Stubborn Sets Pruning for Privacy Preserving Planning

Tim Schulte
Albert-Ludwigs-Universität Freiburg
schultet@cs.uni-freiburg.de

Abstract

We adapt a partial order reduction technique based on *stubborn sets* to the setting of privacy-preserving multi-agent planning. We prove that the presented approach preserves optimality and show experimentally that it can significantly improve search performance on some domains.

Introduction

Recently, privacy preserving planning (Nissim and Brafman 2014) has become an increasingly popular multi-agent planning framework. It enables agents to engage in a cooperative planning process in order to compute joint plans that achieve mutual goals. Notably, the framework allows agents to keep certain information private. There are many settings in which this is of great importance. Consider, for instance, research departments of different companies that want to collaborate on a common project in order to mutually benefit from each others' competence. Exchanging proprietary data could diminish the benefits of this endeavor.

Heuristic search is a particularly successful approach to privacy-preserving planning. Specifically, multi-agent forward search (MAFS) (Nissim and Brafman 2012) has proven to be highly efficient, when coupled with good heuristic functions (Štolba and Komenda 2014; Štolba, Fišer, and Komenda 2015). However, when accurate heuristic estimates are unavailable, the search space is often searched exhaustively (e.g. when the search gets stuck on a plateau). Even with almost perfect heuristic estimates, search effort can scale exponentially (in the size of the planning task), when an optimal solution is sought (Helmert and Röger 2008). In these cases, additional pruning techniques that narrow down the number of state expansions, while preserving optimality, can substantially improve the search performance.

Partial order reduction (POR) techniques exploit that independent actions can be applied in an arbitrary order. Ideally, search algorithms would consider only one such order, thereby reducing the number of expanded states exponentially. Partial order reduction based on *stubborn sets* (Valmari 1989) strives to achieve just that and has successfully been applied to optimal (single agent) planning (Alkhazraji et al. 2012; Wehrle et al. 2013). In this paper we adapt and apply stubborn sets pruning to the privacy-preserving planning setting. The main challenge addressed is how to account for private information without losing completeness or optimality. We show experimentally that the revised algorithm can significantly improve search performance.

As a running example, we use a new domain, inspired by a production site. The goal is to produce a set of products with certain properties. The agents must process the products to establish their required properties. Each property has a corresponding processing action, all of which are private and independent of one another. A concrete example that embodies this type of domain has the agents building personal computers according to a given set of orders. Each order specifies an individual PC setup, i.e. the set of components the PC should consist of. Many components, like hard disc drives, physical drives, sound card, working memory, etc., can independently be installed onto the mainboard.

Background

We consider multi-agent planning in a notational variant of the privacy-preserving planning formalism (Nissim and Brafman 2014). The formalism extends *classical planning* with a notion of *agents*, their respective *action sets*, and a *privacy partition*.

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

- $N = \{1, 2, \ldots, n\}$ *is a finite set of agents,*
- V *is a finite set of* state variables*. Each* $v \in V$ *is associated with a domain* D_v*. A* variable assignment *is a function* s *with domain* $D_s \subseteq V$*, such that* $s(v) \in D_v$ *for all* $v \in D_s$*. A* variable assignment *defined for all variables in* V *is called* state*.*
- s_0 *is the* initial state*,*
- $s_\star$ *is a variable assignment over* V *called the* goal*,*
- A_i *is a finite set of actions available to agent* i*. Each* action $a = \langle \mathrm{pre}(a), \mathrm{eff}(a), \mathrm{c}(a)\rangle \in A_i$ *consists of two variable assignments over* V *called* precondition $\mathrm{pre}(a)$ *and an* effect $\mathrm{eff}(a)$*, and a* cost $\mathrm{c}(a) \in \mathbb{R}_0^+$*. The set of all actions is* $A = \bigcup_{i\in N} A_i$*.*

An action a is *applicable* in state s if s agrees with *pre*(a) wherever *pre*(a) is defined. Application of action a in state s yields the *successor state* $a(s)$ which agrees with *eff*(a)

where *eff*(a) is defined, and agrees with s, elsewhere. The set of all applicable actions in state s is $app(s)$. The solution to a planning task is a sequence of actions $\pi = (a_1, \dots, a_k)$ such that a_1 is applicable in s_0, every subsequent action is applicable in the state generated by its preceding action, and $a_k(\dots(a_1(s_0))\dots) \models s_\star$.

Multi-agent planning tasks can be conceived as "agent-decoupled" classical planning tasks, and are solvable by centralized classical planning systems like Fast Downward (Helmert 2006). Some settings require agents to preserve privacy during the planning process. By constraining the agents to keep certain information on the planning task private, the use of distributed planning techniques becomes sensible. We now introduce the required notation to then define the *privacy-preserving* extension to multi-agent planning.

Definition 2 (Projection)**.** *Let s be a variable assignment over the set of variables V. The projection of s to $V' \subseteq V$ is a variable assignment $s|_{V'}$ that is defined on V' and agrees with s wherever it is defined, i.e. $s|_{V'}(v) = s(v)$, for all $v \in V'$.*

Definition 3 (Action projection)**.** *The* projection *of an action a to the set of variables V' is $a|_{V'} = \langle \mathrm{pre}(a)|_{V'}, \mathrm{eff}(a)|_{V'}, \mathrm{c}(a)\rangle$.*

Consequentially, the projection of a set of actions A to the set of variables V' is defined as $A|_{V'} = \{a|_{V'} | a \in A\}$.

Definition 4 (Privacy partition)**.** *Let $\Pi = \langle N, V, s_0, s_\star, \{A_i\}_{i\in N}\rangle$ be a multi-agent planning task. A* privacy partition *is an indexed family of sets*

$$\mathcal{P} = \{P_v\}_{v \in V}$$

that, for each variable $v \in V$, contains the set of agents $P_v \subseteq N$ that have access to v.

In this paper, we only consider privacy partitions where all sets $P_v, v \in V$ have a cardinality of either one or $|N|$. Furthermore, if $v \in D_{s_\star}$ then $P_v = N$. Thus, $\mathcal{P}$ partitions the set of variables V into a set of public *variables V^{pub}, known to all agents, and $|N|$ sets of* private *variables V_j^{pri}, each known to a single agent $j \in N$ only:*

- $V_j^{pri} = \{v \in V \mid P_v = \{j\}\}$, *for* $j \in N$
- $V^{pub} = \{v \in V \mid P_v = N\}$

Actions are partitioned into a set of public actions A^{pub} and sets of private actions A_j^{pri}, accordingly:

- $A_j^{pri} = \{a \in A_j \mid a = a|_{V_j^{pri}}\}$, *for* $j \in N$
- $A^{pub} = \bigcup_{j\in N} (A_j \setminus A_j^{pri})$

Definition 5 (Local view)**.** *Let $\Pi = \langle N, V, s_0, s_\star, \{A_i\}_{i\in N}\rangle$ be a multi-agent planning task and $\mathcal{P}$ be a privacy partition for Π. The* local view *of agent j on Π is defined as*

$$\Pi^j = \langle N, V^j, s_0^j, s_\star, \{A_i^j\}_{i\in N}\rangle, \text{ where}$$

- $V^j = V^{pub} \cup V_j^{pri}$,
- $s_0^j = s_0|_{V^j}$, *and*
- $A_i^j = (A_i \setminus A_i^{pri})|_{V^j}$ *for* $i \neq j$, *and* $A_j^j = A_j$.

Definition 6 (Privacy preserving planning task)**.** *A* privacy preserving planning task *is a tuple $(\Pi, \mathcal{P})$ consisting of a multi-agent planning task Π and a privacy partition $\mathcal{P}$.*

A multi-agent planning algorithm is *weakly private* if each agent can only access its own *local view* on the planning task and the agents never exchange private information with one another. A multi-agent planning algorithm is *strongly private* if no agent can deduce private information from the course of conversation (message history) between the agents. Private information includes knowledge about the existence or value of a variable private to another agent, or an action model (Brafman 2015).

Multi-Agent Forward Search

Because agents can only access a factor (their local view) of the original multi-agent planning task, cooperation with other agents becomes a necessity.

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 i receives a message $m = \langle s, h_j(s), g_j(s)\rangle$ of some other agent j, it checks whether s is already in its open or closed list. If this is not the case, agent i puts s on its open list. If agent i generated state s previously with higher cost, then 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.

Strong Stubborn Sets

Strong stubborn sets can be used within forward search algorithms to potentially reduce the number of successor states generated in each *state expansion* step. Instead of expanding a state s by generating a successor state $a(s)$ for each applicable action $a \in app(s)$, only a subset of actions $T_{app(s)} \subseteq app(s)$ needs to be considered. Applicable actions that are not contained in $T_{app(s)}$ are said to be *pruned*.

In the following, we provide the definitions of *action dependencies*, *disjunctive action landmarks* (Helmert and Domshlak 2009), and *necessary enabling sets*, which are the three crucial components for the computation of strong stubborn sets.

Definition 7 (Action dependency)**.** *Let $\Pi = \langle N, V, s_0, s_\star, \{A_i\}_{i\in N}\rangle$ be a multi-agent planning task, and let $a_1, a_2 \in A$.*

- *a_1* disables *a_2 if there exists a variable $v \in V$ and facts $\langle v, d_1\rangle \in \mathrm{eff}(a_1)$ and $\langle v, d_2\rangle \in \mathrm{pre}(a_2)$ s.t. $d_1 \neq d_2$.*
- *a_1 and a_2* conflict *if there exists a variable $v \in V$ and facts $\langle v, d_1\rangle \in \mathrm{eff}(a_1)$ and $\langle v, d_2\rangle \in \mathrm{eff}(a_2)$ s.t. $d_1 \neq d_2$.*

Algorithm 1: Strong stubborn set computation of agent i for state s (incomplete)

Input: $\Pi = \langle N, V, s_0, s_\star, \{A_i\}_{i \in N} \rangle$, state s
Result: strong stubborn set $T_s \subseteq A$

```
T_s ← L_s^{s⋆} for some DAL L_s^{s⋆} for s_⋆ in s
repeat
    forall a ∈ T_s do
        if a ∈ app(s) then
            T_s ← T_s ∪ dep(a)
        else
            T_s ← T_s ∪ N_s^a for some NES N_s^a
until T_s reaches a fixed-point
return T_s
```

- *a_1 and a_2 are dependent if a_1 disables a_2, or a_2 disables a_1, or a_1 and a_2 conflict. We write $\mathrm{dep}(a)$ for the set of actions with which a is dependent.*

Definition 8 (Disjunctive action landmark)**.** *A* disjunctive action landmark (DAL) *for a set of facts F in state s is a set of actions L such that every applicable action sequence that starts in s and ends in $s' \supseteq F$ contains at least one action $a \in L$.*

Definition 9 (Necessary enabling set)**.** *A* necessary enabling set (NES) *for action $a \notin app(s)$ in state s is a disjunctive action landmark for $\mathrm{pre}(a)$ in s.*

We can now give the definition of strong stubborn sets. It is identical to the one used in classical single-agent planning (Alkhazraji et al. 2012) except for the input data now being a multi-agent planning problem.

Definition 10 (Strong stubborn set)**.** *Let Π be a multi-agent planning task with actions A and goal $s_\star$, and let s be a state of Π. A* strong stubborn set (SSS) *in s is an action set $T_s \subseteq A$ such that:*

1. *For each $a \in T_s \cap app(s)$, we have $\mathrm{dep}(a) \subseteq T_s$.*
2. *For each $a \in T_s \setminus app(s)$, we have $N_s^a \subseteq T_s$ for some necessary enabling set N_s^a of a in s.*
3. *T_s contains a disjunctive action landmark for $s_\star$ in s.*

The above definition of strong stubborn sets ensures that for every plan π for the current state s, a permutation of π exists which is not pruned. An algorithm to compute a strong stubborn set for a state s is given in Algorithm 1. The state expansion step of forward search algorithms can be modified in the following way: before expanding state s, compute the respective strong stubborn set T_s using Algorithm 1, then expand s by applying the actions in $T_{app(s)} := T_s \cap app(s)$ only. As a consequence, states reached by actions in $app(s)$ but not in T_s are pruned.

Strong Stubborn Sets Revised

We now exemplify two ways in which the pruning of successor states based on strong stubborn sets, as defined above, violates completeness when used in combination with a privacy preserving distributed planning approach, like MAFS. We then propose two possible adaptations that make up for the identified shortcomings and argue that the revised stubborn set approach maintains the completeness property of MAFS.

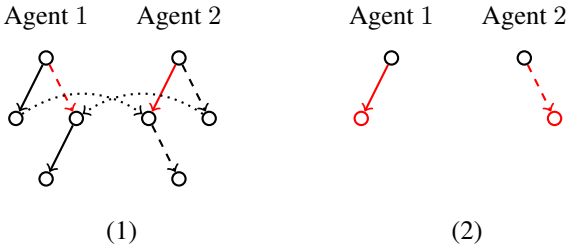
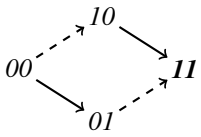


Figure 1: Two ways in which planning for the task of Example 1 goes wrong. Agent 1's action is represented by a solid arrow, agent 2's action by a dashed arrow. Dotted arrows represent state transmissions. Pruned actions are marked red.

The first example fails despite the absence of private information. The second example fails because required information is private to another agent.

Example 1. *Let $(\Pi, \mathcal{P}) = (\langle N, V, s_0, s_\star, \{A_1, A_2\} \rangle, \mathcal{P})$ be a privacy preserving planning task, with*



$$N = \{1, 2\}, V = \{v_0, v_1\}$$
$$\mathcal{P} = \{N, N\}$$
$$s_0 = \{v_0 = 0, v_1 = 0\}$$
$$s_\star = \{v_0 = 1, v_1 = 1\}$$
$$A_1 = \{a\} \text{ with } a = \langle v_0 = 0, v_0 = 1 \rangle$$
$$A_2 = \{b\} \text{ with } b = \langle v_1 = 0, v_1 = 1 \rangle$$

There is no private information (all variables are known to both agents), hence, the agents' public projections are identical, i.e. $\Pi^1 = \Pi^2 = \Pi$. When expanding a state, each agent independently applies stubborn set pruning (Algorithm 1) to potentially reduce the number of generated successor states. Ideally, the agents would either prune state 10 *or state* 01.

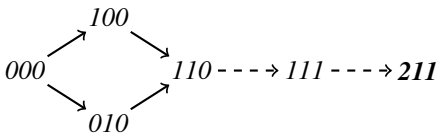
Because actions a and b are independent of one another and both are applicable in the initial state, the respective strong stubborn set contains either a or b (but not both), depending on the choice of the initial disjunctive action landmark. When both agents choose the same DAL, they would virtually agree on either pruning state 01 *or state* 10 *(which is the desired outcome). If the agents select different DALs, however, this leads to the following undesirable outcomes, depicted in Figure 1:*

(1) The agents end up generating all states. This happens because both of them prune the other's initial action. Therefore, they generate different successor states, which they then transmit to the respective other agent.

(2) The agents end up in a livelock*, waiting for one another to apply the first action (which they pruned) and to transmit the resulting state (which never happens).*

In this example, planning fails because the agents do not synchronize their pruning efforts, using different strong stubborn sets, and prune "public" successor states, relevant to the other agent. We can resolve this issue by enforcing that the stubborn sets computed by each agent only include the agent's own actions. That way, the agents would not prune their own public action in the initial state and therefore not end up in a livelock. We will explain this in more detail further below and continue by emphasizing another issue that occurs when dealing with private information.

Example 2. *Let* $(\Pi, \mathcal{P}) = (\langle N, V, s_0, s_\star, \{A_1, A_2\}\rangle, \mathcal{P})$ *be a privacy preserving planning task, with*



$$N = \{1, 2\}, V = \{v_0, v_1, v_2\}, \mathcal{P} = \{N, N, \{2\}\}$$
$$s_0 = \{v_0 = 0, v_1 = 0, v_2 = 0\}$$
$$s_\star = \{v_0 = 2\}$$
$$A_1 = \{a, b\}, A_2 = \{c, d\} \text{ with}$$
$$a = \langle v_0 = 0, v_0 = 1\rangle, b = \langle v_1 = 0, v_1 = 1\rangle$$
$$c = \langle v_0 = 1 \wedge v_1 = 1, v_2 = 1\rangle, d = \langle v_2 = 1, v_0 = 2\rangle$$

Here, agent 1 *can savely prune either action* a *or* b *in the initial state, thereby avoiding either state* 100 *or* 010*, respectively. This, however, does not work out, as both agents are planning with their local view. Consider the local view of agent* 1*:*

$$N = \{1, 2\}, V^1 = \{v_0, v_1\}$$
$$s_0^1 = \{v_0 = 0, v_1 = 0\}, s_\star = \{v_0 = 2\}$$
$$A_1^1 = \{a, b\}, A_2^1 = \{c|_{V^1}, d|_{V^1}\} \text{ with}$$
$$a = \langle v_0 = 0, v_0 = 1\rangle, b = \langle v_1 = 0, v_1 = 1\rangle$$
$$c|_{V^1} = \langle v_0 = 1 \wedge v_1 = 1, \emptyset\rangle, d|_{V^1} = \langle \emptyset, v_0 = 2\rangle$$

To agent 1 *there appears to be no connection between agent* 2*'s actions, i.e.* c *and* d *appear to be independent. Furthermore, action* d *appears to be applicable in the initial state. Applying Algorithm 1 in* s_0 *therefore yields the following strong stubborn set* T_s*:*

$$T_s = \{d\}$$

Hence:

$$T_{app(s_0)} = app(s_0) \cap T_s = \{a, b\} \cap \{d\} = \emptyset$$

Since agent 1 *has no action to apply in its initial state, no goal can be reached and completeness is violated.*

Initially, agent 1 *computes a disjunctive action landmark for the goal (Algorithm 1, line 1). The only action satisfying a goal condition is action* d *of agent* 2*. The stubborn set therefore initially consists of action* d *only. According to Algorithm 1 either the set of dependent actions* $\mathrm{dep}(d)$ *has to be added to* T_s *(if* d *is applicable in* s_0*) or a necessary enabling set for* d *(if* d *is not applicable in* s_0*). Neither is possible for agent* 1*. Additionally, agent* 1 *cannot decide correctly, whether action* d *is applicable in* s_0 *or not. This would require knowledge of private information not available to agent* 1*. In the example, agent* 1 *adds all dependent actions instead of a necessary enabling set for* d*. Since action* d *is not dependent on any other action in agent* 1*'s local view, a fixed point is reached and the algorithm returns* $T_s = \{d\}$*. Even if agent* 1 *could decide that action* d *was not applicable in* s_0*, he could not add a valid necessary enabling set for* d*. The only enabling action is* c *and it enables a private precondition of* d *that is not visible to agent* 1*. Hence, the NES would be empty. (As we have already seen, computing the dependent actions is error prone for the same reasons.)*

Algorithm 2: Strong stubborn set computation of agent i for state s (revised, complete)

Input: $\Pi = \langle N, V, s_0, s_\star, \{A_j\}_{j\in N}\rangle$, state s
Result: strong stubborn set $T_s \subseteq A$

```
T_s ← {a ∈ A_i^pub | a|_{V^pub} appl. in s}
repeat
    forall a ∈ T_s do
        if a ∈ app(s) then
            T_s ← T_s ∪ (dep(a) ∩ A_i)
        else
            T_s ← T_s ∪ (N_s^a ∩ A_i) for some NES N_s^a
until T_s reaches a fixed-point
return T_s
```

Again, we observe that the inclusion of other agents' (publicly projected) actions during the strong stubborn set computation is problematic. For these actions, the agent cannot compute disjunctive action landmarks or necessary enabling sets correctly. Simply excluding other agents' actions from the stubborn set computation, as the solution to the first example suggests, does not resolve the entire issue. In Example 2, agent 1 has no action to satisfy a goal condition, hence, the initial DAL would be empty when simply omitting agent 2's actions.

Having other agents' actions in the stubborn sets is only a superficial cause of failure, while the real problem is deeper. The ultimate cause is that public actions, which are part of a plan, are pruned. States created by public actions resemble potential interaction points between the agents. An agent cannot decide, whether these states are part of a plan leading to a goal or not, because they have only partial knowledge of the other agents' actions. Consequently, we need to alter the definition of strong stubborn sets in the context of privacy preserving planning. Instead of requiring the stubborn sets for state s to contain a disjunctive action landmark for a goal condition (Definition 10, point 3), we constrain them to contain the set of all public actions that are reachable from s by a (potentially empty) sequence of private actions. This definition of strong stubborn sets ensures that for every sequence of actions π, starting in the current state s and end-

ing with a public action, a permutation of π exists which is not pruned. All successor states created by public actions are therefore preserved. Furthermore, we can now savely restrict the stubborn set computation to include only the agents' own actions.

Consider the following, revised definition of strong stubborn sets:

Definition 11 (Strong stubborn set for privacy preserving planning)**.** *Let $(\Pi, \mathcal{P})$ be a privacy preserving planning task and let s be a state of Π. A* strong stubborn set *for agent i in s is an action set $T_s \subseteq A_i$ such that:*

1. *For each $a \in T_s \cap app(s)$, we have $\mathrm{dep}(a) \cap A_i \subseteq T_s$.*
2. *For each $a \in T_s \setminus app(s)$, we have $N_s^a \cap A_i \subseteq T_s$ for some necessary enabling set N_s^a of a in s.*
3. *T_s contains all actions $a \in A_i^{pub}$, such that $a|_{V^{pub}}$ is applicable in s.*

Note how the first two constraints are only subtly different from Definition 10 restricting the included actions to belong to agent i only, while the third constraint ensures that all interaction points are preserved. Algorithm 2 computes stubborn sets consistent with the above definition.

Consider Example 1 again. According to Algorithm 2 each agent initially adds its own public action (line 1). Since they are both applicable in the initial state, all dependent actions are added in the next step (line 4, 5). There are none, hence, the computation reaches a fixed point and the stubborn sets are returned: $T_{00} = \{a\}$ for agent 1 and $T_{00} = \{b\}$ for agent 2. We end up with case (1) depicted in Figure 1. Similarly, in Example 2 agent 1 adds actions a and b to the initial strong stubborn set. Since these two actions are both applicable in the initial state and there are no dependent actions, the computation finishes returning $T_{000} = \{a, b\}$.

In both examples completeness is retained at the expense of pruning capacity. Since an agent cannot savely prune public actions, the pruning potential is restricted to permutations of private action sequences. We now discuss some theoretical properties of the presented stubborn set pruning technique.

Privacy

SSS for privacy preserving planning strives to reduce each agents individual search space without introducing any additional communication. It never transmits a state that is not transmitted by the respective planning algorithm without SSS pruning. We therefore believe that the presented technique is strongly privacy preserving.

Optimality

First, we define the terminology used in the proof.

Definition 12 (Public step)**.** *A* public step *in state s is a sequence of actions πa, where*

- *a is a public action of agent i and*
- *π is a* minimal plan *from s to $\mathrm{pre}(a)$, i.e. $\pi[s] \models \mathrm{pre}(a)$, that consists of private actions of agent i only.*

A plan π from s to $pre(a)$ is minimal*, if there is no subsequence π'' of π that can be moved behind action a, such that $\pi a[s] = \pi' a \pi''[s]$, where π' is the sequence π without π''.*

A public step can be thought of as a sort of "macro action" that encapsulates the execution of private actions followed by a single public action.

Definition 13 (Public state)**.** *A state s is called* public state *if it is reachable from the initial state by a sequence of public steps.*

Lemma 1. *Let $(\Pi, \mathcal{P})$ be a privacy preserving planning problem and $\pi = (a_1, a_2, \ldots, a_k)$ be a solution to Π. Then, there exists a permutation $\pi' = (a'_1, a'_2, \ldots, a'_k)$ of π, such that for all pairs of consecutive public actions[1] a'_i, a'_j in π', $(a'_{i+1}, a'_{i+2}, \ldots a'_j)$ is a public step.*

Proof. Let $\pi = (a_1, a_2, \ldots, a_k)$ be a solution to Π, such that every private action in π is followed by another action (public or private) of the same agent. Only considering solutions of this type preserves optimality and completeness (Nissim and Brafman 2014). Assume that, between two consecutive public actions a_i and a_j we have a sequence of actions (of the same agent) $\pi_{i..j} = (a_{i+1}, a_{i+2}, \ldots, a_j)$ that is not a public step. Then, there must be a subsequence in $\pi_{i..j}$ that can be moved behind a_j. By moving this subsequence behind a_j, just before the next sequence of actions of the same agent, we create a permutation π'' that is a legal plan. Repeating this process until all inconsistencies have been removed yields a plan π' that is a permutation of π and that consists of public steps only. □

Lemma 2. *Restricting the successor generation to a SSS (according to Def. 11) in every state is optimality preserving for privacy preserving planning.*

Proof. Let $(\Pi, \mathcal{P})$ be a privacy preserving planning task. The proof is by induction over $k \in \mathbb{N}$, where S_k is the set of public states that are reachable in at most k public steps from the initial state and S'_k is the set of public states that are reachable in at most k public steps when stubborn set pruning is applied. We show that $S_k = S'_k$ for all k. (It suffices to consider public states instead of all possible states because of Lemma 1.)

The initial state s_0 is reachable by an empty sequence of actions (zero public steps), therefore, $S_0 = \{s_0\} = S'_0$.

Let the set of reachable states expand from S_{k-1} to $S_k \supset S_{k-1}$. For each new state $s^* \in S_k \setminus S_{k-1}$, a state $s \in S_{k-1}$ must exist from which s^* is reachable, in a single public step. Therefore, there must be a public state $s \in S_{k-1}$ and a public step πa, such that $\pi a[s] = s^*$. Let i be the agent, such that $a \in A_i^{pub}$.

According to the induction hypothesis $S_{k-1} = S'_{k-1}$, it holds that $s \in S'_{k-1}$. We argue that SSS preserves a public step (of agent i) σa, such that $\sigma a[s] = s^*$. Observe that a is included in T_s for agent i since $a \in A_i^{pub}$ and its public projection $a|_{V^{pub}}$ is applicable in s (Definition 11, point 3).

If a is applicable in s, i.e. $a(s) = s^*$, then $s^* \in S'_k$. If a is not applicable in s, then a necessary enabling set for a must be contained in T_s (Definition 11, point 2). That is, a

[1] By *consecutive public actions* we mean that there are no other public actions between a'_i and a'_j. There might be private actions in between, however.

	blind		goalcount		FF	
Domain	def	sss	def	sss	def	sss
blocksworld	0	0	1	1	0	**1**
depot	2	2	**6**	4	0	0
driverlog	7	7	**17**	16	16	16
elevators	**3**	2	20	20	12	**14**
logistics	3	3	**18**	14	**17**	15
rovers	20	20	19	**20**	**20**	18
satellites	3	3	20	20	**20**	19
sokoban	**2**	0	2	**4**	7	7
taxi	6	**8**	11	**13**	2	2
wireless	0	0	0	0	**2**	1
woodworking	**2**	1	**2**	1	**2**	1
zenotravel	5	5	**20**	16	**16**	14
prod. site	0	**20**	11	**20**	8	**20**
Total	53	**71**	147	**149**	122	**128**

Table 1: Benchmark results.

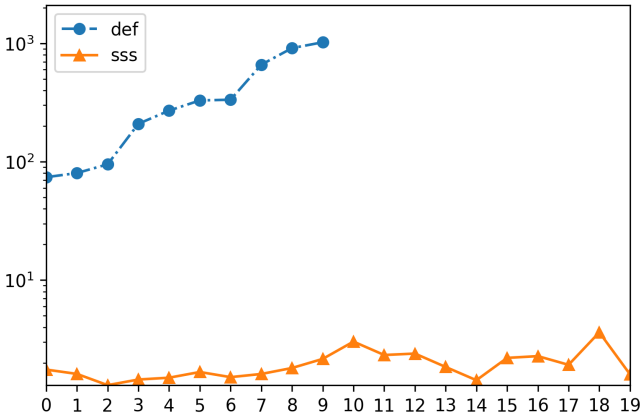


Figure 2: Runtime in seconds for increasingly difficult instances of the production site domain. Both configurations used the goalcount heuristic.

disjunctive action landmark for $pre(a)$ in s. The stubborn sets generated for s according to Definition 11 correspond to the stubborn sets generated for s according to Definition 10 when planning towards the goal $s_\star = pre(a)$ with the set of actions $A = A_i$. Since strong stubborn sets consistent with Definition 10 are optimality and completeness preserving (Alkhazraji et al. 2012), a permutation σ of π must be preserved, such that $\pi a[s] = s^* = \sigma a[s]$. Hence $s^* \in S'_k$. □

Evaluation

The presented algorithms were implemented in a distributed multi-agent planning system written in Go. Experiments were run on a 2.6 Ghz Intel Xeon 8-core CPU. Each problem instance used a single core and 8 GB of RAM, shared by all agents.

We experimented with the benchmarks from the CoDMAP competition (Štolba, Komenda, and Kovacs 2015) consisting of 12 domains with 20 problems each. Of the new *production site* domain 20 problem instances of varying difficulty were included in the benchmarks. Planning time was limited to 30 minutes per problem instance. Table 1 shows coverage results for the tested configurations, while Figure 2 shows the running time for the production site instances.

Production site domain. While plain MAFS solves 0, 11 and 8 instances of the production site domain when using the blind, goalcount and FF heuristic (Hoffmann and Nebel 2001) respectively, MAFS with stubborn set pruning solves all 20 instances, independent of the heuristic used.

Blind MAFS resembles depth-first search and chains together random sequences of actions, most of which do not lead to a goal. Due to the expansive search space, even the easiest instances cannot be solved.

The goalcount and FF heuristics, on the other hand, both guide MAFS towards states with as many subgoals satisfied as possible. That way, the search focuses on one subgoal, or product, after the other and the number of generated states is reduced decisively. Although this behavior seems to be favorable, it has its own shortcomings. The heuristics cannot differentiate between two states in which the same number of subgoals are satisfied, even if one state is significantly closer to satisfying another subgoal than the other. The reason for this is that the heuristics are computed based on each agent's local view. The public actions of other agents establish a subgoal (finish a product) with a cost of one and appear to always be applicable, because their public projections do not include their private preconditions. Because of this heuristic inaccuracy, states that satisfy a larger number of subgoals but which do not lead to a goal are preferred to states that lead to a goal but satisfy fewer subgoals. Processing actions, for instance, cannot be undone. Therefore, if a product is processed in a way not consistent with its goal requirements, the agent cannot finish that product. The respective subgoal can then only be supplied by another agent. If no agent can supply the subgoal, the search has to backtrack to a state in which the faulty processing action has not been applied yet.

This problem does not occur in the stubborn set pruning variant. Counter-productive processing actions that prevent a product from being finished are always pruned. These actions are independent of the other processing actions and therefore may only be included in the stubborn set if they establish a precondition of the public action that finishes the product. Stubborn set pruning therefore effectively restricts the search to consider only such states that can be extended into a goal state. Furthermore, each agent must consider only a single permutation of processing actions to finish each product, where otherwise exponentially many permutations would have to be considered. When FF or goalcount heuristic is used, the stubborn set approach also focuses on one subgoal after the other. The generated plans encourage the division of labor between the agents, each creating a subset of the products, rather than one agent creating them all. Furthermore, plans are found very fast, as all parts of the search

space that do not progress towards a goal are pruned. Figure 2 highlights this fact.

CoDMAP domains. Regarding the CoDMAP domains, the results are mixed. There are no major differences in coverage between the strong stubborn set approach and regular MAFS, although overall the latter configuration solves a few problems more. We believe that this is due to the additional computations required for computing the stubborn sets. Interestingly, some domains seem not to benefit from the stubborn sets based partial order reduction at all.

A possible explanation is that these domains already internalize a form of POR by decoupling the planning task in such a way that each agent has its own individual responsibilities. If in the production site domain each agent had a single processing action only, there would be as good as no pruning potential. This is exactly what we find in some of the CoDMAP domains. The *woodworking* domain is a good example of such an agent decoupling. Here, most of the agents can only perform a single action.

Another explanation is that the pruning potential cannot be exploited, because the agents compute their stubborn sets independent of one another. Therefore, they might end up generating more states than necessary, similar to the first case of Figure 1. Investigating how to get the agents' pruning efforts more in sync seems to be worthwhile.

Conclusion

This paper provides a theoretical basis for stubborn sets pruning in the context of privacy preserving planning. The empirical results show that some domains significantly benefit from partial order reduction. Although the production site domain was created with partial order reduction in mind, we believe that it models a specific situation that can also occur within the search space of other domains. In this situation, the heuristics are blind or misleading and, in consequence, the search exhaustively explores the affected parts of the search space. When these parts consist of many independent actions, then stubborn sets pruning can significantly reduce the search effort.

References

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In *Proc. ECAI*, 891–892.

Brafman, R. I. 2015. A privacy preserving algorithm for multi-agent planning and search. In *IJCAI*, 1530–1536.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS*, 162–169.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *Proc. AAAI*, volume 8, 944–949.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In *Proc. AAMAS*, 1265–1266.

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

Štolba, M., and Komenda, A. 2014. Relaxation heuristics for multiagent planning. In *Proc. ICAPS*.

Štolba, M.; Fišer, D.; and Komenda, A. 2015. Admissible landmark heuristic for multi-agent planning. In *Proc. ICAPS*.

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

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proc. Petri Nets*, 491–515. Springer.

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