major parameter that affects the speed of solving of Boolean formulae is their size Petke (2015). The size of formulae in the BASIC-SAT encoding is affected mostly by the size of the TEGs (this is embodied in the $|E|$ factor in the encoding size). To obtain a significant speedup we reduce the size of TEG_i for agent a_i in terms of number of vertices while the soundness of encoding is preserved.

Let TEG^μ_i denote TEG_i for μ time expansions. We set $\mu = \mu_0 + \Delta$ in our solution. The data structure we use for reducing TEG^μ_i is a *multi-value Decision Diagram* (MDD). MDDs were already used in the search-based MAPF algorithm ICTS Sharon et al. (2013). In our context, MDD^μ_i is a digraph that represents all possible valid paths from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ of cost μ for agent a_i. MDD^μ_i has a single *source node* at level 0 and a single *sink node* at level μ. Every node at depth t of MDD^μ_i corresponds to a possible location of a_i at time t, that is on a path of cost μ from $\alpha_0(a_i)$ to $\alpha_+(a_i)$. It is easy to see that MDD^μ_i is subgraph of TEG_i. While TEG^μ_i includes all vertices of G at each time step, MDD^μ_i includes only those vertices and edges that represent possible valid paths, and thus vertices not in MDD^μ_i can be ignored.

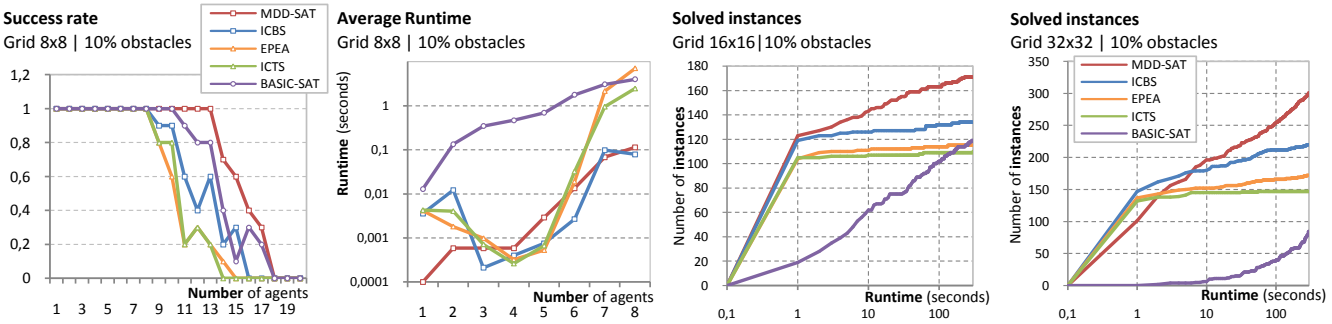


Figure 5: Results on 8×8 grid (left). Number of solved instances in the given runtime on 16×16 and 32×32 grids. (right)

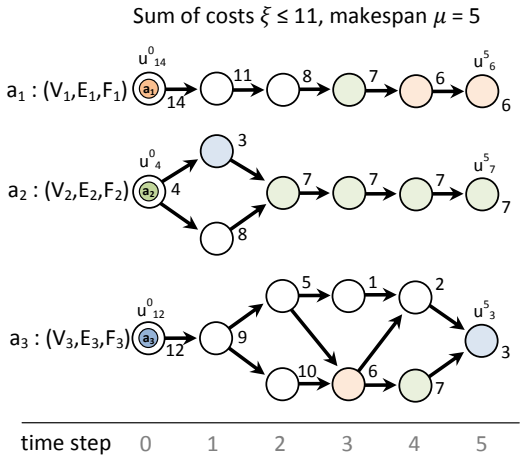


Figure 4: MDDs for agents a_1, a_2, and a_3 for the MAPF from Figure 1 for *sum of individual cost* $\xi \leq 11$

Moreover, the maximum cost that can be consumed by single agent a_i under given sum-of-costs bound ξ is $\xi_0(a_i) + \Delta$ where, as defined above, $\xi_0(a_i)$ is the shortest path connecting $\alpha_0(a_i)$ with $\alpha_+(a_i)$ in G (assuming no other agent exist). Thus, it is sufficient to replace TEG_i^{μ} with $MDD_i^{\xi_0(a_i)+\Delta}$, which is useful since $\xi_0(a_i) + \Delta \leq \mu_0 + \Delta = \mu$.

MDDs for the agents of Figure 1 are shown in Figure 4. Indeed, the size of the MDDs is much smaller than the corresponding TEGs which include all states for all time steps.

The encoding that uses MDD-based time expansion will be called MDD-SAT and the corresponding formulae will be denoted as $\mathcal{F}_{MDD}(\Sigma, \mu, \Delta)$. $\mathcal{F}_{MDD}(\Sigma, \mu, \Delta)$ are similar to BASIC-SAT. The only different is that in BASIC-SAT there is a variable for all vertices and edges of the TEGs while in MDD-SAT, only variables for the vertices and edges of the MDDs are needed. This difference can be significant. Table 1 presents the number of propositional variables and clauses accumulated over all the constructed formulae for a given MAPF instance for BASIC-SAT and for MDD-SAT over 8×8 grid with 10% obstacles. The average values out of 10 random instances per number of agents is shown. Up to two orders of magnitude reduction is shown.

5 Experimental Evaluation

We experimented on 4-connected grids with randomly placed obstacles Silver (2005); Standley (2010) and on *Dragon Age* maps Sturtevant (2012). Both settings are a standard MAPF benchmarks. The initial position of the agents was randomly selected. To ensure solvability the goal positions were selected by performing a long *random walk* from the initial arrangement.

We compared our SAT solvers to several state-of-the-art search-based algorithms: the *increasing cost tree search* - ICTS Sharon et al. (2013), *Enhanced Partial Expansion A** - EPEA* Goldenberg et al. (2014) and *improved conflict-based search* - ICBS Boyarski et al. (2015). For all the search algorithms we used the best known setup of their parameters and enhancements suitable for solving the given instances.

The SAT approaches were implemented in C++ using `Glucose 3.0` Audemard and Simon (2009); Audemard, Lagniez, and Simon (2013); a top performing SAT solver in the *SAT Competition* Järvisalo et al. (2012); Surynek (2014a). The cardinality constraint was encoded using a simple standard circuit based encoding called *sequential counter* Sinz (2005). ICTS and ICBS were implemented in C#, based on their original implementation. All experiments were performed on a Xeon 2Ghz, and on Phenom II 3.6Ghz, both with 12 Gb of memory.

5.1 Square Grid Experiments

We first experimented on 8×8, 16×16, and 32×32 grids with 10% obstacles while varying the number of agents from 1 to 20. Figure 5 presents results over 10 instances where each algorithm was given a time limit of 300 seconds. The leftmost plot shows the *success rate* (=precantage of instances solved within the time limit) as a function of the number of agents. The next plot reports the average runtime for instances that were solved by all algorithms. The right plots visualize the results on 16×16 and 32×32 grids but

Grid 8x8	BASIC-SAT		MDD-SAT	
m	*Variables*	*Clauses*	*Variables*	*Clauses*
1	1 552.8	11 617.6	20.6	27.9
4	14 712.0	127 732.2	276.5	554.0
8	226 391.2	2 099 127.6	18 355.6	68 826.0
16	4 075 187.2	32 108 347.2	2 253 508.2	13 128 646.9

Table 1: The number of variables and clauses

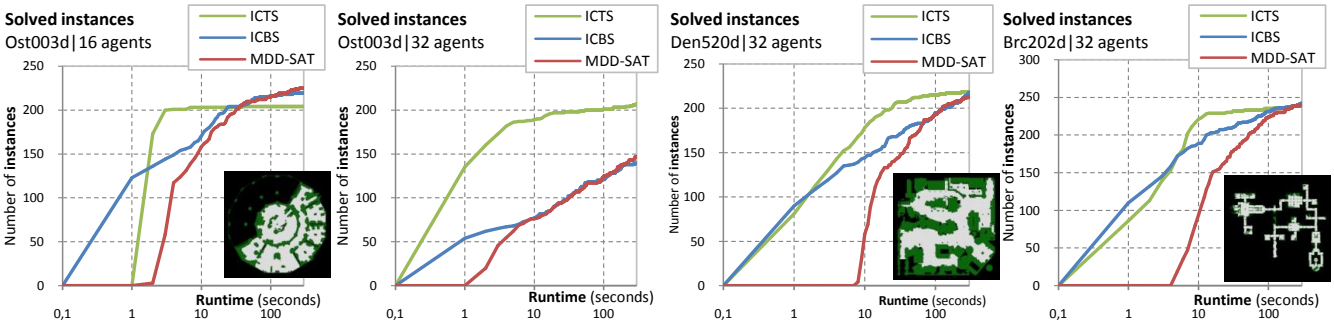


Figure 6: The number of solved instances in the given runtime on Dragon Age maps for 16 and 32 agents.

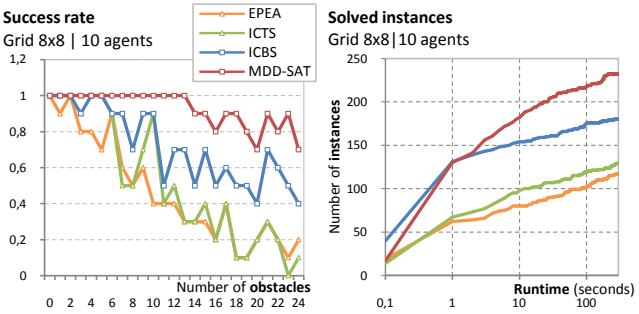


Figure 7: Success rate and runtime on the 8×8 grid with increasing number of obstacles (out of 64 cells).

in a different way. Here, we present the number of instances (out of all 200 instances for all number of agents) that each method solved (y-axis) as a function of the elapsed time (x-axis).

The first clear trend is that MDD-SAT significantly outperforms BASIC-SAT in all aspects. This shows the importance of developing efficient SAT encodings for this problem. In addition, a prominent trend observed in all the plots is that MDD-SAT has higher success rate and solves more instances than all other algorithms. In some cases, however, where the available runtime is very small, MDD-SAT is outperformed by the search-based algorithms.

For the rest of our experiments, we only evaluated the most efficient algorithms, namely, MDD-SAT, ICTS, and ICBS.

Next, we varied the number of obstacles for the 8×8 grid with 10 agents. Results are shown in Figure 7. Clearly MDD-SAT can solve more instances over all settings. MDD-SAT was always faster except for some easy instances where ICBS was slightly faster. Interestingly, increasing the number of obstacles reduces the number of open cells. This is an advantage for the SAT solver as the SAT formula has less variables. By contrast, for the search-based solvers, adding obstacles means that the graphs gets denser and harder to solve.

5.2 Results on the Dragon Age Maps

Next, we experimented on three Dragon-Age maps (*ost003d*, *den520d*, and *brc202d*) commonly used as testbeds. In these maps there is a large number of open cells but the graph is sparse with agents. This gives a clear advantage to the search-based solvers. To obtain instances of various difficulties we varied the distance between start and goal locations. 10 random instances were generated for each distance in the range: $\{8, 16, 24, \ldots, 200\}$. The results are shown in Figure 6 (the number of instances solved as the function of time).

In the Dragon-Age setting there is no universal winner. Each algorithm was the best for some of the instances (especially in case of *ost003d*). When limited time is allowed ICTS or ICBS are better. However, given enough time MDD-SAT catches up and even outperforms the other algorithms. This was evident in all these experiments except for *ost003d* with 32 agents. Concrete runtimes for 10 instances of *ost003d* are given in Table 2. MDD-SAT solves the hardest instance (#1) while other solvers ran out of time. The right part of the table illustrates the cumulative size of the formulae generated during the solving process. Although the map is much larger than the square grids, the size of formulae is comparable to the densely occupied grid (see Figure 1). This is because ξ_0 is a good lower bound of the optimal cost in the sparse maps.

The entire set of experiments show a clear trend. When a small amount of time is given the search-based algorithm may be faster. But, given enough time MDD-SAT is the correct choice, even in the large maps where it has an initial disadvantage. One of the reasons for this is modern SAT solvers have the ability to learn and improve their speed during the process of answering a SAT question. But, this learning needs sufficient time and large search trees to be effective. By contrast, search algorithms do not have this advantage.

6 Summary and Conclusions

We introduced the first state-of-the-art SAT-based solver for the sum-of-costs variant of MAPF. The resulting encoding, called MDD-SAT, was shown to be competitive in comparison with the state-of-the-art search-based solvers over a variety of domains. Nevertheless, as previous authors mentioned Sharon et al. (2015); Boyarski et al. (2015) there is no universal winner and each of the approaches has pros and cons and worsk best in different circumstances. This calls for a deeper study of various classes of MAPF instances and their characteristics.

There are several factors behind the performance of the SAT-based approach: clause learning, constraint propagation, good implementation of the SAT solver. On the other

hand, the SAT solver doesn't understand the structure of the encoded problem which may downgrade the performance. Hence, we consider that implementing techniques such as learning directly into the dedicated MAPF solver may be a future direction.

MAPF	**Ost003d** (seconds) 16 agents, distance=168		
	MDD-SAT	*ICBS*	*ICTS*
1	**101.4**	N/A	N/A
2	12.8	9.7	**2.4**
3	13.2	4.4	**2.4**
4	3.8	**0.6**	1.2
5	13.5	9.6	**3.2**
6	22.7	**10.7**	N/A
7	N/A	N/A	N/A
8	36.9	49.6	**2.5**
9	12.0	2.6	**1.4**
10	N/A	N/A	N/A

m	**MDD-SAT**, 16 agents	
Distance	*Variables*	*Clauses*
8	758.0	1 169.7
64	34 648.7	120 961.1
128	932 440.9	9 128 568.8

m	**MDD-SAT**, 32 agents	
Distance	*Variables*	*Clauses*
8	2 377.6	3 751.3
64	571 915.1	3 672 249.3
128	5 163 157.0	49 201 960.0

Table 2: Runtime for 10 instances (left) and the average size of the MDD-SAT formulae for `ost003d` (right)

References

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI*, 399–404.

Audemard, G.; Lagniez, J.; and Simon, L. 2013. Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *Theory and Applications of Satisfiability Testing - SAT 2013*, 309–317.

Bailleux, O., and Boufkhad, Y. 2003. Efficient CNF encoding of boolean cardinality constraints. In *CP*, 108–122.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. 2015. ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, 740–746.

Cohen, L.; Uras, T.; and Koenig, S. 2015. Feasibility study: Using highways for bounded-suboptimal mapf. In *SOCS*, 2–8.

de Wilde, B.; ter Mors, A.; and Witteveen, C. 2014. Push and rotate: a complete multi-agent pathfinding algorithm. *JAIR* 51:443–492.

Dresner, K., and Stone, P. 2008. A multiagent approach to autonomous intersection management. *JAIR* 31:591–656.

Erdem, E.; Kisa, D. G.; Oztok, U.; and Schueller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R.; and Schaeffer, J. 2014. Enhanced partial expansion A*. *JAIR* 50:141–187.

Järvisalo, M.; Berre, D. L.; Roussel, O.; and Simon, L. 2012. The international SAT solver competitions. *AI Magazine* 33(1).

Khorshid, M. M.; Holte, R. C.; and Sturtevant, N. R. 2011. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Symposium on Combinatorial Search (SOCS)*.

Petke, J. 2015. *Bridging Constraint Satisfaction and Boolean Satisfiability*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer.

Röger, G., and Helmert, M. 2012. Non-optimal multi-agent pathfinding is solved (since 1984). In *SOCS)*.

Ryan, M. 2010. Constraint-based multi-robot path planning. In *ICRA*, 922–928.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence* 195:470–495.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.* 219:40–66.

Silva, J., and Lynce, I. 2007. Towards robust CNF encodings of cardinality constraints. In *CP*, 483–497.

Silver, D. 2005. Cooperative pathfinding. In *AIIDE*, 117–122.

Sinz, C. 2005. Towards an optimal CNF encoding of boolean cardinality constraints. In *CP*, 827–831.

Srinivasan, A.; Ham, T.; Malik, S.; and Brayton, R. 1990. Algorithms for discrete function manipulation. In *(ICCAD*, 92–95.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 173–178.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *Computational Intelligence and AI in Games* 4(2):144–148.

Surynek, P. 2010. An optimization variant of multi-robot path planning is intractable. In *AAAI*.

Surynek, P. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI*. 564–576.

Surynek, P. 2014a. Compact representations of cooperative pathfinding as SAT based on matchings in bipartite graphs. In *ICTAI*, 875–882.

Surynek, P. 2014b. A simple approach to solving cooperative pathfinding as propositional satisfiability works well. In *PRICAI*, 827–833.

Surynek, P. 2014c. Simple direct propositional encoding of cooperative path finding simplified yet more. In *MICAI*, 410–425.

Surynek, P. 2015. Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In *IJCAI*, 1916–1922.

Wagner, G., and Choset, H. 2015. Subdimensional expansion for multirobot path planning. *Artif. Intell.* 219:1–24.

Wang, K., and Botea, A. 2011. MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *JAIR)* 42:55–90.

Yu, J., and LaValle, S. 2013a. Planning optimal paths for multiple robots on graphs. In *ICRA*, 3612–3617.

Yu, J., and LaValle, S. M. 2013b. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*.

Automated Verification of Social Law Robustness in STRIPS

Erez Karpas and **Alexander Shleyfman** and **Moshe Tennenholtz**
Faculty of Industrial Engineering and Management
Technion — Israel Institute of Technology

Abstract

Agents operating in a multi-agent system must consider not just their own actions, but also those of the other agents in the system. Artificial social systems are a well known means for coordinating a set of agents, without requiring centralized planning or online negotiation between agents. Artificial social systems enact a social law which restricts the agents from performing some actions under some circumstances. A good social law prevents the agents from interfering with each other, but does not prevent them from achieving their goals. However, designing good social laws, or even checking whether a proposed social law is good, are hard questions. In this paper, we take a first step towards automating these processes, by formulating criteria for good social laws in a multi-agent planning framework. We then describe an automated technique for verifying if a proposed social law meets these criteria, which is based on a compilation to classical planning.

1 Introduction

The design of an agent which is about to operate in a multi-agent environment is quite different from the design of an agent which performs his activities in isolation from other agents. Typically, a plan that would have allowed an agent to obtain his goals had he operated in isolation might yield unexpected results as a consequence of other agents' activities. Various approaches to multi-agent coordination have been considered in the literature. We could, for instance, subordinate the agents to a central controller. This approach may be useful in various domains but might suffer from well-known limitations, such as bottlenecks at the central site or sensitivity to failure. Another approach is to design rules of encounter, that is, rules which determine the behavior of the agent, and in particular the structure of negotiation, when his activities interfere with those of another agent. Rules of encounter may be quite useful for conflict resolution, but might sometimes be inefficient, requiring repeated negotiations to solve on-line conflicts. In this paper we consider a canonical intermediate approach to coordination, referred to as *artificial social systems* [Tennenholtz, 1991; Shoham and Tennenholtz, 1992a; 1992b; 1995; Moses and Tennenholtz, 1995].

An artificial social system institutes a social law that the agents shall obey. Intuitively, a social law restricts, off-line, the actions legally available to the agents, and thus minimizes the chances of an on-line conflict, and the need to negotiate. Similarly to a code of laws in a human society [Rousseau, 1762], an artificial social law regulates the individual behavior of the agents and benefits the community as a whole. Yet, the agents should still be able to achieve their goals, and restricting their legal actions to a too wide extent might leave them with no possible way to do so.

Consider for instance a domain consisting of roads on which our agents travel. These roads cross one another at junctions where total freedom on the side of the agents makes an accident a likely event. In order to guarantee accident-free traffic we could set a law that allows an agent to enter an intersection only if the crossing road is free. This law certainly prevents accidents, but restricts the agents too much. Although we have guaranteed an accident-free environment, we have also introduced the possibility of a deadlock: when two agents reach the intersection via crossing roads, they might find themselves waiting indefinitely for the crossing road to get free before initiating their move. This example illustrates the fact that we must be careful in designing social laws: Only useful social laws, i.e laws which guarantee that each agent achieves his goals, are to be considered. In the example above, the law could be modified to give one direction the right-of-way, obliging cars coming from the crossing direction to yield.

The artificial social systems approach has become a canonical approach to the design of multi-agent systems [Woolridge, 2001; Shoham and Leyton-Brown, 2008; Horling and Lesser, 2004; d'Inverno and Luck, 2004; Klusch, 1999]. However, while the origins of the artificial social systems approach arise from a knowledge representation and planning perspective, and early work by the founders of that approach had advocated the use of planning paradigms, such as STRIPS-like presentations for multi-agent planning [Tennenholtz and Moses, 1989], the connection between artificial social systems to modern planning techniques has not been crystallized or exploited. The aim of our current line of research is to re-visit the artificial social systems approach in view of progress made in planning. Specifically, the contri-

butions of this paper are threefold: First, we describe a formalism for representing and reasoning over social laws in a multi-agent planning framework. Second, we describe some robustness criteria we believe good social laws should satisfy. Third, we describe an algorithm for verifying if a given social law meets these criteria, which is based on a compilation to classical planning. An empirical evaluation shows this approach scales up very well.

2 Preliminaries

We consider multi-agent planning settings formulated in (a variation of) MA-STRIPS [Brafman and Domshlak, 2008]. Our focus in this paper is on problems which do not require cooperation, but do require coordination, and thus we modify MA-STRIPS to include a goal for each agent, rather than an overall goal. A multi-agent planning setting is defined by a tuple $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$, where: F is a set of facts, $I \subseteq F$ is the initial state, $G_i \subseteq F$ is the goal of agent i, and A_i is the set of actions of agent i. Each action $a \in A_i$ is described by preconditions $pre(a) \subseteq F$, add effects $add(a) \subseteq F$, and delete effects $del(a) \subseteq F$. The result of applying action a in state s is $(s \setminus del(a)) \cup add(a)$.

The projection of Π for agent i is the (single agent) STRIPS [Fikes and Nilsson, 1971] planning problem $\Pi_i = \langle F, A_i, I, G_i \rangle$. A sequence of actions π_i is a solution for Π_i if applying the actions in π_i from state I results in a state s that satisfies the goal, that is, a state such that $G_i \subseteq s$. In the execution model we consider here, each agent attempts to follow its own plan π_i. The plans interact with each other through a scheduler, which determines which agent acts next. We do not make any assumptions about the fairness of the scheduler, and in fact, consider it to be adversarial.

3 Encoding Social Laws

We have described our multi-agent setting, but we have yet to describe how we represent social laws in this setting. To begin with, we will represent social laws as modifications to a MA-STRIPS problem. That is, a social law l takes an input MA-STRIPS problem Π, modifies it, and outputs a new MA-STRIPS problem Π^l. Such a social law can be described by:

1. The facts it adds or removes,
2. The actions it adds or removes,
3. The preconditions, add effects, or delete effects it adds or removes from each existing action,
4. The facts it adds or removes from the initial state
5. The facts it adds or removes from each agent's goal, and
6. The action preconditions which are denoted as *waitfor* preconditions

The first 5 items above are simply syntactic modifications of an MA-STRIPS setting. For example, the social law which says that everyone must drive on the right side of the road can be encoded by removing all actions which drive to the left side of the road.

The *waitfor* precondition annotations, however, require some explanation. Waiting is one of the most basic forms of coordination, and can eliminate some failures. For example, waiting for an intersection to be clear before entering it eliminates the possibility of collision. However, waiting also introduces the possibility of a *deadlock* — if our social law states that to enter the intersection we wait until there are no cars to the right, then a deadlock occurs when there are four cars on the four sides of the intersection, as illustrated in Figure 2(c).

The semantics of executing an action with a *waitfor* precondition in our model is as follows: When an agent invokes an action a with a *waitfor* precondition p in state s, the scheduler will only execute the action if p holds in s. We will denote the *waitfor* preconditions of action a by $pre_w(a)$ and the other preconditions of a by $pre_f(a)$. Then the scheduler can only choose to execute an action a whose *waitfor* preconditions hold in the current state s, that is $pre_w(a) \subseteq s$. If all agents are currently either waiting for some precondition or finished (that is, have already achieved their goal), then the system is in a deadlock.

We conclude this section with a brief discussion of when it does or does not make sense to wait for some precondition of an action. One of the original motivations for social laws comes from robotics, and in the real world, robots can not sense everything. Thus, it only makes sense to wait for something the agent can sense, as otherwise there is no way to implement the action on a robot. This is a subtle point with the assumptions underlying classical planning: assuming the actions are deterministic, do we sense at every state, or only at the initial state? For classical planning, the answer is irrelevant, but when there are multiple agents operating in the world, the answer is very important. *waitfor* precondition annotations answer this question by stating that we sense *before* we start executing an action.

Secondly, it does not make sense to wait for a precondition which the agent can achieve by itself — a private fact in multi-agent planning terms. This would automatically result in an agent entering a deadlock by itself. For example, consider the action $move(A, X, Y)$ which has a precondition $at(A, X)$, which the agent waits for. Unless some other agent can move A to X, and has a good reason to do so, A will be stuck waiting for itself to move to X.

4 Properties of Social Laws

Now that we have formalized the setting in which we consider social laws, we describe what are the criteria that define a *good* social law. We consider two different criteria for social laws, which we call *rational* and *adversarial* robustness. In rational robustness, we assume all agents are rational and want to achieve their goal, and ask whether there is any possible way for them to interfere with each other. In adversarial robustness, we assume all agents except for one specific agent (say agent i) are adversarial, and only want to prevent agent i from achieving its goal, without regard for achieving their own goal later.

These criteria are formally stated in the following definition:

Definition 1. *A social law l for multi agent setting $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ is robust to:*

rational *iff for all agents $i = 1 \dots n$, for all individual solutions π_i for Π_i, for all possible action sequences π resulting from any arbitrary interleaving of $\{\pi_i\}_{i=1}^n$ which respects* waitfor *preconditions, π achieves $G_1 \cup \dots \cup G_n$.*

adversarial against i *iff for all individual solutions π_i for Π_i, for all possible action sequences π resulting from an arbitrary interleaving of π_i which respects* waitfor *preconditions with any valid action sequence of all other agents, π achieves G_i.*

adversarial *iff it is robust to* adversarial against i *for all* $i = 1 \dots n$.

It is easy to see that if a social law is robust to *adversarial*, then it is also robust to *rational*. Conversely, we show that the *verification* problem for adversarial robustness VERIFY-ADVERSARIAL is reducible to the *verification* problem for rational robustness VERIFY-RATIONAL.

Theorem 1.
VERIFY-RATIONAL $\geq_p$ VERIFY-ADVERSARIAL.

Proof. Given a multi-agent setting Π and a social law l, we want to solve VERIFY-ADVERSARIAL(Π^l). From Definition 1, this is equivalent to verifying that Π^l is robust to adversarial against i for all $i = 1 \dots n$. To verify that Π^l is robust to adversarial against a given i, we will construct a 2-agent setting Π' and a social law l, such that Π'^l is robust to rational iff Π^l is robust to adversarial against i.

The facts and the initial state in Π' are the same facts as in Π: F and I, respectively. The first agent in Π' is agent i from Π, and its actions and goal are the same as in Π: A_i and G_i, respectively. The second agent in Π' is a single virtual agent, which controls all agents in Π except for i, that is, its action set is $\bigcup_{j \neq i} A_j$. The goal of this agent is always true ($\top$), that is, it can achieve its goal whenever it wants. The social law l is the same, except for the required modifications introduced by renaming the agents.

To see that Π'^l is robust to rational iff Π^l is robust to adversarial against i, note that for any solution π_i for Π_i, and for any sequence of actions π' of all other agents, the set of interleavings of π_i and π' which respects *waitfor* preconditions is the same in Π'^l and in Π^l. Furthermore, note that given any such interleaving π, it achieves G_i in Π'^l iff it achieves G_i in Π^l. Finally, note that π always achieves the (always true) goal of the second agent in Π'^l, and that the definition for adversarial against i does not require the agents other than i to achieve their goal. □

We conclude this section by noting that Definition 1 is somewhat restrictive. Specifically, it assumes each agent chooses a plan to execute *a-priori*, and then executes that plan. Without introducing the ability to wait, this is similar to conformant planning — whenever the scheduler tells an agent to act, it must execute its next action. If the preconditions of that action do not hold, the agent fails.

In general, we would like to be able to support more general policies for the agents. This would be similar to contingent planning, except that the non-determinism is really the result of other agents acting in the world. However, this makes the non-deterministic planning problem highly intractable, as any action can have many outcomes [Muise *et al.*, 2015].

Therefore, in this paper we adopt a limited form of contingent planning — waiting until some condition holds. This means that each agent can treat its own individual planning problem as a classical planning problem, and does not need to reason about the possible changes introduced by the other agents. In fact, we argue that the purpose of a good social law is to allow agents to do just that — not have to reason about what the other agents are going to do, assuming they respect the social law.

To show that waiting is a natural way to specify social laws, consider a traffic light. Without waiting, if an agent chooses to execute the action which drives into the intersection, and the light happens to be red, the action will fail because one of its preconditions is violated. However, if we denote the precondition of having a green light as *waitfor*, we obtain the desired behavior of waiting for the light to turn green.

5 Verifying Social Laws

Now that we have formally stated the criteria we want in a social law, and showed that verifying adversarial robustness can be compiled into verifying rational robustness, we turn to describing how we can verify that a social law l applied in a multi-agent setting Π is rationally robust. Our algorithm compiles the VERIFY-RATIONAL problem for $\Pi^l = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ into a classical planning problem. This compilation is described formally in full in Figure 1, but we will first provide some explanations of the compilation, and then prove its correctness.

The idea behind this compilation is to create $n + 1$ copies of each fact of the planning problem, and thus of the state. We will refer to copies $1 \dots n$ as local copies (one for each agent), and the final copy as the global copy, which will be denoted by g. Each action of agent i affects both its local copy i and the global copy g. Thus, each agent i acts alone in copy i, and all agents act together in the global copy g. The goal is to find a plan for each agent which works alone (that is, in copy i), but when all plans are joined together (in an order chosen by the planner) in copy g, there is a failure. The goal of this planning task is to achieve G_i in copy i, and have either a deadlock or some action fail in the global copy, as indicated by the flag $failure$. We remark that this duplication of facts is similar to the compilations for discovering worst case distinctiveness in goal recognition design [Keren *et al.*, 2014; 2015; 2016], although the rest of the compilation is very different.

In order to identify failures in the global copy, we create several versions of each action a_i for each of the possible outcomes of a_i: a_i succeeds, a_i fails due to a violated (non-wait) precondition, or a_i leads to a deadlock. Each of these work in copy i as if a_i succeeds, but has different effects on the global copy: the success outcome also succeeds in the global copy, the fail version requires one of the preconditions of a_i to be false in the global copy[1], and the deadlock version

[1] note that this requires disjunctive negative preconditions, which can be easily compiled away by adding more actions

$\Pi' = \langle F', A', I', G' \rangle$, where:

- $F' = \{f_1 \dots f_n \mid f \in F\} \cup \{f_g, f_c \mid f \in F\} \cup \{wt_{f,i} \mid f \in F, i = 1 \dots n\} \cup \{fin_i \mid i = 1 \dots n\} \cup \{failure, act\}$
- $A' = \bigcup_{i=1}^{n} A'_i \cup$ $\{$CHECK-NO-f, CHECK-NO-WAITING-$f \mid f \in F\}$, where: $A'_i = \{\text{END}_i^s, \text{END}_i^f, \text{END}_i^w\} \cup \{a_i^s, a_i^f \mid a_i \in A_i\} \cup \{a_i^{w,x} \mid a_i \in A_i, x \in pre_w(a_i)\}$, such that:

  $pre(a_i^s) = act \wedge (\bigwedge_{f \in pre(a_i)}(f_i \wedge f_g))$,
  $add(a_i^s) = \{f_i, f_g \mid f \in add(a_i)\}$,
  $del(a_i^s) = \{f_i, f_g \mid f \in del(a_i)\}$,

  $pre(a_i^f) = act \wedge (\bigwedge_{f \in pre(a_i)} f_i) \wedge (\bigwedge_{f \in pre_w(a)} f_g) \wedge (\bigvee_{f \in pre_f(a_i)} \neg f_g)$,
  $add(a_i^f) = \{failure\} \cup \{f_i \mid f \in add(a_i)\}$,
  $del(a_i^f) = \{f_i \mid f \in del(a_i)\}$,

  $pre(a_i^{w,x}) = act \wedge (\bigwedge_{f \in pre(a_i)} f_i) \wedge \neg x_g$,
  $add(a_i^{w,x}) = \{failure, wt_{x,i}\} \cup \{f_i \mid f \in add(a_i)\}$,
  $del(a_i^{w,x}) = \{f_i \mid f \in del(a_i)\}$,

  $pre(END_i^s) = \neg fin_i \wedge (\bigwedge_{f \in G_i} f_i) \wedge (\bigwedge_{f \in G_i} f_g)$,
  $add(END_i^s) = \{fin_i\}$,
  $del(END_i^s) = \{act\}$,

  $pre(END_i^f) = \neg fin_i \wedge (\bigwedge_{f \in G_i} f_i) \wedge (\bigvee_{f \in G_i} \neg f_g)$,
  $add(END_i^f) = \{fin_i, failure\}$,
  $del(END_i^f) = \{act\}$

  $pre(END_i^w) = \neg fin_i \wedge (\bigwedge_{f \in G_i} f_i) \wedge (\bigvee_{f \in F} wt_{f,i})$,
  $add(END_i^w) = \{fin_i, failure\}$,
  $del(END_i^w) = \{act\}$

  $pre($CHECK-NO-$f) = (\bigwedge_{i=1\dots n} fin_i) \wedge \neg f_g$,
  $add($CHECK-NO-$f) = f_c$,
  $del($CHECK-NO-$f) = \emptyset$

  $pre($CHECK-NO-WAITING-$f) = (\bigwedge_{i=1\dots n} fin_i) \wedge (\bigwedge_{i=1\dots n} \neg wt_{f,i})$,
  $add($CHECK-NO-WAITING-$f) = f_c$,
  $del($CHECK-NO-WAITING-$f) = \emptyset$

- $I' = \{act\} \cup \{f_i \mid f \in I, i = 1 \dots n\} \cup \{f_g \mid f \in I\}$, and
- $G' = \{failure\} \cup \{f_c \mid f \in F\} \cup \{fin_i \mid i \in \{1 \dots n\}\}$

Figure 1: Formal Description of the Compilation. For ease of exposition, we use logic, rather than sets, to express preconditions.

requires that one of the facts that a_i waits for is false, and remains false when agent i has a chance to act. This is achieved by raising a flag $wt_{f,i}$ that indicates that agent i is waiting for fact f. Since we assume the scheduler is adversarial to the agents, and thus under the control of the planner, the next opportunity when agent i is sure to be able to act is after all other agents have finished (either achieved their goal or are also waiting).

It is important to note here that we are attempting to find a sequence in which actions are *executed* which leads to a failure, not a sequence in which actions are started. Thus, if agent i has started action a_i, which is currently waiting for fact f, this could be reflected in the final plan in two different ways. If this is going to result in a deadlock, meaning that the wait precondition f must not hold at the end, then the plan will contain the deadlock version of a_i, at the point when agent i decides to apply a_i. However, if f is going to be achieved, and the scheduler is going to execute a_i when this happens, then the success version of a_i will appear in the plan *later*, when a_i is actually executed, and not when agent i decides to apply a_i.

In order to know when agents have finished, we also add an END action for each agent, whose preconditions are the goal facts of the agent. We create the success, fail, and deadlock versions of this action, and thus the only failures we need to consider are action preconditions not holding and deadlocks. However, in order to prevent a situation where an agent achieves its goal early, and then another agent invalidates it later, we do not allow any "regular" actions to occur after one agent executed an END action, which is controlled by the act flag.

Finally, in order to make sure that deadlocks are true deadlocks (that is, that if agent i is waiting for fact f, then f will be false after all agents have finished), for each fact f we also add two actions which are meant to verify that no agent is waiting for f at the end, and f holds. These actions are called CHECK-NO-f and CHECK-NO-WAITING-f. The first checks that f does not hold, and the second checks that no agents are waiting for f. Both of these achieve a new fact, f_c, which is also included in the goal, and are only applicable after all agents have executed their END actions. Together, these actions verify that at the end, $wt_{f,i} \rightarrow \neg f$, that is, that if agent i is waiting for fact f, then f does not hold at the end. Note that, since we need this to hold for all agents, this is equivalent to $\neg f \vee (\bigwedge_{1=1\dots n} \neg wt_{f,i})$, and each of these actions is responsible for checking one of the disjuncts.

We now proceed to prove that the compilation is correct, through a series of lemmas. We begin by proving a lemma about the structure of any solution of Π'.

Lemma 1. *Any solution π of Π' can be divided into three subsequences, $\pi = \pi_a \cdot \pi_{END} \cdot \pi_{CHECK}$, such that π_a contains only "regular" actions (a_i^s, a_i^f or $a_i^{w,f}$ for some $a_i \in A_i$), π_{END} contains only END actions, and π_{CHECK} contains only CHECK-NO-f and CHECK-NO-WAITING-f actions.*

Proof. By construction of Π', as soon as one of the END actions is executed, act is deleted. As act is a precondition of all regular actions, they must all precede the first END action. Similarly, $\bigwedge_{i=1\dots n} fin_i$ is a precondition of all CHECK

actions, and since fin_i can only by achieved by one of the END_i actions, all END actions must precede all CHECK actions. □

Next, we prove that any solution for Π' contains valid individual solutions for each of the agents:

Lemma 2. *Let $\pi = \pi_a \cdot \pi_{END} \cdot \pi_{CHECK}$ be an arbitrary solution of Π'. Define π_i to be the subsequence of π_a consisting only of actions of agent i. Then π_i is a solution for Π^l_i — the projection of Π^l for agent i.*

Proof. Let us look at the projection of Π' on $\{f_i \mid f \in F\} \cup \{fin_i\}$, which we will denote by Π'_i. It is easy to see that Π'_i is simply Π^l_i with an END action that achieves fin_i added to it. As Π'_i is an abstraction of Π', any solution for Π' is also a solution for Π'_i. Since actions that do not belong to agent i do not affect $\{f_i \mid f \in F\}$ or fin_i, if π is a solution of Π', $\pi_i \cdot \langle \text{END}_i \rangle$ is a solution of Π'_i, for any of the versions of END_i. Because Π'_i and Π^l_i are equivalent except for the addition of END and fin_i, π_i is a solution for Π^l_i. □

We now prove that any solution of Π' respects the *waitfor* preconditions:

Lemma 3. *Let $\pi = \pi_a \cdot \pi_{END} \cdot \pi_{CHECK}$ be an arbitrary solution of Π'. π respects the* waitfor *preconditions of all actions in π_a, that is, whenever one of the success (a^s_i) or fail (a^f_i) variants of action a_i is executed in π, all* waitfor *preconditions of a_i hold.*

Proof. $pre_w(a_i)$ is in the preconditions of both a^s_i and a^f_i, meaning that any action that is executed, is executed only when the agent would not have waited to execute it. Recall that the meaning of $a^{w,f}_i$ is that agent i is attempting to execute a_i, and will now wait for f forever (that is, until the end of the plan). □

We are now ready to prove our main theorem, about the correctness of the compilation:

Theorem 2. *Assume Π^l_i is solvable for all agents i. Then Π' is not solvable iff Π^l is rationally robust.*

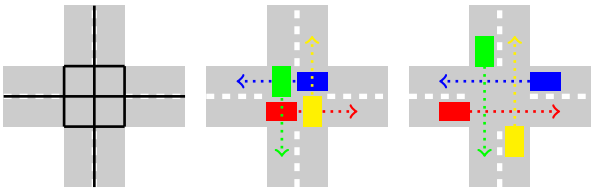
Proof. Assume Π' is solvable, let $\pi = \pi_a \cdot \pi_{\text{END}} \cdot \pi_{\text{CHECK}}$ be a solution for Π', and denote by π_i the subsequence of π_a consisting only of actions of agent i. Let us denote the first non-success action in π_i (that is, a^f_i or $a^{w,f}_i$) by ns_i, where $ns_i = \perp$ if all actions in π_i are success actions (that is, a^s_i). From Lemma 2, each π_i is a solution for Π_i.

First, note that there must exist some j such that $ns_j \neq \perp$. Otherwise, none of the actions in the plan achieve $failure$, which is part of the goal. If there exists some j such that $ns_j = a^f_j$ then π_a gives us an interleaving of $\{\pi_i\}_{i=1}^n$ which violates one of the (non-wait) preconditions of a^f_j. From Lemma 3, π_a respects all *waitfor* preconditions. Thus, we have found an interleaving of valid individual plans, which respects *waitfor* preconditions, but leads to an illegal joint plan, and thus Π^l is not rationally robust.

If there does not exist some j such that $ns_j = a^f_j$ then there must exist some j such that $ns_j = a^{w,f}_j$. We can guarantee that f will not hold at the end, since the only way to achieve f_c, which is part of the goal, would be through CHECK-NO-f (as CHECK-NO-WAITING-f is not applicable after $a^{w,f}_j$ is executed). Thus, if the scheduler does not allow agent j to act until all other agents are done, we have an interleaving in which agent j is in a deadlock. Here again, we have found an interleaving of valid individual plans, which respects *waitfor* preconditions, but leads to an illegal joint plan, and thus Π^l is not rationally robust.

We have shown that if Π' has a solution then Π^l is not rationally robust. Now assume Π' does not have a solution, and let π_i be any solution for Π_i, for $i = 1 \ldots n$. Let π be any interleaving of these individual plans which respects *waitfor* preconditions. All preconditions of all actions in π hold when the action is executed, as otherwise Π' would have been solvable (taking π with the appropriate END and CHECK actions added at the end as a solution). Similarly, π achieves $G_1 \cup \ldots \cup G_n$, and none of the agents is stuck waiting for some fact (as again, either of these scenarios would have led to Π' being solvable). Thus, if Π' does not have a solution then Π^l is rationally robust.

□



(a) (b) (c)

Figure 2: Intersection Example Illustration

6 Empirical Evaluation

We implemented our compilation, based upon the script which was used to convert MA-PDDL to PDDL representing a centralized planning problem from the first Competition of Distributed and Multiagent Planners [Stolba *et al.*, 2015]. We remark that while we have described the compilation for a grounded MA-STRIPS setting, most of the compilation can actually be done on the lifted level of MA-PDDL.

Our empirical evaluation is divided into two parts: First, to demonstrate the benefits of our approach, we describe a scenario in which a human designer uses it to create a useful social law. Second, we provide some empirical results on existing benchmarks, to demonstrate how our technique scales up with problem size.

6.1 Intersection Example

We demonstrate how one might use our proposed compilation on the classical example for deadlock — that of an intersection with entrances from the north, south, east, and west, where each car wants to go straight. This is illustrated in Figure 2.

This example can be modeled with facts: $\text{at}(A, L)$ and $\text{clear}(L)$ where A is an agent and L is one of the 12 locations illustrated in Figure 2(a): both lanes on the north, south, east,

BLOCKSWORLD	
Instance	Time (s)
9-0	0.1
9-1	0.09
9-2	0.11
10-0	0.1
10-1	0.08
10-2	0.09
11-0	0.17
11-1	0.18
11-2	0.11
12-0	0.18
12-1	0.26
13-0	0.32
13-1	0.48
14-0	0.44
14-1	0.19
15-0	4.17
15-1	1.83
16-1	1.83
16-2	0.43
17-0	8.23

DRIVERLOG	
Instance	Time (s)
pfile1	0
pfile2	0
pfile3	0
pfile4	0
pfile5	0
pfile6	0
pfile7	0
pfile8	0
pfile9	0
pfile10	0
pfile11	0.01
pfile12	0.01
pfile13	0.02
pfile14	0.04
pfile15	0.17
pfile16	1.25
pfile17	28.63
pfile18	23.53
pfile19	88.06
pfile20	111.54

ZENOTRAVEL	
Instance	Time (s)
pfile3	0
pfile4	0
pfile5	0
pfile6	0
pfile7	0
pfile8	0
pfile9	0
pfile10	0.01
pfile12	0
pfile13	0.04
pfile14	0.1
pfile15	0.22
pfile16	0.12
pfile17	0.61
pfile18	1.04
pfile19	5.26
pfile20	2.77
pfile21	3.85
pfile22	5.46
pfile23	15.06

SATELLITES	
Instance	Time (s)
p05	0.11
p07	0.24
p21	243.38
p24	—
p25	—

SOKOBAN	
Instance	Time (s)
p06-1	—

Table 1: Solution Times on Benchmark Domains (timeouts indicated by a dash)

or west of the intersection, as well as the southeast, southwest, northeast, or northwest corners of the intersection. The actions are:

- arrive(A, l), which models the arrival of agent A at location l, which must be one of the north, south, east, or west entrances to the intersection. This action adds at(A, l), and can only be applied once per agent[2].
- drive(A, l_1, l_2), which models agent A driving from location l_1 to location l_2. This requires at(A, l_1) and clear(l_2).

When we run our compilation on this problem, we of course obtain a failure, as agents can crash into each other in the intersection, which is represented by violating the clear preconditions of drive.

In an effort to correct this, we add to drive a wait annotation which avoids driving into occupied locations. Running our compilation on this yields a deadlock, as illustrated in Figure 2(b).

Attempting to correct this, we add to drive a precondition stating that a car that is about to enter the intersection must yield to a car which is about to enter the intersection from its right. Running our compilation on this yields a deadlock, as illustrated in Figure 2(c).

Finally, we arrive at a deadlock free solution, by dropping the yield preconditions we added for cars entering from the east and the west, while still making cars entering from the north and south yield, and also adding preconditions which ensure that a car does not enter the intersection when it is blocked. Our compilation then verifies that this social law is rationally robust. All planner runs in this example terminate within fractions of a second.

6.2 Benchmarks

In order to evaluate the effectiveness of our compilation on problems with increasing size, we used the benchmark domains from the first Competition of Distributed and Multiagent Planners [Stolba *et al.*, 2015]. These benchmarks are for cooperative planning, and thus contain a single goal in each instance. We created an instance with a separate goal for each agent by allocating each fact in the goal to one of the agents, in a round-robin manner (except in cases where the first argument of the goal fact mentions a specific agent, in which case it was allocated to that agent). We excluded problem instances in which one of the agents was not able to achieve its goal alone (we checked this by solving the Π_i planning problem for each agent), which left us with only 3 full domains (BLOCKSWORLD, DRIVERLOG, and ZENOTRAVEL), as well as 5 instances from SATELLITE and one instance of SOKOBAN.

The choice of planner to use here poses an interesting question. On the one hand, if the social law we are trying to verify is robust, then the planning problem is going to be unsolvable. In such a case, planners for proving unsolvability [Bäckström *et al.*, 2013; Hoffmann *et al.*, 2014] might be a good choice. On the other hand, if we were sure that the social law is robust, we would not be verifying it, and thus using a planner that is geared towards finding solutions might be better.

While in general it is probably a good idea to combine several planners in a portfolio, we have no reason to assume that the multi-agent planning benchmarks already contain a robust social law. In fact, if they did, they would have been fairly easy planning tasks. Therefore, we used the FF planner [Hoffmann and Nebel, 2001] on the compilation for each of these instances, with a timeout of 5 minutes. Table 1 shows how much time it took to solve each of these instances. As these results show, we are able to solve almost all of them very quickly.

7 Discussion

In this paper, we have connected social laws to model-based planning, and formalized some criteria which we believe "good" social laws should exhibit under this framework. We have also described a compilation to classical planning for verifying whether an artificial social system meets these criteria, and provided an empirical demonstration that this compilation is feasible in practice.

We remark that the principles behind the compilation we present can be easily extended to more realistic settings. First, agents might not know what the goals of other agents are, and only have some idea of what the possible goals are. A simple compilation which eliminates disjunctive goals can solve this problem. Second, agents might enter the system at different places and at different times. It is very easy to define actions for "adding" agents at legal locations, and our compilation will take care of making sure the social law criteria are not violated by this.

We conclude by noting that, in this paper, we focus on the problem of *verifying* whether a social law (encoded in a multi-agent planning framework) meets some desired criterion. However, our ultimate goal is to automatically synthesize such social laws, rather than just verifying them. Having efficient verification techniques is a first step in this direction.

[2] we keep track of this through another fact, which we omit for the sake of brevity

References

[Bäckström *et al.*, 2013] Christer Bäckström, Peter Jonsson, and Simon Ståhlberg. Fast detection of unsolvable planning instances using local consistency. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013, Leavenworth, Washington, USA, July 11-13, 2013*. AAAI Press, 2013.

[Brafman and Domshlak, 2008] Ronen I. Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS 2008*, pages 28–35. AAAI Press, 2008.

[d'Inverno and Luck, 2004] Mark d'Inverno and Michael Luck. *Understanding agent systems*. Springer, 2004.

[Fikes and Nilsson, 1971] Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence*, 14:253–302, 2001.

[Hoffmann *et al.*, 2014] Jörg Hoffmann, Peter Kissmann, and Álvaro Torralba. Distance? who cares? tailoring merge-and-shrink heuristics to detect unsolvability. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 441–446. IOS Press, 2014.

[Horling and Lesser, 2004] Bryan Horling and Victor Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(04):281–316, 2004.

[Keren *et al.*, 2014] Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design. In *ICAPS Conference Proceedings*, June 2014.

[Keren *et al.*, 2015] Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design for non optimal agents. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2015)*, January 2015.

[Keren *et al.*, 2016] Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design with non oservable actions. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2016) - to appear*, February 2016.

[Klusch, 1999] Matthias Klusch. *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999.

[Moses and Tennenholtz, 1995] Yoram Moses and Moshe Tennenholtz. Artificial social systems. *Computers and Artificial Intelligence*, 14(6), 1995.

[Muise *et al.*, 2015] Christian Muise, Paolo Felli, Tim Miller, Adrian R. Pearce, and Liz Sonenberg. Leveraging fond planning technology to solve multi-agent planning problems. In *Workshop on Distributed and Multi-Agent Planning (DMAP'15)*, 2015.

[Rousseau, 1762] J.J. Rousseau. *Du Contrat Social*. 1762.

[Shoham and Leyton-Brown, 2008] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, NY, USA, 2008.

[Shoham and Tennenholtz, 1992a] Yoav Shoham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies (preliminary report). In *AAAI 1992*, pages 276–281, 1992.

[Shoham and Tennenholtz, 1992b] Yoav Shoham and Moshe Tennenholtz. On traffic laws for mobile robots (extended abstract). In *AIPS 1992*, pages 309–310. Kaufmann, San Mateo, CA, 1992.

[Shoham and Tennenholtz, 1995] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1-2):231–252, 1995.

[Stolba *et al.*, 2015] Michal Stolba, Antonín Komenda, and Daniel Laszlo Kovacs. Competition of distributed and multiagent planners (CoDMAP). http://agents.fel.cvut.cz/codmap/, 2015.

[Tennenholtz and Moses, 1989] Moshe Tennenholtz and Yoram Moses. On cooperation in a multi-entity model. In *IJCAI 1989*, pages 918–923. Morgan Kaufmann, 1989.

[Tennenholtz, 1991] M. Tennenholtz. *Efficient Representation and Reasoning in Multi-Agent Systems*. PhD thesis, Weizmann Institute, Israel, 1991.

[Woolridge, 2001] Michael Woolridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

Quantifying Privacy Leakage in Multi-Agent Planning

Michal Štolba and **Jan Tožička** and **Antonín Komenda**
{stolba,tozicka,komenda}@agents.fel.cvut.cz
Department of Computer Science, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic

Abstract

Multi-agent planning using MA-STRIPS-related models is often motivated by the preservation of private information. Such motivation is not only natural for multi-agent systems, but is one of the main reasons, why multi-agent planning (MAP) problems cannot be solved centrally. Although the motivation is common in the literature, formal treatment of privacy is mostly missing. An exception is a definition of two extreme concepts, weak and strong privacy.

In this paper, we first analyze privacy leakage in the terms of secure Multi-Party Computation and Quantitative Information Flow. Then, we follow by analyzing privacy leakage of the most common MAP paradigms. Finally, we propose a new theoretical class of secure MAP algorithms and show how the existing techniques can be modified in order to fall in the proposed class.

Introduction

Multi-agent planning models the problems in which multiple entities (or agents) need to generate a sequence of actions in order to fulfill some specified goal, either common or respective to each agent. If the environment and actions are deterministic (that is their outcome is unambiguously defined by the state they are applied in), we are talking about deterministic multi-agent planning. Here, we focus on a cooperative scenario, where the agents are coordinating their actions in order to fulfill a common goal. The reason the agents cannot simply feed their problem descriptions into a centralized planner typically lies in that although the agents cooperate, they want to share only the information necessary for their cooperation, but not the information about their inner processes.

We denote deterministic cooperative multi-agent planning problems with privacy as Privacy-Preserving Multi-Agent Planning (PP-MAP) problems. A number of planners solving PP-MAP has been proposed in recent years, such as MAFS (Nissim and Brafman 2014), FMAP (Torreño *et al.* 2014), PSM (Tožička *et al.* 2015) and GPPP (Maliah *et al.* 2016). Although all of the mentioned planners claim to be privacy-preserving, thorough formal treatment of such claims is rather scarce. The privacy of MAFS is discussed in (Nissim and Brafman 2014) and expanded upon in (Brafman 2015), proposing Secure-MAFS, a version of MAFS with stronger privacy guarantees.

The most formal definition and treatment was published in (Nissim and Brafman 2014) and further elaborated on in (Brafman 2015). The authors present two notions, weak and strong privacy preservation. It turns out, that the two definitions are both extremities of a whole spectrum. Weak privacy preservation forbids only explicit communication of the private information, which is trivial to achieve and provides no security guarantees. The strong privacy preservation forbids leakage of any information allowing other agents to deduce any private information at all.

The amount of privacy loss in general algorithms (or functions) has been studied in secure Multi-Party Computation (MPC) (Yao 1982) as information leakage in Quantitative Information Flow (Smith 2009; Braun *et al.* 2009). In this paper, we take this general approach and apply it on the problem of PP-MAP, in order to theoretically analyze privacy leakage of the main planning paradigms used in multi-agent planning and also of the particular planning algorithms. We propose a new class of MAP algorithms, SECMAP and show that it preserves more privacy than any of the existing algorithms and how the existing algorithms can be modified to fall in the new class of SECMAP algorithms.

Multi-Agent Planning

The most common model for PP-MAP problems is MA-STRIPS (Brafman and Domshlak 2008) and derived models (such as MA-MPT (Nissim and Brafman 2014) using multi-valued variables). We reformulate the MA-STRIPS definition and we also generalize the definition to multi-valued variables. Formally, for a set of agents $\mathcal{A}$, a PP-MAP problem $\mathcal{M} = \{\Pi_i\}_{i=1}^{|\mathcal{A}|}$ is a set of agent problems. An agent problem of agent $\alpha_i \in \mathcal{A}$ is defined as

$$\Pi_i = \left\langle \mathcal{V}_i = \mathcal{V}^{\mathsf{pub}} \cup \mathcal{V}_i^{\mathsf{priv}}, \mathcal{O}_i = \mathcal{O}_i^{\mathsf{pub}} \cup \mathcal{O}_i^{\mathsf{priv}} \cup \mathcal{O}^{\mathsf{proj}}, s_I, s_\star \right\rangle,$$

where $\mathcal{V}_i$ is a set of variables s.t. each $V \in \mathcal{V}_i$ has a finite domain $\mathsf{dom}(V)$, if all variables are binary (i.e. $|\mathsf{dom}(V)| = 2$), the formalism corresponds to MA-STRIPS. The set of variables is partitioned into the set $\mathcal{V}^{\mathsf{pub}}$ of public variables (with all values public), common to all agents and the set $\mathcal{V}_i^{\mathsf{priv}}$ of variables private to α_i (with all values private), such that $\mathcal{V}^{\mathsf{pub}} \cap \mathcal{V}_i^{\mathsf{priv}} = \emptyset$. A complete assignment over $\mathcal{V}$ is a *state*, partial assignment over $\mathcal{V}$ is a partial state. We denote

$s[V]$ as the value of V in the (partial) state s and $\mathsf{vars}(s)$ as the set of variables defined in s. The state s_I is the initial state and $s_\star$ is a partial state representing the goal condition, that is if for all variables $V \in \mathsf{vars}(s_\star)$, $s_\star[V] = s[V]$, s is a goal state.

The set $\mathcal{O}_i$ of actions comprises of a set $\mathcal{O}_i^{\mathsf{priv}}$ of private actions of α_i, a set $\mathcal{O}_i^{\mathsf{pub}}$ of public actions of α_i and a set $\mathcal{O}^{\mathsf{proj}}$ of public projections of other agents' actions. $\mathcal{O}_i^{\mathsf{pub}}$, $\mathcal{O}_i^{\mathsf{priv}}$, and $\mathcal{O}^{\mathsf{proj}}$ are pairwise disjoint. An action is defined as a tuple $a = \langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{lbl}(a) \rangle$, where $\mathsf{pre}(a)$ and $\mathsf{eff}(a)$ are partial states representing the precondition and effect respectively and $\mathsf{lbl}(a)$ is a unique label. An action a is applicable in state s if $s[V] = \mathsf{pre}(a)[V]$ for all $V \in \mathsf{vars}(\mathsf{pre}(a))$ and the application of a in s, denoted $a \circ s$, results in a state s' s.t. $s'[V] = \mathsf{eff}(a)[V]$ if $V \in \mathsf{vars}(\mathsf{eff}(a))$ and $s'[V] = s[V]$ otherwise. When we are considering the planning problem from the perspective of a single agent α_i, we omit the index i whenever possible.

We model all "other" agents as a single agent (the adversary), which is common in secure MPC, as all the agents can collude and combine their information in order to infer more. The public part of the problem Π which can be shared with the adversary is denoted as a public projection. The public projection of a (partial) state s is $s^\triangleright$, restricted only to variables in $\mathcal{V}^{\mathsf{pub}}$, that is $\mathsf{vars}(s^\triangleright) = \mathsf{vars}(s) \cap \mathcal{V}^{\mathsf{pub}}$. We say that s, s' are publicly equivalent states if $s^\triangleright = s'^\triangleright$. The public projection of action $a \in \mathcal{O}^{\mathsf{pub}}$ is $a^\triangleright = \langle \mathsf{pre}(a)^\triangleright, \mathsf{eff}(a)^\triangleright, \mathsf{lbl}(a)^\triangleright \rangle$ and of action $a' \in \mathcal{O}^{\mathsf{priv}}$ is an empty (no-op) action ϵ. The public projection of Π is $\Pi^\triangleright = \left\langle \mathcal{V}^{\mathsf{pub}}, \{a^\triangleright | a \in \mathcal{O}^{\mathsf{pub}}\}, s_I^\triangleright, s_\star^\triangleright \right\rangle$.

In general, the public projection $\mathsf{lbl}(a)^\triangleright$ of the label $\mathsf{lbl}(a)$, does not have to preserve the uniqueness. Thus a single public projection $a^\triangleright$ can represent multiple actions a, a' such that $\mathsf{pre}(a)^\triangleright = \mathsf{pre}(a')^\triangleright$ and $\mathsf{eff}(a)^\triangleright = \mathsf{eff}(a')^\triangleright$. Even though for an agent trying to hide its private information it is reasonable to publish only one publicly projected action with distinct public precondition and effect $\langle \mathsf{pre}(a)^\triangleright, \mathsf{eff}(a)^\triangleright \rangle$, none of the existing PP-MAP planners does it. Therefore we define also a label preserving projection $^\trianglerighteq$ which differs in that $\mathsf{lbl}(a)^\trianglerighteq = \mathsf{lbl}(a)$. Apart from this difference, $\Pi^\trianglerighteq$ is defined equivalently to $\Pi^\triangleright$. Later we show the label preserving projection leaks a significant amount of information.

Finally, we define the solution to Π and $\mathcal{M}$. A sequence $\pi = (a_1, ..., a_k)$ of actions from $\mathcal{O}$, s.t. a_1 is applicable in $s_I = s_0$ and for each $1 \leq i \leq k$, a_i is applicable in s_{i-1} and $s_i = a_i \circ s_{i-1}$, is a local s_k-plan, where s_k is the resulting state. If s_k is a goal state, π is a local plan, that is a local solution to Π. Such π does not have to be the global solution to $\mathcal{M}$, as the actions of other agents ($\mathcal{O}^{\mathsf{proj}}$) are used only as public projections and may be missing private preconditions and effects of other agents. We define $\pi^\triangleright = (a_1^\triangleright, ..., a_k^\triangleright)$ with ϵ actions omitted to be the public projection of π.

From the global perspective of $\mathcal{M}$ a public plan $\pi^\triangleright = (a_1^\triangleright, ..., a_k^\triangleright)$ is a sequence of public projections of actions of various agents from $\mathcal{A}$ such that the actions are sequentially applicable with respect to $\mathcal{V}^{\mathsf{pub}}$ starting in $s_I^\triangleright$ and the resulting state satisfies $s_\star^\triangleright$. A public plan is i-extensible, if by replacing $a_{k'}^\triangleright$ s.t. $a_{k'} \in \mathcal{O}_i^{\mathsf{pub}}$ by the respective $a_{k'}$ and adding $a_{k''} \in \mathcal{O}^{\mathsf{priv}}$ to required places we obtain a local plan (solution) to Π_i. According to (Tožička *et al.* 2015), a public plan $\pi^\triangleright$ i-extensible by all $\alpha_i \in \mathcal{A}$ is a global solution to $\mathcal{M}$.

Transition System Model

We will define explicitly the transition system induced by the planning problem. A transition system of a problem Π is a tuple $T(\Pi) = \langle S, L, T, s_I, s_\star \rangle$, where $S = \prod_{V \in \mathcal{V}} \mathsf{dom}(V)$ is a set of states, $L = \{\mathsf{lbl}(a) | a \in \mathcal{O}\}$ is a set of transition labels corresponding to the actions in $\mathcal{O}$ and $T \subseteq S \times L \times S$ is a transition relation of Π s.t. $\langle s, l, s' \rangle \in T$ if $a \in \mathcal{O}$ s.t. $\mathsf{lbl}(a) = l$ is applicable in s and $s' = a \circ s$. The state $s_I \in S$ is the initial state and the goal condition $s_\star$ describes all goal states as in Π. A public projection of $T(\Pi)$ is $T(\Pi)^\triangleright = \langle S^\triangleright, L^\triangleright, T^\triangleright, s_I^\triangleright, s_\star^\triangleright \rangle$ such that $S^\triangleright$ is S restricted to $\mathcal{V}^{\mathsf{pub}}$, $L^\triangleright = \{\mathsf{lbl}(a)^\triangleright | a \in \mathcal{O}\}$ and $\langle s^\triangleright, \mathsf{lbl}(a)^\triangleright, s'^\triangleright \rangle \in T^\triangleright$ if $\langle s, \mathsf{lbl}(a), s' \rangle \in T$. We denote $s \in S$ as a private state (although it contains also public variables) and $s^\triangleright \in S^\triangleright$ as a public state.

For the global problem $\mathcal{M}$, we define an intersection of the transition systems $T(\mathcal{M}) = \bigcap_{\alpha_i \in \mathcal{A}} T(\Pi_i)$ as a transition system on states $\bar{s}$ over $\mathcal{V}^{\mathsf{pub}} \cup \bigcup_{\alpha_i \in \mathcal{A}} \mathcal{V}_i^{\mathsf{priv}}$. A transition $\langle s, l, s' \rangle$ in $T(\Pi_i)$ is represented by a set of transitions $\langle \bar{s}, l, \bar{s}' \rangle$ such that $\bar{s}[V] = s[V]$ for all $V \in \mathcal{V}^{\mathsf{pub}} \cup \mathcal{V}_i^{\mathsf{priv}}$, $\bar{s}'[V] = s'[V]$ for all $V \in \mathcal{V}^{\mathsf{pub}} \cup \mathcal{V}_i^{\mathsf{priv}}$ and $\bar{s}[V] = \bar{s}'[V]$ for all $V \in \bigcup_{\alpha_j \in \mathcal{A} \setminus \alpha_i} \mathcal{V}_j^{\mathsf{priv}}$.

Privacy in the Literature

Apart from a specialized privacy leakage quantification by (Van Der Krogt 2009) (which is not practical as it is based on enumeration of all plans and also is not applicable to MA-STRIPS in general), the only rigorous definition of privacy for PP-MAP so far was proposed in (Nissim and Brafman 2014) and extended in (Brafman 2015). Here, we rephrase the definitions in order to build on them in the following sections.

We say that an algorithm is *weak privacy-preserving* if, during the whole run of the algorithm, the agent does not openly communicate private parts of the states, private actions and private parts of the public actions. In other words, the agent openly communicates only the information in $\Pi^\triangleright$. Even if not communicated, the adversary may deduce the existence and values of private variables, preconditions and effects from the (public) information communicated.

An algorithm is *strong privacy-preserving* if the adversary can deduce no information about a private variable and its values and private preconditions/effect of an action, beyond what can be deduced from the public projection $\Pi^\triangleright$ and the public projection of the solution plan $\pi^\triangleright$.

Privacy Leakage

One of the threat models studied in the literature on secure MPC is the threat that an attack will allow the adversary to guess the private information of the agent. In the case of PP-MAP this means that the adversary may be able to guess

the actual transition system of the agent. In the weak privacy case, the probability of the right guess is not considered at all, whereas in the strong privacy case even an unrealistically small probability is considered a breach of the privacy.

Based on (Smith 2009), let us have an algorithm which takes a high (private) input H and produces low (public) output L. What we are interested in is, how much information about H can be deduced by an adversary who sees the output L. We assume that there is an a priori, publicly-known probability distribution of a random variable H with a finite space of possible values $\mathcal{H}$. We denote the a priori probability that H has a value $h \in \mathcal{H}$ by $P[H = h]$, and we assume that each element h of $\mathcal{H}$ has nonzero probability. Similarly, we assume that L is a random variable with a finite space of possible values $\mathcal{L}$, and with probabilities $P[L = l]$. We assume that each output $l \in \mathcal{L}$ is possible, in that it can be produced by some input $h \in \mathcal{H}$.

The leakage of private information is based on the uncertainty of the adversary about the input H. A high-level formula is

$$\text{information leaked} = \text{initial uncertainty} - \text{remaining uncertainty}, \quad (1)$$

where initial uncertainty is related to the probability of guessing the right input without any additional knowledge gained from the output L of the algorithm, whereas remaining uncertainty is the probability of guessing the right input H given the output L.

In the case of PP-MAP, the high (private) input the adversary is attempting to guess is the transition system $T(\Pi)$ of the agent's problem. In agreement with the assumptions above, we assume that an upper bound on the size (number of states) of $T(\Pi)$ is publicly known as n. This bound together with the public projection of the transition system limits the number of the possible transition systems, denoted as t_{apriori}. We also assume that all possible transition systems are equally probable, which gives us an uniform distribution.

The public output consists of all information obtained from the agent during the planning process, that is the public projection of the transition system $T(\Pi)^{\triangleright}$, the resulting plan π and all additional information obtained during the planning process.

By the application of (Smith 2009) to the case of the possible transition system which is deterministic and has assumed uniform distribution, we obtain information leakage measures based on min-entropy $H_\infty(H)$:

initial uncertainty: $H_\infty(H) = \log t_{\mathsf{apriori}}$

remaining uncertainty: $H_\infty(H|L) = \log t_{\mathsf{post}}$

information leaked: $H_\infty(H) - H_\infty(H|L) =$

$$\log t_{\mathsf{apriori}} - \log t_{\mathsf{post}} = \log \frac{t_{\mathsf{apriori}}}{t_{\mathsf{post}}} \quad (2)$$

Where t_{apriori} is the number of possible transition systems known a-priori, that is based on the public projection $T(\Pi^{\triangleright})$, the assumption of maximum n nodes and the public projection of the resulting plan $\pi^{\triangleright}$. Conversely, t_{post} is the number of transition systems based on the posteriori knowledge, that is the public projection of the transition system, public projection of the resulting plan and all other information obtained from the run of the planning algorithm.

The remaining uncertainty gives a security guarantee as the expected probability of guessing H, that is the transition system of the agent, given L, the public output of the planning algorithm, is $2^{-H_\infty(H|L)} = 2^{-\log t_{\mathsf{post}}} = 1/t_{\mathsf{post}}$ where, again, t_{post} is the number of possible transition systems given the public output. Thanks to the determinism and uniform distributions, the result conveys with an intuition that the privacy preservation decreases by lowering the number of possible transition systems. In the next section, we will focus on how to estimate both t_{apriori} and t_{post}.

Leakage Quantification in PP-MAP

In the previous section we have placed an assumption that an upper bound n on the total number of the states of the agent's transition system is publicly known. This results in n^2 possible transitions (including loops) and 2^{n^2} possible transition systems. Moreover, we assume, that bounds on the number of private variables $p \geq |\mathcal{V}^{\mathsf{priv}}|$ and on the size of the private variable domains $d \geq |\mathsf{dom}(V)|$ for all $V \in \mathcal{V}^{\mathsf{priv}}$ are also publicly known.

Similarly to (Brafman 2015), for the simplicity of presentation we assume that $\mathcal{O}^{\mathsf{priv}} = \emptyset$. This assumption can be stated without the loss of generality as each sequence of private actions followed by a public action can be compiled as a single public action (with a potential exponential blow-up in the number of public actions). From the perspective of privacy, it is clear, that when adhering at least to the weak privacy, private action is never communicated and thus can never leak. Any information about private action always leaks in the sense of the described compilation, that is it appears as if the respective public action had some additional private preconditions or effects.

Let us first consider the public projection of the agent's transition system $T(\Pi)^{\triangleright}$. Based on the above bounds, the number of private states $s \in S$ represented by a single public state $s^{\triangleright} \in S^{\triangleright}$ is d^p. The number of possible private transitions $\langle s, l, s' \rangle \in T$ represented by a single public transition $\langle s^{\triangleright}, l^{\triangleright}, s'^{\triangleright} \rangle \in T^{\triangleright}$is $(d^p)^2$, that is from each private state s is a transition to a private state s'.

For a single variable ($p = 1$), there are d private states represented by each public state and for a single action a, the respective public transition $\langle s^{\triangleright}, \mathsf{lbl}(a)^{\triangleright}, s'^{\triangleright} \rangle \in T^{\triangleright}$ represents d^2 possible private transitions between two public states, that is, for each of the first d private states there either is or is not a transition to each of the second d private states, based on the private preconditions and effects of a. This set of transitions represents a transition system of the action a. An upper bound on the number of all such transition systems for a is $t_a = 2^{d^2}$, that is the number of subsets of the d^2 transitions. It is reasonable to assume that a public transition encodes only existing transitions (actions applicable in at least one private state). This means that an empty transition system is not an option, thus $t_a = 2^{d^2} - 1$, for binary (STRIPS) variable, $t_a = 15$. As the variables are independent, for p variables, we get $t_a = (2^{d^2} - 1)^p$, or $t_a = 15^p$ for STRIPS where $d = 2$.

In the case of a label preserving projection, $\mathsf{lbl}(a) = \mathsf{lbl}(a^{\triangleright})$ all actions $a \in \mathcal{O}$ have unique labels, because each public transition $\langle s^{\triangleright}, l^{\triangleright}, s'^{\triangleright} \rangle \in T^{\triangleright}$ represents exactly one action $a \in \mathcal{O}$. As a single action can never produce multiple states when applied in a single state, the number of possible transition systems t_a is significantly reduced. For a single variable with d values, we get the number of partial functions between two sets of size d, which is $t_a = (1+d)^d - 1$ (again -1 for empty transition system), for STRIPS $t_a = 8$, without conditional effects $t_a = 7$. Again, as the variables are independent, the numbers can be multiplied for each variable.

Based on the above, a complete public projection $T(\Pi)^{\triangleright}$ of the agent's transition system restricts the number of possible private transition system to $t_{\mathsf{proj}} = (t_a)^{|\mathcal{O}^{\mathsf{pub}}|}$, where t_a is the number of transition systems represented by a single action.

Sources of Leakage

Before we analyze complex algorithms, let us first focus on elements which have a major impact on the privacy leakage. From the Equation 1 and its particular instance Equation 2 follows, that the source we need to focus on is the information which is communicated in addition to the initial information (the projected problem $\Pi^{\triangleright}$ and the projected plan) which is superfluous. In particular, the sources of information we focus on are superfluous plans, superfluous distinct states and superfluous action applicability information.

Of course, there are plenty of possible other sources of information leakage and their combinations, but we base our analysis on these three prominent sources separately, thus providing a lower bound on the information leaked. In the following, we provide more detailed description of the aforementioned sources of leakage.

Superfluous plans are (partial) plans revealed by the algorithm without being the actual solution (or its prefix).

In the public projection of the transition system, all public projections of actions are independent. One possibility how to interconnect actions is a s_k-plan $\pi = (a_1, ..., a_k)$. As we assume a single initial state s_I, a public projection $\pi^{\triangleright}$ reveals information, that the first action is applicable in s_I. Without the loss of generality, we can assume that all variables have a particular value (e.g., 0 for STRIPS) in s_I, as the values can be arbitrarily renamed. This fixes uncertainty about a_1 in that it is now not possible that a_1 is not applicable in the initial state. But since we know that $s_I[V] = 0$ and application of a single action always results in single state, $s_1 = a_1 \circ s_I$ has also fixed value of V (although we do not know which one). This means that a_2 has also reduced number of possible transition systems.

For the general projection $\pi^{\triangleright}$, as a single public action may represent multiple actions, transition systems with nondeterministic behavior are possible, because even though applied on a single state, each of the represented actions may result in a different state. Thus the only information revealed is about the first action a_1.

Combination of multiple plans does not leak any additional information in comparison to the separate plans. If the projections of all plans of the agent are revealed to the adversary (e.g., after exhaustive search) and the adversary knows that no other plans exist (e.g., the complete state-space was explored), more can be deduced. Let $\pi_1^{\triangleright} = (a_1, a_2, a_3)$ and $\pi_2^{\triangleright} = (a_4, a_5, a_6)$ be two projected plans such that $s_1 = a_2 \circ (a_1 \circ s_I)$ and $s_2 = a_5 \circ (a_4 \circ s_I)$ are two publicly equivalent states. From the information that $\pi_3^{\triangleright} = (a_1, a_2, a_6)$ is not in the set of all possible plans it can be deduced, that s_1 and s_2 are in fact two distinct, although publicly equivalent, states. This leads to the definition of superfluous distinct states.

Superfluous distinct states are publicly equivalent states s, s' revealed that $s \neq s'$ and either s or s' is not part of the solution. In the previous example, the states s_1 and s_2 were discovered to be distinct, but if $\pi_2^{\triangleright} = (a_4, a_5, a_6)$ was the only actual solution, this distinction was not necessary and the private information has leaked.

The most common situation where the superfluous distinct state information leaks is the use of unique state labels. For example, in MAFS, each state communicated with other agents has to have a unique label representing the private part of the state, which is then copied to its successors created by other agents. Thus, when a successor of the state is communicated back to the original agent, it can reconstruct its private part. The same holds also for Secure-MAFS.

The superfluous distinct state information itself does not reduce the number of possible transition systems, but can be used for further deduction of superfluous action applicability, defined as follows.

Superfluous action applicability is an information of applicability of an action a on two distinct publicly equivalent states s, s' s.t. $a^{\triangleright}$ is applicable in both $s^{\triangleright}, s'^{\triangleright}$ known to the adversary that s, s' are distinct states and either s or s' is not part of the solution, in other words, the states s, s' are superfluous distinct states.

In general, if the adversary obtains an information, that an action a s.t. $a^{\triangleright}$ is applicable in public state $s^{\triangleright}$, but is not applicable in some of the private states represented by $s^{\triangleright}$, the number of possible transition systems represented by a is reduced. We refer to such action as *privately-dependent* action, as it must depend on a private variable. If the state s or the action a is not part of the solution, this information is superfluous and considered privacy leakage.

A similar situation appears, when a single action a is applicable in two publicly equivalent, but superfluous distinct states s, s'. In this case, the information learned is that the action does not depend on the private variable which distinguishes s and s', thus reduces the number of possible transition systems for a. We refer to such action as *privately-independent*, as it does not depend on any particular private variable (although it may still depend on other private variable).

A single action a may be privately-dependent for some private variable and privately-independent for some other variable.

Leakage Estimate

Let the number of transition systems represented by a single action a without any information revealed be t_a, which depends on the number of variables, their domain size and other factors. Let $\mathcal{O}^{\mathsf{plan}}$ denote the set of all actions a used in any superfluous plan[1] and t_a^{plan} the number of transition systems represented by a single action a reduced by the information learned from the plan existence. Similarly, we define the set $\mathcal{O}^{\mathsf{pd}}$ of privately-dependent actions and the set $\mathcal{O}^{\mathsf{pi}}$ of privately-independent actions and the respective number of transition systems represented by each such action a as t_a^{pd} and t_a^{pi} respectively, each reduced[2] according to the revealed information. Thus for each action $a \in \mathcal{O}^{\mathsf{pub}}$, we can define the number of transition systems it represents as

$$\tau_{\mathsf{post}}(a) = \min \begin{cases} t_a & \text{always} \\ t_a^{\mathsf{plan}} & \text{if } a \in \mathcal{O}^{\mathsf{plan}} \\ t_a^{\mathsf{pd}} & \text{if } a \in \mathcal{O}^{\mathsf{pd}} \\ t_a^{\mathsf{pi}} & \text{if } a \in \mathcal{O}^{\mathsf{pi}} \end{cases}$$

that is the minimum of the number of transition systems represented by a based on its membership in the $\mathcal{O}^{\mathsf{plan}}, \mathcal{O}^{\mathsf{pd}}, \mathcal{O}^{\mathsf{pi}}$ sets. For example if an action a_1 is revealed as privately-independent and is in some communicated plan, $\tau_{\mathsf{post}}(a_1) = \min(t_a, t_a^{\mathsf{plan}}, t_a^{\mathsf{pi}})$ where for a STRIPS variables, single private variable and label-preserving projection, the constants are $t_a = 7, t_a^{\mathsf{plan}} = 5, t_a^{\mathsf{pi}} = 3$ and thus $\tau_{\mathsf{post}}(a_1) = 3$.

The knowledge obtained about particular actions can be combined to compute the total number of possible transition systems $t_{\mathsf{post}} = \prod_{a \in \mathcal{O}^{\mathsf{pub}}} \tau_{\mathsf{post}}(a)$ giving us an upper bound on the remaining uncertainty as

$$\begin{aligned} H_\infty(H|L) &= \log(t_{\mathsf{post}}) \\ &= \log \textstyle\prod_{a \in \mathcal{O}^{\mathsf{pub}}} \tau_{\mathsf{post}}(a) \\ &= \textstyle\sum_{a \in \mathcal{O}^{\mathsf{pub}}} \log \tau_{\mathsf{post}}(a). \end{aligned} \tag{3}$$

In the above formula, the number of possible transition systems include not only the superfluous (and thus leaked) information, but also the a-priori known information. The a-priori formula for initial uncertainty can be constructed similarly, but using only the information from the projected problem t_a and the projection $\pi^{\triangleright}$ of the solution plan, that is $\mathcal{O}^{\mathsf{plan}} = \{a | a \in \pi^{\triangleright}\}$ resulting in

$$\tau_{\mathsf{apriori}}(a) = \min \begin{cases} t_a & \text{always} \\ t_a^{\mathsf{plan}} & \text{if } a^{\triangleright} \in \pi^{\triangleright}, \end{cases}$$

which for the action a_1 from the previous example, assuming it is not part of the solution plan, is $\tau_{\mathsf{apriori}}(a_1) = \min(t_a) = 7$. The final formula for the leaked information is obtained as $H_\infty(H) - H_\infty(H|L) = \sum_{a \in \mathcal{O}^{\mathsf{pub}}} \log \tau_{\mathsf{apriori}}(a) - \sum_{a \in \mathcal{O}^{\mathsf{pub}}} \log \tau_{\mathsf{post}}(a) =$

$$\sum_{a \in \mathcal{O}^{\mathsf{pub}}} (\log \tau_{\mathsf{apriori}}(a) - \log \tau_{\mathsf{post}}(a)) \tag{4}$$

where clearly, for the actions with no additional information revealed we obtain $\log \tau_{\mathsf{apriori}}(a) - \log \tau_{\mathsf{post}}(a) = 0$ and for actions with leaked information (those which are part of an superfluous plan, or for which the superfluous action applicability has been revealed) we obtain $\log \tau_{\mathsf{apriori}}(a) - \log \tau_{\mathsf{post}}(a) > 0$. Again, for our example action a_1, $\log \tau_{\mathsf{apriori}}(a_1) - \log \tau_{\mathsf{post}}(a_1) = \log 7 - \log 3 \cong 1.2$. Because always $\tau_{\mathsf{apriori}}(a) \geq \tau_{\mathsf{post}}(a)$ as no information can be lost, only obtained, the number of possible transition systems can only decrease. This is important, because it shows that the more actions is revealed by the superfluous plans or superfluous applicability (itself following from superfluous distinct states), the more private information is leaked.

It is also clear, that private actions do not have to be considered, as a private action is never part of a projection of any (partial) plan and no information about its applicability can be learned, thus a private action is never a member of the $\mathcal{O}^{\mathsf{plan}}, \mathcal{O}^{\mathsf{pd}}, \mathcal{O}^{\mathsf{pi}}$ sets and thus does not influence the leaked information measure.

In the above equations, we assume independence of the actions. Obviously, as the sources of information leakage are not exhaustive and the actions are not independent, more information may possibly leak (also by interactions between the sources). This results in possibly lower number of possible transition systems and thus higher information leakage. The results provided here serve as an upper bound on the number of possible transition systems and therefore a lower bound on the information leakage.

Analysis of PP-MAP Algorithms

So far we have shown lower bound on the leakage of privacy based on parameters such as the number of privately-dependent actions observed by the adversary. This measure is largely dependent on the particular problem, planning algorithm and various non-deterministic decisions. In order to compare algorithms, we need to abstract such details. In this section we analyze the worst-case scenarios. To allow such analysis of different planning paradigms, we alter the multi-agent planning problem $\mathcal{M}$ we are solving in the following way. Let $\mathcal{M}^*$ be the problem of finding all solutions of $\mathcal{M}$. From the perspective of a single agent, we are looking for all solutions of Π extensible by all other agents. This modified problem corresponds to the worst-case execution of state-space search algorithms (explore complete search space), partial-order planning algorithms (explore all possible partial plans) and coordination-space search algorithms (explore all possible combinations of local plans).

Minimal Leaked Information

To analyze what private knowledge can be deduced from the solution to $\mathcal{M}^*$, let us first define the solution as a transition system $T^*(\mathcal{M})$, which can be obtained from an intersection of transition systems $T(\Pi_i)$ of all agents, that is $T(\mathcal{M}) = \bigcap_{\alpha_i \in \mathcal{A}} T(\Pi_i)$. Informally, the intersection preserves only nodes and transitions (joining preserved nodes) present in the source transition systems.

From $T(\mathcal{M})$ we obtain $T^*(\mathcal{M})$ by merging all publicly equivalent states s, s' (that is $s[V] = s'[V]$ for all $V \in \mathcal{V}^{\mathsf{pub}}$)

[1] In the case of general projection which does not preserve labels, only the first action of each plan reveals some information.

[2] We omit the numbers for simplicity, but they can be computed for a particular model by enumerating the possible transition systems of an action.

as long as doing so does not introduce any new solution. In other words, the set of paths from s to any goal state is equal to the set of paths from s' to any goal state. This means that only the necessary publicly equivalent states are not merged and thus their difference in private parts revealed. This is clearly the minimal knowledge that is revealed by the solution of $\mathcal{M}^*$ and thus also by the worst case scenario in $\mathcal{M}$. Obviously, it is not tractable to achieve $T^*(\mathcal{M})$ leakage as it would require to solve the $\mathcal{M}^*$ problem first.

The leakage quantification and estimation is applicable to $\mathcal{M}^*$, by defining the a-priori information as all information revealed by $\mathcal{M}^*$, in particular all plans and all privately-dependent and privately-independent actions. The notions of superfluous plans, distinct states and action applicability carry on as well, only with the solution (included in the information which can be revealed) which now consists of $\mathcal{M}^*$ instead of just one solution plan $\pi^\triangleright$.

Multi-Agent Planning Paradigms

Here, we describe the dominating multi-agent planning paradigms. We use a high-level description and present examples of state-of-the-art planners falling into each paradigm.

FS is a forward-chaining (and analogously backward-chaining) state-space search. In the multi-agent version, each state expanded by a public action is sent to all other agents (or just agents with an applicable action). Examples of such planners are MAFS (Nissim and Brafman 2014) and SECURE-MAFS (Brafman 2015). Forward-chaining Partial Order Planning (POP) falls in the same category. In multi-agent POP, the (public projections of) plans are shared in order to coordinate the exploration. Example of such planners is FMAP (Torreño *et al.* 2014).

CS is a coordination-space search, a paradigm specific for multi-agent planning, where agents attempt to agree on a coordination scheme (public projections of local plans) which is then extended by private actions of all agents. Examples of such are the PSM (Tožička *et al.* 2015), COMPLETEPSM (Tožička *et al.* 2014) and GPPP (Maliah *et al.* 2016) planners.

Privacy Leakage of PP-MAP Algorithms

We analyze the leakage of private information of the described planning paradigms and particular planners in the worst-case scenario, that is when solving $\mathcal{M}^*$. Let $\lambda_{\mathbb{A}}(\mathcal{M}^*)$ denote the worst-case information leakage of algorithm $\mathbb{A}$, according to definition of information leakage (Equation 2 and in particular Equation 4).

Forward/Backward State-Space Search The most significant source of leakage in state-space search algorithms is the use of unique IDs representing the private parts of the states, thus distinguishing publicly equivalent states even when it is not necessary[3].

A multi-agent forward search algorithm jointly explores the state-spaces of all agents, s.t. only globally reachable states are explored. A source of superfluous distinct states is that dead-end[4] states are also explored, communicated with other actions and subsequently action applicability is revealed. Analogous situation appears in backward search, where the dead-end states are not explored, but states unreachable from the initial state are. All analyzed algorithms reveal public actions as label-preserving projection, which leaks significant amount of privacy (as showed in previous section) even though it is not necessary.

In the worst-case scenario (that is solving $\mathcal{M}^*$), plain FS (e.g., MAFS) reveals all reachable states and all reachable privately-dependent and privately-independent actions together with all plans. Let $\mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$ denote the set of all privately-dependent and privately-independent actions which are globally reachable in $\mathcal{M}$ from the initial state and are not part of the solution $T^*(\mathcal{M})$ (the actions are applicable to a dead-end state). This means that all actions in $\mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$ reveal their superfluous reachability. There are no superfluous plans in FS. Based on Equation 4, a lower bound on information leakage can be stated as

$$\lambda_{\mathsf{FS}}(\mathcal{M}^*) = \sum_{a \in \mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}} (\log \tau_{\mathsf{apriori}}(a) - \log \tau_{\mathsf{post}}(a)) \qquad (5)$$

where for all other actions $a \notin \mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$, $\log \tau_{\mathsf{apriori}}(a) = \log \tau_{\mathsf{post}}(a)$ and thus the respective sum element results in 0 and can be ignored. Similar bound can be constructed for the BS algorithm, where the set of superfluous actions contains actions applicable in states from which the goal is globally reachable but which are not globally reachable from the initial state.

The SECURE-MAFS algorithm reduces privacy leakage by not communicating a state with equal public and other agents' private parts twice. In general, this approach reduces the number of revealed states and privately-dependent/independent actions, but it does not prevent the exploration of dead-end states. This means that $\mathcal{O}^{\mathsf{sup}}_{\text{SECURE-MAFS}} \subseteq \mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$ and thus $\lambda_{\text{SECURE-MAFS}}(\mathcal{M}^*) \leq \lambda_{\mathsf{FS}}(\mathcal{M}^*)$. There are cases, where the number of revealed actions can be reduced to zero as shown in (Brafman 2015).

Forward-Chaining Plan-Space Search The forward-chaining plan-space search does not have to reveal private state IDs, but instead reveals private ordering constraints. The ordering constraints can be used to deduce at least the same knowledge as can be deduced from the state-space search. Thanks to the forward-chaining property, each public projection of a partial plan encodes a public state, say s. If an action a is applicable in s, eventually, a partial plan containing a will be reached and thus the applicability of a revealed. If s is reachable, but a dead-end state, it forms an superfluous distinct state thus providing excess action applicability information for a. Thus, similarly to MAFS, the forward-chaining principle leads to the exploration of the reachable, but possible dead-end states, resulting in $\lambda_{\mathsf{POP}}(\mathcal{M}^*) = \lambda_{\mathsf{FS}}(\mathcal{M}^*)$ for the forward-chaining variant of POP.

[3] Certainly, it is not necessary to distinguish states which are represented by only one node in $T^*(\mathcal{M})$.

[4] As *dead-end state* is commonly referred to any state from which there is no solution, i.e. no path to any goal state.

Coordination-Space Search Only states and actions which appear in some local plan are explored in the coordination-space search, that is states which are locally reachable and are not local dead-ends. On the contrary, parts of the state-space which are not globally reachable may be explored as well.

In CS, only the necessary publicly equivalent states need to be distinguished, which also results in less privately-dependent/independent superfluous action applicability revealed. The set $\mathcal{O}^{\mathsf{sup}}_{\mathsf{CS}}$ contains all actions in public projections of all local plans (that is projections $\pi_i^{\triangleright}$ of plans π_i solving Π_i) an also all actions with superfluous action applicability revealed. Again, the lower bound on information leakage is analogous to Equation 5 as all actions not in $\mathcal{O}^{\mathsf{sup}}_{\mathsf{CS}}$ leak no information. In general, $\lambda_{\mathsf{CS}}(\mathcal{M}^*)$ and $\lambda_{\mathsf{FS}}(\mathcal{M}^*)$ are not comparable, as the content and relative sizes of $\mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$ and $\mathcal{O}^{\mathsf{sup}}_{\mathsf{CS}}$ are problem-dependent.

Designing a Secure Multi-Agent Planner

Based on the above analysis, we can attempt to improve the existing algorithms to reduce leaked private information when solving $\mathcal{M}^*$. Let us state three rules preventing privacy leakage based on the techniques used in the existing algorithms:

CS-RULE: Before communicating a state s, make sure it is part of a local solution to the agent's problem Π_i.

FS-RULE: Before communicating a state s, make sure it is reachable in $\mathcal{M}$. In the case of plan-space search, before communicating a (partial) plan, make sure it is valid in $\mathcal{M}$.

$\triangleright$-RULE: Do not use label-preserving projection and do not communicate a state with equivalent public and other agent's private parts more than once.

Figure 1 illustrates portions of the state space leaked by application of the CS-RULE, FS-RULE and their intersection. The $\triangleright$-RULE is not shown in the figure as it does not directly influence the search space, but rather make the leaked information less dense. Obviously, the best algorithm would expand and communicate only states of $T^*(\mathcal{M})$, thus resulting in zero leakage, but that would require to check whether a state is part of a global solution (i.e. solving $\mathcal{M}^*$).

We propose a class of algorithms called SECMAP, containing algorithms which follow all three proposed rules when communicating about states, actions and plans. In the rest of this section, we show that SECMAP algorithms leak less private information than all existing algorithms.

Definition 1. An algorithm $\mathbb{A}_1$ leaks less information than algorithm $\mathbb{A}_2$, denoted as $\mathbb{A}_1 \subset \mathbb{A}_2$, if after solving $\mathcal{M}^*$ by both algorithms, $\lambda_{\mathbb{A}_1}(\mathcal{M}^*) < \lambda_{\mathbb{A}_2}(\mathcal{M}^*)$. Analogously, we define $\mathbb{A}_1 \subseteq \mathbb{A}_2$.

The amount of information leaked by a SECMAP algorithm is $\lambda_{\text{SECMAP}}(\mathcal{M}^*)$ and can be estimated as follows. In SECMAP, the only superfluous state is s (and subsequently actions applicable in s) such that s is globally reachable (thanks to the FS-RULE), s is part of a solution of Π_i (thanks to the CS-RULE), but s is not part of a solution of $\mathcal{M}$. In other words, the only leaked information of a SECMAP algorithm is the information about states which are not part of a global solution, thus are not part of $T^*(\mathcal{M})$, but are globally reachable and are part of a local solution of agent i. By application of the $\triangleright$-RULE, even less superfluous states is revealed, similarly to SECURE-MAFS. Let us denote the set of all actions applicable in such states as $\mathcal{O}^{\mathsf{sup}}_{\text{SECMAP}}$ and thus $\lambda_{\text{SECMAP}}(\mathcal{M}^*) = \sum_{a \in \mathcal{O}^{\mathsf{sup}}_{\text{SECMAP}}} (\log \tau_{\mathsf{apriori}}(a) - \log \tau_{\mathsf{post}}(a))$. In the rest section we show that SECMAP leaks at most as much information as any of the existing planning paradigms in general and strictly less information in specific classes of problems.

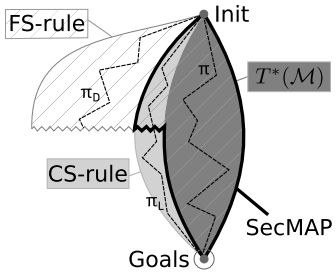


Figure 1: Portions of the state space leaked by application of the CS-RULE, FS-RULE and their combination, where π_D is a sequence of actions leading to a dead-end, π_L is a local plan for Π which cannot be extended to form a global plan for $\mathcal{M}$ and π is a global plan for $\mathcal{M}$.

Lemma 2. *From the combination of the* FS-RULE, CS-RULE *and* $\triangleright$-RULE *follows* $\mathcal{O}^{\mathsf{sup}}_{\text{SECMAP}} = \mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}} \cap \mathcal{O}^{\mathsf{sup}}_{\mathsf{CS}} \cap \mathcal{O}^{\mathsf{sup}}_{\text{SECURE-MAFS}}$.

Proof. For an action $a \in \mathcal{O}^{\mathsf{pub}}_i$ holds $a \in \mathcal{O}^{\mathsf{sup}}_{\text{SECMAP}}$ only if it is applicable in a globally reachable state s, that is $a \in \mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$, a is a part of a local solution to Π_i, that is $a \in \mathcal{O}^{\mathsf{sup}}_{\mathsf{CS}}$ and s' publicly equivalent to s was not already published. □

Now we define two specific classes of MAP problems.

Definition 3. Let $\mathcal{M}$ be a MAP problem. We say that $\mathcal{M}$ is *worst-case* FS*-superfluous* if there exist a state s in $T(\mathcal{M})$ which is globally reachable from s_I, no goal state is locally reachable from s in Π_i for some i and there is an action $a \in \mathcal{O}^{\mathsf{pub}}_i$ applicable in s.

For example, such situation may rise in a factory domain, where a factory consumes too much of a resource, thus making it impossible to reach the goal in the long run, but in FS, many other actions may be explored and published to the adversary before reaching the dead end.

Definition 4. Let $\mathcal{M}$ be a MAP problem. We say that $\mathcal{M}$ is *worst-case* CS*-superfluous* if there exist a state s in $T(\mathcal{M})$ which is part of a local solution of Π_i for some i, s is not globally reachable from s_I and there is an action $a \in \mathcal{O}^{\mathsf{pub}}_i$ for some i applicable in s.

In logistics, an agent α_i may deduce based on its local problem that some other agent α_j is able to deliver a package from A to B and thus α_i explore states and publish actions

based on this assumption. But the assumption may be wrong for some internal reasons of α_j (e.g., not enough fuel or no route between A and B).

Based on the above definitions we can state the following.

Theorem 5. *For a worst-case* FS*-superfluous MAP problem* $\mathcal{M}$, SECMAP $\subset$ SECURE-MAFS $\subseteq$ MAFS.

Proof. SECURE-MAFS $\subseteq$ MAFS has been proven in (Brafman 2015) and it was already shown that $\mathcal{O}^{\mathsf{sup}}_{\text{SECURE-MAFS}} \subseteq \mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$ and thus $\lambda_{\text{SECURE-MAFS}}(\mathcal{M}^*) \leq \lambda_{\mathsf{FS}}(\mathcal{M}^*)$. From Lemma 2 and because $\mathcal{M}$ is worst-case FS-superfluous, there exist an action a such that $a \in \mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$ and $a \notin \mathcal{O}^{\mathsf{sup}}_{\mathsf{CS}}$ and thus $\mathcal{O}^{\mathsf{sup}}_{\text{SECMAP}} \subset \mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$ and $\lambda_{\text{SECMAP}}(\mathcal{M}^*) < \lambda_{\mathsf{FS}}(\mathcal{M}^*)$ and MAFS is an instance of FS. By including the $\rhd$-RULE, we remove the same actions from $\mathcal{O}^{\mathsf{sup}}_{\text{SECMAP}}$ and $\mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$. As the action a was superfluous based on the FS-RULE and does not satisfy the $\rhd$-RULE it is not removed and thus the inequality holds also for SECURE-MAFS, that is $\lambda_{\text{SECMAP}}(\mathcal{M}^*) < \lambda_{\text{SECURE-MAFS}}(\mathcal{M}^*)$. □

Theorem 6. *For a worst-case* CS*-superfluous MAP problem* $\mathcal{M}$, SECMAP $\subset$ PSM.

Proof. From Lemma 2 and because $\mathcal{M}$ is worst-case CS-superfluous, there exist an action a such that $a \notin \mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}}$ and $a \in \mathcal{O}^{\mathsf{sup}}_{\mathsf{CS}}$ and thus $\mathcal{O}^{\mathsf{sup}}_{\text{SECMAP}} \subset \mathcal{O}^{\mathsf{sup}}_{\mathsf{CS}}$ and $\lambda_{\text{SECMAP}}(\mathcal{M}^*) < \lambda_{\mathsf{CS}}(\mathcal{M}^*)$ and PSM is an instance of CS. □

The same results hold for FMAP which is an instance of FS and for GPPP which is an instance of CS. It is obvious, that in general case, a SECMAP algorithm never leaks more information than any of the considered algorithms.

SECMAP Algorithms

The rules defining SECMAP are constructive and thus they can help us modify each of algorithms to fall in the SECMAP class. First, all algorithms need to be modified not to publish the label-preserving projection of public actions. Furthermore, each algorithm has to be extended to comply with all rules of SECMAP.

MAFS, SECURE-MAFS Algorithm 1 (black lines) presents the MAFS algorithm. Let us show how to modify this algorithm to belong to SECMAP (red and black lines).

MAFS already satisfies the FS-RULE as all reached states during the search are globally reachable. The FS-RULE follows form the distributed forward-chaining principle of MAFS which passes through all agents which has to use their private actions to reach further public actions which again passes the process (lines 18 and 3).

To satisfy the CS-RULE in the secure variant SECMAP-MAFS, the agents need to verify that the extracted state s (line 9) is part of some local solution, before sending it to other agents (line 18). Since MAFS assures that s is (globally) reachable, it is enough to check that also the goal is reachable from this state using actions of the agent and public projections of all other actions of all other agents as defined in Π_i. Such check requires to solve new local planning task and if it unsolvable (represented as $\perp$ on line 13) the state s has to be ignored. The lines 12–15

```
1  Algorithm MAFS (M, α_i) and
   SECMAP-MAFS (M, α_i)
2      while TRUE do
3          forall the messages m = ⟨s, ...⟩ in message
           queue do
4              if s is not in open or closed list or ... then
5                  add state from m to open list
6                  ...
7              end
8          end
9          s ← extract-min(open list);
10         move s to closed list
11         check whether s is a goal state
12         π^▷ ← reconstruct-public-plan(s);
13         if local-planning(Π_i, π^▷) = ⊥ then
14             goto line 3
15         end
16         forall the α_j ∈ A do
17             if s is relevant for α_j then
18                 send s to α_j
19             end
20         end
21         expand s
22     end
```

Algorithm 1: MAFS algorithm (shortened to contain only parts relevant to the privacy disclosure) for the agent α_i, see (Nissim and Brafman 2014), and its SECMAP variant when are the red lines included.

describe such process. First, a projected public plan $\pi^{\rhd}$ to the expanded state s is reconstructed using classical recursion over parent actions and states backwards to the initial state (function `reconstruct-public-plan(s)`). Second, the local planning process is run for Π_i (function `local-planning(`Π_i`,` $\pi^{\rhd}$`)`) which searches for a plan solving Π_i and containing actions of $\pi^{\rhd}$ in that specific order. In other words, the planning task is about filling gaps between the initial state, the public actions in $\pi^{\rhd}$ and one of the goal states, such that the potential private preconditions of the public actions in $\pi^{\rhd}$ are satisfied. The $\rhd$-RULE can be satisfied by the same way as in SECURE-MAFS, that is never sending a state s which differs only in the private part of the sending agent.

FMAP We will omit the FMAP algorithm pseudo-code as it is similar to the MAFS but it searches in coordinated way the space of partial plans. Thus, the only differences are that s is a partial plan and that the successors are generated using different expansion function.

PSM The PSM algorithm builds local plans in parallel by all agents . These plans are exchanged among all agents, therefore all agents have (in the end) all agents' local solutions in form of projections . A non-empty intersection of these solutions contains only plans which were local solutions, therefore together they represent global solutions (a public plan $\pi^{\rhd}$ i-extensible by all $\alpha_i \in \mathcal{A}$ is a global solution to $\mathcal{M}$ (Tožička *et al.* 2015)).

PSM already fulfills the CS-RULE, as the agents' sets of local plans Φ are generated considering both the public projection and agent's private part together in Π_i. To satisfy the FS-RULE, the secure variant SECMAP-PSM must not send the whole local solutions π at once , but each agent has to check prefixes of the generated plans in Φ whether they are all globally reachable. Again, the $\rhd$-RULE can be satisfied by never sending a state s which differs only in the agent's own private part from some already sent state s'.

GPPP The Greedy Privacy-Preserving Planner (GPPP) (Maliah *et al.* 2016) fits in the CS paradigm. The algorithm plans centrally in forward-chaining fashion over publicly projected $\Pi^{\rhd}$ and checks the solution by local planning on all agents, filling in the missing private parts of the projected plan $\pi^{\rhd}$ (that is basically checking the i-extensibility of $\pi^{\rhd}$). If all agents can successfully provide local plans for the projected plan $\pi^{\rhd}$the algorithm found a global solution of the problem. Although based on different principle, the CS-RULE is satisfied by GPPP similarly to PSM. In order to get a SECMAP variant of GPPP, the FS-RULE has to be ensured. As in SECMAP-PSM, the check of i-extensibility has to be performed iteratively on the prefix of $\pi^{\rhd}$ starting with the length of 1 and ending with the length of $|\pi^{\rhd}|$. Additionally, the check should never be performed on a state s such that it was already performed on a publicly equivalent s', thus satisfying the $\rhd$-RULE.

Finally, from the above analysis follows:

Theorem 7. *The* SECMAP-MAFS*,* SECMAP-FMAP*,* SECMAP-PSM *and* SECMAP-GPPP *algorithms do not leak more information than* SECMAP.

Proof. All the algorithms satisfy FS-RULE, CS-RULE and $\rhd$-RULE, thus $\mathcal{O}^{\mathsf{sup}}_{\text{SECMAP-MAFS}} \subseteq \mathcal{O}^{\mathsf{sup}}_{\mathsf{FS}} \cap \mathcal{O}^{\mathsf{sup}}_{\mathsf{CS}} \cap \mathcal{O}^{\mathsf{sup}}_{\text{SECURE-MAFS}}$. Then from Lemma 2 $\mathcal{O}^{\mathsf{sup}}_{\text{SECMAP-MAFS}} \subseteq \mathcal{O}^{\mathsf{sup}}_{\text{SECMAP}}$ and analogously for SECMAP-FMAP, SECMAP-PSM and SECMAP-GPPP. □

Conclusions and Future Work

We have proposed a way how to quantify the amount of leaked information during multi-agent planning. The use of this measure is demonstrated on several cases of private information leakage. Furthermore, we have identified which cases of information leakage are presented in the most common multi-agent planning paradigms and a new class SECMAP of privacy preserving algorithms has been proposed. This class is guaranteed to leak less information than any currently known algorithm for a certain classes of problems.

Dominant multi-agent planning algorithms representing different paradigms were analyzed and we proposed how to change them to belong to SECMAP for the price of increased computational complexity as all SECMAP algorithms require another (albeit local) planning process to decrease the need for communicating information which can be used to deduce private parts of the problem. Proposing a practically efficient SECMAP (or at least close to SECMAP) algorithms is left for future work together with experimental comparison against the less privacy preserving algorithms.

Acknowledgments This research was supported by the Czech Science Foundation (grant no. 15-20433Y) and by the Grant Agency of the CTU in Prague (grant no. SGS16/235/OHK3/3T/13).

References

Ronen I. Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS*, pages 28–35, 2008.

Ronen I. Brafman. A privacy preserving algorithm for multi-agent planning and search. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1530–1536, 2015.

Christelle Braun, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Quantitative notions of leakage for one-try attacks. *Electr. Notes Theor. Comput. Sci.*, 249:75–91, 2009.

Shlomi Maliah, Guy Shani, and Roni Stern. Collaborative privacy preserving multi-agent planning. *Autonomous Agents and Multi-Agent Systems*, pages 1–38, 2016.

Raz Nissim and Ronen I. Brafman. Distributed heuristic forward search for multi-agent planning. *J. Artif. Intell. Res. (JAIR)*, 51:293–332, 2014.

Geoffrey Smith. On the foundations of quantitative information flow. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures, FOSSACS , Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS*, pages 288–302, 2009.

Alejandro Torreño, Eva Onaindia, and Oscar Sapena. FMAP: distributed cooperative multi-agent planning. *Appl. Intell.*, 41(2):606–626, 2014.

Jan Tožička, Jan Jakubův, Karel Durkota, Antonín Komenda, and Michal Pěchouček. Multiagent planning supported by plan diversity metrics and landmark actions. In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence, ICAART*, volume 1, pages 178–189, 2014.

Jan Tožička, Jan Jakubův, Antonín Komenda, and Michal Pěchouček. Privacy-concerned multiagent planning. *Knowledge and Information Systems*, pages 1–38 (pre–print), 2015.

Roman Van Der Krogt. Quantifying privacy in multiagent planning. *Multiagent and Grid Systems*, 5(4):451–469, 2009.

Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, SFCS*, pages 160–164, 1982.

Computing Multi-Agent Heuristics Additively

Michal Štolba and **Antonín Komenda**
{stolba,komenda}@agents.fel.cvut.cz
Department of Computer Science, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic

Abstract

Similarly to classical planning, heuristics play a crucial role in most multi-agent and privacy-preserving multi-agent planning systems. It has been shown that distributed heuristics may crucially improve the search guidance, but are costly in terms of communication and computation time and are often a source of privacy concerns. One solution is to compute a heuristic additively, in the sense that each agent can compute its part of the heuristic independently and obtain a complete heuristic estimate by summing up the individual parts. In this preliminary paper, we propose a technique based on cost-partitioning allowing us to use any heuristic in such a way.

Introduction

Modern real-world large-scale personal, corporate or military applications often consist of multiple independent entities. Such entities may need to cooperate in the plan synthesis, while still wanting to protect the privacy of their input data and internal processes. Multi-agent and privacy-preserving multi-agent planning allows the definition of factors of the global planning problem private to the respective entities (i.e. agents) in order to improve the efficiency of planning and/or maintain privacy of the information.

In such privacy-preserving planning systems (Torreño, Onaindia, and Sapena 2014; Nissim and Brafman 2014; Maliah, Shani, and Stern 2014; Tožička, Jakubův, and Komenda 2014), each agent has access only to its part of the global problem, thus can plan only using its operators. The agent can compute a heuristic from its view on the global problem, its projection. Such projection also contains view of other agent's public operators, which allows for a heuristic estimate of the entire problem, but such estimate may be significantly misguided as shown in (Štolba, Fišer, and Komenda 2015). The reason is that the projection does not take into account the parts of the problem private to other agents, moreover in some problems, the optimal heuristic estimate may be arbitrarily lower for the projection than for the global problem.

To obtain a better guidance, a global heuristic estimate can be computed using a distributed process while in some cases still preserving privacy. A number of inadmissible heuristics has been treated this way such as the FF heuristic (Štolba and Komenda 2014), a DTG-based heuristic (Torreño, Onaindia, and Sapena 2014) and a landmark-based heuristic (Maliah, Shani, and Stern 2014). The admissible LM-Cut heuristic (Helmert and Domshlak 2009) is computed in a distributed way in (Štolba, Fišer, and Komenda 2015) and in (Maliah, Shani, and Stern 2015), the authors distribute an admissible pattern database heuristic. The recent class of potential heuristic has been computed distributedly in (Štolba, Fišer, and Komenda 2016).

MAD-A* (Nissim and Brafman 2012) and its secure variant secure-MAFS (Brafman 2015) are the only optimal privacy-preserving multi-agent planners. There is a number of optimal multi-agent planners not concerning privacy (Dimopoulos, Hashmi, and Moraitis 2012; Jezequel and Fabre 2012).

All distributed heuristics published up-to-date present ad-hoc techniques to distribute each particular heuristic. Typically, the distributed computation of heuristic estimate requires cooperation of all (or at least most of) the agents and incurs a substantial amount of communication. In many scenarios, the communication may be very costly (multi-robot systems) or prohibited (military) and even on high-speed networks, communication takes significant time compared to local computation. In such cases it may pay off to use the projected heuristic instead of its better informed counterpart. Most of the referenced heuristics is also missing any formal treatment of privacy, which is for complex algorithms indeed a nontrivial undertaking.

In (Nissim and Brafman 2014), the authors propose an idea of an additive heuristic such that projected estimates of two agents could be added together and still maintain admissibility. In this paper we apply the results of research of additive heuristics, namely the approach of cost-partitioning, to the case of distribution of heuristics for multi-agent planning. This way we obtain a fully general approach allowing us to compute any heuristic additively in a distributed way. Also, it allows us to combine different heuristics, which adheres to the idea of independent agents (that is, each agent can use the heuristic it sees most fit). Last but not least, the presented approach allows us to compute admissible sum of admissible heuristics.

In classical planning, the cost partitioning is typically computed for each state evaluated during the planning process. In PP-MAP, such approach does not make much sense

as we want to keep local as much computation as possible. Thus, the envisioned use of such cost-partitioning is to compute it once at the beginning of the planning process, use the cost-partitioned problems to evaluate heuristics locally and sum the local heuristics to obtain a global estimate.

Since this paper is preliminary, we present mostly theoretical results and leave a proper evaluation for the future work.

Formalism

In this section we present the formalism used throughout the paper. First of all, we define a general (that is single-agent) planning task in the form of Multi-Valued Planning Task (Helmert 2006) (MPT). The MPT is a tuple

$$\Pi = \langle \mathcal{V}, \mathcal{O}, s_I, s_\star, \mathsf{cost} \rangle$$

where $\mathcal{V}$ is a finite set of finite-domain variables, $\mathcal{O}$ is a finite set of operators, s_I is the initial state, $s_\star$ is the goal condition and $\mathsf{cost} : \mathcal{O} \mapsto \mathbb{R}_0^+$ is a cost function. Each V in the finite set of variables $\mathcal{V}$ has a finite domain of values $\mathsf{dom}(V)$. A *fact* $\langle V, v \rangle$ is a pair of a variable V and one of the values v from its domain (i.e. an assignment). Let p be a partial variable assignment over some set of variables $\mathcal{V}$. We use $\mathsf{vars}(p) \subseteq \mathcal{V}$ to denote a subset of $\mathcal{V}$ on which p is defined and $p[V]$ to denote the value of V assigned by p. Alternatively, p can be seen as a set of facts $\{\langle V, p[V] \rangle \,|V \in \mathsf{vars}(p)\}$ corresponding to that partial variable assignment. A complete assignment over $\mathcal{V}$ is a *state* over $\mathcal{V}$. A (partial) assignment p is *consistent* with a (partial) assignment p' iff $p[V] = p'[V]$ for all $V \in \mathsf{vars}(p)$.

An *operator* o from the finite set $\mathcal{O}$ has a precondition $\mathsf{pre}(o)$ and effect $\mathsf{eff}(o)$ which are both partial variable assignments. An operator o is applicable in state s if $\mathsf{pre}(o)$ is consistent with s. Application of operator o in a state s results in state s' such that all variables in $\mathsf{eff}(o)$ are assigned to the values in $\mathsf{eff}(o)$ and all other variables retain the values from s, formally $s' = o \circ s$.

A solution to MPT Π is a sequence (a plan) $\pi = (o_1, ..., o_k)$ of operators from $\mathcal{O}$, such that o_1 is applicable in $s_I = s_0$, for each $1 \leq l \leq k$, o_l is applicable in s_{l-1} and $s_l = o_l \circ s_{l-1}$ and s_k is a goal state (i.e. $s_\star$ is consistent with s_k).

Similarly as MA-STRIPS (Brafman and Domshlak 2008) is an extension of STRIPS (Fikes and Nilsson 1971) towards privacy and multi-agent planning, we now present MA-MPT as a multi-agent extension of the Multi-Valued Planning Task. For n agents, the MA-MPT problem $\mathcal{M} = \{\Pi^i\}_{i=1}^n$ consists of a set of n MPTs. Each MPT for an agent $\alpha_i \in \mathcal{A}$ is a tuple

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

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

All variables in $\mathcal{V}^{\mathsf{pub}}$ and all values in their respective domain are public, that is known to all agents. All variables in $\mathcal{V}^{\mathsf{priv}_i}$ and all values in their respective domains are private to agent α_i which is the only agent aware of such V and allowed to modify its value.

A *global state* is a state over $\mathcal{V}^{\mathsf{G}} = \bigcup_{i \in 1..n} \mathcal{V}^i$. A global state represents the true state of the world, but no agent may be able to observe it as a whole. Instead, each agent works with an *i-projected state* which is a state over $\mathcal{V}^i$ such that all variables in $\mathcal{V}^{\mathsf{G}} \cap \mathcal{V}^i$ are equal in both assignments (the assignments are consistent).

The set $\mathcal{O}^i$ of operators of agent α_i consists of private and public operators such that $\mathcal{O}^{\mathsf{pub}_i} \cap \mathcal{O}^{\mathsf{priv}_i} = \emptyset$. The precondition $\mathsf{pre}(o)$ and effect $\mathsf{eff}(o)$ of private operators $o \in \mathcal{O}^{\mathsf{priv}_i}$, are partial assignments over $\mathcal{V}^{\mathsf{priv}_i}$, whereas in the case of public operators $o \in \mathcal{O}^{\mathsf{pub}_i}$ the assignment is over $\mathcal{V}^i$ and either $\mathsf{pre}(o)$ or $\mathsf{eff}(o)$ assigns a value to at least one public variable from $\mathcal{V}^{\mathsf{pub}}$. Because $\mathcal{V}^{\mathsf{pub}}$ is shared, public operators can influence other agents. A function $\mathsf{cost}^i : \mathcal{O}^i \mapsto \mathbb{R}_0^+$ assigns a cost to each operator of agent α_i. The initial state s_I and the partial goal state $s_\star$ (partial variable assignment over $\mathcal{V}^{\mathsf{G}}$) are in each agent's problem represented only as i-projected (partial) states.

We define a *global problem* (MPT) as a union of the agent problems, that is

$$\Pi^{\mathsf{G}} = \left\langle \bigcup_{i \in 1..n} \mathcal{V}^i, \bigcup_{i \in 1..n} \mathcal{O}^i, s_I, s_\star, \mathsf{cost}^{\mathsf{G}} \right\rangle$$

where $\mathsf{cost}^{\mathsf{G}}$ is a union of the cost functions cost^i. The global problem is the actual problem the agents are solving.

An *i-projected problem* is a complete view of agent α_i on the global problem Π^{G}. The i-projected problem of agent α_i contains i-projections of all operators of all agents. Formally, an i-projection $o^{\triangleright i}$ of $o \in \mathcal{O}^i$ is o. For a public operator $o' \in \mathcal{O}^{\mathsf{pub}_j}$ of some agent α_j s.t. $j \neq i$, an i-projected operator $o'^{\triangleright i}$ is o' with precondition and effect restricted to the variables of $\mathcal{V}^i$, that is $\mathsf{pre}(o'^{\triangleright i})$ is a partial variable assignment over $\mathcal{V}^i$ consistent with $\mathsf{pre}(o')$ ($\mathsf{eff}(o')$ treated analogously). An i-projection of a private operator $o'' \in \mathcal{O}^{\mathsf{priv}_j}$ s.t. $j \neq i$ is $o''^{\triangleright i} = \epsilon$, that is a no-op action with $\mathsf{cost}^{\triangleright i}(o''^{\triangleright i}) = \mathsf{cost}^i(\epsilon) = 0$. The cost of i-projection of $o'' \in \mathcal{O}^{\mathsf{pub}_j}$ is preserved, formally $\mathsf{cost}^{\triangleright i}(o^{\triangleright i}) = \mathsf{cost}^j(o)$.

The set of i-projected operators is

$$\mathcal{O}^{\triangleright i} = \{o^{\triangleright i} | o \in \bigcup_{j \in 1...n} \mathcal{O}^j\}$$

and an i-projected *problem* is

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

The set of all i-projected problems is then $\mathcal{M}^{\triangleright} = \{\Pi^{\triangleright i}\}_{i=1}^n$.

Example

Here we present a small running example with two agents α_1 and α_2. The problem of agent α_1 is Π^1:

$$\begin{aligned}
\mathcal{V}^{\mathsf{pub}} &= \{V_3 \in \{u, g\}\} \\
\mathcal{V}^{\mathsf{priv}_1} &= \{V_1 \in \{i_1, p_1\}\} \\
\mathcal{O}^{\mathsf{pub}_1} &= \{b_1\} \\
\mathcal{O}^{\mathsf{priv}_1} &= \{a_1\} \\
s_I^1 &= V_1 \mapsto i_1, V_3 \mapsto u \\
s_\star^1 &= V_3 \mapsto g
\end{aligned}$$

where the actions a_1, b_1 are:

a	$\mathsf{pre}(a)$	$\mathsf{eff}(a)$	$\mathsf{cost}^1(a)$
a_1	$V_1 \mapsto \mathsf{i}_1$	$V_1 \mapsto \mathsf{p}_1$	$\mathsf{cost}^1(a_1) = 1$
b_1	$V_1 \mapsto \mathsf{p}_1$	$V_1 \mapsto \mathsf{i}_1, V_3 \mapsto g$	$\mathsf{cost}^1(b_1) = 2$

The problem of agent α_2 is Π^2:

$$\begin{aligned}
\mathcal{V}^{\mathsf{pub}} &= \{V_3 \in \{u, g\}\} \\
\mathcal{V}^{\mathsf{priv}_2} &= \{V_2 \in \{i_2, p_2\}\} \\
\mathcal{O}^{\mathsf{pub}_2} &= \{b_2\} \\
\mathcal{O}^{\mathsf{priv}_2} &= \{a_2\} \\
s_I^2 &= V_2 \mapsto i_2, V_3 \mapsto u \\
s_\star^2 &= V_3 \mapsto g
\end{aligned}$$

where the actions are:

a	$\mathsf{pre}(a)$	$\mathsf{eff}(a)$	$\mathsf{cost}^2(a)$
a_2	$V_2 \mapsto \mathsf{i}_2$	$V_2 \mapsto \mathsf{p}_2$	$\mathsf{cost}^2(a_2) = 1$
b_2	$V_2 \mapsto \mathsf{p}_2$	$V_2 \mapsto \mathsf{i}_2, V_3 \mapsto g$	$\mathsf{cost}^2(b_2) = 2$

In addition, the actions of projected problem $\Pi^{\triangleright 1}$ are $\mathcal{O}^{\triangleright 1} = \{a_1^{\triangleright 1}, b_1^{\triangleright 1}, b_2^{\triangleright 1}\}$, where $a_1^{\triangleright 1}, b_1^{\triangleright 1}$ are unchanged and $b_2^{\triangleright 1}$:

a	$\mathsf{pre}(a)$	$\mathsf{eff}(a)$	$\mathsf{cost}^1(a)$
$b_2^{\triangleright 1}$	$\emptyset$	$V_3 \mapsto g$	$\mathsf{cost}^1(b_2^{\triangleright 1}) = 2$

Analogously, the actions of projected problem $\Pi^{\triangleright 2}$ are $\mathcal{O}^{\triangleright 2} = \{a_2^{\triangleright 2}, b_2^{\triangleright 2}, b_1^{\triangleright 2}\}$, where $a_2^{\triangleright 2}, b_2^{\triangleright 2}$ are unchanged and $b_1^{\triangleright 2}$:

a	$\mathsf{pre}(a)$	$\mathsf{eff}(a)$	$\mathsf{cost}^2(a)$
$b_1^{\triangleright 2}$	$\emptyset$	$V_3 \mapsto g$	$\mathsf{cost}^2(b_1^{\triangleright 2}) = 2$

Obviously, a global solution to the problem is either (a_1, b_1) or (a_2, b_2), both of cost 3. The optimal solution of $\Pi^{\triangleright 1}$ is $(b_2^{\triangleright 1})$ with the cost of 2 and symmetrically for $\Pi^{\triangleright 2}$. Thus if we take the baseline approach and maximize the two optimal costs we obtain 2 which is a bound on the value any two admissible heuristics can give as a maximum of projected heuristics.

Abstractions

The set $\mathcal{M}^{\triangleright}$ of all i-projected problems can be seen as a set of abstractions of the global problem Π^{G}. To do so, we first define the transition system of a MPT problem Π.

Definition 1. (Transition system) A transition system of a planning task Π is a tuple $\mathcal{T}(\Pi) = \langle S, L, T, s_I, S_\star \rangle$, where $S = \prod_{V \in \mathcal{V}} \mathsf{dom}(V)$ is a set of states, L is a set of transition labels corresponding to the actions in $\mathcal{O}$ and $T \subseteq S \times L \times S$ is a transition relation of Π s.t. $\langle s, a, s' \rangle \in T$ if $a \in \mathcal{O}$ s.t. a is applicable in s and $s' = a \circ s$. A state-changing transition is $\langle s, a, s' \rangle \in T$ such that $s \neq s'$. The state $s_I \in S$ is the initial state and $S_\star$ is the set of all goal states (that is all states s s.t. $s_\star$ is consistent with s).

Next, we proceed with the definition of an abstraction.

Definition 2. (Abstraction) Let $\mathcal{T} = \langle S, L, T, s_I, S_\star \rangle$ and $\mathcal{T}' = \langle S', L', T', s'_I, S'_\star \rangle$ be transition systems with the same label set $L = L'$ and let $\sigma : S \mapsto S'$. We say that T' is an abstraction of T with abstraction function (mapping) σ if

- $s'_I = \sigma(s_I)$,
- for all $s \in S_\star$ also $\sigma(s) \in S'_\star$, and
- for all $\langle s, a, s' \rangle \in T$, $\langle \sigma(s), a, \sigma(s') \rangle \in T'$.

To conclude this section, we show that an i-projection is an abstraction.

Theorem 3. *(Projection is an abstraction) Let* $\mathcal{T}(\Pi^{\mathsf{G}}) = \langle S^{\mathsf{G}}, \bigcup_{i \in 1..n} \mathcal{O}^i, T^{\mathsf{G}}, s_I, S_\star \rangle$ *be the transition system of the global problem* Π^{G} *and* $\mathcal{T}(\Pi^{\triangleright i}) = \langle S^{\triangleright i}, \mathcal{O}^{\triangleright i}, T^{\triangleright i}, s_I^i, S_\star^i \rangle$ the transition system of the i-projected problem $\Pi^{\triangleright i}$. Then $\mathcal{T}(\Pi^{\triangleright i})$ is an abstraction of $\mathcal{T}(\Pi^{\mathsf{G}})$ with respect to the state-changing transitions.

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

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

The missing loops never influence the shortest path and thus can be ignored (or added at will).

Cost Partitioning

In this section we describe the idea of cost-partitioning (Katz and Domshlak 2010) as used in classical planning and define a novel notion of multi-agent cost-partitioning. We will be talking about non-negative cost-partitioning, where the costs of actions are not allowed to be less than 0, but all notions and techniques generalize to the case of general cost-partitioning without such restriction.

Definition 4. (Cost partitioning). Let Π be a planning task with operators $\mathcal{O}$ and cost function cost. A cost partitioning for Π is a tuple $\mathsf{cp} = \langle \mathsf{cp}_1, ..., \mathsf{cp}_k \rangle$ where $\mathsf{cp}_l : \mathcal{O} \to \mathbb{R}_0^+$ for $1 \leq l \leq k$ and $\sum_{l=1}^k \mathsf{cp}_l(o) \leq \mathsf{cost}(o)$ for all $o \in \mathcal{O}$.

As shown in (Katz and Domshlak 2010), a sum of admissible heuristics computed on the cost-partitioned problem is also admissible, formally

Proposition 5. *(Katz and Domshlak 2010). Let* Π *be a planning task, let* $h_1, ..., h_k$ *be admissible heuristics for* Π*, and let* $\mathsf{cp} = \langle \mathsf{cp}_1, ..., \mathsf{cp}_k \rangle$ *be a cost partitioning for* Π*. Then* $h_{\mathsf{cp}} = \sum_{l=1}^k h_l(s)$ *where each* h_l *is computed with* cp_l *is an admissible heuristic estimate for a state* s*.*

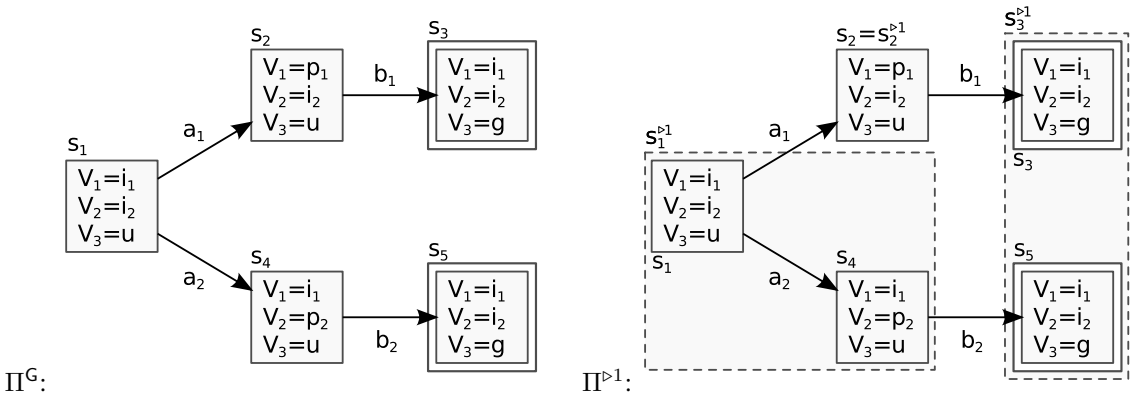


Figure 1: a) Transition system of the global problem Π^{G} respective to the example. b) Example transition system, 1-projection (abstraction).

Based on the particular cost partitioning cp, the heuristic estimate can have varying quality. By optimal cost-partitioning (OCP) we mean a cost-partitioning which maximizes h_{cp}.

Now we proceed with definition of a multi-agent variant of cost-partitioning, which differs in that the partitions are defined a-priori by the set of i-projected problems.

Definition 6. (Multi-agent cost partitioning). Let $\mathcal{M}^{\triangleright} = \{\Pi^{\triangleright i}\}_{i=1}^{n}$ be the set of all i-projected problems with respective cost functions $\mathsf{cost}^{\triangleright i}$. A multi-agent cost-partitioning for $\mathcal{M}^{\triangleright}$ is a tuple of functions $\mathsf{cp} = \langle \mathsf{cp}_1, ..., \mathsf{cp}_n \rangle$ where $\mathsf{cp}_i : \mathcal{O}^{\triangleright i} \to \mathbb{R}_0^+$. For $1 \leq i \leq n$ and for each $o \in \mathcal{O}^{\mathsf{G}}$ holds $\sum_{i=1}^{n} \mathsf{cp}_i(o^{\triangleright i}) \leq \mathsf{cost}^{\triangleright j}(o^{\triangleright j})$ where α_j is the owner of o, that is $o \in \mathcal{O}^j$.

Theorem 7. *Let $\mathcal{M}^{\triangleright} = \{\Pi^{\triangleright i}\}_{i=1}^{n}$ be the set of all i-projected problems, Π^{G} the global problem respective to $\mathcal{M}$ and cp a multi-agent cost-partitioning for $\mathcal{M}^{\triangleright}$. Then cp is a cost-partitioning for Π^{G}.*

Proof. The theorem follows from Definition 4, Definition 6 for all public actions and from setting $o^{\triangleright i} = \epsilon$ for all $o \in \mathcal{O}^{\mathsf{priv}_j}$ s.t. $j \neq i$. As $\mathsf{cost}^{\triangleright i}(o^{\triangleright i}) = \mathsf{cost}^{\triangleright i}(\epsilon) = 0$ and $\mathsf{cost}^{\triangleright j}(o^{\triangleright j}) = \mathsf{cost}^{j}(o)$, the cost-partitioning property $\sum_{i=1}^{n} \mathsf{cp}_i(o^{\triangleright i}) \leq \mathsf{cost}^{j}(o)$ holds also for private operators. □

Thanks to Theorem 7 we can apply the Proposition 5 also in the multi-agent setting using a multi-agent cost-partitioning. Thus, each agent α_i can compute its part of the heuristic locally on $\Pi^{\triangleright i}$ using cp_i instead of cost^i as the cost function. To obtain the global heuristic, the individual parts can be simply summed

$$h_{\mathsf{G}}(s) = \sum_{i=1}^{n} h^{\triangleright i}_{\mathsf{cp}_i}(s^{\triangleright i}) \tag{1}$$

where $h^{\triangleright i}_{\mathsf{cp}_i}$ is an i-projected heuristic computed on $\Pi^{\triangleright i}$ using cp_i. We contrast this approach to the current state of the art, which is taking the maximum, formally

$$h_{\max}(s) = \max_{1 \leq i \leq n} h^{\triangleright i}(s^{\triangleright i}) \tag{2}$$

where $h^{\triangleright i}$ is any (admissible) heuristic computed on $\Pi^{\triangleright i}$ using the original cost^i.

Optimal Cost Partitioning

To compute the optimal cost partitioning (OCP) for i-projections, based on Theorems 3 and 7 we can readily apply the results of optimal cost partitioning for abstractions (Pommerening et al. 2014).

The idea behind the following LP is to encode the abstract transition systems and possible shortest paths in it. The LP variables used for each $\alpha_i \in \mathcal{A}$ are $\bar{h}^{\triangleright i}$ encoding the i-projected heuristic value (given the cost-partitioning), $\bar{s}'^{\triangleright i}$ representing the cost of shortest path from a state s (or actually $s^{\triangleright i}$) to $s'^{\triangleright i}$ in the i-projected problem given the cost partitioning and $\bar{a}^{\triangleright i}$ representing the cost-partitioned cost of action $a^{\triangleright i} \in \mathcal{O}^{\triangleright i}$. The LP is formulated as follows:

Maximize $\sum_{i=1}^{n} \bar{h}^{\triangleright i}$ subject to

$$\begin{array}{rcll}
\bar{s}'^{\triangleright i} & = & 0 & \text{for all } s'^{\triangleright i} = s^{\triangleright i} \\
\bar{s}''^{\triangleright i} & \leq & \bar{s}'^{\triangleright i} + \bar{a}^{\triangleright i} & \text{for all } \langle s'^{\triangleright i}, a^{\triangleright i}, s''^{\triangleright i} \rangle \in T^{\triangleright i} \\
\bar{h}^{\triangleright i} & \leq & \bar{s}'^{\triangleright i} & \text{for all } \bar{s}'^{\triangleright i} \in s^i_\star \\
\sum_{j=1}^{n} \bar{a}^{\triangleright j} & \leq & \mathsf{cost}^i(a) & \text{for all } a \in \mathcal{O}^{\mathsf{pub}_i} \\
\bar{a}^{\triangleright i} & \leq & \mathsf{cost}^i(a) & \text{for all } a \in \mathcal{O}^{\mathsf{priv}_i}
\end{array}$$

where the first set of constraints sets all states equal (in the i-projection) with the current state s to have zero cost of shortest path. The second set of constraints encode the actual (abstracted) transitions and their costs (transitions where $s'^{\triangleright i} = s''^{\triangleright i}$ can be ignored), the third set of constraints places an upper bound on the actual heuristic estimate to keep it admissible. The fourth and fifth sets of constraints represent the cost partitioning of public and private actions respectively. Note, that private actions of agent α_i always occur only as i-projections and are not partitioned (i.e. any other projection of such action has the cost of 0).

Let us show how the optimal cost partitioning is computed on the running example. The global transition system is shown in Figure 1 a) and the transition system projected to agent α_1 in Figure 1 b) (transition system projected to α_2 is symmetrical). The LP is built based on the projected problems as follows:

Maximize $\bar{h}^{\triangleright 1} + \bar{h}^{\triangleright 2}$ subject to

$$\begin{aligned}
\bar{s}_1^{\triangleright 1} &= 0 \\
\bar{s}_2^{\triangleright 1} &\leq \bar{s}_1^{\triangleright 1} + \bar{a}_1^{\triangleright 1} \\
\bar{s}_3^{\triangleright 1} &\leq \bar{s}_2^{\triangleright 1} + \bar{b}_1^{\triangleright 1} \\
\bar{s}_3^{\triangleright 1} &\leq \bar{s}_1^{\triangleright 1} + \bar{b}_2^{\triangleright 1} \\
\bar{h}^{\triangleright 1} &\leq \bar{s}_3^{\triangleright 1} \\
&\ldots \\
\bar{a}_1^{\triangleright 1} &\leq 1 \\
\bar{b}_1^{\triangleright 1} + \bar{b}_1^{\triangleright 2} &\leq 2 \\
&..
\end{aligned}$$

where the omitted parts are defined for agent α_2 analogously. The solution gives $h^{\triangleright 1} + h^{\triangleright 2} = 3$ as the value of the objective function and $\bar{b}_1^{\triangleright 1} = 1, \bar{b}_1^{\triangleright 2} = 1, \bar{b}_2^{\triangleright 1} = 2, \bar{b}_2^{\triangleright 2} = 0$ as the values of (relevant) LP variables. The values directly give the cost partitioning. When applied, the optimal solutions of $\Pi^{\triangleright 1}$ and $\Pi^{\triangleright 2}$ has the cost of 1 and 2 respectively, which is the maximal value so that the sum does not violate admissibility.

In contrast to the use in classical planning, we intend to compute the cost-partitioning LP only once at the beginning of the planning process. Obviously, this results in a possibly sub-optimal cost-partitioning for other states than the initial one, but still should give better informed heuristics than just taking maximum of the projections.

Unfortunately, even computing such OCP once may be intractable in general, as the i-projected problems may be as large and as hard as the global problem e.g., in a scenario where all (or most of) actions and variables are public. Even though typically the projected problems are significantly smaller and thus it is reasonable to experimentally evaluate this approach.

Approximate OCP

Another approach is to approximate the optimal cost-partitioning. The most obvious approach is to compute a smaller abstraction of each of the $\Pi^{\triangleright i}$ and compute the OCP as above on that set of smaller abstractions. Moreover, a cost-partitioning LP formulation is known also for other heuristics, such as LM-Cut and even more heuristics can be expressed as a LP (Pommerening et al. 2014) and modified to compute the cost-partitioning. In the following text, we describe two such examples, a LM-Cut (Helmert and Domshlak 2009) based cost-partitioning and a cost partitioning modification of a State Equation (SEQ) heuristic (Van Den Briel et al. 2007) LP.

Finally we describe a number of ad-hoc cost-partitioning techniques which are very easy to compute (without the use of LP) and still may lead to interesting results.

Landmarks

The LM-Cut heuristic proceeds by computing disjunctive landmarks in the relaxed problem and iteratively reducing their cost.

Definition 8. (Disjunctive Landmark) For a planning task Π, a disjunctive landmark $L = \{a_1, ..., a_k\}$ is a set of actions from $\mathcal{O}$ such that each solution π of Π contains at least one action $a \in L$. The cost of landmark L is defined as $\mathsf{cost}(L) = \min(\mathsf{cost}(a_1), ..., \mathsf{cost}(a_k))$.

Its LP formulation (Pommerening et al. 2014) starts with a set $\mathcal{L}$ of landmarks and assigns a LP variable to the cost of each $L \in \mathcal{L}$ respective to each cost-partitioning. Also, cost of each action respective to each CP is represented by a LP variable. The LP maximizes the sum of all landmark costs subject to

$$\sum_{a \in L} \bar{L} \leq \bar{a}$$

for each a and the cost-partitioning constraints. The LP variables $\bar{a}$ and $\bar{L}$ represent the costs of respective actions and landmarks.

In the running example, there is one disjunctive landmark for agent α_1, that is $L_1^{\triangleright 1} = \{b_1^{\triangleright 1}, b_2^{\triangleright 1}\}$ and symmetrically $L_2^{\triangleright 2} = \{b_1^{\triangleright 2}, b_2^{\triangleright 2}\}$ for α_2. The private actions do not form a disjunctive landmark as the plan $(b_2^{\triangleright 1})$ solves $\Pi^{\triangleright 1}$ without using a_1. The LP is then formulated as follows:

Maximize $\bar{L}_1^{\triangleright 1} + \bar{L}_2^{\triangleright 2}$ subject to

$$\begin{aligned}
\bar{b}_1^{\triangleright 1} &\leq \bar{L}_1^{\triangleright 1} \\
\bar{b}_2^{\triangleright 1} &\leq \bar{L}_1^{\triangleright 1} \\
&\ldots \\
\bar{b}_1^{\triangleright 1} + \bar{b}_1^{\triangleright 2} &\leq 2 \\
\bar{b}_2^{\triangleright 1} + \bar{b}_2^{\triangleright 2} &\leq 2
\end{aligned}$$

where the constraints for $\bar{L}_1^{\triangleright 2}$ are analogous to those for $\bar{L}_2^{\triangleright 1}$. The solution is $\bar{L}_1^{\triangleright 1} = 2, \bar{L}_2^{\triangleright 2} = 0, \bar{b}_1^{\triangleright 1} = 2, \bar{b}_1^{\triangleright 2} = 0, \bar{b}_2^{\triangleright 1} = 2, \bar{b}_2^{\triangleright 2} = 0$, from which is clear that given the resulting CP, the optimal solution for $\Pi^{\triangleright 1}$ is $(b_2^{\triangleright 1})$ with cost 2 and for $\Pi^{\triangleright 2}$ is $(b_1^{\triangleright 2})$ with cost 0. Thus the sum of optimal costs is 2 which is not more than the maximum using the original costs.

It might help to compute the landmarks globally (as in (Štolba, Fišer, and Komenda 2015)). It would make no difference in the example above, but in general, including private and public actions of different agents in a single landmark might improve the quality of the resulting cost-partitioning.

State Equation

State equation heuristic (SEQ) (Van Den Briel et al. 2007) builds on the idea of counting the operators necessary to change the values of variables from the initial state values to the goal state values. It is naturally formulated as a LP (Pommerening et al. 2014) and can be easily modified so that the resulting values can be interpreted as a multi-agent cost-partitioning.

In the original formulation, there is a LP variable for each action, encoding the number of times it has to be used in any optimal plan. There is a constraint for each fact, that is a variable-value pair $\langle V, v \rangle$ for each $v \in V$ and each variable V. In order to formulate the constraint, we need to determine the set $\mathcal{O}^{\mathsf{AP}}$ of actions which always produce the fact (i.e. $\langle V, v \rangle \in \mathsf{eff}(a)$ and $\langle V, v' \rangle \in \mathsf{pre}(a)$ for some $v' \in V$),

a set $\mathcal{O}^{\mathsf{SP}}$ of actions which sometimes produce the fact (i.e. $\langle V, v\rangle \in \mathsf{eff}(a)$ and $V \notin \mathsf{vars}(\mathsf{pre}(a))$) and analogously a set $\mathcal{O}^{\mathsf{AC}}$ of actions which always consume the fact (i.e. $\langle V, v\rangle \in \mathsf{pre}(a)$ and $\langle V, v'\rangle \in \mathsf{eff}(a)$ for some $v' \in V$) and a set $\mathcal{O}^{\mathsf{SC}}$ of actions which sometimes consume the fact (i.e. $\langle V, v\rangle \in \mathsf{pre}(a)$ and $V \notin \mathsf{vars}(\mathsf{eff}(a))$). The constraints for each fact $\langle V, v\rangle$ are

$$\sum_{a\in\mathcal{O}^{\mathsf{AP}}} \bar{a} + \sum_{a'\in\mathcal{O}^{\mathsf{SP}}} \bar{a'} - \sum_{a''\in\mathcal{O}^{\mathsf{AC}}} \bar{a''} \geq L$$
$$\sum_{b\in\mathcal{O}^{\mathsf{AP}}} \bar{b} - \sum_{b'\in\mathcal{O}^{\mathsf{AC}}} \bar{b'} - \sum_{b''\in\mathcal{O}^{\mathsf{SC}}} \bar{b''} \leq U$$

where the bounds L, U are determined based on the initial and goal state. The optimization function of the LP is minimize $\sum_{a\in\mathcal{O}} \mathsf{cost}(a)\bar{a}$, where $\mathcal{O}$ is a set of all actions.

In order to compute a multi-agent CP based on the SEQ LP, we simply express all constraints respective to the i-projected problem $\Pi^{\triangleright i}$ for each agent and add them to a single LP. The optimization criterion is modified so that it minimizes the sum of $\mathsf{cost}(a)\bar{a}$ for all actions and all agents. The computed values of the LP variables then represent how many times each projection of each action has to be used in a solution of each projected problem. That is, for an action $a \in \mathcal{O}^{\mathsf{pub}_j}$ of some agent $\alpha_j \in \mathcal{A}$, we obtain $\bar{a}^{\triangleright 1}, ..., \bar{a}^{\triangleright n}$. Subsequently, the cost partitioning cp_k for agent $\alpha_k \in \mathcal{A}$ can be computed as

$$\mathsf{cp}_k(a^{\triangleright k}) = \mathsf{cost}^j(a)\frac{\bar{a}^{\triangleright k}}{\sum_{i=1}^{n} \bar{a}^{\triangleright i}} \tag{3}$$

that is, based on the ratio of the use of the action projections in the respective projected problems. Of course, if $\sum_{i=1}^{n} \bar{a}^{\triangleright i} = 0$, we need to determine the cost partitioning some other way. The LP for the example problem is formulated as follows:

Minimize $\bar{a}_1^{\triangleright 1} + 2\bar{b}_1^{\triangleright 1} + 2\bar{b}_2^{\triangleright 1} + \bar{a}_2^{\triangleright 2} + 2\bar{b}_1^{\triangleright 2} + 2\bar{b}_2^{\triangleright 2}$ subject to

$$\begin{array}{lcll}
\langle V_1, i_1\rangle & : & \bar{b}_1^{\triangleright 1} - \bar{a}_1^{\triangleright 1} & \geq -1 \\
\langle V_1, i_1\rangle & : & \bar{a}_1^{\triangleright 1} - \bar{b}_1^{\triangleright 1} & \leq 0 \\
\langle V_1, p_1\rangle & : & \bar{a}_1^{\triangleright 1} - \bar{b}_1^{\triangleright 1} & \geq 0 \\
\langle V_1, p_1\rangle & : & \bar{a}_1^{\triangleright 1} - \bar{b}_1^{\triangleright 1} & \leq 1 \\
 & & ... & \\
\langle V_3, g\rangle & : & \bar{b}_1^{\triangleright 1} + \bar{b}_2^{\triangleright 1} & \geq 1 \\
\langle V_3, g\rangle & : & \bar{b}_1^{\triangleright 1} + \bar{b}_2^{\triangleright 1} & \leq 1
\end{array}$$

where the constraints for V_2 are symmetric to the constraints for V_1. The resulting values are $\bar{b}_1^{\triangleright 2} = 1, \bar{b}_2^{\triangleright 1} = 1$ and 0 for all other LP variables. According to Equation 3, the costs of $b_1^{\triangleright 1}, b_2^{\triangleright 1}$ is computed as $\mathsf{cp}_1(b_1^{\triangleright 1}) = 2 \cdot 0/1 = 0$ and $\mathsf{cp}_1(b_2^{\triangleright 1}) = 2 \cdot 1/1 = 2$ respectively and analogously for $b_1^{\triangleright 2}, b_2^{\triangleright 2}$, which gives exactly the same results as the solution based on LM-Cut formulation.

Orthogonal Abstractions

Let us, again, have a closer look on the i-projections as abstractions. What is the reason, that the i-projections cannot be admissibly summed by default? It is the use of the same actions (the public ones) in multiple abstractions. This means, that the abstractions are not orthogonal, formally:

Definition 9. (Orthogonal Abstractions) Let $\mathcal{T}_1, \mathcal{T}_2$ be two abstractions of a planning task Π with transition system $\mathcal{T}$ and let σ_1, σ_2 be their respective abstraction functions. The abstractions $\mathcal{T}_1, \mathcal{T}_2$ are orthogonal if for each transition $\langle s, l, s'\rangle$ in $\mathcal{T}$ holds $\sigma_k(s) = \sigma_k(s')$ for at least one $k \in \{1, 2\}$.

Orthogonal abstractions can be admissibly summed, according to the following proposition.

Proposition 10. *(Helmert, Haslum & Hoffmann 2007) Let $\mathcal{T}_1, ..., \mathcal{T}_k$ be pairwise orthogonal abstractions of the same transition system $\mathcal{T}$ and let $h_1, ..., h_k$ be admissible heuristics computed on the respective transition systems. Then $\sum_{l=1}^{k} h_l$ is an admissible heuristic.*

This means, that in order to be admissibly summed, each action has to be represented by a loop in all but one abstraction. In the i-projected problems, the only problematic actions are projections of public actions, which are counted (non-loop) in each of the projections. A straightforward remedy is to introduce two new types of projection. We will refer to the first one as i-private projection.

Definition 11. (i-private Projection) Let $\Pi^i = \langle \mathcal{V}^i, \mathcal{O}^i, s_I^i, s_\star^i, \mathsf{cost}^i\rangle$ be the problem of an agent α_i. Then the i-private projection of Π^i is $\Pi^{\triangledown i} = \langle \mathcal{V}^{\mathsf{priv}_i}, \mathcal{O}^{\triangledown i}, s_I^{\triangledown i}, s_\star^{\triangledown i}, \mathsf{cost}^{\triangledown i}\rangle$, where $\mathcal{O}^{\triangledown i}$ is the set of i-private projected operators from $\mathcal{O}^{\mathsf{priv}_i}$, $s_I^{\triangledown i}, s_\star^{\triangledown i}$ are the i-private projected initial and goal (partial) states and $\mathsf{cost}^{\triangledown i}$ is cost^i restricted to the operators in $\mathcal{O}^{\triangledown i}$. An i-private projected operator $o^{\triangledown i}$ is o with $\mathsf{pre}(o)$ and $\mathsf{eff}(o)$ restricted to the variables in $\mathcal{V}^{\mathsf{priv}_i}$. An i-private projected (partial) state $s^{\triangledown i}$ is s restricted to the variables in $\mathcal{V}^{\mathsf{priv}_i}$.

Analogously, we define a public projection.

Definition 12. (Public Projection) Let $\Pi^{\mathsf{G}} = \langle \bigcup_{i\in 1..n} \mathcal{V}^i, \bigcup_{i\in 1..n} \mathcal{O}^i, s_I, s_\star, \mathsf{cost}^{\mathsf{G}}\rangle$ be the global problem. Then the public projection of Π^{G} is $\Pi^{\triangleright\mathsf{pub}} = \left\langle \mathcal{V}^{\mathsf{pub}}, \mathcal{O}^{\triangleright\mathsf{pub}}, s_I^{\triangleright\mathsf{pub}}, s_\star^{\triangleright\mathsf{pub}}, \mathsf{cost}^{\triangleright\mathsf{pub}}\right\rangle$, where $\mathcal{O}^{\triangleright\mathsf{pub}}$ is the set of public projection of operators from $\bigcup_{i=1}^{n} \mathcal{O}^{\mathsf{pub}_i}$, $s_I^{\triangleright\mathsf{pub}}, s_\star^{\triangleright\mathsf{pub}}$ are the public projections of initial and goal (partial) states and $\mathsf{cost}^{\triangleright\mathsf{pub}}$ is $\mathsf{cost}^{\mathsf{G}}$ restricted to the operators in $\mathcal{O}^{\triangleright\mathsf{pub}}$. A public projection of operator o is $o^{\triangleright\mathsf{pub}}$ with $\mathsf{pre}(o)$ and $\mathsf{eff}(o)$ restricted to the variables in $\mathcal{V}^{\mathsf{pub}}$. A public projection $s^{\triangleright\mathsf{pub}}$ of a (partial) state s is s restricted to the variables in $\mathcal{V}^{\mathsf{pub}}$.

By computing an admissible heuristic on the public projection and on each of the i-private projections and summing the results, we obtain an admissible heuristic.

Theorem 13. Let $\mathcal{T}(\Pi^{\mathsf{G}})$ be the transition system of the global problem Π^{G}, $\mathcal{T}(\Pi^{\triangledown i})$ the transition system of the i-private projected problem $\Pi^{\triangledown i}$ for each $1 \leq i \leq n$ and $\mathcal{T}(\Pi^{\triangleright\mathsf{pub}})$ the transition system of the public projection $\Pi^{\triangleright\mathsf{pub}}$. Let $h^{\triangledown 1}, ..., h^{\triangledown n}, h^{\triangleright\mathsf{pub}}$ be admissible heuristics computed on the respective projections. Then $h^{\triangleright\mathsf{pub}} + \sum_{i=1}^{n} h^{\triangledown i}$ is an admissible heuristic.

Proof. The public and i-private projections are abstractions by the same reasoning as in Theorem 3. They are (pairwise)

orthogonal from definition and from $\mathcal{O}^i \cap \mathcal{O}^j = \emptyset$ for each $j \neq i$ and $\mathcal{O}^{\mathsf{pub}_i} \cap \mathcal{O}^{\mathsf{priv}_i} = \emptyset$ for each i, thus by application of Proposition 10 the theorem holds. □

A question remains, whether the i-private and public projections can be expressed in the form of cost partitioning of the i-projected problems. An obvious answer is yes, they can. By setting $\mathsf{cp}_i(a^{\triangleright i}) = 0$ for all i-projections of public actions a, the heuristic computed on $\Pi^{\triangleright i}$ using cp_i as a cost function ignores the public actions as if they were self-loops and thus computes the heuristic on $\Pi^{\triangledown i}$. Similar treatment of private actions (i.e. retaining costs only of i-projections of public actions) leads to computation of the heuristic on the public projection $\Pi^{\triangleright \mathsf{pub}}$.

In order to maintain only the n cost-partitioned problems, one of the agents (say α_j) may keep the problem not partitioned, resulting in a heuristic $h^{\triangleright j} + \sum_{i=1, i\neq j}^{n} h^{\triangledown i}$, where $h^{\triangleright j}$ is an admissible heuristic computed on the j-projected problem $\Pi^{\triangleright j}$. In such case, the orthogonality of abstractions still holds and thus the resulting heuristic is also admissible (and possibly more informative).

Let us now apply this approach on the running example. A public projection $\Pi^{\triangleright \mathsf{pub}}$ of the problem reflects only the variable V_3 and actions $b_1^{\triangleright \mathsf{pub}}, b_2^{\triangleright \mathsf{pub}}$. The transition system $\mathcal{T}(\Pi^{\triangleright \mathsf{pub}})$ has two states, an initial state $s_I^{\triangleright \mathsf{pub}}$ where $V_3 = u$ and a goal state $s_\star^{\triangleright \mathsf{pub}}$ where $V_3 = g$. There are two transitions (one for each actions) from $s_I^{\triangleright \mathsf{pub}}$ to $s_\star^{\triangleright \mathsf{pub}}$ with cost 2, thus the cost of optimal solution is $h^{\triangleright \mathsf{pub}}(s_I^{\triangleright \mathsf{pub}}) = 2$. A 1-private projection $\Pi^{\triangledown 1}$ reflects only the variable V_1, thus has two states, which are both goal states (the goal condition is empty). Thus $h^{\triangledown 1} = 0$ and similarly $h^{\triangledown 2} = 0$, resulting in total estimate of $h^{\triangleright \mathsf{pub}} + h^{\triangledown 1} + h^{\triangledown 2} = 2$. Using $h^{\triangleright 1}$ instead of $h^{\triangleright \mathsf{pub}} + h^{\triangledown 1}$ does not help in this particular case as $h^{\triangleright 1} = 2$.

Ad-hoc Cost Partitioning

So far, we have presented a number of more or less involved multi-agent cost-partitioning schema, but more trivial approaches should not be omitted. First is the very baseline uniform cost-partitioning, where

$$\mathsf{cp}_j(a^{\triangleright j}) = \frac{\mathsf{cost}^i(a^{\triangleright i})}{n}$$

for each action $a \in \mathcal{O}^{\mathsf{pub}_i}$ and each agent $\alpha_j \in \mathcal{A}$. Private actions are not partitioned as in the other cases.

Often, the costs of plans using projections of other agent's actions are underestimated as the cost of their private preconditions (that is the cost of private actions achieving them) is not reflected. The aim of presented cost-partitioning techniques is to balance this out. Instead of complex optimization, a simple rule of thumb may work in many cases. We denote such simple approach as projection-compensating cost-partitioning and base it on the following equation

$$\begin{aligned} \mathsf{cp}_j(a^{\triangleright j}) &= \tfrac{1-k}{n-1}\mathsf{cost}^i(a^{\triangleright i}) \quad \text{for } j \neq i \\ \mathsf{cp}_i(a^{\triangleright i}) &= k\mathsf{cost}^i(a^{\triangleright i}) \end{aligned}$$

where $k \in \langle 0, 1\rangle$. For $k = 1/n$, we obtain the uniform cost-partitioning. For $k = 0$, the cost of action $a^{\triangleright i}$ s.t. $a \in \mathcal{O}^i$ in $\Pi^{\triangleright i}$ is 0 and the cost is uniformly distributed among all other agents. For $k = 1$, the cost is retained by the owner agent and the costs of projections are 0. In general, as k is the same for all actions, the OCP cannot be achieved.

On the running example, the uniform CP results in $\mathsf{cp}_1(b_1^{\triangleright 1}) = 1, \mathsf{cp}_1(b_2^{\triangleright 1}) = 1$ and $\mathsf{cp}_2(b_1^{\triangleright 2}) = 1, \mathsf{cp}_2(b_2^{\triangleright 2}) = 1$. The sum of optimal costs computed on such cost-partitioning is 2. We obtain the same result for $k = 0$, where $\mathsf{cp}_1(b_1^{\triangleright 1}) = 0, \mathsf{cp}_1(b_2^{\triangleright 1}) = 2$ and $\mathsf{cp}_2(b_1^{\triangleright 2}) = 2, \mathsf{cp}_2(b_2^{\triangleright 2}) = 0$ and 0 for $k = 1$. In this particular example, we can express the OCP by setting $k = 3/4$, where $\mathsf{cp}_1(b_1^{\triangleright 1}) = 0.5, \mathsf{cp}_1(b_2^{\triangleright 1}) = 1.5$ and $\mathsf{cp}_2(b_1^{\triangleright 2}) = 1.5, \mathsf{cp}_2(b_2^{\triangleright 2}) = 0.5$ and the resulting cost of the sum of optimal solutions is 3.

Privacy

So far, we have avoided the question of privacy which is crucial in privacy-preserving multi-agent planning. In spite of its importance, privacy has been scarcely formally treated in the literature. Apart from an attempt to quantify privacy leakage by (Van Der Krogt 2009) (which is not applicable on heuristic computation as it evaluates possible plans), there are basically two commonly accepted definitions of privacy according to (Nissim and Brafman 2014).

Definition 14. (Weak Privacy) A *weak privacy-preserving* algorithm is a distributed algorithm, such that during its execution no agent $\alpha_i \in \mathcal{A}$ communicates any private part of Π^i, that is any $V \in \mathcal{V}^{\mathsf{priv}_i}$, its value, any $a \in \mathcal{O}^{\mathsf{priv}_i}$ and the actions $a' \in \mathcal{O}^{\mathsf{pub}_i}$ are communicated only in the form of $a'^{\triangleright \mathsf{pub}}$.

The private aspects of Π^i may be deduced from the information communicated.

Definition 15. (Strong Privacy) A *strong privacy-preserving* algorithm is a distributed algorithm, such that after its execution no agent α_j can deduce an isomorphic (that is differing only in renaming) model of a private variable $V \in \mathcal{V}^{\mathsf{priv}_i}$ and its values, a private operator $a \in \mathcal{O}^{\mathsf{priv}_i}$ and its cost or an i-private projection $a'^{\triangledown i}$ of a public operator $a' \in \mathcal{O}^{\mathsf{pub}_i}$ of an agent α_i, beyond what can be deduced from the projected problem $\Pi^{\triangleright j}$ and the output of the algorithm.

It is clear, that the privacy of a heuristic planner depends not only on both the planning algorithm and the heuristic, but also on their interaction. That said, here we focus only on the heuristic itself.

The privacy of the additive computation of cost-partitioned heuristics can be split in two separate issues, computation of the sum of cost-partitioned heuristics and computation of the cost-partitioning itself. In the following sections we briefly analyze both cases.

Privacy of Additive Heuristics

Clearly, any heuristic, such that for agent α_i, $h(s^{\triangleright i}) \neq h(s'^{\triangleright i})$ for states s, s' such that $s^{\triangleright i} = s'^{\triangleright i}$ reveals the information that s and s' are in fact different states, although the same from the perspective of the agent α_i. That is, s, s' differ in at least one variable $V \in \mathcal{V}^{\mathsf{priv}_j}$ for some $j \neq i$, although for $n > 2$, α_i does not know the value of j. It is

not perfectly clear, whether this violates strong privacy, as only the existence of V and $|\mathsf{dom}(V)| \geq 2$ is revealed. It is also clear, that no heuristic which provides the same search guidance can reveal less information.

In the case of a multi-agent cost-partitioning heuristic in Equation 1 where h_{cp_i} is a heuristic computed by α_i on $\Pi^{\triangleright i}$ based on the cost-partitioning cp_i, the situation is a little bit more complex. If the sum is computed plainly, that is each agent α_j provides agent α_i with the value of $h_{\mathsf{cp}_j}(s^{\triangleright j})$, the information revealed is not only that $s \neq s'$, but also that $s^{\triangleright j} \neq s'^{\triangleright j}$ for each j such that $h_{\mathsf{cp}_j}(s^{\triangleright j}) \neq h_{\mathsf{cp}_j}(s'^{\triangleright j})$. Nevertheless, this is the same information revealed by the baseline max-heuristic in Equation 2, where $h^{\triangleright i}$ is an i-projected heuristic computed by α_i on $\Pi^{\triangleright i}$ (without any cost-partitioning).

In addition to that, there exist algorithms for secure sum computation in the literature (Sheikh, Kumar, and Mishra 2010). Moreover, in some situations, it may not be necessary to compute such sum altogether. For example in distributed forward state-space search (e.g. MAD-A* (Nissim and Brafman 2012)), the value of $h(s)$ is sent together with the state. Then if agent α_i wants to expand $s^{\triangleright i}$ with a private action $a \in \mathcal{O}^{\mathsf{priv}_i}$ such that $s' = a \circ s$, because a does not change any part of $\Pi^{\triangleright j}$ for any $j \neq i$, the heuristic of state s' can be computed as

$$h(s') = h(s) - h_{\mathsf{cp}_i}(s^{\triangleright i}) + h_{\mathsf{cp}_i}(s'^{\triangleright i})$$

so that the only information revealed is the same as in the very first case. Of course, this approach cannot be used for public actions, except for the orthogonal abstraction based cost-partitioning and is not practical if $h_{\mathsf{cp}_i}(s^{\triangleright i})$ is computationally intensive.

It is worth noting, that in the MAD-A* algorithm and also its more secure variant Secure-MAFS (Brafman 2015), each state is accompanied with unique state IDs for each agent, thus effectively revealing the information that $s^{\triangleright j} \neq s'^{\triangleright j}$ straight away. In that situation, the plain sum of cost-partitioned heuristics does not reveal any additional information.

Privacy of the Cost-Partitioning Computation

Theorem 16. *(Strongly Private Cost-Partitionings) The computation of Orthogonal Abstraction cost-partitioning and Privacy-Compenstaing cost-partioning are strong privacy-preserving.*

Proof. There is no information exchanged between any agents in computation of either cost-partitioning. □

Regarding the LP-based cost-partitioning computation, there are techniques for secure LP computation such as (Mangasarian 2011; Dreier and Kerschbaum 2011), which were already applied in (Štolba, Fišer, and Komenda 2016) to securely compute LP for a multi-agent version of potential heuristics (Pommerening et al. 2015).

The technique of (Mangasarian 2011) is applicable only on vertically partitioned LPs, which means that each agent has to own a subset of the LP variables. In the case of the OCP, the LP consists of state and heuristic LP variables for each projection thus satisfying the requirement (each projection falls into the partition of the respective agent). The same can be said about the landmark-based and SEQ-based LPs, where the actions and landmarks are represented as projections and thus each variable falls to the respective partition. The constraints can be shared by multiple partitions (agents) in which case the constraint is split according to the variables. This technique does not encrypt the right-hand side vector of the LP, which is not a problem in the OCP and landmark-based case, where the right-hand side is either 0, or the cost of a public action. In the case of the SEQ-based LP, the right-hand side of the LP represents the upper and lower bounds which may potentially leak some information. The technique of (Dreier and Kerschbaum 2011) is applicable in the general case and encrypts the whole LP.

Both techniques use encryption, which has some probability of revealing the actual LP, in (Mangasarian 2011), the probability is not discussed, but in (Dreier and Kerschbaum 2011) it is analyzed and quantified. In PP-MAP, there is no theory which would account for such probabilistic approach to privacy, thus we cannot say any formal conclusion about the usage of such techniques in PP-MAP. Nevertheless, in practice, such approaches should be enough to assure reasonable degree of privacy.

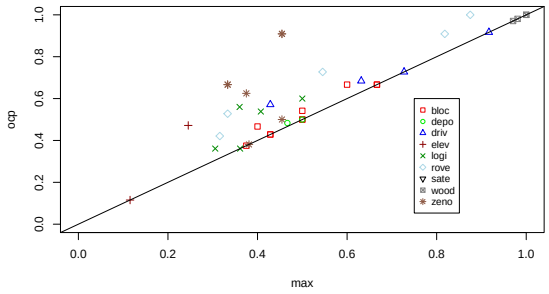


Figure 2: Preliminary comparison on various domains. Let h^* be the cost of global optimal solution, h_i^* the cost of optimal solution of the projected problem of agent α_i and $h_i^{\mathsf{OCP}*}$ the cost of optimal solution of projected problem of agent α_i using the OCP cost. Then the x axis shows $(\max_{i=1}^n h_i^*)/h^*$ and the y axis shows $(\sum_{i=1}^n h_i^{\mathsf{OCP}*})/h^*$.

Conclusions and Future Work

To draw definitive conclusions, we need to perform thorough experimental evaluation of both the cost-partitioning techniques themselves (that is how well do they estimate the global optimal solution) and the practical use of the cost-partitioning and heuristics in a MAD-A*-style planner. The Figure 2 shows preliminary results where the OCP approach provides better estimates for a number of domains and problems. The h_{cp} using optimal cost-partitioning (OCP) always gives better or equal results to $h_{\max}$, as $h_{\max}$ can also be expressed via cost-partitioning, but as already mentioned, computing the OCP is often not tractable. In the example,

all other cost-partitioning techniques resulted in the same heuristic estimate, but in practice, we expect diverse results.

A very promising next step seems to be the use of general cost-partitioning which should allow to compensate more private actions in the costs of the projected operators. On the other hand, use of negative costs would disqualify many heuristics to be used on top of the cost-partitioned problem.

Acknowledgments This research was supported by the Czech Science Foundation (grant no. 15-20433Y) and by the Grant Agency of the CTU in Prague (grant no. SGS16/235/OHK3/3T/13).

References

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, 28–35.

Brafman, R. I. 2015. A privacy preserving algorithm for multi-agent planning and search. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 1530–1536. AAAI Press.

Dimopoulos, Y.; Hashmi, M. A.; and Moraitis, P. 2012. μ-satplan: Multi-agent planning as satisfiability. *Knowledge-Based Systems* 29:54–62.

Dreier, J., and Kerschbaum, F. 2011. Practical privacy-preserving multiparty linear programming based on problem transformation. In *Proceedings of IEEE 3rd International Conference on Privacy, Security, Risk and Trust (PASSAT) and IEEE 3rd Third Inernational Conference on Social Computing (SocialCom)*, 916–924.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI)*, 608–620.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 162–169.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Jezequel, L., and Fabre, E. 2012. A#: A distributed version of a* for factored planning. In *Proceedings of the 51th IEEE Conference on Decision and Control, CDC 2012, December 10-13, 2012, Maui, HI, USA*, 7377–7382.

Katz, M., and Domshlak, C. 2010. Implicit abstraction heuristics. *Journal of Artificial Intelligence Research* 51–126.

Maliah, S.; Shani, G.; and Stern, R. 2014. Privacy preserving landmark detection. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, 597–602.

Maliah, S.; Shani, G.; and Stern, R. 2015. Privacy preserving pattern databases. In *Proceedings of the 3rd Distributed and Multiagent Planning (DMAP) Workshop of ICAPS'15*, 9–17.

Mangasarian, O. L. 2011. Privacy-preserving linear programming. *Optimization Letters* 5(1):165–172.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1265–1266.

Nissim, R., and Brafman, R. 2014. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research* 51:293–332.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-Based heuristics for cost-optimal planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 226–234.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 3335–3341.

Sheikh, R.; Kumar, B.; and Mishra, D. K. 2010. A distributed k-secure sum protocol for secure multi-party computations. *arXiv preprint arXiv:1003.4071*.

Štolba, M., and Komenda, A. 2014. Relaxation heuristics for multiagent planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 298–306.

Štolba, M.; Fišer, D.; and Komenda, A. 2015. Admissible landmark heuristic for multi-agent planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 211–219.

Štolba, M.; Fišer, D.; and Komenda, A. 2016. Potential heuristics for multi-agent planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS)*.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014. FMAP: distributed cooperative multi-agent planning. *Applied Intelligence* 41(2):606–626.

Tožička, J.; Jakubův, J.; and Komenda, A. 2014. Generating multi-agent plans by distributed intersection of finite state machines. In *Proceedings of 21st European Conference on Artificial Intelligence (ECAI)*, 1111–1112.

Van Den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An lp-based heuristic for optimal planning. In *Principles and Practice of Constraint Programming–CP 2007*. Springer Berlin Heidelberg. 651–665.

Van Der Krogt, R. 2009. Quantifying privacy in multiagent planning. *Multiagent and Grid Systems* 5(4):451–469.

Generating Collaborative Behaviour through Plan Recognition and Planning

Christopher Geib
Department of Computer Science
Drexel University
Philadelphia, PA 19104, USA
cgeib@drexel.edu

Bart Craenen
School of Computing Science
Newcastle University
Newcastle NE1 7RU, England, UK
Bart.Craenen@newcastle.ac.uk

Ronald P. A. Petrick
Department of Computer Science
Heriot-Watt University
Edinburgh EH14 4AS, Scotland, UK
R.Petrick@hw.ac.uk

Abstract

This paper presents a framework for integrated plan recognition and automated planning, to produce collaborative behaviour for one agent to help another agent. By observing an "initiator" agent performing a task, the plan recognizer hypothesises how a "supporter" agent could help the initiator by proposing a set of subgoals to be achieved. A lightweight negotiation process mediates between the two agents to produce a mutually agreeable set of goals for the supporter. The goals are passed to a planner which builds an appropriate sequence of actions for satisfying the goals. The approach is demonstrated in a series of experimental scenarios.

Introduction

The ability of an agent to *help* another agent is a desirable attribute when designing artificial entities, such as robots, that must operate together with humans in real-world environments. Indeed, the idea of building assistive agents that must work alongside humans in a cooperative fashion has been a long-standing goal of artificial intelligence and robotics since its earliest days. However, the task of deciding when and how to help another agent can be difficult. Effective helping involves recognising the goals or intentions of other agents, reasoning about opportunities to contribute to existing plans, generating appropriate actions, and potentially communicating such information to the agents involved. In the worst case, identifying an opportunity to help, and generating an appropriate response, may require reasoning over the entire joint space of goals and actions for all the agents.

While the computational cost of reasoning about cooperative action in its most general form may be entirely impractical, constrained forms of reasoning do exist that could be used as the basis for helpful behaviour. For instance, consider the case of two agents setting a table for dinner, where the first agent sets the plates and glasses, and the second agent sets the knives, forks, and spoons. The subgoals pursued by each agent are disjoint but together they contribute to a shared overall goal. Moreover, each action is performed by a single agent, with no action requiring the joint coordination of multiple agents (e.g., two agents lifting a table). Finally, the order of subgoal achievement is independent of the actions of the other agent (e.g., it makes no difference if the knives are placed before the forks or vice versa).

In this paper we consider scenarios of the above form, where one agent, called the *supporter*, must decide how to act to help a second agent, called the *initiator*, achieve its goals. While the supporter is considered to be an artificial agent, no assumption is made about the initiator which may either be a human or artificial agent. In this work, we consider goals which can be decomposed as in the above example, and tasks that consist of independent sequences of actions for each agent. While such conditions may appear to be restrictive, they nevertheless characterise a useful collection of problem scenarios whose solution is far from trivial: the goals of the initiator must be identified and suitable subgoals must be appropriately selected for the supporter to achieve.

To do so, we combine *plan recognition* and *automated planning*, together with a lightweight negotiation process for ensuring that a set of supporter goals is acceptable to both agents. Plan recognition and plan generation are provided by two existing frameworks: the ELEXIR plan recognizer (Geib 2009) and the PKS planner (Petrick and Bacchus 2002; 2004). As such, we focus on the high-level (symbolic) reasoning involved in this task, rather than the low-level processes (e.g., involving continuous models or geometric reasoning) that are part of the design of certain artificial agents like robots. Moreover, the novelty of the approach arises from the particular combination of these two general techniques (i.e., plan recognition and planning), rather than the specific tools used to implement them.

In this approach, the supporter will infer the high-level plans of the initiator and identify possible subgoals that contribute to the initiator's plan. Pairs consisting of the initiator's hypothesised high-level goal, and a candidate subgoal, will then be proposed to the initiator as possible helpful subgoals that the supporter could accomplish. This involves a directed search that first attempts to find the hypothesised goal of the initiator, followed by a search of the remaining subgoals that could be performed by the supporter.

Once negotiation is complete, the agreed upon goals are passed to an automated planner which constructs an independent sequence of actions for the supporter to execute to help the initiator. In particular, no centralised planning or scheduling component is used to enforce collaborative behaviour through joint plans. For example, after observing the initiator place spoons on the table, the supporter might infer that the initiator is setting the table, and that the plates

still need to be set. After confirming with the initiator that setting the plates would help the initiator achieve its goals, the supporter can build and execute a plan for this subgoal. However, if the initiator denies either the hypothesised goal (e.g., the initiator is instead placing the spoons onto the table in order to polish them) or the proposed subgoal (e.g., only bowls need to be set), then an alternative goal/subgoal pair could be proposed to find another way to help the initiator.

The rest of this paper is organised as follows. First, we review the relevant related work. Next, we highlight the main components in our approach, notably the ELEXIR plan recognizer and the PKS planner, and discuss the integration of these systems. We then present the results of our approach tested in three experimental domains. Finally, we discuss the limitations of our approach and highlight future directions.

Related Work

The idea of constructing cooperative agents has been a longstanding area of research (Nwana 1996). Moreover, the task of building artificial agents (especially robots) that can proactively achieve goals has been an active area of study (Schrempf et al. 2005; Pandey, Ali, and Alami 2013), as has the idea of human-robot collaboration (Bauer, Wollherr, and Buss 2008; Chandrasekaran and Conrad 2015). As a result, this work follows a long tradition of prior approaches addressing various aspects of this complex problem.

The general idea of agents that help other agents (including humans) has variously been viewed as a primary property of a plan, or as implicit in multiagent actions. For example, (Pollack 1990; Lochbaum, Grosz, and Sidner 1990) explicitly reason about coordination and helping in the form of shared plans and mutual beliefs. However, establishing agreement of such plans or beliefs has typically relied on shared knowledge which has long been a stumbling block of such theories. Similarly, action representations for multiagent joint actions (i.e., actions that require two or more agents for their execution) (Brafman and Domshlak 2008; Boutilier and Brafman 2001) could be used to model situations where one agent helps another agent. However, these representations do not address the case where helping is not a consequence of such multiagent joint actions.

There has also been significant prior research on a variety of approaches to multiagent planning (e.g., (Nau 2007; Brenner 2003; Brafman and Domshlak 2008; Crosby, Jonsson, and Rovatsos 2014)), along with work on the decentralised solving of constraint optimisation problems (Modi et al. 2003). Approaches have also considered the use of plan recognition (Talamadupula et al. 2014) and intent recognition (Karpas et al. 2015) as a means of coordinating human-robot teams. The idea of ambient intelligence (Augusto 2007) also has connections to the problem of designing systems that proactively aid humans in achieving their goals.

The role of natural language dialogue as an effective means of coordinating actions between a robot and a human has also been previously studied (Fong, Thorpe, and Baur 2003). Moreover, the combination of natural language and goal inference has been explored for the task of selecting actions to contribute to an ongoing task, or for correcting the action of a human already engaged in the task (Foster et al. 2008; Giuliani et al. 2010). Finally, hybrid architectures have been used to integrate diverse components with different representational requirements, particularly when a robot must cooperate with a human (Hawes et al. 2007; Kennedy et al. 2007; Zender et al. 2007).

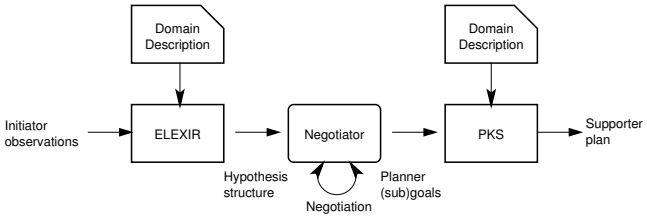


Figure 1: Components and interactions in the framework.

A Framework for Collaborative Behaviour

We now present our approach to collaborative behaviour by describing the main components in our work: the Engine for LEXicalized Intent Recognition (ELEXIR) plan recognizer, the negotiation process, and the Planning with Knowledge and Sensing (PKS) planner. The relationship between these components is shown in Figure 1 and discussed below.

Plan Recognition with ELEXIR

We begin by first distinguishing between work in *activity recognition* (also called *goal recognition* (Liao, Fox, and Kautz 2005; Hoogs and Perera 2008; Blaylock and Allen 2003)) and *plan recognition* in this context. Activity recognition is the creation of a single unstructured label that represents the overarching goal of the activity being observed. For example, such an algorithm would recognize a sequence of pick and place actions of forks, knives, spoons, and plates as an instance of setting the table. This kind of single label is insufficient for our purposes. We need to know the steps in the plan already completed by the initiator, and whether or which future subgoals the supporter can still contribute to.

In contrast, plan recognition attempts to identify not only the goal being pursued by the agent but also the subgoals of the plan that have already been accomplished, and those that are anticipated to be part of the plan in the future. Thus, a plan recognition algorithm is able to produce the complete unexecuted frontier of a hierarchical plan (Kautz 1991; Blaylock and Allen 2003; Geib 2009). For example, following observations of picking and placing forks followed by knives, such a system could identify that the goal was to set the table, the current subgoal was to set the knives, and that in the future, the agent would be setting spoons and plates. These predicted future subgoals are required to effectively reason about possible collaborative contexts.

In this work, we use ELEXIR (Geib 2009) to perform the kind of plan recognition described above. ELEXIR is a probabilistic plan recognition system that views the problem as an instance of parsing a probabilistic grammar. As such, ELEXIR takes as input a formal probabilistic grammar that specifies the set of plans to be recognized and a set of observed actions. ELEXIR represents its plans using Combina-

set-forks :=*SetTable*/{*SetKnives*, *SetSpoons*, *SetPlates*, *SetGlasses*} | (*CleanForks*/{*PutAwayForks*})/{*WashForks*}.
set-knives :=*SetKnives*. **set-spoons** := *SetSpoons*.
set-plates :=*SetPlates*. **store-forks** := *PutAwayForks*.

Figure 2: Portion of a CCG action grammar in ELEXIR.

tory Categorial Grammars (CCGs) (Steedman 2000). While a full discussion of CCGs in ELEXIR is not possible in the space available, we include an example to aid our discussion.

Figure 2 shows a portion of a CCG action grammar that captures a plan for *SetTable* and *CleanForks*. *SetTable*, *SetKnives*, *SetSpoons*, *SetPlates*, *SetGlasses*, *CleanForks*, *PutAwayForks*, and *WashForks* are all symbols that represent goals or subgoals within the plan library. As such, this grammar already encodes some abstraction in the plans. For example, as we will see, a **set-forks** action can be realized by the planner as a sequence of four lower level actions. We assume activity recognition is able to produce observations of the defined high-level actions (e.g., **set-forks**, **set-knives**, **set-spoons**,....) from observations of lower level actions. We could have encoded the grammar at a finer granularity, but this would have added significant unnecessary complexity to the example. Further, because the actions are being executed by the initiator, this would not have eliminated the need for assuming activity recognition of the observed actions. Thus, our example grammar is presented at this abstract level.

That said, the grammar doesn't make commitments about the level of subgoal abstraction. For example, *WashForks* is likely a complex subplan in its own right. Finally, we note that while ELEXIR does support actions with variable arguments, all of our examples use propositional actions, again to simplify the discussion (see (Geib 2009) for more details about ELEXIR's handling of non-propositional actions).

The grammar in Figure 2 specifies that two possible plans can account for an observation of the action **set-forks**: *SetTable* and *CleanForks*. *SetTable* requires that *SetKnives*, *SetSpoons*, *SetPlates*, and *SetGlasses* follow the observed occurrence of **set-forks**, but the subgoals are unordered with respect to each other. *CleanForks* can explain the observed **set-forks**, but only if *WashForks* follows it, and followed by *PutAwayForks*.

Given a set of observed actions, and a formal grammar as above, ELEXIR produces the complete set of hierarchical plan structures that conforms to the grammar and is consistent with the observations, along with a probability for each. These structures represent the hypothesised plans being executed by the agent. Using it we can extract an ordered set of subgoals from each hypothesis that must still be executed for the goal to be achieved, associated with the probability of the hypothesis.

Note that ELEXIR supports both the possibility that a given agent can be pursuing multiple plans as well as the possibility of partially ordered plans. Therefore, for this discussion we will represent a hypothesis produced by ELEXIR as a tuple of the form:

$$(P, [\{G_i : \{\mathrm{sg}_1, ..., \mathrm{sg}_n\}^*\}^+]),$$

where P is the probability of the hypothesis, G_i is the goal of the hypothesised plan, and sg_j the remaining sets of possibly partially ordered subgoals that must be achieved for G_i to be completed. The sg_j within one set of braces are treated as unordered with respect to each other, but all the sg_j within one set must be achieved before those in the next set.

Thus, the three hypotheses from the table setting example (after observing the setting of forks) might be captured as:

$(.95, [\{\textit{SetTable} : \{\textit{SetKnives}, \textit{SetSpoons}, \textit{SetPlates}, \textit{SetGlasses}\}\}]),$
$(.045, [\{\textit{CleanForks} : \{\textit{WashForks}\}\{\textit{PutAwayForks}\}\}]),$
$(.005, [\{\textit{CountingForks} : \{\}\}]).$

The first tuple captures the hypothesis that with 95% probability the agent is following a plan to set the table, and still has the subgoals to set the knives, spoons, and plates. These subgoals are unordered with respect to each other within the plan. The second tuple captures the hypothesis that with 4.5% probability the agent is cleaning the forks and still needs to wash them and put them away, in that order. The third tuple captures the hypothesis that with only 0.5% probability the agent is simply counting the forks and is done with its plan. Thus, each hypothesis provides us with access to the probability of the plans being executed, the goals they are intended to achieve, and the subgoals in the plan that have yet to be achieved. This is precisely the information that we need in order to identify opportunities where the supporter can help the initiator. We discuss how this is done in the next section.

Subgoal Identification and Negotiation

In order to efficiently negotiate collaboration, a supporter must first confirm that it understands the objective of the initiator's high-level plan. Without this confirmation, the supporter might waste significant amounts of time suggesting subgoals that it could achieve, but that may not contribute to the initiator's goal and plan. Using the hypothesis structures from ELEXIR this can be done in a straightforward way.

In the case where a single plan is being pursued by the initiator, sorting the hypotheses by their probabilities ranks the goals of the plan being pursued. This makes it relatively easy for the supporter to verify the initiator's actual plan by a simple query to the initiator.

Having thus identified the goal of the initiator's plan, the supporter can then attempt to identify a future subgoal within those hypotheses that share the identified goal. Returning to our example, when considering the hypotheses for the setting of forks, the first hypothesis is the most likely:

$(.95, [\{\textit{SetTable} : \{\textit{SetKnives}, \textit{SetSpoons}, \textit{SetPlates}, \textit{SetGlasses}\}\}]).$

If the initiator confirms that SetTable is in fact the goal of its plan, the supporter could then suggest that it take on the subgoals of *SetKnives*, *SetSpoons*, *SetPlates*, and *SetGlasses*, or some subset thereof. As we will see in Section , a maximally helpful agent would volunteer to do all of these subgoals. Note, however, that the negotiation process could also result in a number of other outcomes, whereby the supporter agrees to some subset of the subgoals, or none of them at all.

In effect, the process of negotiating collaboration between the initiator and the supporter is then a directed search: first to identify the goal of the initiator's plan, and then to find appropriate subgoals from the set of known unaccomplished subgoals of the plan the supporter has inferred for the goal.

Automated Planning with PKS

Once negotiation is complete and has produced a set of subgoals for helping the initiator, the supporter must generate a concrete sequence of actions to execute in the world. To do so, we use the off-the-shelf PKS planning system.

PKS (Planning with Knowledge and Sensing) (Petrick and Bacchus 2002; 2004) is a contingent planner that builds plans using incomplete information and sensing. PKS operates at the knowledge level (Newell 1982) by reasoning about how the planner's knowledge state changes due to action. PKS is based on a generalisation of STRIPS (Fikes and Nilsson 1971). In PKS, the planner's knowledge state (rather than the world state) is represented by a set of databases, each of which models a particular type of knowledge. The contents of each database have a formal interpretation in a modal logic of knowledge. Actions can modify the databases, which has the effect of updating the planner's knowledge. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) it can represent. The information in PKS's databases can also be incomplete, and PKS does not make a closed world assumption. PKS also supports features like functions and run-time variables that arise in real-world planning scenarios.

Like other planners, a PKS planning domain consists of an initial state, a set of actions, and a set of goals. The initial state is simply the planner's initial knowledge (databases). Goals specify the knowledge conditions that the planner is trying to achieve, formed from the supporter's agreed upon subgoals through a syntactic compilation process which transforms the subgoals into a form understandable by PKS. Actions in PKS are modelled by their preconditions that query the planner's knowledge state, and effects that change the knowledge state by updating particular databases. Plans are constructed by a forward-chaining heuristic search, starting from the initial knowledge state, and continuing until the goal conditions are satisfied or the search fails.

For instance, Figure 3 shows two PKS actions taken from our experimental domains (Section). A precondition `K`(ϕ) queries PKS's knowledge to determine if the planner knows ϕ, while an effect that references `Kf` updates PKS's database of known world facts. Using these actions, a plan such as:

```
grasp(left,drawer,fork1),
putdown(left,table_pos1,fork1),
grasp(left,drawer,fork2),
putdown(left,table_pos2,fork2)
```

might be built in support of a goal to put forks on the table.

Integration and Operation

From a technical point of view, both ELEXIR and PKS are implemented as C++ libraries, with ELEXIR structured into core and recognizer parts. Both libraries expose a user interface through ZeroC's Internet Communication Engine (ICE), a modern distributed computing platform (Henning 2004). This allows both ELEXIR and PKS to be used as standalone servers by a client application implementing the framework, in a traditional client-server architecture.

```
action grasp(?h : hand, ?l : loc, ?o : obj)
   preconds: K(graspable(?o, ?h)) &
             K(objectAt(?o, ?l)) &
             K(holding(?h) = nil)
   effects:  add(Kf, holding(?h) = ?o),
             del(Kf, objectAt(?o, ?l))

action putdown(?h : hand, ?l : loc, ?o : obj)
   preconds: K(holding(?h) = ?o)
   effects:  add(Kf, objectAt(?o, ?l)),
             add(Kf, holding(?h) = nil)
```

Figure 3: PKS actions in the experimental domain.

Figure 1 illustrates the flow of control between the plan recognition, negotiation, and automated planning components in the framework. At system initialisation time, both the plan recognizer and planner are provided with domain-dependent knowledge in the form of their respective domain descriptions. This information is minimally aligned to ensure interoperation between these components (see below).

The process then starts with the supporter observing actions performed by the initiator. These observations are fed into ELEXIR, which produces a set of hypotheses about the initiator's high-level plan, as goal/subgoal pairs, in a hypothesis structure. This structure is then handed over to the negotiation process which mediates the negotiation between the supporter and initiator. Negotiation proceeds by applying directed search to the hypothesis structure to produce a set of goals for the planner. In the final step, PKS uses these goals to attempt to generate a plan to be executed by the supporter.

Experimental Demonstration and Validation

We now present three scenarios, based on the table setting running example, as an experimental demonstration of the proposed framework, integrated as depicted in Figure 1. The underlying domain setting for the three scenarios remains the same: an initiator agent has begun setting a table for a dinner for two people, where each place setting should include a knife, fork, spoon, plate, and glass. The aim of the supporter agent is to help the initiator complete the overall goal of setting the table. Knowledge about the operating environment and the requirements for setting tables is supplied to both the plan recognizer and the planner using appropriate domain descriptions, as detailed earlier.

The observations provided to the ELEXIR plan recognizer remain the same for each scenario: one by one the initiator picks up two forks and two knives and puts them down in their appropriate positions on the table. The scenarios differ in the way these observations are interpreted, and the, way in which differences in the negotiation process can lead to the identification of different subgoals, and how that can affect the resulting plans. The process ends when the planner builds a plan for the supporter to perform, based on the goals identified during the negotiation process.

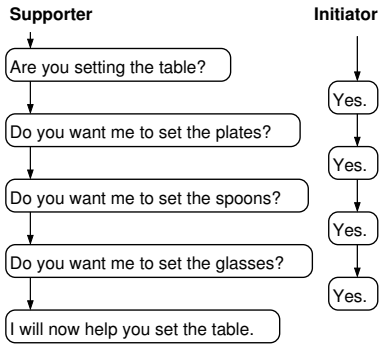


Figure 4: Negotiation in Scenario 1.

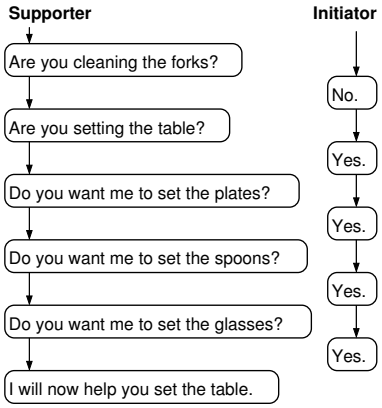


Figure 5: Negotiation in Scenario 2.

For each scenario the correctness of the approach is validated during experimentation. Validation focuses on two points in the process: first, whether the plan recognizer interprets the observation correctly, and, second, whether the planner produces the correct plans. Validation of this form is possible in this case because the example scenarios are designed so that we know, beforehand, what the negotiation process should look like and, as a consequence, how the supporter is supposed to help the initiator set the table.

The computational requirements for all three example scenarios are minimal. Both plan recognition and planning in this domain context takes minimal time, while the computational cost of the negotiation process, excluding the required by the negotiation exchange, is negligible. Total execution time for these, admittedly small-scale, scenarios, on contemporary hardware, takes only seconds. For larger scenarios, and more ambiguous domains, the time required for plan recognition and planning is expected to increase, although ELEXIR and PKS, as well as the negotiation process, scale well. Experience thus far indicates that both scenarios and domains can substantially increase in size before computational costs, and thus execution time, become an issue.

Scenario 1: We begin with the base-case scenario. In this scenario, the plan recognizer correctly identifies the initiator's goal of setting the table, as well as the subgoals the initiator would like the supporter to fulfil. The hypothesis the negotiator examines first is given by:

$$(0.8, [\{\mathit{SetTable} : \{\mathit{SetSpoons}, \mathit{SetPlates}, \mathit{SetGlasses}\}\}]).$$

In this scenario, there is no need for a directed search of the hypothesis structure supplied by ELEXIR. Using this hypothesis, the negotiation then takes the form in Figure 4.

Once completed, the *SetSpoons*, *SetPlates*, and *SetGlasses* subgoals are syntactically translated into PKS goals and the planner attempts to generate a plan. For example, the partial plan for the *SetPlates* subgoal may be:

```
grasp(left,sidetable,plate1),
grasp(right,sidetable,plate2),
putdown(left,table_pos1,plate1),
putdown(right,table_pos2,plate2).
```

(The plans for the other two subgoals will be similar.) Since the hypothesis and the resulting plan(s) are both known beforehand, they can be used to verify that the experimental results match the expected outcome in this scenario.

Scenario 2: This scenario extends the first scenario, and is designed to test the use of directed search to correctly identify the initiator's goal from the hypothesis structure supplied by ELEXIR. In particular, the search focuses on high-level goal identification during negotiation, with the initiator rejecting the hypothesis initially presented by the supporter.

In the first iteration of the negotiation process, the supporter presents the initiator with the following hypothesis:

$$(0.8, [\{\mathit{CleanForks} : \{\mathit{WashForks}\}\{\mathit{PutAwayForks}\}\}]).$$

This hypothesis incorrectly identifies the initiator's goal to be that of cleaning the forks. The initiator rejects this hypothesis, with the supporter moving to the next most probable hypothesis, thus iteratively negotiating with the initiator until the correct goal is found. The number of negotiation iterations can be reduced by adding further reasoning logic about the hypothesis. For simplicity, the next hypothesis correctly identifies the goal of the initiator, so further directed search and negotiation iterations are unnecessary. The correct hypothesis is then the same as the one in Scenario 1:

$$(0.8, [\{\mathit{SetTable} : \{\mathit{SetSpoons}, \mathit{SetPlates}, \mathit{SetGlasses}\}\}]).$$

Negotiation would then take the form as shown in Figure 5.

The remainder of the process then follows the one given in the first scenario: the subgoals are translated for use by PKS; the planner builds plans for setting the plates, spoons, and glasses; and the supporter performs the plan.

This scenario demonstrates that by considering all hypotheses, the framework can recover from an initially incorrect identification of the initiator's goal through the use of a lightweight negotiation strategy and directed search of the hypothesis structure provided by ELEXIR.

Scenario 3: The final scenario we consider is designed to test the framework when dealing with the situation in which the goal of the plan pursued by the initiator is correctly identified, but one (or more) of the hypothesised subgoals is not, and is thus rejected by the initiator. If this happens, the supporter, using directed search of the hypothesis structure, will

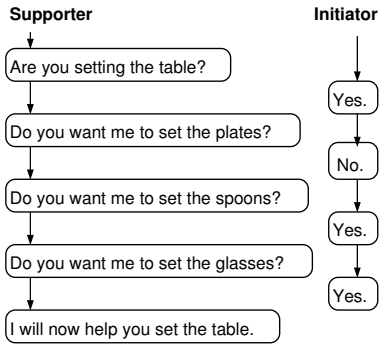


Figure 6: Negotiation in Scenario 3.

iteratively negotiate with the initiator until it finds an acceptable subgoal. It is possible for the supporter to run out of subgoals, if none of the (remaining) subgoals contained in the hypothesis are acceptable to the initiator. If this occurs, the supporter can then revert back to the hypothesis structure to find another hypothesis with the same goal, and continue negotiation with the initiator to see if the (other) subgoals are acceptable. This eventuality is not examined in this scenario due to lack of space. Instead, this scenario considers the same hypothesis as in the first scenario:

$$(0.8, [\{\mathit{SetTable} : \{\mathit{SetSpoons}, \mathit{SetPlates}, \mathit{SetGlasses}\}\}])$$

with negotiation taking the form shown in Figure 6.

The remainder of this process differs from the above scenarios in that the rejected subgoal is not translated and passed to the planner. Instead, only a plan for setting the spoons and glasses is built and performed by the supporter.

This scenario demonstrates that by using ELEXIR's hypothesis structure, the initiator is not limited to accepting all subgoals in a hypothesis: the framework provides enough flexibility for the initiator to decide how, and in which way, he wants to be helped, without the need for elaborate reasoning or goal decomposition on the part of the supporter.

Discussion

The three experimental scenarios demonstrate that our approach successfully generates cooperative plans: for each scenario, ELEXIR interprets the observations correctly, supplying the correct hypothesis structure to the negotiation process; and the negotiator subsequently presents PKS with the expected subgoals, with the planner producing the correct plans. Thus, in each case the framework produces the expected behaviour, thereby validating the process.

However, the framework also relied on certain assumptions concerning the knowledge of the initiator and supporter. For instance, the plan inferred by the supporter is never shared with the initiator, and this approach does not generate plans with joint actions, where multiple agents must coordinate to perform the same task (e.g., lifting a table). Instead, it only generates independent action sequences for the supporter once there is mutual agreement as to the supporter's subgoals. It is also possible that different agents might use different terms to refer to the same objects. If there is sufficient disagreement on such terms, negotiation will simply break down in the face of failed communication. Likewise, a high degree of overlap between the knowledge of the agents, and a tighter correspondence in the names used to identify domain concepts, should give rise to situations where cooperation is more easily negotiated.

In this work, we have also focused on the importance of the supporter being *proactive* in suggesting goals that it could help the initiator with, based on an understanding of the initiator's plan as identified through plan recognition. While an alternative strategy on the part of the supporter may be to simply ask the initiator how it can help, this is not the focus of our approach. For instance, if the initiator is a human, and the supporter is an artificial agent, the human may be forced to respond to a large number of requests (including clarifications) as to what the initiator is doing. Conversely, the approach in the paper could be adapted to scenarios where an initiator may tell a supporter to achieve certain subgoals. In this case, we could simply bypass the plan recognition process and negotiation stages, using goal translation to pass goals directly to the planner. However, both scenarios require knowledge of the terms used by the initiator, which could be much more extensive than the restricted domain descriptions we work with.

During plan execution, there is no direct reasoning of goal changes on the part of the initiator, except as detected through additional plan recognition. Similarly, the adoption by the initiator of a subgoal assigned to the supporter may result in the initiator performing tasks that have already been planned by the supporter. In such a case, we rely on plan execution monitoring and replanning techniques to generate appropriate behaviour to avoid a duplication of tasks.

Another representational problem that must be overcome involves the correspondence between the domain descriptions used by the plan recognizer and the planner. In particular, it is not unusual for a plan recognizer and a planner to have different representations for the same domain, resulting from differences in the underlying representation languages and problems being solved. However, since the plan recognizer and planner must operate within the same reasoning framework, the onus is currently placed on the domain designer to ensure that domains are appropriately engineered to interoperate correctly. One area of future work is to find a common representation that can be used for both tasks, or to automatically induce one representation from the other.

Finally, in this first stage of our work we have placed greater emphasis on the role of plan recognition, compared with that of planning. However, a key direction of future work is to extend our approach to more complex real-world domains, such as those involving incomplete information and uncertainty, where we can take advantage of PKS's ability to build plans with sensing actions (including communicative actions (Petrick and Foster 2013)) to gather information from the world or other agents at execution time. For instance, if in the example scenario the supporter agreed to place wine glasses on the table, then it may first need to query the initiator as to who is drinking wine (and what type of wine) to ensure the table is properly set. One way to do this is by building a contingent plan with information-

gathering actions completely at the planning stage. Thus, plans could be significantly more complex compared to those in the experimental scenarios.

Another potential use of the planner in the next phase of the work is to address the problem of subgoal achievability during the negotiation stage. Currently, the supporter proposes subgoals to the initiator without determining a priori whether those goals are actually achievable by the supporter. Instead, we are exploring the feasibility of trying to generate (partial) plans for particular subgoals at negotiation, in an attempt to limit the supporter's subgoal proposals to achievable subgoals (or subgoals that at least appear likely to be achievable). While it is not expected that this can be done for all subgoals, due to the time that plan generation could take in complex domains, we are nevertheless exploring this approach as a possibility in smaller domains. One additional advantage of such a technique is that in cases where the supporter knows a subgoal is achievable, the supporter could also explain *how* the subgoal could be achieved, by presenting or summarising the plan. Such an approach may also lead to further opportunities for collaborative behaviour between the supporter and initiator as part of such plans.

Conclusion

This paper presented a framework for combining plan recognition and automated planning to produce collaborative behaviour between a pair of agents. Successful integration of the plan recognition and planning components centred around appropriate subgoal identification by the plan recognizer, combined with a lightweight negotiation process which generated goals to be used by the planner for constructing appropriate action sequences. A set of experiments demonstrated the potential of our approach, and helped motivate our ongoing and future work to extend these techniques to more complex real-world situations.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant no. 270273 (XPERIENCE, `xperience.org`) and grant no. 610917 (STAMINA, `stamina-robot.eu`).

References

Augusto, J. C. 2007. *Intelligent Computing Everywhere*. London: Springer. chapter Ambient Intelligence: The Confluence of Ubiquitous/Pervasive Computing and Artificial Intelligence, 213–234.

Bauer, A. M.; Wollherr, D.; and Buss, M. 2008. Human-robot collaboration: a survey. *International Journal of Humanoid Robotics* 5(1):47–66.

Blaylock, N., and Allen, J. 2003. Corpus-based statistical goal recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1303–1308.

Boutilier, C., and Brafman, R. 2001. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research* 14:105–136.

Brafman, R., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 28–35.

Brenner, M. 2003. A Multiagent Planning Language. In *Proceedings of the Workshop on PDDL at ICAPS 2003*.

Chandrasekaran, B., and Conrad, J. M. 2015. Human-robot collaboration: A survey. In *Proceedigs of the IEEE SouthEastCon*, 1–8.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A single-agent approach to multiagent planning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 237–242.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Fong, T.; Thorpe, C.; and Baur, C. 2003. Collaboration, dialogue, and human-robot interaction. In *Robotics Research, Volume 6 of Springer Tracts in Advanced Robotics*. Springer. 255–266.

Foster, M. E.; Giuliani, M.; Müller, T.; Rickert, M.; Knoll, A.; Erlhagen, W.; Bicho, E.; Hipólito, N.; and Louro, L. 2008. Combining goal inference and natural-language dialogue for human-robot joint action. In *ECAI Workshop on Combinations of Intelligent Methods and Applications*.

Geib, C. W. 2009. Delaying commitment in probabilistic plan recognition using combinatory categorial grammars. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1702–1707.

Giuliani, M.; Foster, M. E.; Isard, A.; Matheson, C.; Oberlander, J.; and Knoll, A. 2010. Situated reference in a hybrid human-robot interaction system. In *Proceedings of the International Natural Language Generation Conference (INLG)*, 67–75.

Hawes, N.; Sloman, A.; Wyatt, J.; Zillich, M.; Jacobsson, H.; Kruijff, G.-J. M.; Brenner, M.; Berginc, G.; and Skočaj, D. 2007. Towards an integrated robot with multiple cognitive functions. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1548–1553.

Henning, M. 2004. A new approach to object-oriented middleware. *IEEE Internet Computing* 8(1):66–75.

Hoogs, A., and Perera, A. A. 2008. Video activity recognition in the real world. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1551–1554.

Karpas, E.; Levine, S. J.; Yu, P.; and Williams, B. C. 2015. Robust execution of plans for human-robot teams. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 342–346.

Kautz, H. A. 1991. A formal theory of plan recognition and its implementation. In Allen, J. F.; Kautz, H. A.; Pelavin, R. N.; and Tenenberg, J. D., eds., *Reasoning About Plans*. Morgan Kaufmann. 69–126.

Kennedy, W. G.; Bugajska, M. D.; Marge, M.; Adams, W.; Fransen, B. R.; Perzanowski, D.; Schultz, A. C.; and Trafton, J. G. 2007. Spatial representation and reasoning for human-

robot collaboration. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1554–1559.

Liao, L.; Fox, D.; and Kautz, H. A. 2005. Location-based activity recognition using relational Markov networks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 773–778.

Lochbaum, K.; Grosz, B.; and Sidner, C. 1990. Models of plans to support communication: An initial report. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 485–490.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2003. An asynchronous complete method for distributed constraint optimization. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 161–176.

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

Newell, A. 1982. The Knowledge Level. *Artificial Intelligence* 18(1):87–127.

Nwana, H. S. 1996. Software agents: an overview. *The Knowledge Engineering Review* 11(3):205–244.

Pandey, A. K.; Ali, M.; and Alami, R. 2013. Towards a task-aware proactive sociable robot based on multi-state perspective-taking. *International Journal of Social Robotics* 5(2):215–236.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 212–221.

Petrick, R., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2–11.

Petrick, R., and Foster, M. E. 2013. Planning for social interaction in a robot bartender domain. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 389–397.

Pollack, M. 1990. Plans as complex mental attitudes. In *Intentions in Communication*. MIT Press. 77–103.

Schrempf, O. C.; Hanebeck, U. D.; Schmid, A. J.; and Worn, H. 2005. A novel approach to proactive human-robot cooperation. In *Proceedings of the IEEE International Symposium on Robot and Human-Robot Interactive Communication (RO-MAN)*, 555–560.

Steedman, M. 2000. *The Syntactic Process*. MIT Press.

Talamadupula, K.; Briggs, G.; Chakraborti, T.; Scheutz, M.; and Kambhampati, S. 2014. Coordination in human-robot teams using mental modeling and plan recognition. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2957–2962.

Zender, H.; Jensfelt, P.; Óscar Martínez Mozos; Kruijff, G.-J. M.; and Burgard, W. 2007. An integrated robotic system for spatial understanding and situated interaction in indoor environments. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1584–1589.

Increased Privacy with Reduced Communication and Computation in Multi-Agent Planning

Shlomi Maliah
Information Systems Engineering
Ben Gurion University
shlomima@post.bgu.ac.il

Ronen I. Brafman
Computer Science
Ben Gurion University
brafman@cs.bgu.ac.il

Guy Shani
Information Systems Engineering
Ben Gurion University
shanigu@bgu.ac.il

Abstract

Multi-agent forward search (MAFS) is a state-of-the-art privacy-preserving planning algorithm. We describe a new variant of MAFS, called *multi-agent forward-backward search* (MAFBS) that uses both forward and backward messages to reduce the number of messages and obtain new privacy properties. While MAFS requires agents to send a state s produced by an action a to all agents that can apply any action in s, MAFBS sends such messages forward only to agents that have an action that requires one of the effects of a. To achieve completeness, it sends messages backward to agents that can supply a missing precondition. This more focused message passing scheme reduces states exchanged, and requires that agents be aware only of other agents that they directly interact with, leading to *agent privacy*.

1 Introduction

In various settings, agents may wish to cooperate to achieve joint goals, while concealing certain private facts. For example, different manufacturers may want to collaborate in the production of a good without disclosing their entire supply-chain, inventory levels, and local processes. An attractive framework for such planning problems is privacy preserving planning [Nissim and Brafman, 2014] which has gained increasing attention in recent years. The latest CoDMAP competition [Štolba *et al.*, 2015b] accepted numerous submissions from 10 different groups. While several approaches were suggested, heuristic search algorithms [Maliah *et al.*, 2015; Štolba and Komenda, 2014; Štolba *et al.*, 2015a; Maliah *et al.*, 2014a] seem to be the best performers, at least in the centralized track. In particular, the Multi-Agent Forward Search algorithm (MAFS) [Nissim and Brafman, 2012], combined with strong heuristic estimates [Štolba *et al.*, 2015a] produces high coverage over the competition domain.

The typical notion of privacy used so far in most work on cooperative, privacy preserving planning, is dichotomic: every action and variable is either private to a single agent or public and accessible to all agents. However, as noted by Bonisoli *et al.* [Bonisoli *et al.*, 2014], various facts or actions are naturally described as private to a strict subset of agents. For example, a supplier and a customer must know the content of a package, but the courier that delivers it need not. Moreover, when two organizations interact to achieve joint goals, each employing sub contractors, it may well be that each wishes that the identity, or even the existence, of its sub contractors would not be known to the other. This is called in the DisCSP literature as *agent privacy* [Faltings *et al.*, 2008].

In this paper, we describe a model of refined privacy following [Bonisoli *et al.*, 2014], and a new algorithm, which we call *forward-backward* MAFS (MAFBS), which not only provides this type of (weak) privacy, but also ensures that two agents that do not share a private variable, never communicate with each other, and hence, need not be aware of the existence of each other. Moreover, through the use of focused communication, we not only obtain the above privacy property, but also reduce the number of messages sent.

Our algorithm, while still a forward search algorithm, introduces elements of regression, in the form of backward request messages, allowing for subset privacy and efficiency. Our efficiency gains stem from the use of goal driven expansion. In MAFS, agents send a state s generated by a non-private action to any agent that can apply a public action in s. In MAFBS, when an agent generates a new state using an action that has non-private effects, it will send this state only to agents that have an action that requires one of these effects. Unfortunately, this state sharing scheme is incomplete.

Too see this, consider 3 agents, φ_1,φ_2,φ_3 , and a solution plan a_1, a_2, a_3, with a_i an action of φ_i. a_1, a_2 have no preconditions and generate p_1 and p_2 respectively. a_3 requires both p_1 and p_2 and produces the goal. Upon applying a_1, agent φ_1 sends the resulting state to φ_3, while φ_2 sends the result of applying a_2 to φ_3. However, φ_3 never gets a state where both p_1 and p_2 exist, and cannot execute a_3.

To address this problem we use simple backward reasoning – if agent φ receives a state s because a precondition of an action a of φ was just produced, it sends s backward to all agents that can supply some precondition for φ. In the example above, when φ_3 receives a state s where p_1 holds from φ_1 it will send it to φ_2, which can then apply a_2, achieving p_2 and sending the state back to φ_3. These backward messages ensure completeness.

Thus, the main contribution is a new sound and complete privacy preserving multi-agent planning algorithm that differs from the two main current alternatives: forward search, and

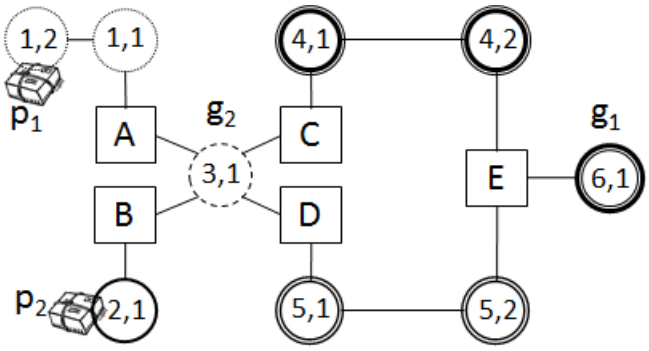


Figure 1: A logistics example.

constraint-based search. The algorithm reduces the message transmission in many domains and provides agent privacy. A secure variant of MAFBS, in the sense of secure-MAFS [Brafman, 2015] can be defined on top of MAFBS.

2 Background

We describe the model of privacy preserving collaborative planning and the MAFS algorithm.

2.1 Privacy Preserving Planning

An MA-STRIPS problem [Brafman and Domshlak, 2013] is represented by a tuple $\langle \Phi, P, \{A_i\}_{i=1}^{|\Phi|}, I, G \rangle$ where:

- Φ is a set of agents.
- P is a finite set of primitive propositions (facts).
- A_i is agent φ_i's action set. $A_i \cap A_j = \emptyset$ if $i \neq j$.
- I is the start state.
- G is the goal condition.

Each action $a = \langle pre(a), \mathit{eff}(a) \rangle$ is defined by its preconditions ($pre(a)$), and effects ($\mathit{eff}(a)$). Preconditions and effects are conjunctions of primitive propositions and literals, respectively. A state is a truth assignment over P. G is a conjunction of facts. $a(s)$ denotes the result of applying action a to state s. A *plan* $\pi = (a_1, \ldots, a_k)$ is a solution to a planning task iff $a_k(\ldots(a_1(I)\ldots) \models G$.

An important assumption we make, for the ease of exposition of the correctness of our algorithm, is that actions are in *transition normal form* [Pommerening and Helmert, 2015]. More specifically, we assume that a primitive proposition appears in a precondition iff it (or its negation) appears in the effect of the action. It is easy to convert any problem into transition normal form.

Privacy-preserving MA-STRIPS extends MA-STRIPS by defining sets of facts and actions as private, known only to a single agent. We extend this, in the spirit of [Bonisoli *et al.*, 2014], allowing a fact to be private to multiple agents.[1] We refer to this as *subset privacy* Thus, for each $p \in P$ we associate a set, $pr(p) \subseteq \Phi$, the set of agents to whom p is private. All agents in $pr(p)$ are immediately aware of changes in the value of p during runtime. We require that for every fact p, if p appears in the description of an action $a \in A_i$, then $\varphi_i \in pr(p)$. That is, an agent is aware of facts that appear as precondition or effect of one of its actions. Similarly, let $pr(\varphi)$ be the set of propositions that φ is aware of. That is, for each $p \in pr(\varphi)$, $\varphi \in pr(p)$, and for each $q \notin pr(\varphi)$, $\varphi \notin pr(q)$.

Compared with the standard definition of multi-agent privacy, the so-called private facts, are now facts private to a single agent only. We support subset-private facts that are not supported in the standard definition. *public* facts in the standard definition are now simply a special case of subset-privacy, where the subset is the entire agent set.

In what follows, when we say that a proposition p is *private to* φ_i we mean that $\varphi_i \in pr(p)$. When we say that a proposition p is private to φ_i *only*, this means that $pr(p) = \{\varphi_i\}$. Finally, when we say that p is *public* we mean that p is not private to a single agent only. Similarly, we refer to an action as *public* when it has some public proposition in its description.

Recently, there is growing awareness of the need to better quantify and improve the privacy guarantees in privacy preserving planning [Brafman, 2015]. A well known privacy property in the area of DisCSP is *agent privacy* [Faltings *et al.*, 2008]. Applying this idea to distributed planning we say that a multi-agent planning algorithm satisfies *agent privacy* if an agent φ_i cannot learn from participation in the algorithm about the existence of an agent φ_j with whom it shares no variable (that is, $pr(\varphi_i) \cap pr(\varphi_j) = \emptyset$). The agents that share at least one variable with φ_i are called the *n*eighbors of φ_i. Thus φ_i is only aware of the existence of its neighbors. As explained above, this seems to be a desirable property in many multi-agent collaboration settings. Moreover, to the best of our knowledge, no current MA planning algorithm satisfies this property.

Figure 1 illustrates a simple logistics example in which the agents are trucks tasked with delivering packages. The set of facts P represents the location of two packages and six trucks. Each truck has three actions: move, load, and unload, corresponding to moving between locations, loading a package and unloading it. Trucks can only drive along the edges in Figure 1. Agents are heterogeneous and their range is restricted, such that location i, j can only be reached using the truck of agent φ_i. The rectangles are logistic centers visited by multiple trucks that load or unload packages.

Truck are owned by different companies that do not want to share their locations and coverage (which locations it can reach) with other companies. Thus, all the facts representing the location of trucks are private, while the facts representing whether a package is at a logistic center are shared among all agents that can reach that logistic center. Only the load/unload actions at the logistic centers are not fully private, whereas the move actions are private for each agent, as well as loading and unloading at private locations.

In the example above, agent privacy requires that agent φ_1 will be unaware of the existence of agents $\varphi_2, \varphi_4, \varphi_5, \varphi_6$. No agent in the above example is aware of all other agents, yet these agents must collaborate to move the packages to their target locations.

[1] Given multi-valued variables, this definition can be further refined, allowing for private variable-value pairs.

2.2 Multi-Agent Forward Search

Multi-Agent Forward Search (MAFS) [Nissim and Brafman, 2014] is a distributed algorithm schema for forward-search planning that also preserves privacy. MAFS was designed following the standard definition of privacy, where facts are either private to a single agent, or public to all agents, and we present it as such.

In MAFS, each agent maintains a separate search space with its own *open* and *closed* lists. The agent expands states using its own actions only. This means that two agents (that have different actions) expanding the same state, will generate *different* successor states. When it generates a state s using a public action of φ_i, then it must send the new state to all other agents that can apply a public action in s. When agent i receives a state s that does not appear in it or in its closed list, it adds s to its open list. For more details, see [Nissim and Brafman, 2014].

Messages sent between agents contain the full state s but the values of private facts in s are encrypted so that only the relevant agent can decipher them. This is typically done by sending an identifier to the private state, rather than encrypting each fact independently. By definition, if q is private to an agent, other agents do not have operators that affect its value, and so they do not need to know or manipulate its value. They simply copy the encrypted value to the next state. As noted by [Bonisoli *et al.*, 2014], by using a slightly more involved encryption scheme, one can also support subset privacy within MAFS.

3 Forward-Backward MAFS

Forward-Backward MAFS (MAFBS) employs a high-level concept similar to MAFS: cooperative state-space search by a group of agents, where each agent expands a state using its own operators only. MAFBS differs from MAFS in its message passing scheme. Whereas MAFS sends a state s generated by a public action to all agents that can apply an action in s, MAFBS sends similar, yet more restricted forward messages, as well as backward messages.

3.1 Forward-Backward Messages

A forward message $a(s)$ is sent to all agents that require a precondition that a produces. Thus, forward messages progress the state forward in a focused manner. The next action of φ_i must use an effect of the last action in the expanded state.

Backward messages use relevance reasoning to identify which agents can supply a missing precondition. In practice, regression is used to identify agents that can help to satisfy these preconditions. A message is sent to an agent that can supply the missing precondition(s), and it may continue to regress them further.

Specifically, if $a(s)$ produces a precondition p of action $a_i \in A_i$, and φ_j has an action a_j that produces another, missing public precondition q of a_i, then φ_i sends to φ_j the state $a(s)$. If φ_j can apply a_j in $a(s)$, it will do so, and return $a_j(a(s))$ to φ_i (and only to φ_i). If φ_j cannot apply a_j in $a(s)$, as a precondition r of a_j is missing, it will send $a(s)$ backward to any agent that can produce r, and so on.

The above schema, requires that forward-backward message passing be applied both between agents *and within agents*. Thus, internally, agents cannot use off-the-shelf forward search algorithms, as they must implement forward-backward search. This is highly inconvenient, and most likely inefficient.

Instead, internally, we allow agents to use standard forward search with their own operators, and adapt the message passing scheme as follows: forward messages are sent following any action that impacts another agent, as described above. We use a more complex mechanism for backward messages. When the current state expanded by φ_i is s, and there is an action $a_i \in A_i$ such that: (1) at least one precondition of a_i achieved by the action that lead to s; (2) at least one public precondition of a_i - p_i does not hold in s. A message is then sent backward to all agents that can supply p_i to φ_i. However, as in the above case, if an action is applied to a state that was received through a backward message, the resulting state is sent only to the agent that sent the backward message.

Note that the agents maintain closed lists. In the case of forwards messages, this the usual closed list. In the case of backwards messages, agent need to check whether they have already expanded that state in a forward fashion in the past, or whether they expanded this state following a backwards message from the same agent. In these cases, they can ignore the message.

In the pseudo-code below, we describe the latter version of MAFBS. We use *forward state* to denote a state received in a forward message and *backward state* to denote a state received in a backward message. We will assume that for every state, we know whether it is a forward or backward state, which agent sent it, and what was the last non-private action in the path that generated each state.

3.2 Goal Detection

Most previous privacy preserving algorithms assume that all goals are public to all agents, or that agents can define artificial public goal actions, and thus report goal states to all other agents. However, by adding artificially shared variables, these schemes violate agent privacy.

Hence, we create a message passing goal identification mechanism. Our goal detection mechanism is similar in nature to consensus with no failures in distributed system literature [Berman *et al.*, 1992]. Agents that identify a local goal notify all their neighbors. Each agent may reject the notification if it knows of some goal that has not been achieved. If all agents confirm the goal, then the search is terminated.

In what follows, we assume that at least one agent has a private goal and that this is common knowledge. It should be straightforward to keep this information private by using a privacy preserving implementation of the *Or* function. We define 4 goal messages — goal proposal, goal confirmation, goal rejection, and termination. When φ_i generates a state in which all sub-goals private to it as well as all public goals are satisfied, it sends a goal-proposal message to all its neighbors.This message contains the (proposed) goal state, and a unique identifier. An agent that receives a goal proposal message with state s checks whether s satisfies its part of the goal. If not, it sends a rejection message to all its neighbors (with

the appropriate state id). Otherwise, it forwards the message to all its neighbors (except the sending agent). Agents that receive the same message twice (identified using the message identifier) ignore it. Agents that receive goal rejection messages forward them to all their neighbors.

If an agent received a goal proposal from all its neighbors, and does not reject it, it sends back a goal confirmation. An agent that receives a goal confirmation message from all its neighbors (except for the one that sent it the goal proposal), and does not reject it, forwards the confirmation to all its neighbors.

Algorithm 1: MAFBS for Agent φ_i

```
1  MAFBS(i)
2      Insert I into open list
3      while TRUE do
4          foreach state s in message queue do
5              process-message(s)
6          s ← extract-min(open list)
7          expand(s)
8  process-message(s)
9      if s is a forward state not in the closed-list or
       (s, j), where j is the sending agent is a backward
       state not in the backwards closed-list then
10         add s to open list
11 expand(s)
12     if s is a goal state then
13         traceback solution and terminate
14     if s is a forward state then
15         move s to closed list;
16         foreach a_i ∈ A_i s.t. pre(a_i) ⊨ s, and a_i(s)
           not in the closed list do
17             add a_i(s) to open;
18             foreach agent φ_j (j ≠ i) that has an action
               that requires an effect of a_i do
19                 send forward a_i(s) to φ_j
20         let a be the last public action performed on the
           path to s;
21         foreach a_i ∈ A_i that (1) consumes an effect of
           a, and (2) has a public precondition p that does
           not hold in s do
22             send s backward to all agents φ_j (j ≠ i)
               that have an action that produces p
23     if s is a backward state then
24         Let φ_j be the sending agent;
25         move (s, j) to the backwards closed list;
26         foreach a_i ∈ A_i applicable in s that satisfies a
           precondition of some a_j ∈ A_j do
27             send a_i(s) forward to φ_j;
28             add a_i(s) to open as a backward state;
29         foreach φ_k that can supply some precondition
           to φ_i do
30             send s backward to φ_k;
```

When the agent that initiated the goal proposal receives a goal confirmation, it sends to all its neighbors a termination message. An agent that receives a termination message forward it to all its neighbors, and terminates the search. Agents that receive the termination message can extract their personal plan from the state.

As with the forward backward messages, when agents forward the goal state to others, they adjust the state to expose only the relevant parts to the receiving agent. Every agent that receives one of the messages above (except for termination), extracts the state and adds it to its open list.

4 MAFBS Properties

We now discuss the key properties of MAFBS: soundness, completeness, and agent privacy.

Claim 1. MAFBS *is sound.*

Each state generated in MAFBS is obtained by applying an action to a state that was previously generated, starting at the initial state. Thus, all states are reachable, and if a goal state is found, there must be a plan.

A key technical result of this section is the completeness of MAFBS.

Theorem 1. MAFBS *is complete.*

Proof. We prove that if a solution plan π exists, then MAFBS will find it. We assume that π includes no action that is private to a single agent only. We later remove this assumption. We also assume that π is minimal in the following sense: we cannot remove some subsequence of actions from π while maintaining the current order of the remaining actions and end up with a valid plan.

The causal structure of π [Karpas and Domshlak, 2012] is a graph defined as follows: The nodes in this graph are the actions of π, and a is a parent of a' if a supplies a precondition p to a' in the plan. That is, a appears before a' in π, and no action between a and a' produces p. In other words, there is a causal link between a and a' [Tate, 1977].

Our first step is to transform π as follows: starting with $i = 1$, we consider the actions between a_i and a_j, the first action to consume an effect of a_i. By minimality, a_j must exist, or a_i is a goal achieving action and is not the producer in any causal link. To address this latter case, just for the sake of the transformation described, we add an END action that consumes the goal literals and must appear last in the plan. For every a_k ($i < k < j$) that is not an ancestor of a_j in the dependency graph, we try to move a_k forward as much as possible by exchanging it with the actions that follows it without invalidating the plan. If possible, we move a_k beyond a_j, at which point we stop moving a_k (in this iteration). We perform this backward, starting at $k = j - 1$ and ending at $k = i + 1$. Once we are done with the current i, we increase i to the (possibly new) index of a_j – the first action to consume an effect of a_i – plus one. Clearly, this is a finite process. Let π now denote the plan following this transformation. By construction, π is a valid plan. We call plans with this property, *goal focused plans*. Notice that any action that can be moved beyond the END action has no role in the plan, and its existence would thus contradict our minimality assumption.

	Messages		Time		States($\times$1000)	
	MAFS	MAFBS	MAFS	MAFBS	MAFS	MAFBS
MALogistics	2086	**582**	15.7	**14**	579.5	**514.8**
MABlocksWorld	129.5	**84.3**	294.9	**292.1**	**272.3**	281.1
Driverlog	622.5	**116.5**	2.7	**0.7**	197.1	**36.8**
Elevators08	519.9	**361**	31.9	**11.9**	845.4	**312.5**
Logistics00	902.5	**361.4**	1.8	**1.6**	50.6	**44.6**
Rovers	676.7	**182.7**	23.6	**16.8**	330.8	**182.9**
Satellites	2174	**69.8**	24.9	**7**	1482.3	**359.3**
Zenotravel	117	**52.1**	6.1	**5.1**	302.4	**221.8**
Depot	1668	**799**	1.3	**0.8**	72.3	**56.6**
Woodworking08	2040	**531.6**	7.7	**2.1**	242.6	**64.3**

Table 1: Comparing the performance of MAFBSand MAFS. For each domain we report the average over all problems.

Claim 2. *A goal-focused plan π satisfies the following property: for every action a_i and a_j such that a_j is the first consumer of an effect of a_i, the actions between a_i and a_j contain only ancestors of a_j in the dependency graph of π.*

Proof. First, we make the following observation. If a_{k-1} cannot be exchanged with a_k without invalidating the plan, then a_{k-1} supplies a_k with a precondition. To see this, suppose to the contrary that a_{k-1} does not supply a_k with some precondition. Then, we cannot exchange them because either a_k destroys a precondition of a_{k-1} or a_{k-1} destroys an effect of a_k required by some future action.

Recall that our actions are in transition normal form. Thus, since in both of the above cases some primitive proposition p appears in the description of both actions, it must appear in their preconditions and effects as well. In that case, the value of p following a_{k-1} must equal its value in the precondition of a_k for π to be applicable. Thus, a_{k-1} supplies a precondition to a_k, and hence, it is one of its parents in the dependency graph.

Thus, proceeding backward from a_j, we see that a_{j-1} must be an ancestor of a_j. Now, suppose that $a_{k+1}, \ldots, a_{j-1}$ are all ancestors of a_j. Consider a_k. If a_k cannot be exchanged with a_{k+1}, then it is a parent of a_{k+1}, and by transitivity, an ancestor of a_j. Otherwise, pre-transformation, a_k supplied some precondition to a_j or one of its ancestors. However, it could be the case that some action a that was before a_k originally, was moved forward beyond a_k, and now supplies this same precondition instead of a_k. This means that both supply some condition p. But since actions are in transition normal form, a must also have p as a precondition to appear after a_k, and hence a_k is an ancestor of a in π now, and thus an ancestor of a_j. If a moved forward beyond another action that supplies p, the same argument can be applied repeatedly. □

Thus, considering an action a_i, we see that all the actions between it and a_j, the first action to consume an effect of a_i must be ancestors of a_j in the dependency graph. Not only that, it is clear that, to preserve plan validity, if some a_k for $i < k < j$ is an ancestor of a_j, then all the ancestors of a_j that are descendants of a_k have to appear between a_j and a_k for the plan to be valid.

Claim 3. *π can be generated by* MAFBS.

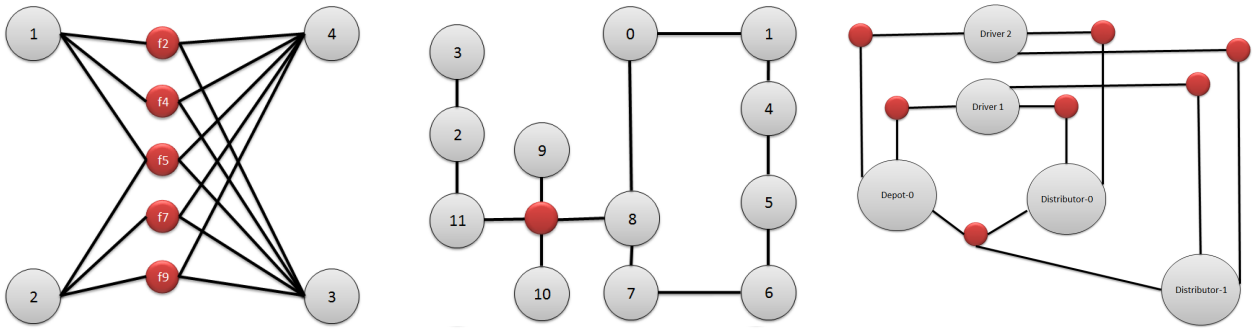
Proof. We now explain how MAFBS will generate π in goal-focused form. Let a_j be the first action that consumes a precondition of a_1. MAFBS sends $a_1(I)$ to the agent that owns a_j (Line 18). This agent will send $a_1(s)$ to all its parents in the dependency graph (Line 21), who will continue to send it to their parents, etc. (Line 28). As we observed, the entire line of ancestors between any two nodes must be present. This continues until one of the earliest ancestors (one that does not have another ancestor between a_1 and a_j) will receive it. Amongst them must be a_2, who will apply its action. Now, there must be an action that consumes the effect of a_2 between a_3 and a_j, as it is an ancestor of a_j and the plan is minimal. The process now continues up and down the tree until a_3 can apply its action and so on. Eventually, we reach the state $a_j(\cdots(a_1(I)))$ at which point we continue in the same manner. □

We assumed so far, that all actions in π are public. To see that the proof holds when private actions are allowed too, notice that within the search space of the agent, full forward search is used. The only caveat is that it is possible that an agent will receive a state s from another agent in which it cannot apply any non-private action. That is, it must first apply a few private actions, and only then does some precondition for a non-private action becomes satisfied. At this point, it may need another precondition from a different agent in order to apply its non-private action. This is handled in lines 20-22, where we allow backward messages not only after a forward message was received. □

An important, technically straightforward property is:

Theorem 2. MAFBS *with a heuristic whose computation preserves agent privacy preserves agent privacy.*

Proof. In MAFBS agents communicate only with agents that share a private variable with them. The information they get from other agents includes the value of variables that are private to them, with the rest of the state encrypted. Assuming the encryption scheme is secure, they cannot deduce the existence of any other agent, as the state of agents that are not their neighbors is concealed in the encrypted state, and they cannot differentiate between aspects of that state that are private to their neighbors and those that are private to agents they



(a) Elevators (b) MALogistics (c) Depot

Figure 2: Agent interaction graph. Larger nodes represent agents while smaller nodes represent shared facts.

do not know. If the heuristic computation preserves agent privacy too, then the entire algorithm respects agent privacy. □

So as not to make this claim somewhat vacuous, we note that the landmarks heuristics of Maliah *et al.* [Maliah *et al.*, 2014b] respects agent privacy: agents are aware only of landmarks of agents with whom they share some variable.

Finally, here are a number of optimizations that can be made to improve the simple MAFBS algorithm above.

1. When a message is sent backwards, the required condition is associated with it, so that only actions that achieve it will be sent forward. Notice that this requires slightly more careful book-keeping in the backwards closed list.
2. If φ_i sent a state s backwards to φ_j, it will not send backwards to φ_j any descendant of s obtained by applying only φ_i's actions.

The above changes preserve completeness because the basic scheme that is used to show that we can generate sequences that correspond to goal-focused plans requires only the type of focused forward and backward message passing that is captured by these changes.

5 Empirical Evaluation

We conduct an empirical analysis of MAFBS. We experiment with a set of benchmarks from the CoDMAP competition [Štolba *et al.*, 2015b], and two more complicated domains — MA-Blocks and MA-Logistics, where a larger number of private actions need to be executed between two consecutive non-private actions, and agents choose between several paths for achieving goals.

We compare MAFBS to MAFS. Both algorithms use a landmark heuristic [Maliah *et al.*, 2014a], which naturally extends to preserve agent privacy. We experiment only with CoDMAP instances of each problem that both algorithms were able to solve. Table 1 shows the results of the experiments. We report 3 metrics — the number of messages that were sent (a message broadcasted to k agents is counted as k separate messages, MAFBS messages contain one extra bit), the time to completion, and the number of expanded states. While MAFBS is advantageous in all metrics on (almost) all domains, the differences are especially pronounced in the case of the number of messages. MAFBS sends from 3% to 69% of the messages sent by MAFS, and 36% on average. This is not surprising, as MAFS sends new states to all agents that can apply an action, while MAFBS sends forward messages only to agents that use an effect of the generating action. The added backward messages do not reduce the advantage of MAFBS.

The number of generated states is often much smaller for MAFBS and is slightly worse only in one domain. Runtime improves across all domains.

To further understand the behavior of MAFBS we take a closer look at the domains where MAFBS does best and worst. Satellites is by far the easiest domain for MAFBS. This is because in satellites no agent ever generates a precondition for another agent. Agents share however resources that are consumed by actions, but never produced. Different agents can achieve different subgoals. Thus, there is no need for an agent to ever send forward (or backward) messages. Instead, the only messages that are sent are goal messages, allowing one agent to achieve goals that the other could not.

MAFBS also performs well on MALogistics, which is a more complex domains similar to the running example in Figure 1. We created domains where the number of interacting agents is limited, that is, the area under the control of each truck is connected to at most 2 other agents. Thus, each time an agent drops a package at some logistic center it needs to inform only a small number of agents, greatly reducing the number of messages.

We show (Figure 2) the agent interaction graphs for example problems from 3 additional domains — MALogistics and Depot, where MAFBSperforms well, and Elevators where it does not. In the graphs, the larger nodes represent agents, and the smaller nodes represent facts that agents share. As can be seen, in MALogistics there is only one fact that is shared between 4 agents. All other facts are shared between exactly 2 agents. In Depot, only one fact is shared between 3 agents, and all other facts are shared only between two agents. Given these sparse dependencies, it is no surprise that MAFBS send much less messages than MAFS.

In Elevators, on the other hand, changes in each floor affect 3 out of 4 elevators (floor f_5 is shared between all agents). Thus, the number of sent messages is not much smaller than what MAFS sends.

These domains demonstrate that loose coupling of agents

is advantageous for MAFBS, while tightly coupled agents with many almost public facts, makes MAFBS almost identical to MAFS.

6 Conclusion

We describe the multi-agent forward-backward search algorithm. This algorithm improves upon the state-of-the-art MAFS algorithm in all parameters measured, but most significantly, in the number of messages passed. This is not only an important performance measure for distributed algorithms, it might be that with fewer messages sent, less private information is leaked. In future work, we will study this intuitive relation between the amount of messages and the amount of leaked information. Moreover, MAFBS is the first MA planning algorithm to provide agent privacy — hiding the existence of an agent from others that have no direct impact on it. It is not difficult to see that forward-backward messages are orthogonal to the changes made in secure-MAFS. Thus, in future work hope to adapt MAFBS to display the same essential property of secure-MAFS, namely, that an agent never sends two states that differ only in the value of the private variables.

MAFBS is also interesting because it combines both forward and backward reasoning, in a manner that is very different from current single and multi-agent planning (and search) algorithms. We believe that additional optimization that help focus its message passing are possible, and could lead to further improvements in performance.

Acknowledgments: We thank the reviewers for their useful comments. This work was supported by ISF Grant 933/13, by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev, and by the Lynn and William Frankel Center for Computer Science.

References

[Berman *et al.*, 1992] Piotr Berman, Juan A Garay, and Kenneth J Perry. *Optimal early stopping in distributed consensus*. Springer, 1992.

[Bonisoli *et al.*, 2014] Andrea Bonisoli, Alfonso Emilio Gerevini, Alessandro Saetti, and Ivan Serina. A privacy-preserving model for the multi-agent propositional planning problem. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 973–974, 2014.

[Brafman and Domshlak, 2013] R. I. Brafman and C. Domshlak. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence*, 198:52–71, 2013.

[Brafman, 2015] Ronen I. Brafman. A privacy preserving algorithm for multi-agent planning and search. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1530–1536, 2015.

[Faltings *et al.*, 2008] Boi Faltings, Thomas Léauté, and Adrian Petcu. Privacy guarantees through distributed constraint satisfaction. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Sydney, NSW, Australia, December 9-12, 2008*, pages 350–358, 2008.

[Karpas and Domshlak, 2012] Erez Karpas and Carmel Domshlak. Optimal search with inadmissible heuristics. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*, 2012.

[Maliah *et al.*, 2014a] Shlomi Maliah, Guy Shani, and Roni Stern. Privacy preserving landmark detection. In *the European Conference on Artificial Intelligence (ECAI)*, pages 597–602, 2014.

[Maliah *et al.*, 2014b] Shlomi Maliah, Guy Shani, and Roni Stern. Privacy preserving landmark detection. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 597–602, 2014.

[Maliah *et al.*, 2015] Shlomi Maliah, Guy Shani, and Roni Stern. Privacy preserving pattern databases. In *ICAPS workshop on Distributed and Multi-Agent Planning (DMAP)*, 2015.

[Nissim and Brafman, 2012] R. Nissim and R. I. Brafman. Multi-agent A* for parallel and distributed systems. In *AAMAS*, pages 1265–1266, 2012.

[Nissim and Brafman, 2014] Raz Nissim and Ronen I. Brafman. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research (JAIR)*, 51:293–332, 2014.

[Pommerening and Helmert, 2015] Florian Pommerening and Malte Helmert. A normal form for classical planning tasks. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 188–192, 2015.

[Štolba and Komenda, 2014] Michal Štolba and Antonín Komenda. Relaxation heuristics for multiagent planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2014.

[Štolba *et al.*, 2015a] Michal Štolba, Daniel Fišer, and Antonín Komenda. Admissible landmark heuristic for multi-agent planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2015.

[Štolba *et al.*, 2015b] Michal Štolba, Antonın Komenda, and Daniel L Kovacs. Competition of distributed and multi-agent planners (codmap). *The International Planning Competition (WIPC-15)*, page 24, 2015.

[Tate, 1977] Austin Tate. Generating project networks. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, August 1977*, pages 888–893, 1977.