Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)

Increased Privacy with Reduced Communication in Multi-Agent Planning

Shlomi Maliah
Information Systems Engineering
Ben Gurion University
shlomima@post.bgu.ac.il

Ronen I. Brafman
Computer Science Dept.
Ben Gurion University
brafman@cs.bgu.ac.il

Guy Shani
Information Systems Engineering
Ben Gurion University
shanigu@bgu.ac.il

Abstract

Multi-agent forward search (MAFS) is a state-of-the-art privacy-preserving planning algorithm. We describe a new variant of MAFS, called *multi-agent forward-backward search* (MAFBS) that uses both forward and backward messages to reduce the number of messages sent and obtain new privacy properties. While MAFS requires agents to send a state s produced by an action a to all agents that can apply any action in s, MAFBS sends such messages forward only to agents that have an action that requires one of the effects of a. To achieve completeness, it sends messages backward to agents that can supply a missing precondition. This more focused message passing scheme reduces states exchanged, and requires that agents be aware only of other agents that they directly interact with, leading to *agent privacy*.

1 Introduction

In various settings, agents may wish to cooperate to achieve joint goals, while concealing certain private facts. For example, different manufacturers may want to collaborate in the production of a good without disclosing their entire supply-chain, inventory levels, and local processes. An attractive framework for such planning problems is privacy preserving planning (Nissim and Brafman 2014) which has gained increasing attention in recent years. While several approaches to privacy preserving planning were suggested, heuristic search algorithms (Maliah, Shani, and Stern 2015; Štolba and Komenda 2014; Štolba, Fišer, and Komenda 2015; Maliah, Shani, and Stern 2014a) seem to produce good results. In particular, the Multi-Agent Forward Search algorithm (MAFS) (Nissim and Brafman 2012), combined with strong heuristic estimates (Štolba, Fišer, and Komenda 2015) demonstrated high performance over benchmarks.

The typical notion of privacy used so far in most work on cooperative, privacy preserving planning, is dichotomic: every action and variable is either private to a single agent or public and accessible to all agents. However, in many cases, various facts or actions are naturally described as private to a strict subset of agents (Bonisoli et al. 2014). For example, a supplier and a customer must know the content of a package, but the courier that delivers it need not. Moreover, when two organizations interact to achieve joint goals, each employing sub contractors, it may well be that each wishes that the identity, or even the existence, of its sub contractors would not be known to the other. This is known in the DisCSP literature as *agent privacy* (Faltings, Léauté, and Petcu 2008).

In this paper, we describe a model of refined privacy following (Bonisoli et al. 2014), and a new algorithm, which we call *forward-backward* MAFS (MAFBS), which not only provides weak privacy (like MAFS), but also ensures that two agents that do not share a private variable, never communicate with each other, and hence, need not be aware of the existence of each other. Moreover, through the use of focused communication, we not only obtain the agent privacy property, but also significantly reduce the number of sent messages.

Our method, while still a forward search algorithm, sends fewer messages by introducing elements of regression, or relevance, through backward request messages, supporting agent privacy. Our efficiency gains results from goal driven expansions; In MAFS, agents send a state s generated by a non-private action to any agent that can apply an action in s. In MAFBS, an agent sends state s generated by action a only to agents that have an action that requires one of a's effects.

Unfortunately, sending states only to agents that require an effect is incomplete. Consider, e.g., 3 agents, φ_1,φ_2,φ_3 , and a solution plan a_1, a_2, a_3, with a_i an action of φ_i. Assume a_1, a_2 generate p_1 and p_2 respectively. a_3 requires both p_1 and p_2 and produces the goal. Suppose I is the initial state. φ_1 will send $a_1(I)$ to φ_3, but not to φ_2, while φ_2 sends $a_1(I)$ to φ_3, but not to φ_1. Thus, φ_3 never receives a state where both p_1 and p_2 hold, and cannot execute a_3.

We address this problem using simple backward reasoning. Suppose φ received state $a(s)$ because a produces a precondition of φ's action a', but $a(s)$ does not satisfy another precondition q of a'. φ now sends $a(s)$ backwards to all agents that can supply q. In the example above, when φ_3 receives $a_1(I)$, which satisfies p_1, but not p_2, it sends $a_1(I)$ to φ_2, who can produce p_2. φ_2 applies a_2, achieving p_2, and sends $a_2(a_1(I))$ back to φ_3. These backward messages ensure completeness.

The resulting algorithm, MAFBS, is sound and complete for privacy preserving multi-agent planning, enhancing MAFS with agent privacy. When forward messages are given priority over backward messages, MAFBS is also very effi-

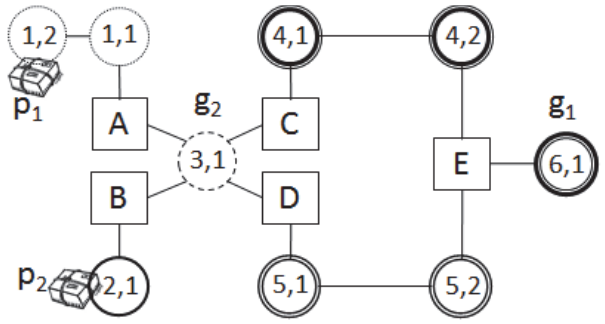


Figure 1: A logistics example.

cient, improving coverage, reducing computation time and state expansions, and reducing, often considerably, the number of messages transmitted in all domains – a crucial parameter in distributed computation.

2 Background

We describe the model of privacy preserving collaborative planning and the MAFS algorithm.

2.1 Privacy Preserving Planning

An MA-STRIPS problem (Brafman and Domshlak 2013) is represented by a tuple $\langle \Phi, \mathcal{P}, \{A_i\}_{i=1}^{|\Phi|}, I, G \rangle$ where:

- $\Phi = \{\varphi_1, \ldots, \varphi_{|\Phi|}\}$ is a set of agents.
- $\mathcal{P}$ is a finite set of primitive propositions (facts).
- A_i is agent φ_i's action set. $A_i \cap A_j = \emptyset$ if $i \neq j$.
- I is the start state.
- G is the goal condition.

Each action $a = \langle pre(a), \mathit{eff}(a) \rangle$ is defined by its preconditions ($pre(a)$), and effects ($\mathit{eff}(a)$). Preconditions and effects are conjunctions of primitive propositions and literals, respectively. A state is a truth assignment over $\mathcal{P}$. G is a conjunction of literals. $a(s)$ denotes the result of applying action a to state s. A *plan* $\pi = (a_1, \ldots, a_k)$ is a solution to a planning task iff $a_k(\ldots(a_1(I)\ldots) \models G$.

An important assumption we make is that actions are in *transition normal form* (Pommerening and Helmert 2015). That is, a primitive proposition (or its negation) appears in a precondition iff it (or its negation) appears in the effect of the action. Every problem is easily converted into transition normal form.

Privacy-preserving MA-STRIPS extends MA-STRIPS by defining sets of facts and actions as private, known only to a single agent. We extend this, in the spirit of (Bonisoli et al. 2014), allowing a fact to be public only to a subset of the agents.[1] We refer to this as *subset privacy*. Thus, with each $p \in \mathcal{P}$ we associate a set, $pr(p) \subseteq \Phi$, the set of agents that are aware of p and its value. We require that for every fact p, if p appears in the description of an action $a \in A_i$, then $\varphi_i \in pr(p)$. That is, an agent must be aware of facts that appear as precondition or effect of one of its actions. Let $pr_p(\varphi)$ be the set of propositions that φ is aware of. That is, for each $p \in pr_p(\varphi)$, $\varphi \in pr(p)$, and for each $q \notin pr_p(\varphi)$, $\varphi \notin pr(q)$.

For clarity of exposition, we refer to a fact p where $|pr(p)| = 1$ as *private* and to a fact p where $pr(p) = \Phi$ as *public*, while facts p such that $1 < |pr(p)| < |\Phi|$ will be called *subset-public*.

We define similar notions for actions. Let $pr(a)$ be the set of agents that are aware of action a. Let $pr_a(\varphi) \supseteq A_\varphi$ be the actions that φ is aware of. An action a is private if $|pr(a)| = 1$, public if $pr(a) = \Phi$, and subset-public otherwise. Furthermore, if p is an effect or precondition of a, then $pr(p) \subseteq pr(a)$. In what follows we assume that the set $pr(p)$ is known to all agents in it. Removing this assumption is an interesting challenge for future work.

Recently, there is growing awareness of the need to better quantify and improve the privacy guarantees in privacy preserving planning (Brafman 2015). A well known privacy property in the area of DisCSP is *agent privacy* (Faltings, Léauté, and Petcu 2008). Applying this idea to distributed planning we say that a multi-agent planning algorithm satisfies *agent privacy* if an agent φ_i cannot learn from participation in the algorithm about the existence of an agent φ_j with whom it shares no variable (that is, $pr(\varphi_i) \cap pr(\varphi_j) = \emptyset$). The agents that share at least one variable with φ_i are called the *neighbors* of φ_i. Thus φ_i is only aware of the existence of its neighbors. This is a desirable property in many multi-agent collaboration settings, which is not satisfied by any current MA planner.

Figure 1 illustrates a simple logistics example in which the agents are trucks tasked with delivering packages. The set of facts $\mathcal{P}$ represents the location of two packages and six trucks. Each truck has three actions: move, load, and unload, corresponding to moving between locations, loading a package and unloading it. Trucks can only drive along the edges in Figure 1. Agents are heterogeneous and their range is restricted, such that location $[i, j]$ can only be reached using the truck of agent φ_i. The rectangles are logistic centers visited by multiple trucks that load or unload packages.

Trucks are owned by different companies that do not want to share their locations and coverage (which locations they can reach) with other companies. Thus, all the facts representing the location of trucks are private, while the facts representing whether a package is at a logistic center are shared among all agents that can reach that logistic center. Only the load/unload actions at the logistic centers are not fully private, whereas the move actions are private for each agent, as well as loading and unloading at private locations.

For each location l and package p there is a fact $at(p, l)$ denoting whether package p is at location l. In this example, $at(p, [4, 2])$ is private to agent φ_4, while $at(p, E)$ is subset-public to agents $\varphi_4, \varphi_5, \varphi_6$. In this example there are no public facts shared between all agents. Similarly, the action $unload(p, E)$ is subset-public to agents $\varphi_4, \varphi_5, \varphi_6$.

In the example above, agent privacy requires that agent φ_1 will be unaware of the existence of agents $\varphi_2, \varphi_4, \varphi_5, \varphi_6$. No agent in the above example is aware of all other agents, yet these agents must collaborate to move the packages to their

[1] Given multi-valued variables, this definition can be further refined, allowing for private variable-value pairs.

target locations.

2.2 Multi-Agent Forward Search

Multi-Agent Forward Search (MAFS) (Nissim and Brafman 2014) is a distributed algorithm schema for forward-search planning that also preserves privacy. MAFS follows the standard definition of privacy, where facts are either private to a single agent, or public to all agents.

In MAFS, each agent maintains a separate search space with its own *open* and *closed* lists. The agent expands states using its own actions only. Thus, two agents (that have different actions) expanding the same state, generate different successor states. An agent generating a state s using a public action, sends s to all other agents that can apply a public action in s. When an agent receives a state s that does not appear in its open or closed lists, it adds s to its open list.

In messages sent between agents, the value of private variables in a state are encrypted so that only the relevant agent can decipher them. This is typically done by sending an identifier to the private state, rather than encrypting each fact independently. By definition, if q is private to an agent, other agents do not have operators that affect its value, and so they do not need to know or manipulate its value. When generating new states by applying their own actions to a received state s, they only copy the encrypted values in s to the next state.

3 Forward-Backward MAFS

Forward-Backward MAFS (MAFBS) employs a high-level concept similar to MAFS: cooperative state-space search by a group of agents, where each agent expands a state using its own operators only. MAFBS differs from MAFS in its message passing scheme. Whereas MAFS sends a state s generated by a public action to all agents that can apply an action in s, MAFBS sends similar, yet more restricted forward messages, as well as backward messages, requesting other agents to achieve facts that the agent needs.

For ease of exposition, we describe MAFBS in terms of multiple search processes running concurrently. We also focus on the question of when can a non-private action (which here we refer to simply as *public*) be applied by the agent, as the application of private actions by an agent is not restricted.

The algorithm is message driven, with each forward or backward message starting a new search process. Forward searches are standard, attempting to advance the agent towards achieving G, while backward searches are goal oriented, where an agent attempts to achieve a specific set of facts P.

Of course, instead of using multiple processes, we can use a single open list, choosing at each iteration one state to expand, progressing the process that this state originated from. Below, we provide a crude pseudo-code of MAFBS (Algorithm 1), ignoring many details for clarity.

3.1 Message Types

MAFBS requires 3 types of messages:

- **Forward messages:** messages sent after an agent φ_i executes a public action a, to all agents that have an action a' such that $pre(a') \cap eff(a) \neq \emptyset$, that is, a' requires a precondition that a supplies. The message contains the new state, as well as the public effects of a.
- **Backward messages:** sent from an agent φ_i to an agent φ_j, requesting φ_j to supply a fact p that φ_i requires. This message is sent only to an agent which has actions that achieve p. The message also contains an identifier, allowing the requesting agent to map this message to a specific search process after a reply. The message contains the current state, the requested fact p, and the requesting agent φ_i.
- **Reply messages:** sent once an agent φ_j has managed to produce the fact p requested in a backward message, containing the state s where p is achieved. This message is sent only to the requesting agent φ_i which requested p. The message also contains the identifier allowing the the requesting agent to map the reply to the request.

3.2 Initialization

MAFBS begins by initializing the open list of each agent with the forward message $\langle I, \mathcal{P} \rangle$. As we explain below, this allows each agent to apply any action applicable in I.

3.3 Handling Forward Messages

An agent that receives a forward message $\langle s, P \rangle$, checks all of its actions that have a precondition in P. For each such action a that can be executed, the agent starts a new search process over $a(s)$. A forward search process is a standard local heuristic forward search, in which the agent extracts a state from the open list, applies all relevant actions, and adds the resulting states back into the open list.

If a public action a was applied, a forward message as described earlier is sent. If a state that is potentially a goal state is generated, a goal detection algorithm, described later, commences.

When a forward search process identifies an action a' with a precondition in P such that all private preconditions of a' hold, but some public preconditions of a' must be achieved by *other* agents, the forward process sends again backward messages requesting the missing preconditions of a'.

3.4 Handling Backward Messages

An agent that receives a backward message $\langle s, p, \varphi_i \rangle$, begins a restricted forward search process. This process is designed to achieve p, and can hence be restricted only to actions that are relevant to p, which can be identified by regression, or by a rapid complete approximated regression.

During this search, the agent may again identify an action a' whose private preconditions are satisfied, yet has some unsatisfied public precondition, that must be produced by other agents. In this case, the agent sends additional backward messages.

When an agent manages to achieve p, it sends a reply message to the requesting agent, containing the state where p was achieved, and an identifier allowing the receiving agent to associate the reply with a request.

Algorithm 1: MAFBS for Agent φ_i

```
1  MAFBS(i)
2      foreach Action a executable from I do
3          start a new forward search process with a(s) in its
             open list
4          send forward message ⟨a, eff(a)⟩ to all relevant
             agents
5      while TRUE do
6          process all received messages
7          choose an active search process and advance it
8  process-forward-message(s, P)
9      foreach a ∈ A_i s.t. pre(a_i) ∩ P ≠ ∅ do
10         if s ⊨ pre(a) then
11             start a new forward search process with a(s) in
                 its open list
12             send forward message ⟨s, eff(a)⟩ to all
                 relevant agents
13         else
14             P_m ← pre(a) \ s
15             foreach φ_j that can satisfy a fact p in P_m do
16                 send backward message ⟨s, p, i⟩ to φ_j
17 process-backward-message(s, p, j)
18     A_p ← actions relevant for p
19     start a new backward search process with goal p over
         actions A_p
20 process-reply-message(s)
21     proc ← the search process that sent the backward
         message that was replied to
22     a ← the action in proc with missing precondition p
23     if s ⊨ pre(a) then
24         Add a(s) to the open list of proc
25     else
26         P_m ← pre(a) \ s
27         foreach φ_j that can satisfy a fact p in P_m do
28             send backward message ⟨s, p, i⟩ to φ_j from
                 proc
```

```
1  advance-forward-search()
2      s ← extract min from open list
3      if s is a goal state then
4          traceback solution and terminate
5      foreach a ∈ A_i do
6          if s ⊨ pre(a) then
7              add a(s) to the open list
8              send forward message ⟨a, eff(a)⟩ to all
                 relevant agents
9          else
10             if private pre(a) satisfied in s then
11                 foreach p ∈ pre(a), s ⊭ p do
12                     send backward message ⟨s, p, i⟩ to
                         relevant agents
13 advance-backward-search(p, A_p, j)
14     s ← extract min from open list
15     if s ⊨ p then
16         Send reply message s to the requesting agent φ_j
17     foreach a ∈ A_p do
18         if s ⊨ pre(a) then
19             add a(s) to the open list
20         else
21             if private pre(a) satisfied in s then
22                 P_m ← pre(a) \ s
23                 foreach φ_j that can satisfy a fact p in P_m
                     do
24                     send backward message ⟨s, p, i⟩ to φ_j
```

3.5 Handling Reply Messages

An agent that receives a reply message containing state s, identifies the search process from which the reply originated, as well as the action a that required the missing preconditions. If a can be executed in the received state s, then a is executed, and the resulting state $a(s)$ is added to the open list of the search process.

If a still has some missing preconditions, new backwards messages are sent with the state s, requesting achievement of the missing preconditions.

Optimizations It was previously observed in MAFS that sending messages whenever a public state is generated, produces too many messages. Instead, in MAFS agents send messages only when a state generated by a public action is extracted from the open list, that is, when that state is heuristically deemed to be the best current state. We use a similar approach, sending both forward and backward messages only after a state is extracted from the open list.

Intuitively, forward messages advance the plan towards the goal, while backward messages are a necessary setback because a needed action cannot be executed. Following this intuition, we give priority to forward messages. When a state is extracted from the open list, we apply all private actions and the relevant public actions, we send only forward messages, and insert the state back into the open list. If that state is extracted again, we send backward messages for relevant public actions with missing public preconditions. That is, backward messages are sent only once no further forward progress can be made in the heuristically best state.

In addition, when deciding which search process, forward and backward, to advance next, we again give priority to the standard forward search over the backward search in a $2:1$ ratio. That is, for each 2 states processed from the standard open list, we process one state from a backward open list.

3.6 Goal Detection

Many existing privacy preserving algorithms assume that all goals are public to all agents, or that agents can define artificial public goal actions, and thus report goal states to all other agents. However, by adding artificially shared variables, these schemes violate agent privacy. Furthermore, allowing public goals only when propositions can be subset-public, seems overly restrictive.

To allow for private sub-goals and yet retain agent privacy,

we suggest using a distributed goal detection mechanism. The essential steps are broadcast and consensus. For goal broadcast, when an agent (the initiator) generates a state in which all its goals, private and public, are satisfied, it sends this state to all its neighbors. When an agent receives a proposed goal state, it forwards it to all agents that have not sent it the proposed goal. An agent that receives a proposed goal from all its neighbors sends them back its decision (i.e., whether the state satisfies its own goals). For consensus, an agent that received a decision updates it based on its own goals, and forwards it to all neighbors who did not send a decision. If the initiator receives a positive decision from all its neighbors, it starts a second broadcast stage, announcing that the search has terminated successfully. Naturally, various optimizations are possible.

This mechanism contains some minimal information leakage, because an agent that broadcasts a goal state and receives a negative decision, learns that some other agent has a private goal. Agent privacy, however, is preserved.

3.7 Private State Encryption

Earlier, we described how agents in MAFS maintain the privacy of their private state information by encrypting it, and that since other agents do not use or change this state information, they can simply copy the encrypted values to newly generated states.

In the case of subset-public variables, the encryption scheme is a bit more involved, but similar. Each variable will have its own encryption/decryption key. For each subset-public variable p, all agents in $pr(p)$ are aware of its initial value, the identity of $pr(p)$ members, and its encryption/decryption key. Of course, each agent will have its own private encryption key (or it could simply maintain a cache of IDs) for its private variables. The encrypted state is a list of encrypted variables values, and each agent can only decrypt the values to which it has a key, as needed. Thus, the state sent by one agent to another will contain the encrypted value of all variables.

3.8 Distributed Plan Execution

When a goal is reached, the agents need to traceback the path leading to the goal. We assume that agents that receive a new state, record the identity of the sending agent (including backward messages). Once a goal is detected, a traceback process ensues. When the traceback reaches a state received from another agent, the token is passed to that agent who continues the process. Note that sometimes, because of backwards messages, the agent simply passes the token on to another neighbor without introducing new actions, as in the plan generation process. By definition, the sending and receiving agents are aware of their mutual existence, as they exchanged states before, and must share some variable. The process stops when the initial state is reached. Now, each agent knows when (at which state) and how it should act, and the plan can be executed in a distributed manner

4 MAFBS Properties

We now discuss the key properties of MAFBS: soundness, completeness, and agent privacy.

Claim 1. MAFBS *is sound.*

Proof. Each state generated in MAFBS is obtained by applying an action to a state that was previously generated, starting at the initial state. Thus, all generated states are reachable, and if a goal state is found, there must be a plan. □

Unfortunately, MAFBS is incomplete. One way to achieve completeness, which we call *Modified* MAFBS is the following: suppose that an agent receives a message, both forwards and backwards, such that it can apply an action (i.e., the conditions in line 10 or line 23 are true), it still sends a backwards message (lines 16 & 28) to agents that can supply some precondition of that action (it may suffice to do this only if a precondition that holds is destroyed by this action.). Whether weaker conditions suffice remains an open question. When the prioritizations described earlier is used, such messages are never actually sent in our experiments.

For our completeness proof, we assume that the goal is a single proposition. If not, the goal detection algorithm is modified to ensure that a state in which a subgoal is achieved is added to the open list of each agent. This, in essence, restarts the search from states in which a subgoal was achieved. We shall also assume, without loss of generality, that all actions in the plan are *public*. Private actions can be compiled away by considering all private action sequences that achieve some private state. In our algorithm, this is not really needed, provided we allow agents complete flexibility with the application of private actions – i.e., run the algorithm as is, except that before applying a public action, the agent can freely apply any private actions.

We require the following definitions: The *causal structure* of a valid plan π (Karpas and Domshlak 2012), denoted $CS(\pi)$ is a DAG whose nodes are the actions of π. a is a parent of a' iff a precedes a' in the π, and a has an effect, say p, that is a precondition of a', and no action between a and a' produces p. That is, there is a causal link between a and a' in π (Tate 1977). As we assume that the goal is a single literal, there is a single leaf node in $CS(\pi)$. We will use $InvCS(\pi)$ to denote $CS(\pi)$ with edge directions reversed, which by the above is a DAG with a single root node. Finally, we say that a plan is *minimal* if whenever any action is removed from the plan, it is no longer a valid plan (i.e., it is either not executable or does not achieve the goal).

Lemma 1. *Let a, a' be two actions in a plan π such that a precedes (not necessarily immediately) a' in π, and p appears in the description of a and a' (possibly negated). Then, a is an ancestor of a' in $CS(\pi)$.*

Proof. The proof is by induction on the number of actions between a and a' in π in whose description p appears. First, suppose that there are no such actions. Because we assume actions are in transition normal form, then p (possibly negated) appears in both the preconditions and effects of a and a'. Therefore, a must supply the correct value of p to a'. Consequently, by definition, a is a parent of a' in $CS(\pi)$.

For the inductive step, suppose the above holds when there are k actions between a and a', and consider the case where there are exactly $k+1$ actions, $a^1, a^2, \ldots, a^{k+1}$ between a and a' that contain p in their description. By the

inductive hypothesis, a^1 is an ancestor of a', and by the argument above, a is a parent of a^1, and thus, an ancestor of a'. □

An immediate consequence of the above Lemma and the definition of post-order traversal of a graph is:

Lemma 2. *The order of actions that mention p in their descriptions in any post-order traversal of $InvCS(\pi)$ is identical.*

Proof. By Lemma 1, every two actions that mention p have an ancestor/descendant relation, which must be maintained in any post-order traversal. □

Lemma 3. *Every post-order traversal of $InvCS(\pi)$ is a valid plan.*

Proof. In any post-order traversal of the graph, for every action a, the relative order of all actions supplying a with some precondition must be the same. Therefore, the value of the propositions in the precondition of a prior to the execution of a will be identical to their value prior to the execution of a in π, and therefore, the preconditions of a are satisfied and a is executable, and hence the entire sequence is executable, and in particular the last action that achieves the goal. □

We now prove:

Theorem 1. *Modified* MAFBS *is complete.*

Proof sketch. Suppose a MA-STRIPS planning problem is solvable. Let π be such a plan. We show that MAFBS generates a post-order traversal of $InvCS(\pi)$, which by Lemma 3 is a plan.

Let a be the first action a in π. a must be a leaf node of $InvCS(\pi)$. After executing a, the algorithm continues to execute one descending path from a using forward messages until a_p, the first parent of a that has other children (which means that this action requires additional actions performed before it). According to the algorithm, the current state is sent backwards from a_p to one of the children and along some branch up to one of its leaf descendants a_l (leaf w.r.t. the tree w/o the actions executed so far). In the modified version of the algorithm, we are guaranteed that the messages are sent backwards by a_p and its descendants even if the action is applicable. Next, we apply a_l, and by virtue of backwards messages, the resulting state is passed and updated along the branch from a_l to a_p. This processes continues until we finish the graph traversal. The above is the key argument in a formal inductive proof. □

An important, technically straightforward property is:

Theorem 2. MAFBS *with a heuristic whose computation preserves agent privacy preserves agent privacy.*

Proof sketch. The key intuition is that an agent cannot distinguish between having multiple "hidden" neighbors of its neighbor, and the case of a "larger" neighbor that encompasses all of these hidden neighbors.

In more detail: Suppose that agent φ_j is not a neighbor of agent φ_i, and let us consider the case that φ_i and φ_j have a common neighbor φ_k. φ_i and φ_j do not communicate with each other directly in any stage of the algorithm (search, goal detection, and plan reconstruction). Thus φ_i can learn about the existence of φ_j only via the content or existence of messages. First, we note that the nature and order of messages sent by φ_k to φ_i are identical whether φ_k has additional neighbors hidden from φ_i or not. Next, let us consider the contents of the messages during search. The messages that φ_i receives contain, aside from the values of variables it is aware of, the values of other variables. These values are encrypted so that their value cannot be determined. Their owner, as well, cannot be detected. In particular, whether the encrypted variables belong to φ_k or to another agent. The same holds during goal detection – if φ_i receives a goal proposal or a decision from φ_k, it cannot know whether φ_k proposed it or accepted/rejected it, or some agent unknown to it. Finally, during plan reconstruction, we repeat the process of search, but backwards, and the same argument applies.

The above arguments are true also if φ_i and φ_j have other shared neighbor φ'_k. The only additional information transmitted is heuristics estimates, whose computation must also preserve agent privacy. □

The landmarks heuristics (Maliah, Shani, and Stern 2014b) respects agent privacy: agents are aware only of landmarks of neighbor agents. Hence, using this heuristic estimate with MAFBS maintains agent privacy.

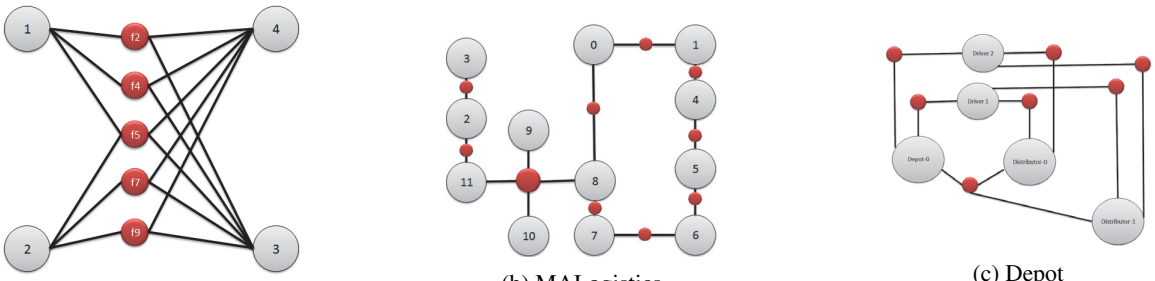
Finally, although initially MAFBS sends a state s only to agents that can use the effects of the last action, s may be sent to additional agents via backwards messages. Still, while MAFS sends s to all agents, MAFBS will send s only to agents relevant to the last action performed. While this set could contain all agents, it is often much smaller.

5 Empirical Evaluation

We conduct an empirical analysis of MAFBS. We experiment with a set of benchmarks from the CoDMAP competition (Štolba, Komenda, and Kovacs 2015), and two more complicated domains — MA-Blocks and MA-Logistics, where a larger number of private actions need to be executed between two consecutive non-private actions, and agents must choose between several paths for achieving goals.

We compare MAFBS to MAFS. Both algorithms use a landmark heuristic (Maliah, Shani, and Stern 2014a), which naturally extends to preserve agent privacy. Table 1 shows the results of the experiments. We report coverage, as well as the number of messages that were sent (a message broadcasted to k agents is counted as k separate messages, MAFBS messages contain one extra bit), the time to completion, and the number of expanded states. We report averages only over problems that both algorithms were able to solve. The table is sorted by decreasing improvement in sent messages. In addition, we report the geometric means in Table 2.

MAFBS is advantageous across all metrics in (almost) all domains. Differences are especially pronounced in the case of the number of messages. MAFBS sends from 3% to 71% of the messages sent by MAFS, and 38% on average. This is not surprising, as MAFS sends new states to all agents that can apply an action, while MAFBS sends forward messages only to agents that use an effect of the generating action.



(a) Elevators (b) MALogistics (c) Depot

Figure 2: Agent interaction graph. Larger nodes represent agents while smaller nodes represent shared facts.

	Coverage		Messages			Time			States		
Domain	MAFS	MAFBS	MAFS	MAFBS	Ratio	MAFS	MAFBS	Ratio	MAFS	MAFBS	Ratio
satellites	13/20	**19**/20	6518	225	3.5 %	411.9	109.8	26.7 %	16692.6	1147.8	6.9 %
rovers	20/20	20/20	2091.3	197.5	9.4 %	121.3	39.1	32.2 %	1894.6	742.5	39.2 %
depot	10/20	**13**/20	58117	6904	11.9 %	148.2	11.9	8 %	30541	4683.7	15.3 %
MALogistics	26/26	26/26	2117.8	343.7	16.2 %	16.9	15.5	91.7 %	3453.1	2840.8	82.3 %
woodworking	8/20	8/20	6068.3	1878.4	31 %	125.3	19	15.2 %	3393.3	1072.5	31.6 %
zenotravel	20/20	20/20	235.1	74.9	31.9 %	209.8	155.5	74.1 %	528.1	435.2	82.4 %
blocksworld	14/20	14/20	58080.3	21779.2	37.5 %	216.3	72	33.3 %	28905.8	18172.5	62.9 %
logistics	20/20	20/20	596.9	227.5	38.1 %	1.2	1	83.3 %	356.1	347.3	97.5 %
MABlocks	4/8	**5**/8	580.3	272	46.9 %	1382.7	992	71.7 %	405.8	316.5	78 %
sokoban	5/20	**8**/20	11760.6	5764	49 %	214.5	75.7	35.3 %	17695.4	10714.6	60.6 %
elevators	18/20	**20**/20	355.2	210.4	59.2 %	19	7.2	37.9 %	783.4	490.8	62.6 %
driverlog	14/20	14/20	783.9	517.4	66 %	51.3	34	66.3 %	3121.4	2944.1	94.3 %
Taxi	20/20	20/20	451.9	323.7	71.6 %	0.3	0.2	66.7 %	419.3	298	71.1 %
Average					38.1 %			53 %			61.6 %

Table 1: Comparing the performance of MAFBS and MAFS. For each domain we report the average over all problems.

	Messages		Time	
Domain	MAFS	MAFBS	MAFS	MAFBS
Satellites	1698.09	**84.29**	73.03	**10.06**
Rovers	908.57	**129.67**	17.39	**9.05**
Depot	7838.33	**1441.28**	13.47	**2.39**
MALogistics	1937.92	**335.40**	15.75	**14.67**
Woodworking	1081.31	**644.46**	2.87	**1.72**
Zenotravel	90.10	**31.89**	5.65	**4.57**
BlocksWorld	9567.31	**5187.88**	43.32	**20.08**
Logistics	374.87	**124.24**	0.58	**0.52**
MABlocks	555.12	**266.19**	1304.10	**919.56**
Sokoban	2804.44	**1167.77**	15.31	**6.11**
Elevators08	222.37	**165.05**	4.16	**2.33**
Driverlog	152.23	**117.39**	1.64	**0.74**
Taxi	322.33	**230.46**	0.20	**0.16**

Table 2: Geometric means for MAFBS and MAFS.

	Coverage		Time		Messages	
Domain	FwdBwd	Fwd	FwdBwd	Fwd	FwdBwd	Fwd
MALogistics	26	26	15.5	15.5	343.7	343.7
blocksworld	14	14	72	72.1	21779.2	21779.2
rovers	20	20	41.1	41.1	197.5	197.5
satellites	19	19	110.4	110.4	225.8	225.8
zenotravel	20	20	155.5	150.9	74.9	74.9
elevators	20	20	7.2	7.1	210.4	195.3
logistics	20	20	1	1	227.5	197
driverlog	14	14	34	28	517.4	252.7
woodworking	8	8	19	17	1878.4	1713.5
depot	**12**	7	15	11	8012	3078
MABlocks	**5**	3	686	690	246	202
sokoban	**8**	2	1.3	1.4	267	288

Table 3: Comparing forward-backward to sending forward messages only

The added backward messages do not reduce the advantage of MAFBS. This reduction in messages is important, as in many distributed settings the cost of message passing dominates all other costs. Our implementation mimics a multi-agent scenario on a single machine, with messages passed through shared memory. Presumably, in a truly distributed setting, the advantage of MAFBS will be more pronounced. The number of generated states as well as runtime improves in MAFBS across all domains, although not as much as the reduction in the number of messages.

To further understand the behavior of MAFBS we take a closer look at the domains where MAFBS does best and worst. Satellites is by far the easiest domain for MAFBS, because no agent ever generates a precondition for another agent. However, agents share resources consumed, but never produced, by actions, and different agents can achieve different subgoals. Thus, there is no need for an agent to ever send forward (or backward) messages. Instead, the only messages that are sent are goal messages, allowing one agent

Domain	Forward	Backward	Bwd actions in plan	Max bwd depth
MALogistics	343.7	0	0	0
blocksworld	21779.2	0	0	0
rovers	197.5	0	0	0
satellites	225	0	0	0
zenotravel	74.9	0	0	0
elevators	210.4	15.1	0	1
logistics	227.5	30.4	0	1
driverlog	517.4	200.3	0	2
woodworking	1878.4	303.4	0	2
sokoban	5764	68.2	1	1
MABlocks	272	43.5	0.8	2
Taxi	323.7	62.5	0.4	5
depot	6904	4823.2	3.9	6

Table 4: Analyzing message behavior.

to achieve goals that the other could not. Rovers has similar properties and presents similar results. Taxi and Driverlog are the domains where the reduction in the number of messages is lowest. In these domains there are public facts that are shared between all agents. Hence, executing some of the actions, results in sending forward messages to all agents.

We show (Figure 2) the agent interaction graphs for example problems from 3 domains — MALogistics and Depot, where MAFBS reduces the amount of messages considerably, and Elevators where it does not. In the graphs, the larger nodes represent agents, the smaller nodes represent facts, and an edge represents that an agent uses a fact.

MALogistics is a complex domain similar to the running example in Figure 1. We created domains where the number of interacting agents is limited, that is, the area under the control of most trucks is connected to at most k other agents. Thus, each time an agent drops a package at some logistic center, it needs to inform only a small number of agents, greatly reducing the number of messages. Indeed, as the graph shows, in MALogistics there is only one fact that is shared among 4 agents. All other facts are shared between exactly 2 agents. In Depot, only one fact is shared among 3 agents, and all other facts are shared only between two agents. Given these sparse dependencies, it is no surprise that MAFBS sends many fewer messages than MAFS.

In Elevators, on the other hand, changes in each floor affect 3 out of 4 elevators, while floor f_5 is shared between all agents. Thus, agents need to inform almost all other agents of changes in a floor, and the number of messages sent by MAFBS is not much less than those sent by MAFS.

These domains demonstrate that loose coupling of agents is advantageous for MAFBS, while tightly coupled agents with many almost public facts, make MAFBS less advantageous. In all domains, supporting agent privacy does not result in reduced performance.

To better understand MAFBS, we take a deeper look at the messages sent by the algorithm. In Table 4 we split the domains into 3 sets: The first set contains domains where no backward messages were ever sent. In these domains an agent never needed a precondition supplied by another agent to apply a public action. Of course, Rovers and Satellite belong to this set. In the second set of domains, agents sometimes requested other agents to achieve preconditions using a backward message, but these requests never entered the final plan. The third set of domains are the most interesting ones, where the backward messages generated actions that were added to the final plan. Of these, Depot and Taxi were the ones where the backward search was deepest, and in Depot the most backward actions were added to the plan.

Depot is the most interesting domain in our experiments. In Depot the agent needs to place boxes in a specific order at a target location. Depot is especially difficult for landmark based search, because the planner may need to give up on achieved landmarks to reach the goal. In Table 1 we see that the reduction in messages, runtime, and expanded states is very pronounced there. This is because limiting agents to apply only actions that depend on the latest action, reduced the problem of placing boxes in the wrong order.

As in many domains MAFBS does not require any backward messages, we ran a forward only version, removing the considerations for sending backward messages altogether (Table 3). While in domains from the first class, there is little benefit in this version, in domains from the second class, this resulted in significantly less messages and run time. These results demonstrate that many domains are, in a sense, easy to solve, using a single forward path in plan space. In addition, we can see that in domains form the third class, sending backward massages significantly improve the coverage.

6 Conclusion

We described the multi-agent forward-backward search algorithm. This algorithm improves upon the state-of-the-art MAFS algorithm, most significantly in the number of messages passed. It also improves upon MAFS in terms of privacy in three ways: 1) Guarantees agent privacy. 2) Due to the reduction in messages, it may leak less private information. Fewer messages sent are likely to lead to less private information leakage (Štolba, Tožička, and Komenda 2016). 3) MAFBS adapts the more refined privacy model of (Bonisoli et al. 2014) supporting agent privacy. The original algorithm suggested by (Bonisoli et al. 2014) modifies MAFS with additional encryption, but keeps the same messages passing protocol. Thus, in terms of performance, number of messages, and agent privacy it is identical to MAFS.

In the future, we intend to examine a fwd/bwd version of secure-MAFS. The fwd/bwd behavior appears orthogonal to the changes made in secure-MAFS, so their combination could be both more secure and send fewer messages.

MAFBS is also interesting because it combines both forward and backward reasoning in a manner that is very different from current single and multi-agent planning (and search) algorithms. We believe that additional optimizations that will focus the message passing are possible, and could lead to further improvements in performance.

Acknowledgments: Supported by ISF Grant 933/13, by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center, and by the Lynn and William Frankel Center for Computer Science.

References

Bonisoli, A.; Gerevini, A. E.; Saetti, A.; and Serina, I. 2014. A privacy-preserving model for the multi-agent propositional planning problem. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, 973–974.

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.

Brafman, R. I. 2015. A privacy preserving algorithm for multi-agent planning and search. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, 1530–1536.

Faltings, B.; Léauté, T.; and Petcu, A. 2008. Privacy guarantees through distributed constraint satisfaction. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Sydney, NSW, Australia, December 9-12, 2008*, 350–358.

Karpas, E., and Domshlak, C. 2012. Optimal search with inadmissible heuristics. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*.

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

Maliah, S.; Shani, G.; and Stern, R. 2014b. Privacy preserving landmark detection. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, 597–602.

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

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

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

Pommerening, F., and Helmert, M. 2015. A normal form for classical planning tasks. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, 188–192.

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

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

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

Tate, A. 1977. Generating project networks. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, August 1977*, 888–893.

Štolba, M.; Tožička, J.; and Komenda, A. 2016. Secure multi-agent planning. In *Proceedings of the 1st International Workshop on AI for Privacy and Security*, PrAISe '16, 11:1–11:8. New York, NY, USA: ACM.