Stronger Privacy Preserving Projections for Multi-Agent Planning

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

Negev, Israel

Abstract

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. In many CPPP algorithms the individual agents reason about a *projection* of the multi-agent problem onto a single-agent classical planning problem. For example, an agent can plan as if it controls the public actions of other agents, ignoring their unknown private preconditions and effects, and use the cost of this plan as a heuristic for the cost of the full, multi-agent plan. Using such a projection, however, ignores some dependencies between agents' public actions. In particular, it does not contain dependencies between actions of other agents caused by their private facts. We propose a projection in which these *private dependencies* are maintained. The benefit of our dependency-preserving projection is demonstrated by using it to produce high level plans in a new privacy preserving planner that is able to solve more benchmark problems than any other state-of-the-art privacy preserving planner. This more informed projection does not explicitly share private information. In addition, we show that even if an adversary agent knows that an agent has some private objects of a given type (e.g., trucks), it cannot infer how many such private objects the agent controls. This introduces a novel strong form of privacy that is motivated by real-world requirements.

Introduction

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. CPPP has important motivating examples such as planning for organizations that outsource some of their tasks and collaboration between military units where compartmentalization is required. CPPP recently received much attention and new efficient solvers, scaling up to larger, more challenging domains, are constantly proposed (**?**; **?**; **?**; **?**; **?**, inter alia).

Even though research on CPPP is flourishing, classical planners are substantially more mature, offering fast solvers with strong heuristics (**?**; **?**). An attractive approach, hence, is to *project* the multi-agent problem onto a classical planning problem, employ a classical solver, and then extend the resulting public plan using private actions (**?**).

Due to privacy, each individual agent has only a partial view of the world, containing its own public and private facts and actions. The agent also knows about the public actions of other agents, but is aware only of their public preconditions and effects. One can treat the agent view of the world as a classical planning problem, pretending that the agent controls the public actions of others. Then, one could solve this classical projection and obtain a plan.

In this single-agent projection of the MA world, the agent is unaware of dependencies between other agents' public actions, originating from their private facts and actions. This results in an over optimistic assumption about agents' abilities. Take, for example, a blocksworld domain with two agents, where each agent controls a different arm, and conceals which block is held by the arm. The precondition of unloading a block A on top of block B by agent 1 is that the agent is holding block A. As this is a private information, agent 2 is unaware of this precondition, and thus believes that agent 1 can put A everywhere without picking it up.

Thus, a solution to this single-agent projection is typically not a sound solution to the mulit-agent planning task, and some mechanism to communicate additional information is needed for constructing a sound plan. For example, in the MAFS algorithm (**?**) agents broadcast partially encrypted states to all agents whenever a public action is performed. Methods of this type use a multi-agent planning mechanism, where all agents plan together, jointly revealing additional important information.

In this paper we take a different approach. We suggest a stronger projection, where all agents publish a projection of their private view of the world, containing only public actions. The joint projection is fed into a classical planner, resulting in a high level plan for achieving the goals, composed only of public actions (**?**; **?**). Then, each agent extends the plan by adding private actions that allow it to accomplish its part in the public plan.

The main drawback of the simple projection is the lack of information concerning dependencies between the public actions of an agent and between public actions and the initial state of the world. Therefore, we focus on identifying these dependencies and sharing them in a privacy preserving manner. We define new facts that represent action dependencies, and explain when to add or delete these dependency facts.

The usefulness of the proposed projection is demonstrated

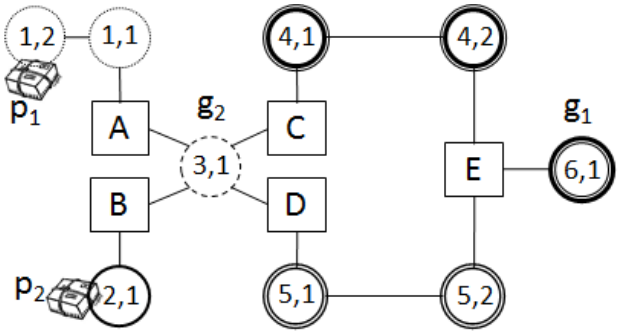

Figure 1: A logistics example, where trucks deliver packages between logistics centers, denoted by squares. Each agent covers a set of cities, denoted by circles, and labeled i, j where i is the agent covering the city and j is the local city index. The logistic centers can be entered by several agents, serving as collaboration sites.

in practice in a simple planner that uses a solution of the projected problem as a high-level plan, extending it to a full plan jointly by all agents. While the resulting planner is not complete, it is able to solve more benchmarks from the distributed multi-agent competition (CoDMAP) (**?**) than any other state-of-the-art privacy preserving planner.

Sharing the proposed projection between the agents does not violate the privacy requirement as the proposed projection does not contain any private information (fact or action) explicitly. This form of privacy is the current standard in the privacy preserving planning literature. Another contribution of this work is the introduction of a more strict form of privacy that we call *object cardinality privacy*. This form of privacy addresses a realistic requirement in which agents are interested in hiding from other agents the number of private objects they control of specific types.

For example, a logistic company may wish to hide the number of trucks it controls, and the military would not disclose the number of bases in an area. An algorithm is said to preserve object cardinality privacy if an adversary, listening to the communication during planning, cannot distinguish between worlds where agents possess different quantities of private objects. We discuss the relationship between object cardinality privacy and other definitions of privacy, and show that our projection also preserves this type of privacy.

The contributions of this paper are hence two-fold: we provide the new definition of privacy – object cardinality privacy– and suggest a new strong projection, showing it both preserves object cardinality privacy and is key to the construction of a new planner to outperform existing state-of-the-art planners.

Privacy Preserving Model

Research on privacy preserving planning focuses on variants of the multi-agent STRIPS model (MA-STRIPS) (**?**) mainly due to its simplicity.

Definition 1 (MA-STRIPS)**.** *An MA-STRIPS problem 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 of false).*
- A_i *is the set of actions agent* i *can perform.*
- I *is the start state.*
- G *is the goal condition.*

Each action $a = \langle \mathit{pre}(a), \mathit{eff}(a), c(a)\rangle$ is defined by its preconditions ($\mathit{pre}(a)$), effects ($\mathit{eff}(a)$), and cost ($c(a)$). Preconditions and effects are logical formulas of P. A state is a conjunction of facts (true or false). The goal G is also a conjunction of facts. The result of applying an action a to a state s is denoted by $a(s)$. A solution to a planning task is a *plan* $(a_1, \ldots, a_k)$ such that $G \subseteq a_k(\ldots(a_1(I)\ldots)$, i.e., a sequence of actions that transforms the initial state (I) to a state satisfying the goal condition (G).

Figure 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 the six trucks. Each truck has three actions: move, load, and unload, corresponding to moving the truck 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 truck i. The rectangles are logistic centers visited by multiple trucks that load or unload packages.

Privacy

Privacy-preserving MA-STRIPS extends MA-STRIPS (**?**) by defining sets of variables and actions as private, known only to a single agent. Let $\mathit{public}(P)$ be a set of public facts, whose value is always known to all agents. Let $\mathit{private}_i(P)$ be a set of facts whose existence and value is known only to agent i. Each agent has a set of actions it can execute, where $\mathit{private}_i(A)$ is a set of private actions, that no other agent knows about, and their execution is hidden from other agents, and $\mathit{public}_i(A)$ is a set of public actions. A private action can have only private effects. A public action may have both public and private effects. Upon the execution of a public action, all agents observe its public effects. For ease of exposition we assume that all goals are public, although our method can be extended to handle private goals.

A solution to a privacy-preserving MA problem, is a sequence of public and private actions. We say that the sequence of public actions in such a solution is a *valid* high-level plan if it can be extended to a full plan using the agents' private actions.

Extending a valid high-level plan to a full plan can be done by all agents concurrently, where each agent plans independently to achieve the preconditions of its actions in the high-level plan (**?**).

Consider a public action a of agent j from the perspective of agent i. The action is public, so agent i knows about a's existence. However, if a has private preconditions and effects, they are only known to agent j and are hidden from agent i. Formally, we define the *local view* of action $a = \langle \mathit{pre}(a), \mathit{eff}(a), c(a)\rangle$ by agent i, denoted $\pi_i(a)$, as

$$\pi_i(a) = \langle \mathit{public}(P) \cap \mathit{pre}(a), \mathit{public}(P) \cap \mathit{eff}(a), c(a)\rangle$$

Clearly, $\pi_i(a) = a$ if a is an action of agent i.

Definition 2 (Local View)**.** *Given an MA-STRIPS problem* $\pi = \langle P, \{A_i\}_{i=1}^k, I, G\rangle$*, the* local view *of agent* i *is defined by the tuple* $\pi_i = \langle \pi_i(P), \{\pi_i(A_j)\}_{j=1}^k, \pi(I), \pi(G)\rangle$ *where:*

- $\pi_i(P) = \textit{public}(P) \cup \textit{private}_i(P)$
- $\pi_i(I) = I \cap \pi_i(P)$
- $\pi_i(G) = G \cap \pi_i(P)$

In our running logistic example, assume now that each truck is owned by a different company, and that one company does not want to share its location and coverage (which locations it can reach) with other companies. The only actions that are public are load/unload actions from/at logistic centers. The only facts that are public are facts representing whether a package is at one of the logistic centers. Load/unload actions from/at private locations and all move actions are private actions. The location of the trucks, and facts representing whether a package is at one of the private locations, are all private facts. The local view of agent 3 consists of the facts representing packages at A, B, C, D, E, and (3,1); the facts representing possible locations of truck 3; the move, load, and unload actions of truck 3; and the load/unload actions of all agents from/to logistics centers. For example, the local view of a load action of another agent has a precondition regarding the location of the package only, ignoring the private precondition of the location of the truck.

Definition 3 (Privacy-Preserving MA-STRIPS)**.** *A privacy-preserving MA-STRIPS problem is defined by an MA-STRIPS problem and the* $\textit{public}(\cdot)$ *and* $\textit{private}_i(\cdot)$ *functions for every* $i \in [1, n]$*. The task is to find a solution without revealing private information during planning.*

In Section 11 we discuss concrete definitions for private information that must not be revealed during planning.

Classical Projections and Regression

We now discuss projections of a privacy-preserving multi-agent problem into classical single-agent planning, and the regression of formulas through action sequences, both of which are key concepts in our projection algorithm.

Classical Projections

A *classical projection* of a privacy-preserving multi-agent problem is a compilation of the problem into a classical single-agent problem. For example, one can consider the local view (Definition 2) of the agent as a classical planning problem, pretending that the agent controls the public actions of other agents as well.

A classical projection is *sound* if the sequence of public actions in any solution to the projected problem, can be extended using only private actions to a solution to the original multi-agent problem. As efficient privacy preserving projections tend to be lossy, not capturing all the information in the original problem, achieving soundness is difficult.

We say that a classical projection is *complete* if given a solution to the original privacy preserving problem, there exists a solution to the projected planning problem with the same sequence of public actions. An example of a complete classical projection is the local view projection.

Classical projections are of interest because they can be solved by each agent, providing a heuristic estimation for the cost of solving the original problem (**?**). Moreover, some privacy preserving algorithms use the solution to the projected planning problem as a high level plan and try to extended it to a complete plan by adding private actions of the individual agents (**?**; **?**).

Dependencies While a local view of an agent is a complete single-agent projection, it fails to consider some dependencies between public actions of another agent. In particular, the local view of agent i contains no information about *private dependencies* of other agents' public actions.

Definition 4. *A public action* a facilitates *the achievement of a private fact* f*, if* f *is an effect of* a*, or there exist 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.* $j < i$*,*
- a_1 *takes as precondition an effect of* a*.*

Definition 5 (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. We call the first type of private dependency an action-to-action dependency and call the second type an init-to-action dependency.*

In our logistic example (Figure 1), the public action *(unload p t3 A)* has a private precondition *(on p t3)*, and the public action *(load p t3 B)* achieves it. Thus, *(unload p t3 A)* has a private dependency of the type "action-to-action". The local views of the other agents do not include this private dependency: the public *(unload p t3 A)* action will appear in the local views of all agents (except agent 3) without the precondition *(on p t3)*, since it is a private fact. As a result, agents will believe that unloading a package at arbitrary public locations is always possible. As an example of an "action-to-init" private dependency, consider the public action *(unload p2 t2 B)*, which has a private precondition *(on p2 t2)*. This precondition can be achieved by a private action *(load p t2 (2,1))*, which in turn require the private fact *(at p2 B)*, which is true in the initial state. All agents (except agent 2) do not know about this precondition of *(unload p2 t2 B)* and would assume that t2 can always unload p2 at B, even if p2 was already moved beyond the reach of t2. In this work we suggest a stronger projection where information about private dependencies is shared, based on the concept of *regression*.

Regression

Regression is a fundamental tool in automated planning, in which we seek the conditions needed to reach a state, achieve a fact, or perform an action. More generally, if a formula ϕ describes a constraint on state facts and a is an action then the regression of ϕ through a, denoted $rg_a(\phi)$, is a formula that describes the minimal constraints on state

facts such that if we apply a to a state that satisfies $rg_a(\phi)$ we will reach a state that satisfies ϕ.

In the context of classical planning, Rintanen (**?**) describes how to compute the regression of a formula ϕ through an action a. We now review these definitions.

For ease of exposition, we assume here that both the effect and the precondition of an action are conjunctions of literals. That is, we do not allow conditional effects or disjunction of preconditions. We assume that the initial state is induced by an $init$ action, which assigns to all predicates their value at the initial state of the problem. The $init$ action can only be applied once and has no preconditions. We also assume that both the preconditions and effects of all actions are consistent, e.g., for a given literal l, no effect or precondition contains both l and $\neg l$. These are not real restrictions, and we could use Rintanen's full regression definition at the cost of a more complicated exposition.

In this restricted representation, Rintanen's regression of a formula ϕ through an action a, defined only over a single literal or a conjunction of literals, is simplified as follows:

Definition 6 (Regression)**.**

$$rg_a(l) = \textit{pre}(a) : l \in \textit{eff}(a) \quad (1)$$

$$rg_a(l) = false : \exists l' \in \textit{eff}(a) \textit{ s.t. } l, l' \textit{ are mutex} \quad (2)$$

$$rg_a(l) = l : l, \neg l \notin \textit{eff}(a) \quad (3)$$

$$rg_a(\phi) = \bigwedge_{l \in \phi} rg_a(l) : \phi = l_1 \wedge l_2 \wedge ... \wedge l_k \quad (4)$$

Rintanen shows that if $s \models rg_a(\phi)$ then $a(s)$ satisfies ϕ.

Given a set of actions A, we define the regression tree of a conjunction formula ϕ using actions in A, denoted $T_{rg}(\phi, A)$. Intuitively, the regression tree of ϕ represents all possible sequences of regression functions $rg_a(\cdot)$ that can be applied to ϕ, i.e., all sequences of actions in A that achieve ϕ. The nodes of the regression tree are labeled by formulas (given our restricted action definition, these formulas are only conjunctions of literals), and the edges are labeled by actions. We denote the formula associated with node n by $n.\phi$. The set of outgoing edges of a node n are labeled by all actions $a \in A$, s.t. $\exists l \in n.\phi : l \in \textit{eff}(a)$. That is, all actions that provide at least one literal in $n.\phi$.

Algorithm 1 shows the construction of the regression tree. The tree is constructed top down, starting from the root. Leaves are labeled by either $true$ or $false$. Every sequence of edges in every path from a $true$ leaf to the root in the regression tree corresponds to a plan that achieves ϕ — the formula at the root of the tree. During the construction of the tree, we may reach a node n that has an ancestor node n' for which $n.\phi \models n'.\phi$. This means a cycle has been identified in the regression tree, and we need not further develop n. To remove the cycle, we set $n.\phi$ to $false$.

Dependency-Preserving Projection

We now present our novel projection, which we call the *dependency-preserving projection* (DP projection). DP projection shares private dependencies without explicitly sharing private information. Unlike the local view projection, a DP projection is generated by all agents collaboratively, either sequentially, one agent at a time, or in parallel.

Intuitively, the DP projection contains all public facts, and replaces every public action a with a set of actions, $\alpha(a)$, where each action in $\alpha(a)$ represents executing a after a set of public actions that enable achieving a's private preconditions. More formally, let A_b be a set of public actions that facilitates the achievement of all private preconditions of a public action a. For each such set we create an action a_b, whose preconditions ensure that a is executed after the execution of all actions in A_b. The set $\alpha(a)$ is composed of all such actions a_b.

Ensuring that a is executed only after all actions in A_b is done by maintaining a set of facts $f_{a'}$ for every public action a', which is true if the effects of a' were achieved. Thus, the projection π_{DP} consists all public facts in π, a public fact f_a and a set of actions $\alpha(a)$ for every public action a.

Next, we present and explain the exact details of how each agent generates appropriate $\alpha(a)$ actions from its local view. The conjunction of the projected actions and facts by all agents produces the complete DP projection.

Creating the Regression Tree

For a public action a of agent i, we use regression to identify unique sets of public actions of i that enable the agent to execute a. Each of these sets of actions will be mapped to a single action in $\alpha(a)$ and added to π_{DP}. In more details, we construct a regression tree for $\textit{pre}(a)$ over a slightly revised local view of agent i in which all public actions (except for a) have no preconditions. This revised local view (denoted π_i^r) is used instead of agent i's local view (π_i) because in constructing $\alpha(a)$ we do not model the conditions required for executing other public actions. Rather, $\alpha(a)$ models how

Algorithm 1: Computing the regression tree $T_{rg}(\phi, A)$ for a conjunction of literals ϕ with a set of action A

```
1  Regress(φ, A)
       Input: φ, a conjunction of literals
       Input: A, a set of actions to regress with
       Output: n, the root of the regression tree
2      n ← a tree node
3      if φ = false then return false;
4      foreach ancestor n' of n do
5          if φ ⊨ n'.φ then
6              n.φ ← false
7              return n
8      if φ is not empty then
9          n.φ ← φ
10         foreach action a do
11             if eff(a) ∩ φ ≠ ∅ then
12                 n_a ← Regress(rg_a(n.φ), A)
13                 Add n_a to n.children
14     else
15         n.φ ← true
16     return n
```

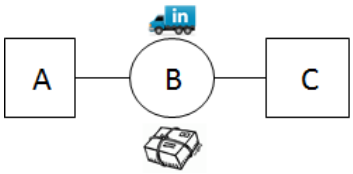

Figure 2: A simple logistics example.

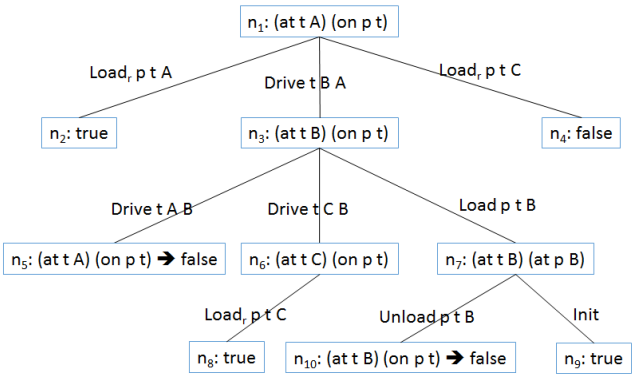

Figure 3: Partial regression tree for *(unload p t A)* in Fig. 2.

other public actions enable executing a.

Specifically, π_i^r is equal to π_i except that every public action a_p in π_i is replaced by a revised action a_p^r:

$$pre(a_p^r) = true \tag{5}$$

$$eff(a_p^r) = \left(\bigwedge_{l \in eff(a_p)} l \right) \wedge \left(\bigwedge_{l' \in pre(a_p), \neg l' \notin eff(a_p)} l' \right) \tag{6}$$

That is, a_p^r has no preconditions and has the effects of a_p. In addition, the preconditions of a_p are added to the effects of a_p^r if they are not deleted by a_p as these facts must be true after a_p was executed. This captures the state of the world immediately after a_p was executed. Consider, for example, an action a with precondition p and effect q. Immediately after a was executed, $(p \wedge q)$ must hold. Furthermore, this process helps us in detecting conflicts between public actions, as we later show in Example 0.1.

Every agent i uses the set of revised actions $\pi_i^r.A$ to construct a regression tree for every action a by calling Algorithm 1 with parameters $pre(a)$ and $\pi_i^r.A$. For brevity, we will denote the resulting regression tree as $T_{rg}(a)$. The branches of $T_{rg}(a)$ ending at leaves labeled by $true$ represent plans to achieve the precondition of a, taking into consideration agent i's local view and assuming that other agents' public actions can always be executed.

Example 0.1. Let us look at the smaller logistics example in Figure 2, where an agent has one private city B and two public logistic centers A and C. Figure 3 shows the regression tree constructed for action *(unload p t A)* with precondition *(at t A)* and *(on p t)*. As the revised action $load_r$ for the public *load p t A* action has the precondition *true* and effect *(on p t) (at t A)*, regressing through this action results in the *true* formula at node n_2. On the other hand, due to the conflict between *(at t A)* and the effect *(at t C)* of the revised action for *(load p t C)*, the regression result is *false* for node n_4 (following rule 2 in Definition 6). $n_5.\phi$ is satisfies the formula at n_1, and is hence replaced by *false*, denoting a cycle. The same holds for $n_{10}.\phi$, which satisfies $n_3.\phi$. The *true* leaves of the tree correspond to the plans *(load p t C), (drive t C B), (Drive t B A)*, *(init), (load p t B), (Drive t B A)*, and *(load p t A)*.

Creating Projected Actions

Using the regression tree described above, we can finally generate for a public action a of agent i the set of actions $\alpha(a)$ that will be added to the DP projection instead of a. Next we describe how $\alpha(a)$ is constructed.

Recall that every branch b in $T_{rg}(a)$ from a leaf n where $n.\phi = true$ to the root represents a plan for achieving $pre(a)$. Every such a branch b is mapped to a set of public actions A_b, which are all the public actions in b that achieve a private precondition for another action in b. For every set of public actions A_b we add a single action a_b to $\alpha(a)$. Executing a_b is intended to represent executing a after the actions in A_b have been executed. Thus, the preconditions of a_b must verify that the actions in A_b were executed.

To keep track of which public actions have been performed so far, we define a fact $f_{a'}$ for every action $a' \in A_b$. There is a single fact for every public action, used in all occurrences of a' in $\alpha(\cdot)$ of all agents. We also add a fact f_{init}, representing the execution of the $init$ action.

The preconditions of a_b are the public preconditions of a in conjunction with $\bigwedge_{a' \in A_b} f_{a'}$. The add effects of a_b are the effects of a in conjunction with setting f_a to be true, allowing other actions that depend on the execution of a, to be executed. We discuss the delete effects of a_b later on. Throughout the rest of this paper, we refer to actions in $\alpha(a)$ as *projected actions* and refer to the added facts $f_{a'}$ as *dependency facts*.

Note that the same set of public actions may appear in different branches of the regression tree, possibly in different orders. Thus, multiple branches may be represented by a single action in $\alpha(a)$. We intentionally group branches that have the same set of public actions and do not add a projected action for every branch in $T_{rg}(a)$ for privacy reasons. Imagine, for example, that a logistics agent can transfer a package from public location A to public location B in two possible routes, each passing through a different sequence of private locations, and thus, resulting in two different sequences of private and public actions. If the agent published two different projected actions, each corresponding to a different route, then the other agents would learn that the agent surely covers more than a single private location, thus revealing private information.

Removing Consumed Dependencies

Let a_b be a projected action and let b be a plan in the regression tree that generated a_b (i.e., the set of public actions that a_b represents). It may be that the actions in b or the action a itself consume some private effect e of a public action a'.

In that case, all other projected actions that rely on a' can no longer depend on the effects of a' to hold. To take this into account, we add $\neg f_{a'}$ to the delete effects of a_b. We denote by *delete*(a_b) the set of such public actions (actions whose private effects are deleted by other actions along b).[1] In summary, for a set of public actions A_b used in a branch b of the regression tree for action a we generate an action a_b such that:

$$pre(a_b) = public(pre(a)) \cup \bigwedge_{a' \in A_b} f_{a'} \quad (7)$$

$$eff(a_b) = public(eff(a)) \cup f_a \cup \bigwedge_{a' \in delete(a_b)} \neg f_{a'} \quad (8)$$

Example 0.2. Consider the regression tree given in Figure 3, created for the action *unload* p t A. The regression tree has three branches ending at *true* leaves – the branches ending at n_2, n_8, and n_9. We create three projected actions a_{n_2}, a_{n_8}, and a_{n_9} corresponding to these branches. The projected action a_{n_8}, for example, has precondition $f_{(load\ p\ t\ C)}$, because *(load p t C)* adds a private fact *(on p t)*. The effects of a_{n_8} are to add the facts *(at p A)* and $f_{(unload\ p\ t\ A)}$ and delete the fact $f_{(load\ p\ t\ C)}$, as it destroys the private effect *(on p t)* of the action *(load p t C)*.

Note that replacing location B with a set of connected private locations $B_1, ..., B_n$, would result in a different regression tree, but all the *true* leaves in this tree would correspond to either the package at A or C, or loading the package at its initial private location. Thus, the projection would be identical regardless of the number of private locations. We exploit this property of the DP projection in Section 11.

Constructing the DP Projection

Algorithm 2 summarizes the process of how the DP projection is constructed collaboratively by all agents. Every agent adds for each of its public actions a (1) a single dependency fact f_a and (2) the set of projected actions $\alpha(a)$. In addition, we add a dependency fact f_{init}, initialized to true, denoting the initialization of the private facts in the initial state of each agent to true. Importantly, we do not specify or share which private facts are true in the initial.

[1] As discussed above, a_b may represents multiple branches in the regression tree having the same set of public actions. In this case, *delete*(a_b) is the union of all public actions whose private effects were delete along the same branch they appeared in.

Algorithm 2: Computing the DP projection

```
1  DP-Project(π)
       Input: π, a privacy preserving MA-STRIPS problem
       Output: π^DP, a DP-projection of π
2      π_DP.P ← public π.P ∪ {f_init}
3      π_DP.I ← public π.I ∪ {f_init = true}
4      π_DP.G ← public π.G
5      π_DP.A ← ∅
6      foreach agent i = 1 ... k do
7          foreach action a ∈ public(A_i) do
8              Add f_a to π_DP.P
9              Add f_a = false to π_DP.I
10             Add α(a) to π_DP.A
11     return π_DP
```

Space Complexity

Next, we analyze the space complexity of our DP projection. In Algorithm 2, every agent i constructs for each of its public action a a single dependency fact f_a and a set of projected actions $\alpha(a)$. There is a single projected action in $\alpha(a)$ for every set of public actions that appear in the same branch in the regression tree of a and achieve a private fact. Since only actions of agent i can achieve its private facts, the number of projected action in $\alpha(a)$ is at most exponential in the number of agent i's public actions. Thus, the main complexity in the space complexity of our DP projection is the number projected actions. However, as we show later in our experimental evaluation, the size of the DP projection in all current benchmarks was manageable as the number of branches in the generated regression trees and the number of public actions along these branches were relatively small. This is because each regression tree only models the private dependencies of the action it was generated for, and not the private dependencies of other actions.

Application of a DP Projection

We employ the DP projection in a privacy preserving planner. First, the agents collaboratively generate the DP projection following Algorithm 2. Then, an off-the-shelf classical planner finds a solution to the DP projection, resulting in a high-level plan. Finally, each agent extends this high-level plan to be able to execute the public actions in it. This means each agent successively solves single agent planning problems, designed to ensure that the high level actions are executed in the designated order. Solving each of these problems is done using an off-the-shelf single agent planner.

Theoretically, it is possible that the generated high-level plan cannot be extended to a complete plan. While in our experiments this never happened, it is possible in case of such a failure to call a complete privacy preserving planner. We call the resulting planner the *DP-projection planner*, or simply DPP.

Preserving Object Cardinality Privacy

An algorithm is privacy preserving, as defined in Definition 3, if one can prove that it does not "reveal private information". Most research on privacy preserving planning considers a private information as revealed only if it is explicitly communicated to another agent. For example, if agent a_1 publishes during planning that it intends to bring a truck t to a private location loc, then clearly a_1 revealed the existence of this private location, as well as an ability to achieve the private fact *(at t loc)*, breaking the privacy constraint. On the other hand, if the agent only publishes that it can achieve a private fact with the arbitrary name p, then it is unclear what private information has been revealed. Thus, some privacy preserving MA-STRIPS planners are built on *obfuscating* the private information

they publish by applying some cryptographic tool (**?**; **?**; **?**).

As noted by Brafman (**?**), this form of privacy preserving is weak, as there is no constraint on what other agents can *infer* from the information sent during planning and during execution. For example, if the public plan consists of an agent a_i picking up a package and the pickup action requires a truck to be present at the location of the package, then all agents now know that a_i controls at least one truck. Recent work by Brafman (**?**) considered a stronger and well-defined form of privacy preserving.

Definition 7 (Strongly Private)**.** *A variable or a specific value of a variable is* strongly private *if other agents cannot deduce its existence from the information available to them.*

The information available to an agent is assumed to be (1) its local view, (2) the messages passed between the agents during planning, and (3) the sequence of public actions (of all agents) in the resulting plan. A multi-agent planning algorithm is said to be *strongly privacy preserving* if all private information remains strongly private (**?**).

While appealing, achieving such a strong form of privacy may be difficult. In fact, the only algorithm proven so far to have this strict form of privacy – a secure version of MAFS – is only guaranteed to preserve this privacy in a short list of specific domains (logistics, satellites, rovers) and under very restricted conditions (unit action cost and heuristic functions that ignore private actions).

Object-Cardinality Privacy In the creation of the DP-projection we proposed as well as in the corresponding algorithm DPP, some information about the dependencies between public actions is published. Thus, any planner using it cannot by strongly privacy preserving. On the other hand, the DP-projection does not share explicitly any private information, and thus it does preserve the more common weaker form of preserving privacy claimed by most privacy preserving planners. Next, we define stronger privacy preserving property in which the *cardinality of private objects* is strongly private (Definition 7), and prove that generating the DP projection preserves this property.

Definition 8 (Cardinality of Private Objects)**.** *Overloading previous notation, denote by $private_t(a_i)$ the set of objects of type t that are private for agent a_i. The cardinality of private objects of type t for agent a_i is $|private_t(a_i)|$.*

The notion of *objects* and *types* has been native to the planning domain description language (PDDL) for a while now, and is used in the recent multi-agent extension of PDDL (**?**) and in the official domain descriptions used in the recent CoDMAP (**?**). Briefly, objects and types are used to conveniently parameterize actions and facts, such that an action or a fact can be defined with respect to an object of a given type. For example, in a logistics domain, the trucks able to pick up packages are all objects of the type *truck*. Then, the pick up action and location fact can be defined once to all trucks, parameterized by an object of type *truck*. For example, the action *load* is parameterized by an object t of type truck and an object p of type package, and its preconditions include the fact that t is at the same location as p. Concretely, the privacy extension of multi-agent PDDL (**?**) supports defining for every agent a_i a list of private objects of any type t, which is exactly $private_t(a_i)$.

Hiding the cardinality of private objects is motivated by real-world scenarios. Consider, for example, the logistic problem above. It is realistic to assume that the agents that collaborate in the planning task, i.e., the various delivery companies, know that packages are delivered using trucks between logistic centers. On the other hand, it is likely that each company would like to hide its logistics capabilities, such as the number of trucks that it controls, or the number of private logistic centers it maintains.

Formally, let us denote by M_i the messages sent by agent i to other agents during planning, and denote by T the set of all object types.

Definition 9 (Preserving Cardinality of a Type)**.** *The cardinality of type t for agent i is preserved during a planning process if no other agent a_j can infer the value of $|private_t(a_i)|$ from the set of messages M_i.*

There are cases where the cardinality of private objects is revealed by the local view of other agents. Consider for example an agent a_i controlling two trucks. If other agents are aware of two separate actions for loading the same package at the same logistic center by a_i, then the other agents can easily infer that a_i controls at least two trucks, even if the names of the objects or the action are obfuscated. Hence, the cardinality of objects that participate in preconditions of public actions may be compromised by the domain description, even before planning commences. Thus, in our logistics example, agents can only hide the number of private locations that they control.

Definition 10 (Preserving object cardinality privacy)**.** *A planning algorithm preserves object cardinality privacy if for any agent and type $t \in T$ the cardinality of private objects of type t can only be revealed during the planning process if it could have already been inferred from the local view prior to planning.*

Clearly, generating the DP projection (Algorithm 2) does not violate the weak form privacy, in the sense that no private fact is shared between the agents. Theorem 1 shows that this process also preserves object cardinality privacy.

Theorem 1 (DP Projection Preserves object cardinality privacy)**.** *The process of generating the DP projection preserves object cardinality privacy, and sharing the DP projection in a planning algorithm does not break object cardinality privacy.*

Proof outline: The DP projection relies on the dependency tree for each public action a_p of agent i. Given a regression tree, one could add private objects of any type t and private actions that are parametrized by them such that they will not modify the number of *true* leaves, or the sequence of public actions in each *true* branch (see Example 0.2). Moreover, we could add any number of private objects of type t and private actions that would modify the number of leaves, yet each leaf in the new tree would contain the same set of public actions as a leaf in the original tree. In these cases, although the number of private objects was modified, the projection does not change. □

Domain	GPPP	MAPR -p	PMR	MAPlan/ FF+DTG	PSM- VRD	DPP
blocksworld	12	**20**	**20**	**20**	**20**	**20**
depot	11	0	0	13	17	**19**
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**	17
taxi	**20**	0	19	**20**	0	**20**
wireless	3	2	7	4	0	**9**
woodworking	18	0	0	16	**19**	**19**
zenotravel	**20**	**20**	18	**20**	13	**20**
sum	184	139	147	197	167	**224**

Table 1: Coverage results for a timeout of 30 minutes over the CoDMAP instances.

Domain	Action count Public	Private	Projection	Depth	Time
blocks	2448	0	23256	2	0.057
depot	6134	0	74174	3	0.142
driverlog	56448	1248	1110720	2	2.66
elevators	768	112	447747	11	498.2
logistics	264	52	480	3	0.019
rover	1196	1642	1196	3	1.53
satellite	3139	42099	3139	2	19.9
sokoban	1856	0	4172	2	0.665
taxi	153	0	153	2	0.001
wireless	5814	0	113414	3	7.98
woodworking	8226	0	8226	2	0.168
zeno-travel	288	1098	1008	3	0.266

Table 2: Projection computation metrics over a large problem from each domain.

Experimental Results

We experimented with benchmarks from the 2015 CoDMAP competition (**?**).[2] We run DPP on a 2.66 GHz machine with 4 cores and 8 GB of memory. Note taht while the computer had multiple cores, we implemented DPP to run on a single core. DPP was implemented using the Fast-Downward (FD) planner (**?**) for the high-level planning and the FastForward (FF) planner (**?**) for extending high-level plans. FD was configured to use preferred operators, deferred heuristic evaluation, and two heuristics: FF and a landmark-based heuristic. This is a common configuration used also by the LAMA planner (**?**). We compare DPP with the best performing and most relevant privacy-preserving planners: GPPP (**?**), MAPlan/FF+DTG (an extension of the MAFS algorithm (**?**) using the FF and DTG heuristics together), MAPR-p, PMR (**?**), and PSM-VRD (**?**; **?**). Details on these planners can be found in the CoDMAP website.

the domains in the competition. Our projection approach – DPP – solves more problems than all other approaches. In 7 of the 12 domains, DPP solved all instances, and in 2 more it solved 19 out of 20 problems. In all domains DPP is either the best performing algorithm (in terms of coverage) or within 3 problems of the best performing algorithm. This is in contrast to its closest rivals – GPPP and MAPlan – where GPPP is non-competitive in 4 domains and MAPlan in 2 domains. Thus, DPP shows robustness across all domains.

wireless. This is because the wireless domain has many dead-ends that are difficult to escape. Both FF and FD do not have a sufficiently strong mechanism to detect and avoid dead-ends. Still, in this domain too, DPP is the best-preforming planner.

DPP and all other planners is that once the projection was constructed we use an off-the-shelf classical planner, rather than a coordinated joint search mechanism. Even the construction of the DP projection can be easily distributed, having each agent add its projected actions and dependency facts. Our results point to the strengths of the mature classical planners over the rather new joint search mechanisms, such as GPPP (**?**) and MAFS (**?**), which is used by MAPlan.

We further analyze the actual size of the DP projection and runtime of creating it (using Algorithm 2). We chose a challenging problem on each domain, and compute various metrics during the projection construction. Table 2 reports the size of the original problem, in terms of the number of public and private actions of all agents, and the number of actions in the projection. We further report the maximal depth of a regression tree in our experiments, as well as the time it took to compute the projection in seconds. Obviously, in domains where there are no private actions, our method is extremely fast. Satellite instances took much time, mainly due to the large number of private actions. Still, there is minor dependencies between the actions, resulting in shallow regression trees with a large branching factor.

more difficult instances of this domain contain private floors, that only a single elevator can reach. As boarding and exiting various quantities of passengers in floors requires different actions, there are many possible combinations for achieving the preconditions of actions, as reflected by the maximal depth of the regression trees in this domain. Hence, this is the only domain in the set of benchmarks that poses a true challenge to the regression tree computation.

ignore the identity of passengers, and focus only on their number, we can learn more general regression trees faster. Another method for scaling up would be to use an approximate regression mechanism, by, e.g., regressing only one precondition at a time.

Conclusion

process in which some information about private dependencies is shared while privacy is preserved. We showed that the DP projection also preserves a strong form of privacy in which the cardinality of private objects of a given type cannot be inferred by any adversary agent. The benefit of our DP projection is demonstrated in a planner that effectively uses the DP projection to solve more benchmark problems than any other state-of-the-art privacy preserving planner.

the DP projection as a heuristic for guiding the search of a complete solver, such as MAFS.

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

[2] http://agents.fel.cvut.cz/codmap/results/

Gurion University of the Negev.