Partial Disclosure of Private Dependencies in Privacy Preserving Planning

Rotem Lev Lehman[1] and **Guy Shani**[1] and **Roni Stern**[1,2]
[1]Software and Information Systems Engineering, Ben Gurion University of the Negev, Be'er Sheva, Israel
[2]Palo Alto Research Center, Palo Alto, CA, USA
levlerot@post.bgu.ac.il, shanigu@bgu.ac.il, sternron@post.bgu.ac.il

Abstract

In collaborative privacy preserving planning (CPPP), agents can plan together by revealing private dependencies between their public actions to other agents. Perhaps one of the best methods for computing plans under privacy constraints uses a projection of the complete problem that captures these private dependencies. In this paper we investigate the partial disclosure of such private dependencies. We create a projection where agents publish only a part of their dependencies, and attempt to create a complete plan using these dependencies only. We investigate different strategies for deciding which dependencies to publish, and how they affect both the coverage and the privacy leakage of the solutions. Experiments over standard CPPP domains show that the proposed dependency-sharing strategies enable creating an effective projection without sharing all private dependencies.

1 Introduction

Designing autonomous agents that act collaboratively is an important goal. A fundamental requirement of such collaboration is to plan for multiple agents acting to achieve a common set of goals. *Collaborative Privacy-Preserving Planning (*CPPP*)* is a multi-agent planning task in which agents need to achieve a common set of goals without revealing certain private information (Brafman and Domshlak 2008). In particular, in CPPP an individual agent may have a set of private facts and actions that it does not share with the other agents. CPPP has important motivating examples, such as planning for organizations that outsource some of their tasks.

There are two common approaches to CPPP: *single search* and *two-level search*. Single search solvers operate by running a joint forward search (Nissim and Brafman 2014; Štolba and Komenda 2017) where agents that apply public actions send the resulting state to other agents that continue the forward search. Two-level search solvers operate by creating a public plan that is shared by all agents, and then have each agent extend it locally with private actions. In either case, the agent publish *dependencies* between the public actions. For example, in a solver from the first approach, an agent i that receives a state s from another agent j that applied a public action a_1, also receives from j an index for its own private facts. Later, when i continues the search from s and executes another public action a_2, it returns the state to j with the same index. Thus, j learns that applying a_1 may have helped i in making a_2 possible. We call this a *private dependency* between the actions.

Maliah, Shani, and Stern (2016a) take this idea further, and compute and publish a set of such private dependencies in the form of artificial facts. This allows the agents to jointly create and publish a *projection* of the problem that contains all the public facts, public actions, and published artificial facts that capture private dependencies. Such a projection can be used to construct heuristics for single search CPPP algorithms such as MAFS. In two-level search solvers, this projection can be used to generate the public plan, defining the public plan to be a solution to the problem defined by the projection.

In many cases, however, the agents can construct a plan requiring only a small portion of the private dependencies. In such cases, it may be preferable to reveal only a part of the dependencies, intuitively reducing the amount of disclosed private information. In this paper we focus on the settings where agents publish only a limited number of such dependencies. We suggest 4 different methods for deciding which dependencies should be published first, in order to construct a plan with as little disclosed dependencies as possible.

We provide experiments on standard benchmarks from the CPPP literature, showing that our methods, in many domains, publish as little private dependencies as possible in order to reach a plan. We also analyze the privacy leakage of our methods (Štolba, Tožička, and Komenda 2018), showing that, as one would intuitively expect, publishing more dependencies leaks more private information.

2 Background

A 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 primitive propositions (facts), A_i is the set of actions agent i can perform, I is the start state, and G is the goal condition.

Each action $a = \langle pre(a), \mathit{eff}(a)\rangle$ 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

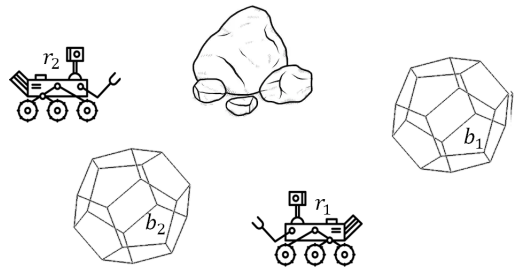


Figure 1: The rovers domain, where two rovers, r_1 and r_2 can access two base stations b_1 and b_2, collaborating to take measurements of a rock.

action a to state s. A *plan* $\pi = (a_1, \ldots, a_k)$ is a solution to a planning task iff $G \subseteq a_k(\ldots(a_1(I)\ldots)$.

Privacy-preserving MA-STRIPS extends MA-STRIPS by defining sets of variables and actions as private, known only to a single agent. $private_i(P)$ and $private_i(A_i)$ denote the variables and actions, respectively, that are private to agent i. $public(P)$ is the set of public facts in P. $public_i(A_i)$, the complement of $private_i(A_i)$ w.r.t. A_i, is the set of public actions of agent i. Some preconditions and effects of public actions may be private, and the action obtained by removing these private elements is called its *public projection*, and it is known to other agents. When a public action is executed, all agents are aware of the execution, and view the public effects of the action. The goals can be public, but can also be private to a single agent. An agent is aware only of its *local view* of the problem, that is, its private actions and facts, its public actions, the public facts, and the public projection of the actions of all other agents. That is, for public actions of other agents, the agent's local view contains only the public preconditions and effects of these actions.

In the Rovers example in Figure 1, 2 Mars Rovers collaborate to explore rocks on the surface of Mars. The Rovers need to perform some sensor measurements on rocks, and, due to limited carriage capacity, can only carry 2 sensors at a time. Unused sensors are stored in base stations, and can be taken and returned to the base stations as needed.

The public facts in this problem are the sensors located in bases, and the current condition of the target rock. The public actions are taking and returning sensors to the base stations, putting collected rock samples at the base stations, and performing various examination actions on rocks, such as taking an image, mining a mineral, or collecting a sample. The sensors held by a rover and its position are private, and the private actions are movement actions.

3 Algorithmic Approaches

There are two major approaches to planning in CPPP (Torreño et al. 2017). The first approach begins by computing a public plan, which is known as a coordination scheme (Nissim, Brafman, and Domshlak 2010; Brafman and Domshlak 2013; Torreno, Onaindia, and Sapena 2014). Then, the agents independently extend the public plan into a complete plan by adding private actions. In this extension each agent attempts to achieve the preconditions of its own public actions in the public plan.

For example, in the DPP planner (Maliah, Shani, and Stern 2016b) the agents compute together a single agent projection of the CPPP problem that captures the dependencies between public actions. That is, which public actions facilitate the execution of other public actions of that agent. In our running example, such a dependency exists, e.g., for agent 1 between picking up a camera at base b_1 and taking a photo of the rock. These dependencies are computed using limited regression from the precondition of a public action to the effects of other public actions. Given this projection, one can compute a public plan using a standard classical planner. The projection is incomplete, and it is hence possible that the generated public plan cannot be extended to a complete plan, in which case DPP fails.

An alternative approach to computing a high level scheme is to compute a complete plan directly. This can be done by each agent running a distributed forward search algorithm over its own action space, informing other agents of advancements in the search process.

The first algorithm in this family is MAFS (Nissim and Brafman 2014) — a distributed algorithm in which every agent runs a best-first forward search to reach the goal. Each agent maintains an open list of states, and in every iteration each agent chooses a state in the open list to expand, generating all its children and adding them to the open list (avoiding duplicates). Whenever an agent expands a state that was generated by applying a public action, it also broadcasts this state to all other agents. An agent that receives a state adds it to the open list. For example, in Figure 1, when agent 1 puts down the camera at base b_2, it broadcasts this state to all other agents. Agent 2 can now use this state to pick up the camera and take a photo of the rock. To preserve privacy, the private part of a state is obfuscated when broadcasting it, e.g., by replacing the private facts with some index, such that only the broadcasting agent knows how to map this index to the corresponding private facts. Once the goal is reached, the agent achieving the goal informs all others, and the search process stops. MAFS can be extended in several ways.

Štolba and Komenda extended MAFS by applying two different open lists, one ordered by a local heuristic and the other by a global heuristic. Maliah et al. Maliah, Shani, and Brafman (2016) compute macros — sequences of private actions bounded by public actions — to expedite the local search process of the agent. For example, in Figure 1, once agent 1 has found the sequence of actions allowing it to get from base b_1 with the camera to the rock, it can save this sequence as a macro, allowing agent 1 to apply this macro in all future explored states where he wants to get from b_1 to the rock, expediting the search process.

4 Partial Disclosure of Private Dependencies

We now present the main contribution of this paper — reducing the amount of disclosed private dependencies and hence, the amount of disclosed information. In this paper, we discuss this in the context of the *DP Projection* method, which can be used either in a dedicated planner called the DP Planner or as a heuristic for MAFS-based solvers (Maliah, Shani,

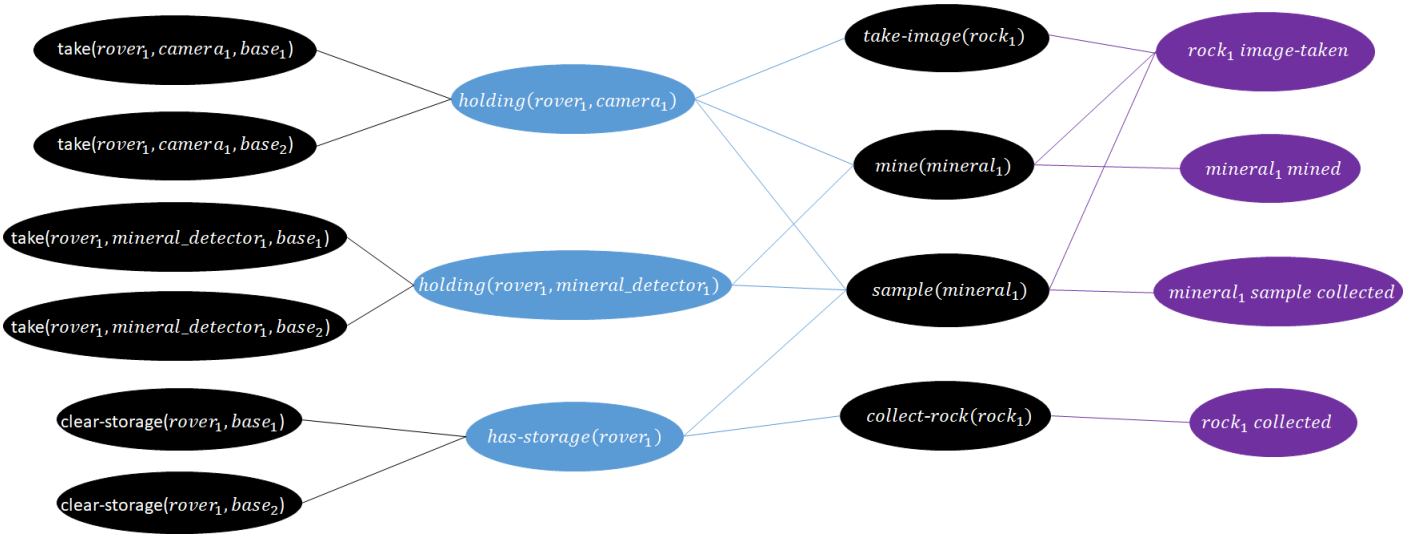


Figure 2: Local perspective of agent 1 private dependencies in the Rovers domain.

and Stern 2016a). We focus on the former application of the DP projection and only briefly discuss possible extensions to MAFS-based solvers.

For completeness and ease of exposition, we now describe a brief and slightly modified version of the DP projection. We say that a public action a *facilitates* the achievement of a private fact f, if (1) f is an effect of a, or (2) there exists a sequence of private actions $a_1, ..., a_k$ such that: f is an effect of a_k, each a_i takes as precondition an effect of some a_j s.t. $j < i$, and a_1 takes as precondition an effect of a.

Definition 1 (Private Dependency)**.** *An action a is said to have a private dependency if it has a private fact f as a precondition such that one of the following hold: (1) there is another public action a' that facilitates achieving f, (2) f is either true in the start state or can be achieved from it by only applying private actions.*

For example, in the *rovers* domain the action *take-image*$(rover_1, rock_1)$ has a private dependency because it has a private precondition *holding*$(rover_1, camera_1)$ that the public action *take*$(rover_1, camera_1, base_1)$ facilitates to achieve.

The agent jointly create a projection of the public problem, containing all the public facts, and a projected version of the public actions. For each public action a_i that requires a private precondition f_j, we create an artificial public fact f_j^i. The projected public version of a requires f_j^i as precondition. For each action a' of the agent that facilitates the achievement of the private f_j, the agent publishes f_j^i as an effect of a', thus publishing the private dependency of a on a', although the way that f_j is achieved remains obscured.

For a public action a_1 of an agent i we introduce a public artificial fact f_{a_1} into the projection, signifying that a_1 was executed. If a_1 facilitates the achievement of a private precondition of an action a_2 of agent i, then the projected a_2 will have f_{a_1} as precondition. Hence, the artificial predicates f_a capture the private dependencies between public actions.

The DP projection described by Maliah, Shani, and Stern (2016a) is created by having all agents compute and publish all their private dependencies. In this work, we limit to k the number of private dependencies of each agent is allowed to publish, where k is a parameter. Technically, each agent publishes all the artificial preconditions of all public actions, as well as k artificial effects of public actions. All public preconditions are published to avoid an over optimistic projection, where agents believe that they can execute public actions with no previous requirements.

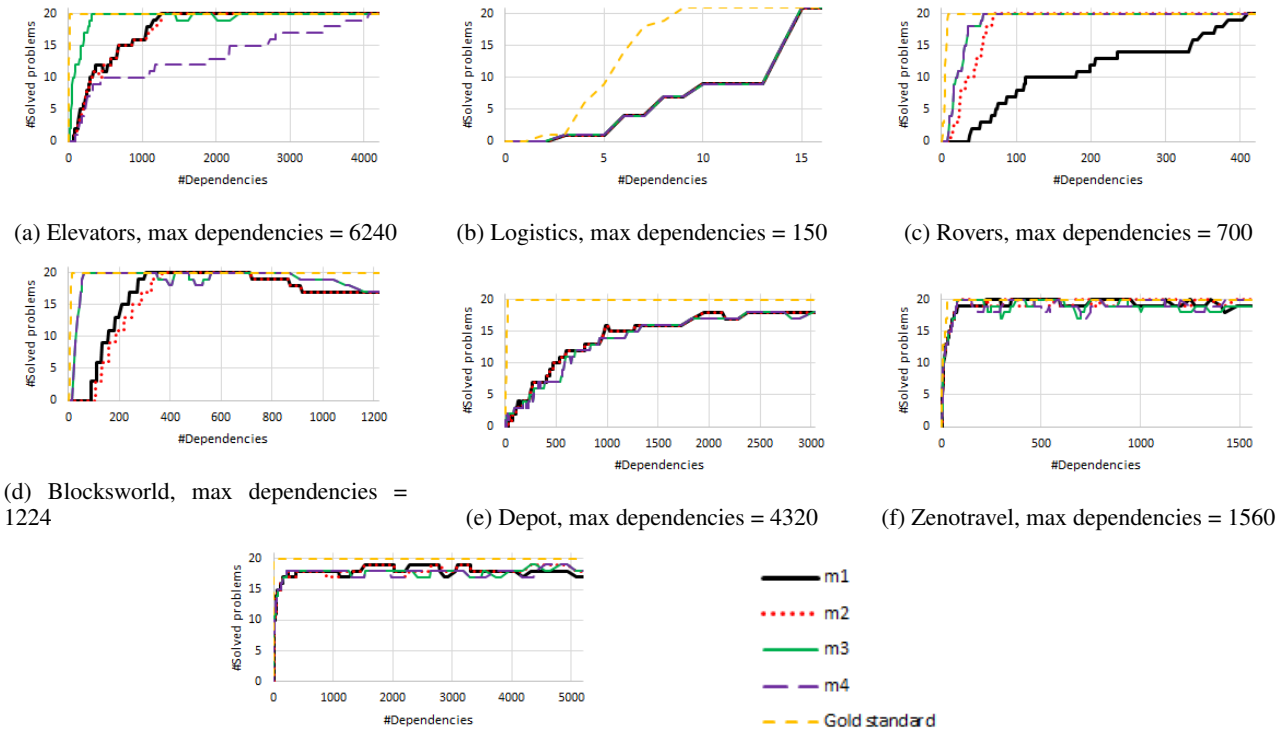
4.1 Theoretical Analysis

Before describing several heuristics for choosing which k private dependencies to share, we explore the theoretical implications of limiting the number of private dependencies that are being shared.

Let A and B be two DP projections. We denote by $A \subseteq B$ iff the set of private dependencies shared by A is a subset of the set of private dependencies shared by B. Let $plans(X)$ be the set of all plans that can be generated by a DP projection X. Let $P_{solve}(X)$ be the probability that a public plan chosen randomly from $plans(X)$ can be extended to a full plan (i.e., include also the private actions of the agents).

Theorem 1. *Given two DP projections A and B of the same problem, such that $A \subseteq B$, then: (1) $plans(A) \subseteq plans(B)$, and (2) $P_{solve}(A) \geq P_{solve}(B)$.*

Proof outline: Sharing a private dependency reveals that an additional artificial facts is achieved by some known public action. Artificial facts are only used as preconditions for some public actions. Thus, revealing private dependencies only facilitates performing additional actions, and thus more plans. The second part of Theorem 1 can be deduced from the fact that each link in the high level plan has a certain probability for extendability, and when increasing the amount of dependencies, you increase the links amount, bringing the probability to be lower than it was without that link.



(a) Elevators, max dependencies = 6240

(b) Logistics, max dependencies = 150

(c) Rovers, max dependencies = 700

(d) Blocksworld, max dependencies = 1224

(e) Depot, max dependencies = 4320

(f) Zenotravel, max dependencies = 1560

(g) Driverlog, max dependencies = 5800

(h) Legend.

Figure 3: Number of solved problems for each amount of published dependencies. Graphs truncated after all methods solve all problems. The maximal number of private dependencies in a problem in the chosen domain is shown for each domain

4.2 Ranking Published Dependencies

Above we discussed the implications of sharing k private dependencies, but we did not specify how to choose which k private dependencies to share. We now close this gap and suggest 4 different methods for selecting which dependencies to publish first. To better illustrate our methods, Figure 2 shows the local perspective of the private dependencies of agent 1 in the Rovers domain. Black nodes in the first and third column represent public actions, purple nodes in the fourth column represent public facts, and blue nodes in the second column denote artificial facts that capture private dependencies between public actions. The public actions on the first column generate the artificial facts while the public actions on the third column require them as preconditions, and generate the public facts. For ease of exposition, we label the artificial facts by the intuitive link that they represent, although in reality, of course, the artificial facts have no such meaningful names.

We take an iterative approach — all agents publish one artificial effect of one public action at each iteration. If the public projection cannot be solved, all agents publish a second artificial effect of a public action, and so forth. Hence, at each iteration an agent must decide on the next artificial effect to publish, given the effects it has published thus far.

Our first method, which we denote m_1, publishes artificial effects that are used as preconditions in as many public actions as possible. In the example in Figure 2, we can publish either the effect of *take*$(rover_1, camera_1, base_1)$, or the effect of *take*$(rover_1, camera_1, base_2)$, representing taking the camera from either base, as it supplies a precondition for 3 public actions (taking an image, mining a mineral, and collecting a stone sample). Let us assume that the effect of *take*$(rover_1, camera_1, base_1)$ was published first. To avoid that in the next round the effect of *take*$(rover_1, camera_1, base_2)$ will be chosen, providing the same preconditions, we subtract from the number of preconditions the amount of times this artificial effect was already published. Hence, at the next step, all unpublished effects have the same ranking.

The second method, which we denote m_2, publishes an effect that *enables the achievement* of as many public facts as possible. An effect enables the achievement of a public fact f if it is a precondition to an action that achieves f. In our illustration, this is when there is a path from a blue private fact to a purple public fact via a black node. Again, we subtract the number of times that an artificial fact was published as an effect to avoid repeatedly choosing the same effect. Hence, if we again select at the first iteration the effect of *take*$(rover_1, camera_1, base_1)$, at the second iteration we will select another artificial fact.

The third method, denoted m_3, attempts to maximize the amount of public actions that can be executed. That is, instead of publishing the artificial fact that provides a precondition for as many actions, we publish the artificial fact

Domain	Method	Min. cost	Max. cost	Min. dep. cost	Max. dep. cost	Improvement
Blocksworld	m1	47.70	95.90	75.00	48.10	34.92%
	m2	47.70	95.50	73.70	48.10	33.49%
	m3	43.50	101.00	74.90	43.90	37.69%
	m4	43.50	101.00	74.80	43.90	37.52%
Depot	m1	48.89	65.00	61.37	48.89	16.97%
	m2	48.89	64.63	61.37	48.89	16.97%
	m3	45.06	50.28	49.94	45.06	8.87%
	m4	45.06	50.28	49.94	45.06	8.87%
Driverlog	m1	37.58	67.32	63.63	38.63	28.12%
	m2	37.11	67.37	63.63	38.63	28.43%
	m3	38.00	67.05	63.32	38.63	27.37%
	m4	38.00	67.11	63.32	38.63	27.37%
Elevators	m1	65.15	87.10	68.00	80.60	4.60%
	m2	65.05	88.05	70.55	80.60	8.23%
	m3	64.60	84.45	68.85	80.60	8.37%
	m4	65.15	84.10	68.75	80.60	5.87%
Logistics	m1	59.65	66.10	65.45	60.25	8.52%
	m2	59.65	66.10	65.45	60.25	8.52%
	m3	59.75	66.20	65.45	60.25	8.47%
	m4	59.75	66.20	65.45	60.25	8.47%
Rovers	m1	72.85	80.10	78.35	75.30	6.59%
	m2	72.60	77.40	75.20	75.30	3.58%
	m3	72.90	77.75	75.85	75.30	3.76%
	m4	73.10	77.70	75.80	75.30	3.36%
Zenotravel	m1	68.25	73.20	69.90	71.20	2.97%
	m2	68.25	73.20	69.90	71.20	2.97%
	m3	68.00	73.45	69.90	71.20	2.73%
	m4	68.00	73.45	69.90	71.20	2.73%
Average	m1	57.15	76.39	68.81	60.43	14.67%
	m2	57.04	76.04	68.54	60.43	14.60%
	m3	55.97	74.31	66.89	59.28	13.89%
	m4	56.08	74.26	66.85	59.28	13.46%

Table 1: Averaged cost over all of the problems for each domain. The data in the "Improvement" column is the percentage of the minimal cost out of the first solved cost.

that enables as many public actions as possible. Here, in the first iteration, publishing the effect of taking the camera from either base, or the effect of clearing the storage on the rover, both enable immediately one public action (taking an image, and picking a sample into the storage bin, respectively). Assuming that we first select, again, *take*$(rover_1, camera_1, base_1)$, at the next iteration, taking the mineral detector from either base station or clearing the storage, both enable one additional action, and are hence tied.

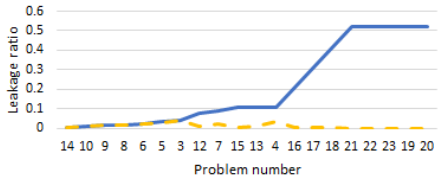
In this method we also need to balance between enabling new public actions that have not been available given the already published effects, and enabling new ways to apply already possible actions, that may result in better plans. As such, for each action a that is enabled by the published effect, we discount its score by $\frac{1}{c_a+1}$ where c_a is the number of times that a was enabled by previously published facts. Hence, at the first time that an action is enabled, it provides a score of 1, at the second time, a score of $\frac{1}{2}$, and so forth.

The last method, denoted m_4, is similar to m_3, but focuses not on the public actions, but on the public effects. That is, we publish an effect that enables achieving as many public facts as possible. Again, we discount the score of a public fact by $\frac{1}{c_f+1}$ where c_f is the number of times that public fact f was enabled by previously published facts. In this method, again taking the camera and clearing the storage are tied in the first iteration. In the second iteration, however, if we again select *take*$(rover_1, camera_1, base_1)$ first, taking the mineral detector has a better score, because it enables the mineral drilling action, that has 2 public effects, one of which was already obtained, and hence a score of $1\frac{1}{2}$.

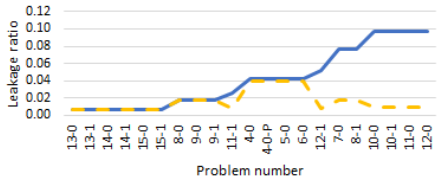
5 Empirical Analysis

We now evaluate our methods using benchmarks from the CPPP literature (Štolba, Komenda, and Kovacs 2015). For each problem in each benchmark domain, we ran the projection-based solver (Maliah, Shani, and Stern 2016a) with a growing number of revealed dependencies.

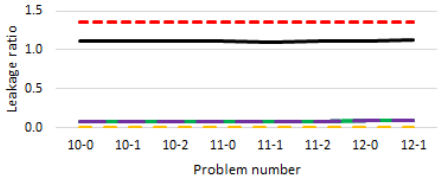
Figure 3 shows the number of problems that were solved on each domain given the number of revealed dependencies



(a) Zenotravel. Blue line represents all methods.



(b) Logistics. Blue line represents all methods.



(c) Blocksworld.

(d) Legend.

Figure 4: Privacy leakage in the various domains. Problems on the x-axis are sorted by increasing leakage.

by each agent. For each problem, we also computed a "Gold Standard" value, which is the number of private dependencies used by the plan that was generated when all dependencies are known. The gold standard serves as a baseline for comparison, indicating how many private dependencies are sufficient to find a solution.[1] In all domains, method m_3, which prioritize enabling additional public actions, performs the best. The rest of the methods vary in their performance. For example, on Rovers and Blocksworld, m_4, that prioritizes achieving additional public facts, performs the same as m_3 and better than the other methods, but on Elevators m_4 performs the worst. On Elevators, arguably the domain with the highest amount of required collaboration, differences between methods are most pronounced.

An interesting phenomena occurs, e.g., in Blocksworld, where some problems are solved when not all dependencies are available, but cannot be solved when more dependencies are published. This is because the projection method may produce a public plan that cannot be extended into a complete plans. Our methods were often able to publish dependencies that resulted in plans that could be extended. Later, additional dependencies confused the planner to choose plans that could not be extended. Zenotravel and Logistics are the easiest domains in the sense that all methods managed to obtain solutions for all problems after sharing only a small number of dependencies (note the scale in the x axis of the Logistics results). Most problems in the Driverlog domain were also easy in this sense, except for a few difficult problems which required more dependencies. On Depot, on the other hand, all methods perform much worse than the gold standard, leaving much room for improvement.

Next, consider the cost of the generated public plan. Table 1 shows for each domain the results of the following averaged factors: (1) Min. cost, (2) Max. cost, (3) Min. dep. cost – the solution cost for a projection with the least amount of dependencies revealed, and (4) Max. dep. cost – the solution cost for a projection with the maximal amount of dependencies revealed. In addition, the column "Improvement" shows the percentage of the minimal cost out of the first solved cost ($\frac{First\ solved\ cost - Minimal\ cost}{First\ solved\ cost} * 100\%$). We can see that on average (marked with green color at Table 1), the improvement is only about 14% for the different methods, which means that the first time the problem was solved gave us a pretty good solution (only 14% worse than the best solution that was found when we revealed more dependencies). However, the impact of sharing private dependencies on solution cost varies significantly per domain. For example, in Blocksworld the improvement is almost 40% while for Zenotravel it is always less than 3%. Also, note that in most cases there was almost no difference between the different heuristics (m1-4) in terms of solution cost.

Different methods require a different amount of published dependencies to solve the problem. While it is intuitive that revealing less dependencies preserves more privacy, this is not necessarily so. To measure the amount of privacy preserved by each algorithm, we use the privacy leakage tool[2] (Štolba, Tožička, and Komenda 2018) for measuring the amount of private information that is leaked by each method. Figure 4 shows the privacy leakage in 3 domains. For each problem in each domain, we take the first time that it was solved by each method, and generate the input for the privacy leakage tool. We measure the privacy leakage ratio from the perspective of agent 1 in each problem. The "Gold Standard" series's privacy leakage is calculated as if we only revealed those dependencies.

In Zenotravel and Logistics all methods publish the same number of dependencies before reaching the goal, although not necessarily the same dependencies. In these domains, the privacy leakage was identical for all methods, except for the gold standard, and is hence represented by a single line. In Blocksworld, however, the privacy leakage ratio of the different methods was not the same. As can be seen, there is a direct correlation between the amount of published dependencies required for computing a plan (Figure 3d), and the amount of leaked privacy (Figure 4c). This further sup-

[1] This does not mean, however, that it is not possible to find a solution if fewer private dependencies are revealed.

[2] http://github.com/stolba/privacy-analysis

ports our intuition that publishing less dependencies results in higher privacy. However, analysis of privacy leakage in CPPP is an ongoing research topic and it is difficult at this stage to draw general conclusions.

6 Conclusion and Future Work

In this work we suggest methods for publishing only a part of the private dependencies between public actions of agents, in order to reduce the amount of privacy leakage. We focus on the projection method that uses all private dependencies to compute a public plan, showing that in many cases a public plan can be computed with only a small portion of the dependencies, and that our heuristic methods rapidly find a good subset to share. We provide experiments over standard benchmark domains, comparing the coverage of our methods, as well as the amount of leaked privacy.

In the future we will expand our methods to MAFS, where an agent may publish only some public states that it reaches, and to other privacy preserving planning techniques as well. Another thing that we shall do in the future is finding the optimal set of dependencies that need to be revealed in order to solve each problem. This shall replace the gold standard that we have used now and will help us find better methods that will outperform the methods that were described in this paper. Future work will also include an analysis of the privacy leakage in the context of using the shared dependencies in the MAFS planner. Finally, it is also worthwhile to assign importance values to different dependencies, where each dependency has an intrinsic value that needs to be considered.

Acknowledgements

This work is partially funded by ISF grant # 210/17 to Roni Stern and by ISF grant # 1210/18 to Guy Shani.

References

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

[Brafman and Domshlak 2013] 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.

[Maliah, Shani, and Brafman 2016] Maliah, S.; Shani, G.; and Brafman, R. I. 2016. Online macro generation for privacy preserving planning. In *ICAPS*, 216–220.

[Maliah, Shani, and Stern 2016a] Maliah, S.; Shani, G.; and Stern, R. 2016a. Stronger privacy preserving projections for multi-agent planning. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 221–229.

[Maliah, Shani, and Stern 2016b] Maliah, S.; Shani, G.; and Stern, R. 2016b. Stronger privacy preserving projections for multi-agent planning. In *ICAPS*, 221–229.

[Nissim and Brafman 2014] Nissim, R., and Brafman, R. I. 2014. Distributed heuristic forward search for multi-agent planning. *JAIR* 51:293–332.

[Nissim, Brafman, and Domshlak 2010] Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *AAMAS*, 1323–1330.

[Štolba and Komenda 2017] Štolba, M., and Komenda, A. 2017. The madla planner: Multi-agent planning by combination of distributed and local heuristic search. *Artificial Intelligence* 252:175–210.

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

[Štolba, Tožička, and Komenda 2018] Štolba, M.; Tožička, J.; and Komenda, A. 2018. Quantifying privacy leakage in multi-agent planning. *TOIT* 18(3):28.

[Torreño et al. 2017] Torreño, A.; Onaindia, E.; Komenda, A.; and Štolba, M. 2017. Cooperative multi-agent planning: A survey. *ACM Comput. Surv.* 50(6).

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