Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-016-9333-9

Collaborative privacy preserving multi-agent planning

Planners and heuristics

Shlomi Maliah[1] · **Guy Shani**[1] · **Roni Stern**[1]

Abstract In many cases several entities, such as commercial companies, need to work together towards the achievement of joint goals, while hiding certain private information. To collaborate effectively, some sort of plan is needed to coordinate the different entities. We address the problem of automatically generating such a coordination plan while preserving the agents' privacy. Maintaining privacy is challenging when planning for multiple agents, especially when tight collaboration is needed and a global high-level view of the plan is required. In this work we present the Greedy Privacy-Preserving Planner (GPPP), a privacy preserving planning algorithm in which the agents collaboratively generate an abstract and approximate global coordination plan and then individually extend the global plan to executable plans. To guide GPPP, we propose two domain independent privacy preserving heuristics based on landmarks and pattern databases, which are classical heuristics for single agent search. These heuristics, called privacy-preserving landmarks and privacy preserving PDBs, are agnostic to the planning algorithm and can be used by other privacy-preserving planning algorithms. Empirically, we demonstrate on benchmark domains the benefits of using these heuristics and the advantage of GPPP over existing privacy preserving planners for the multi-agent STRIPS formalism.

Keywords Artificial intelligence · Automated planning · Multi-agent planning

Parts of this paper appeared as [34].

✉ Roni Stern
roni.stern@gmail.com

Shlomi Maliah
shlomima@post.bgu.ac.il

Guy Shani
shanigu@bgu.ac.il

1 Ben Gurion University of the Negev, Be'er Sheva, Israel

Published online: 22 March 2016

1 Introduction

Many modern organizations outsource some of their tasks to outside companies. The organization must then work together with the outsourcing companies to achieve its goals, while disclosing as little as possible about its abilities. Consider, for example, the case of a military organization that has outsourced its food service to an outside company. The food service company must deliver food into logistics centers, from where it is picked by army trucks and distributed to the army bases. The military would not want, however, to disclose the whereabouts of these bases, the number of people in each base, or the number and location of army trucks. The two organizations must then collaborate while maintaining privacy about their actions, and communicating only over public actions, such as the interactions at the logistics centers.

More generally, we focus on a setting where a team of agents collaborates to achieve a set of goals while constrained to preserve the privacy of other agents. The planning task in this setting is to generate a plan that achieves goals without invalidating pre-defined privacy constraints. This planning task is known as *Collaborative Privacy Preserving Planning* (CPPP).

We study CPPP in the context of the multi-agent STRIPS (MA-STRIPS) domain specification [11,35]. MA-STRIPS is an extension of the classical STRIPS planning language to define multiple agents, each having a potentially different set of capabilities. A planner for an MA-STRIPS problem generates a plan for all agents, dictating when and how each agent must act in order to achieve a set of goals. A CPPP problem can be defined over an MA-STRIPS problem by defining some actions and facts about the world as private. A private action is only known to the agent capable of performing it. A private fact is only known to a single agent, while the other agents are unaware of its existence.

Several approaches have been proposed for building privacy preserving MA-STRIPS planners. One approach is to first “guess” a *coordination* scheme that dictates how the agents should interact, and then have each agent generate a plan to fulfill the decided coordination scheme. We call this approach “Coordinate & Plan” (CP). The CSP+planning [11] and Planning first [36] algorithms are representative of the CP approach. A major challenge in this approach is to “guess” good coordination schemes.

An alternative approach is to let each agent run forward search concurrently. The agents coordinate their search only when needed. For example, when the agent reveals new achievable public facts about the world it may broadcast this information to the other agents. We call this approach “Plan & Coordinate” (PC). The multi-agent forward search algorithm (MAFS) [35] is a representative of this approach. The challenge in this approach is in guiding the forward search performed by the different agents to consider possible collaborations between the agents.

In this work we propose the Greedy Privacy Preserving Planner (GPPP), which is motivated by the CP approach, but can be viewed as a hybrid between the CP and PC approaches. In GPPP, agents plan collaboratively in a relaxed planning problem. The result of this plan is a global plan that contains the coordination scheme. This global plan is then extended by each of the individual agents locally to a concrete plan, where each agent now considers its private information. Constructing a global plan for coordination, and afterwards extending it with private actions can be seen as a CP approach. On the other hand, the global plan is constructed by a planning process, with coordination occurring when needed, which can be seen as a PC approach.

GPPP relies on an efficient privacy preserving heuristic to guide the search. A privacy preserving heuristic is a function that estimates, for a given agent, the cost of achieving all goals from a given state, without disclosing private information of other agents. Generating a heuristic that is accurate while preserving the agents’ privacy is a major challenge that we discuss in this paper. One way to obtain a privacy preserving heuristic is to create an agent-specific projection of the planning problem that ignores all private actions and facts of other agents. Then, the agents can compute over the projection any existing single agent heuristic, such as Pattern Databases (PDB) [15,16] or landmark-based heuristic [28,30,40]. Such a projection-based approach has various problems, such as an inability to account for complex interactions between agents. The limitations of projection-based heuristics are especially manifested in domains where collaboration between the agents is necessary to achieve all goals, as we will later demonstrate.

In this work we propose two privacy preserving heuristics that substantially outperform such projection-based heuristics. Both heuristics are inspired by effective single agent heuristics. The first is inspired by landmark-based heuristics. A landmark is a set of literals or actions that must be achieved or performed before reaching the goal. Discovering all landmarks is computationally hard [28], but there are polynomial methods that are able to identify a subset of the landmarks [28,39]. We propose here a distributed privacy preserving algorithm for detecting landmarks, called PP-LM, and show how these landmarks can be used by the agents to collaboratively compute a heuristic for a state.

The second privacy preserving heuristic we propose is called *Privacy Preserving Pattern Database* (3PDB). Like single agent PDB, the 3PDB heuristic requires a preprocessing stage in which a lookup table—the PDB—is populated with costs of plans for transitioning between public facts. This PDB is populated in a collaborative and privacy preserving manner. Then, when an agent needs to evaluate a state, it searches for public facts it can achieve from that state using its private actions. The cost associated with that state considers the distance stored in the PDB for reaching public goal facts from achievable public facts. The resulting heuristic is privacy preserving and is shown to be highly effective in guiding the planner.

Empirical results over benchmark domains show that GPPP is able to find more solutions faster than MAFS, using the proposed heuristics. Both of the proposed heuristics are shown to be more effective in guiding the search than a projection-based heuristic, in domains where collaboration is required. GPPP with the best performing heuristic is shown to consistently outperform MAFS with projection-based heuristics by more than two orders of magnitude in some domains.

An additional set of experiments compares GPPP with other planners used in the recently introduced Competition of Distributed and Multiagent Planners (CoDMAP) [45]. These experiments show that while GPPP is not always the best performing planner, it is the best performing planner in many domains.

Thus, our contributions are threefold—(1) we suggest a new algorithm for solving CPPP problems represented using MA-STRIPS, (2) we suggest the PP-LM heuristic for computing landmarks in a privacy preserving setting, and (3) we suggest an adaptation of pattern databases to a privacy preserving setting, through the 3PDB heuristic.

This paper is structured as follows. Section 2 defines CPPP in the context of MA-STRIPS and provides necessary background. Section 3 presents the GPPP algorithm and discusses key design details. Sections 5 and 6 present our two novel privacy preserving heuristics. Section 7 briefly discussed what occurs if some of the goals are private. Section 8 presents experimental results and Sect. 9 discusses related work. Finally, Sect. 11 concludes the paper and suggests future work.

2 Background and problem definition

We begin by formally defining the privacy preserving planning problem in MA-STRIPS and providing related background, including a brief description of existing planners.

There are several existing models for defining multi-agent planning problems. In this work we focus on the recent MA-STRIPS model due to its simplicity. MA-STRIPS is an extension of the well-known STRIPS model and is defined as follows.

Definition 1 (*MA-STRIPS* [10]) An MA-STRIPS problem is represented by a tuple $\langle P, \{A_i\}_{i=1}^k, I, G\rangle$ where:

- P is the set of possible facts about the world, represented as propositional variables.
- $I \subseteq P$ is the set of facts true at the start state
- k is the number of agents
- A_i is the set of actions that agent i can perform.
- $G \subseteq P$ is the goal state, i.e., the set of facts that must be jointly achieved by the agents.

To simplify presentation, we assume that the sets A_i are disjoint, that is, no action can be executed by two different agents. An action $a \in A_i$ has three elements:

- $pre(a)$: the preconditions of a.
- $add(a)$: the add-list of a.
- $del(a)$: the delete-list of a.

The precondition of a ($pre(a)$) is a logical formula that must hold before a can be executed. Following standard STRIPS conventions, $pre(a)$ is a conjunction of facts, although this is not truly a limitation for our specific approach. The sets $add(a)$ and $del(a)$ represent the effects of executing a. The add-list ($add(a)$) and delete-list ($del(a)$) are the facts set to be true and false, respectively, in the state following the execution of a. A solution to an MA-STRIPS model is a *plan*—a sequence of actions that achieves the goal from the initial state.

2.1 Privacy in MA-STRIPS

Following previous work [35], we define privacy in the context of MA-STRIPS as follows.

Definition 2 (*Privacy Aware MA-STRIPS*) A privacy-aware MA-STRIPS problem is represented by the tuple

$$\left\langle P, \{A_i\}_{i=1}^k, I, G, \left\{private_i(P), private_i(A_i)\right\}_{i=1}^k\right\rangle,$$

where $P, \{A_i\}_{i=1}^k, I$, and G are defined as in Definition 1, and $private_i(P) \subseteq P$ and $private_i(A_i) \subseteq A_i$ represent the facts and actions that are private for agent i.

There are several ways to define the semantic meaning of private facts and actions. We assume a strict form of privacy, where each agent is not even aware of the private facts and actions of other agents, and private information must not be shared during execution and during planning. For example, in the logistics domain described earlier the locations of trucks are private facts. Thus, each agent is only aware of the location of the trucks it controls. Furthermore, the agent is not even aware of the existence of the other agents' trucks. Technically speaking, this means that not only the current truth assignment of the fact variables is unknown, but other agents are not even aware of the existence of these variables themselves.

Following this strict meaning of private information, we point out that the input to each agent consists of its private information and the public information about the other agents. Formally, each agent $j \in [1, k]$ accepts as input the following:

- Projected facts: $\Pi_j(P) = P \setminus \bigcup_{i=1, i\neq j}^k private_i(P)$

– Projected start state: $\Pi_j(I) = I \setminus \bigcup_{i=1,i\neq j}^{k} private_i(P)$
– Projected actions: $\Pi_j(A) = A \setminus \bigcup_{i=1,i\neq j}^{k} private_i(A_i)$
– Projected goal: $\Pi_j(G) = G \setminus \cup_{i=1,i\neq j}^{k} private_i(P)$

Facts and actions that are directly observable by all agents are referred to as *public* and denoted by $public(P)$ and $public(A)$, respectively. In our logistics example, both the army agent and the food supplier agent know when a truck unloads a food shipment in the logistic centers or when the shipment is loaded and is no longer at the center. Thus, having a food shipment in the logistic center is a public fact.

For convenience we will use the following notation. Let $A = \bigcup_{i=1}^{k} A_i$ be the set of all action, and $private(P)$ and $private(A)$ denote the set of private facts and private actions, respectively, of every agent. It is easy to see that: $private(P) = \bigcup_{i=1}^{k} private_i(P) = P \setminus public(P)$ and similarly $private(A) = \bigcup_{i=1}^{k} private_i(A) = A \setminus public(A)$.

Our definition of privacy-aware MA-STRIPS is slightly different from the privacy definition in earlier work on MA-STRIPS [11], where the private facts and actions were automatically inferred from the MA-STRIPS problem definition (Definition 1). A fact is defined private if only a single agent has an action that has this fact as a precondition or as an effect. An action is defined public, if at least one of its preconditions or effects is a public fact. One motivation for automatically identifying private and public facts is to split the problem into loosely coupled components.

The definition of privacy-aware MA-STRIPS used in this paper is more general. For example, a fact that is only used by preconditions and effects of an agent must be private according to Brafman and Domshlak's original definition of privacy in MA-STRIPS [11]. In our definition, such a fact may be defined as public, which has a direct implication on the effectiveness of the heuristics. For example, projection-based heuristics are more powerful as more facts are public, as the projected state space becomes closer to the original state space. We make the assumption that an agent knows when it can apply its actions. This imposes some restriction on which facts and actions can be defined as private. Namely, an action of one agent cannot be a private action of another, and the preconditions and effects of an agent's action cannot be private facts of another agent.

Strictly speaking, our privacy definition is not more expressive than Brafman and Domshlak's original definition. This is because one can take a privacy-aware MA-STRIPS definition and add a "dummy" public action to every public fact such that the public fact is a precondition of this "dummy" action. This would result in all public facts being identified as such by Brafman and Domshlak's privacy-preserving MA-STRIPS variant.

Recently, MA-PDDL [31], an extension of the PDDL 3.1 planning language to support MAP, was extended to express privacy in planning. This was done as part of the recent CoDMAP [45] and the full specification is available online [32]. Their definition is more general than the one used by us, as it supports defining facts as private to a subset of agents. We refer to such an extended definition of privacy as *subset privacy* and briefly discuss in Sect. 10.2 how our work can be extended to support subset privacy.

2.1.1 Degrees of privacy preserving planning

In this work we aim for a privacy preserving planning in which private facts and actions are not directly shared between agents during both planning and execution.

This is a weak form of privacy preserving in the sense that it does not mean that an adversary agent cannot learn about the abilities of other agents. For example, when agent *a* publishes that it can achieve a fact using *n* actions, the other agents may deduce some lower bound on the number of private actions of a. In principle, one can measure the information revealed by every message sent to other agents, for example by computing the entropy reduction on the possible states of the agent following the message. This approach is used in privacy preserving distributed constraint optimization [19]. That being said, we leave a thorough discussion of privacy measurements to future research, and constrain our algorithms here not to reveal explicitly any private facts that can be achieved, or private actions that can be executed. This is the privacy definition that is currently widely used in MA-STRIPS research.

A notable exception is the recent work by Brafman [9] that described a more strict privacy definition where agents learn nothing of the capabilities of other agents while planning. It appears that to accomplish such privacy constraints, very strict restrictions must be made on the domains that can be handled, which most current benchmark domains do not comply with.

2.1.2 Privacy preserving plans

As defined above, a plan in privacy preserving MA-STRIPS is a sequence of actions. Naturally, such a plan is composed of both public and private actions. The public actions within the plan and their order are known to all agents, yet an agent is unaware of the private actions of other agents, and their ordering within the sequence of public actions.

Thus, a sequential plan can be decomposed into a *joint coordination scheme* and a set of *individual private plans*. The joint coordination scheme is the sequence of public actions that the agents must perform. This coordination scheme is shared by all agents, and each agent is committed to executing the public actions that are assigned to it in the coordination scheme. The individual private plans are sequences of private actions that each agent needs to perform between the execution of public actions, to achieve their private preconditions. In addition, consider the case in which the goal contains some private facts. We refer to these facts as *private goals*. In such a case agents will have one or more individual private plan that achieve these private goals.

Thus, a privacy preserving plan inherently supports concurrency, as agents can perform their private plans in parallel (subject to the synchronization constraints imposed by the public actions in the coordination scheme). Symmetrically, we can express the decomposition in terms of public and private facts, where the coordination scheme is a sequence of public facts, rather than actions, that needs to be achieved in some order. Below, we use the action oriented view for clarity of exposition.

2.1.3 Privacy preserving planners

Several privacy preserving planners have been proposed for MA-STRIPS. The first planner proposed for MA-STRIPS, called CSP+Planning [11], partitions the planning process into two. First, the sequence of public actions that needs to be performed by all agents is guessed. Then, each agent plans to be able to perform its assigned public actions. One of the appealing properties of CSP+Planning is that it enables, at least theoretically, the use of distributed CSP algorithms [20,21,51, inter alia]. The main difficulty in using CSP+Planning in practice is that it is not clear how to guess which public actions to commit to before planning. Thus, CSP+Planning was shown to be inferior to other approaches empirically [46].

A different approach to privacy preserving planning for MA-STRIPS that was shown to be more effective is represented by the MAFS algorithm [35]. In MAFS, each agent plans individually to achieve the goals using a best-first search (BFS) guided by some heuristic. All agents plan concurrently, and coordination between the agents is used when new information is revealed. When an agent generates a state by applying a public action, it broadcasts that state to all other agents. This state is then being considered by the rest of the agents in their planning computation. This process continues until a goal state is reached by one of the agents.

Privacy may be compromised when agents broadcast states. Nissim et al. [35] proposed a technique to preserve privacy when broadcasting a state. The facts in a state s can be partitioned into $k+1$ disjoint sets: one set of facts containing the public facts that are true in s (denoted by $public(s)$), and k sets of facts, one per agent, specifying which private facts of that agent are true in s (denoted by $private(s, i)$ for agent i). When an agent i broadcasts a state s, it encrypts $private(s, i)$ before broadcasting it. When an agent j receives a state s, it decrypts $private(s, j)$. Existing cryptographic tools can be used for implementing these encryptions and decryptions. This technique was used in MAFS, a state-of-the-art privacy preserving MA-STRIPS planner. For a complete survey of other MA-STRIPS algorithms and their performance compared to MAFS, see Nissim et al. [35].

One way to implement this state encryption mechanism, which we employ in GPPP, is to have each agent maintain a *private state index* (PSI) for every generated state. This is an index that only $agnt_i$ can map to $private(s, i)$. We denote by $PSI(s, i)$ the PSI of agent $agnt_i$ for state s. Every state can be fully represented by a set of public facts and a set of PSIs, one per agent. This representation of a state can be safely broadcasted and shared by all agents without loss of privacy. Below, we assume that states are always represented in such a way.

MAFS has several limitations. First, in MAFS the plan is developed sequentially in the sense that one agent plans the first steps, then passes the state to another agent to plan the subsequent steps and so on. Thus, while the agents in MAFS search for plans concurrently, the possible speedup that can be achieved from a distributed computation is limited by the length of the solution. For example, if the solution is composed of X actions performed by agent $agnt_i$ and then Y actions performed by agent $agnt_j$ then the construction of the solution in MAFS would require agent $agnt_i$ planning these X actions and only then having agent $agnt_j$ planning the additional Y actions. This is true even if the agents are totally independant.

Second, and perhaps more important, is that for MAFS to be efficient, each agent needs a heuristic to guide its search. To preserve privacy, such a heuristic cannot directly use the private states of other agents and thus using heuristics developed for single agent planning is problematic. In Sects. 5 and 6 we propose effective privacy preserving heuristics. Next, we describe a novel privacy preserving planner that first performs global planning of the coordination needed between agents, and then lets the agents concurrently generate plans to meet this planned coordination scheme.

Other MA-STRIPS planners were recently suggested. Some use techniques from partial ordered planning [46–48], others use different methods [29,49]. While a full comparison of all MA-STRIPS planners is beyond the scope of this paper, it is generally agreed that MAFS is a state-of-the-art MA-STRIPS planner. We provide a broader discussion on other MAP algorithms in Sect. 9.

3 The Greedy Privacy Preserving Planner (GPPP)

In this section we present our first major contribution—the GPPP algorithm. First we describe the basic GPPP algorithm as a centralized algorithm, specifying the details of its two main phases: global planning and local planning. Then, we analyze its theoretical properties and discuss how GPPP can be distributed. For ease of exposition we will assume that the goal consists of only public facts and explain in Sect. 7 how to relax this assumption.

GPPP takes as input a privacy-aware MA-STRIPS specification and computes a privacy preserving plan. That is, the agents agree on a joint coordination scheme and each agent computes internally a set of *individual private* plans to allow executing the agreed upon joint coordination scheme. GPPP has two phases.

– *Global planning* The agents collaboratively plan in order to find a joint coordination scheme that the agents should follow to achieve all goals. This is done by solving a *relaxed* privacy-preserving MA-STRIPS planning problem in which only public actions can be performed.[1]
– *Local planning* The agents individually solve a set of single agent planning problems, in order to find a set of individual private plans that achieve the preconditions of the public actions agreed upon in the joint coordination scheme.

As the global planning is done on a relaxation of the original planning problem, the local planning phase may fail. This means that some agent may not be able to perform all the public actions assigned to it in the coordination scheme. In such a case, the global planning phase continues to search for a different coordination scheme. This process is repeated until a privacy preserving plan is found. Next, we describe in detail the phases of GPPP.

3.1 Global planning

The goal of the global planning phase in GPPP is to find a joint coordination scheme without solving the full planning problem. This is done by solving a relaxed MA-STRIPS planning problem, which is defined next. In this relaxed planning problem only public actions can be performed while private actions are assumed to have no delete effects or cost and to be automatically performed when possible. By *automatically performed*, we mean that for a given state s, each agent $agnt_i$ computes $private(s, i)$ by repeatedly applying all its private actions, accumulating achieved private facts until no more facts can be achieved. Thus, transitions between states in this relaxed problem only occur by performing public actions. This relaxed problem is inspired by *delete relaxation*, a form of planning problem relaxation that is at the core of many successful planners and heuristics [27].

The global planning phase uses a best-first search to solve this relaxed planning problem. Algorithm 1 lists the pseudo-code of GPPP, describing the best-first search. The agents share an open list (denoted OPEN) of nodes that are considered for expansion. First, the initial state is inserted into OPEN (line 2). As we explained above, we do not add the actual initial state, but its encrypted representation, where the public facts are explicit, yet the private state of each agent is represented by its PSI.

At every iteration, the best state s according to a heuristic function (such as the heuristics presented later in Sects. 5 and 6) is extracted from OPEN (line 4). Each agent then checks which public actions it can perform in that state, and generates the corresponding states. Before inserting the generated states to OPEN, each agent adds to the state the private facts it can achieve without performing additional public actions. As mentioned above, this is computed by having the agent repeatedly apply all the private actions it can apply in the current state ignoring delete effects.

If one of the generated states is a goal state, then the corresponding plan is extracted (line 9) and is proposed as a joint coordination scheme. Then, the local planning phase begins (line 10). By construction, the joint coordination scheme, denoted by P_{public}, consists of

[1] The exact details of this relaxed problem are more involved, and are explained later in the paper.

Algorithm 1: GPPP

```
1  Plan($start$, $goal$)
       Input: $start$, the start state
       Input: $goals$
2      OPEN ← {$start$}
3      while OPEN is not empty and goal not found do
4          $s$ ← choose best $s$ from OPEN
5          foreach Agent $agnt_i$ do
6              $children$ ← GetChildren($s$,$agnt_i$)
7              foreach $child \in children$ do
8                  if child is a goal node then
9                      $P_{public}$ ← the plan for this goal
10                     $P_{full}$ ← LocalPlanning($start$,$P_{public}$)
                       /*If the coordination scheme is valid        */
11                     if $P_{full}$ is not null then
12                         return $P_{full}$
13                 else
14                     Compute $h(child)$ and insert $child$ to OPEN
15     return Null (no plan found)
16 end
17 LocalPlanning($start$,$P_{public}$)
       Input: $start$, the start state
       Input: $P_{public}$, a sequence of public actions
18     $P_{full}$ ← ⟨⟩
19     $current$ ← $start$
20     foreach action $a \in P_{public}$ do
21         $P_a$ ← PlanForPreconditions($current$,$a$)
22         if $P_a$ was found then
23             $current$ ← execute $P_a$ on $current$
24             Append $P_a$ to $P_{full}$
25         else
26             return Null
27     return $P_{full}$
28 end
```

only public actions. The private actions are added during the local planning phase (Sect. 3.2). If the local planning phase is successful, the algorithm halts with a complete plan. If not, it means that the agents did not find a plan that enables following P_{public}. In such a case, the global planning phase resumes, searching for another solution to the relaxed planning problem.

To increase the likelihood of the global planning phase producing a extendable coordination scheme, the agents perform the following refinement when generating a state s. Each agent removes from $private(s, i)$ the set of private facts in s that are mutually exclusive with the preconditions or effects of the actions used to generate s.

To identify mutually exclusive facts, we use the following simple approach. Every fact pair p, q for which each action that adds p removes q and vice versa is considered a mutex. Each agent runs this process independently, using its actions as well as the public projections of actions of other agents. As opposed to standard classical planning mutex identification we cannot use the initial state as it is not shared between the agents. Thus, this approach is not sound in general.

3.2 Local planning

The main component of the local planning phase is a single-agent planning procedure ($PlanForPreconditions$) that accepts as input a state $current$ and an action a and outputs a plan for transitioning from $current$ to a state in which the preconditions of a are satisfied (line 20 in Algorithm 1). During the local planning phase this single-agent planning procedure is invoked once for each action in the coordination scheme. Note that each action in the coordination scheme is associated with an agent that is tasked to perform it—this is the agent that performed it in the global planning phase.

Let $P_{public}[i]$ be the ith action in the coordination scheme and let $agnt_{P(i)}$ be the agent tasked to perform it. Agent $agnt_{P(i)}$ must plan given the previous state to achieve the preconditions of $P_{public}[i]$.

Starting with $i = 1$, agent $agnt_{P(1)}$ is required to solve the first single-agent planning problem in the local planning phase. It searches for a plan from the initial state to a state that enables $agnt_{P(1)}$ to perform the action $P_{public}[1]$ (line 20). Agent $agnt_{P(1)}$ then applies this plan and action $P_{public}[1]$ to the start state (line 22). The resulting state, denoted $current$, is sent to the agent tasked with performing the second action in the coordination scheme, i.e., agent $agnt_{P(2)}$. $current$ is the initial state for the next single-agent planning problem, in which agent $agnt_{P(2)}$ searches for a plan that will enable it to perform action $P_{public}[2]$.

This continues until either the goal state is reached or until one of the agents fails to find a solution for a single-agent planning problem. If a goal state has been reached, the local planning succeeded and the agents have a complete privacy preserving plan that achieves the goal (line 12 in Algorithm 1). In the latter case, the local planning fails and the global planning phase continues to search for a different coordination scheme (line 25).

Privacy is preserved during local planning by using the same encryption techniques used during the global planning. States sent between agents are encrypted, such that the public facts are visible but the private facts are encrypted (represented by PSIs). Note that in the pseudo code of Algorithm 1 each agent returns its extended plan but in practice, all that is shared between the agents is the current state in an encrypted form. In fact, the current state does not even need to be shared between the agents, and each agent can locally plan for the actions allocated to it independently. This is discussed later in Sect. 3.4.

3.3 Theoretical properties

Intuitively, the soundness of GPPP results from the fact that executing the actions in the coordination scheme guarantees that a state is reached in which all facts in the goal are satisfied (assuming all goals are public).

Theorem 1 (Soundness) *Any multi-agent plan returned by GPPP is valid, i.e., if all the agents follow the plan then the goal state is reached.*

Proof outline GPPP returns a plan if a coordination scheme was returned by the global planning phase and that coordination scheme was made concrete in the local planning phase. If a coordination scheme is returned by the global planning phase this means that applying the corresponding public actions results in a state in which all public facts in the goal are satisfied. Since the coordination scheme has been extended, this means that each agent can perform the public actions specified to it in the coordination scheme. Thus, the goal state is achieved by following the plan returned by GPPP. We later discuss in Sect. 7 the soundness of GPPP in the presence of private goals, i.e., facts in the goal state that are private.

GPPP achieves the weak form of privacy preserving defined in Sect. 2.1.2. This is because the communication between the agents consists of only public facts and PSIs—no private information is ever sent.

GPPP, as defined so far, is not complete, in the sense that there may be solvable MAP problems that GPPP will not be able to solve. This is because the local planning for each action in the coordination scheme is done without consideration of the other actions that the same agent is tasked to perform according to the coordination scheme. For example, assume that agent $agnt_1$ is tasked to plan for actions $P_{public}[1]$ and $P_{public}[3]$. It may be the case that the plan for achieving the precondition of $P_{public}[1]$ prevents planning for $P_{public}[3]$, due to modifications to the private state done during the local planning $P_{public}[1]$. One approach to resolve this is to allow backtracking in the local planning phase, forcing a different private plan for the earlier public actions. If all options to backtrack were exhausted and no local plan was found, the algorithm will return to the global planning phase to search for a new coordination scheme.

For efficiency reasons, we did not implement this procedure, and in cases where the coordination scheme could not be extended to a concrete plan, we did not try to backtrack the local plan and immediately searched for a new coordination scheme. To ensure completeness, one can employ a complete CPPP algorithm, like MAFS, after several iterations of this process.

3.4 Distributed computation

As mentioned earlier in the paper, we explain GPPP as a centralized algorithm for ease of exposition, but it can be implemented as a distributed algorithm.

The best-first search done in the global planning phase needs to be coordinated between the agents, as they share OPEN and jointly select which state to expand. This is done by having the agents broadcast generated states and having each agent maintain a copy of OPEN. Then, a consensus mechanism can be used to jointly agree on which state to expand. Similarly, a consensus mechanism is needed for halting the search when a goal is found.

Expanding a node is done in a distributed fashion: each agent applies all its actions to the current state and broadcasts the resulting generated states. Each agent computes and broadcasts the heuristic function for every generated state it receives. Aggregating the heuristic values computed by each agent for a given state depends on the heuristic function used. See below for how this is done for the heuristics we propose in Sects. 5 and 6.

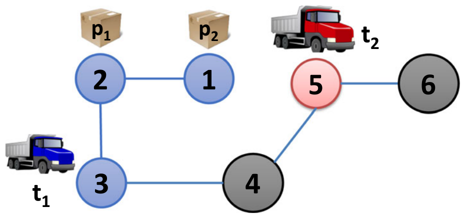
The local planning phase can be completely distributed, each agent responsible for the actions allocated to it in the coordination scheme. This is because all public actions are defined by the coordination scheme, and every action that impacts facts that may affect more than a single agent are by definition public.

Each agent must locally plan for the actions allocated to it in a sequential manner. This is because local planning of the j^{th} action allocated to agent $agnt_i$ may depend on the private facts of the current state, and thus depends on how the first $j-1$ actions allocated to $agnt_i$ were achieved.

4 Improved local planning

Local planning, as outlined in Algorithm 1, is done sequentially, one action at a time. As discussed above, even if GPPP is run in a distributed manner, each agent must locally plan for the actions allocated to it sequentially.

Fig. 1 A scenario demonstrating the problem of sequential local planning



Coordination scheme	local planning	Cost
Unload p_1 at 4	Move 3 $\longrightarrow$ 2	1
	Load p_1	1
	Move 2$\longrightarrow$ 4	2
	Unload p_1	1
Unload p_2 at 4	Move 4 $\longrightarrow$ 1	3
	Load p_2	1
	Move 1 $\longrightarrow$ 4	3
	Unload p_2	1
Load p_2 at 4	Move 5 $\longrightarrow$ 4	1
	Load p_2	1
Unload p_2 at 6	Move 4 $\longrightarrow$ 6	2
	Unload p_2	1
Total:		**18**

(a)

Coordination scheme	local planning	Cost
Unload p_1 at 4 and Unload p_2 at 4	Move 3 $\longrightarrow$ 1	2
	Load p_2	1
	Move 1 $\longrightarrow$ 2	1
	Load p_1	1
	Move 2 $\longrightarrow$ 4	2
	Unload p_1	1
	Unload p_2	1
Load p_2 at 4	Move 5 $\longrightarrow$ 4	1
	Load p_2	1
Unload p_2 at 6	Move 4 $\longrightarrow$ 6	2
	Unload p_2	1
Total:		**14**

(b)

Fig. 2 Plans for different local planning algorithms, for the scenario in Fig. 1. **a** Regular local planning and **b** improved local planning

This can be inefficient, both in terms of runtime and in terms of solution cost. Consider, e.g., the logistics example shown in Fig. 1, where the goal is to put package p_1 at location 4, and package p_2 at location 6. Locations 4 and 6 are known to both agents. Facts related to locations 1, 2, and 3 are private facts of agent t_1, and facts related to location 5 are private facts of agent t_2. The coordination scheme generated for this problem consists of the following public actions:

- a_1: Agent t_1 unloading p_1 at location 4
- a_2: Agent t_1 unloading p_2 at location 4
- a_3: Agent t_2 loading p_2 at location 4
- a_4: Agent t_2 unloading p_2 at location 6

Locally planning to execute these public actions one at a time would result in truck t_1 driving to location 2, loading p_1, driving to location 4, and unloading p_1. Then, the agent will drive back to location 1 to pick up p_2. Figure 2a lists the result of the local planning from coordination scheme to concrete plans. Clearly, it would be more efficient for t_1 to drive to pick up both packages and then drive to location 4 to unload both packages. A better concrete plan could have been found if agent t_1 was tasked to achieve the effects of a_1 and a_2 (which are that p_1 and p_2 are at location 4) in a single planning problem. Figure 2b lists this improved local planning, showing that the resulting plan has a lower cost (14 instead of 18) compared to the plan generated by the regular local planning. Moreover, this solution may be found faster, as fewer planning problems are solved in order to extend the global plan (3 instead of 4).

Our improved GPPP tries to capture the intuition behind the example given in Fig. 1. The first step in our improved local planning is to consider *locally planning for the public effects of actions instead of actions*. This means that the task of a local planning agent is not to find a plan that reaches a state where a public action can be performed, but to find a plan that reaches a state where the required public facts hold.

Definition 3 (*Public effect*) The public effect of an action a is a tuple $\langle add_P(a), del_P(a)\rangle$, where $add_P(a)$ is the list of public facts in a's add list, and $del_P(a)$, is the list of public facts in a's delete list. Achieving a public effect of an action a means reaching a state in which the facts in $add_P(a)$ are true and the facts in $del_P(a)$ are not.

Locally planning for a public effect corresponds to finding a plan in which that public effect is achieved. Planning for public effects instead of actions offers greater flexibility, as there may be several public actions that achieve the same public effect.

The second step in our improved local planning is to identify which action effects can be planned for together. Planning for a set of public effects E corresponds to reaching a state in which all the facts in the add lists of all the public effects in E are true, and all the facts in the delete lists of all the public effects are false.

Identifying which public effects can be planned for together and ordering these sets of public effects is challenging. One may consider applying plan parallelization algorithms over the coordination scheme [3] to identify which public actions can be performed in parallel, and plan together their public effects. Plan parallelization algorithms, however, cannot be directly applied here, as these algorithms require that preconditions and effects of all actions in the plan are known, while in privacy preserving planning, some of these preconditions and effects may be private.

We propose a privacy preserving method to estimate which public effects can be planned together. Our approach only allows locally planning for public effects together if it does not prevent any public effect of the coordination scheme from being achieved. More formally, let $E_P[i]$ be the public effects of $P_{public}[i]$. We allow locally planning a public effect $E_P[i]$ together with a public effect $E_P[j]$, for $j < i$ if:

1. $E_P[i]$ and $E_P[j]$ can be achieved by a single agent. This is needed as local planning involves applying private actions and thus cannot be done by more than one agent without breaking privacy constraints.
2. $E_P[i]$ can be achieved when planning for $E_P[j]$. That is, whether $E_P[i]$ can be achieved after the first j-1 public effects were achieved.
3. Achieving $E_P[i]$ together with $E_P[j]$ does not prevent achieving any other public effects in the coordination scheme. Here we verify that planning for $E_P[i]$ earlier than planned, i.e., with $E_P[j]$, does not prevent achieving the public effects that were planned to be achieved after achieving $E_P[j]$ and before achieving $E_P[i]$.
4. Public effects $E_P[k]$ for all $j < k < i$ do not negate the effects of $E_P[i]$. This condition checks that the effects of $E_P[i]$ will not be deleted by the public effects originally planned to be achieved after achieving $E_P[j]$ and before achieving $E_P[i]$.

Testing whether these conditions hold is not trivial, and we now suggest a possible implementation. To implement these conditions, we need to know which effects enable or prevent achieving other effects. Given the privacy constraints, this is not simple to verify. As an approximation, we consider for every action in the coordination scheme the impact of performing it on the set of public effects that can be achieved.

Let $Pos[i]$ be the list of public effects that were not achievable prior to performing the ith action in the coordination scheme, and became achievable after performing $P_{public}[i]$, and let $Neg[i]$ be the list of public effects that were achievable prior to $P_{public}[i]$ but became unachievable after performing $P_{public}[i]$.

We compute $Pos[i]$ and $Neg[i]$ by storing for every action in the coordination scheme the list of alternative public actions that could have been chosen instead of it. That is, public actions whose preconditions hold in the state where $P_{public}[i]$ was chosen. Identifying this set of alternative actions can be done easily during the global planning phase.

Let $Alt[i]$ be this set of public effects of the alternative actions of $a_{P(i)}$. Using $Alt[i]$, we can compute $Pos[i]$ and $Neg[i]$ as follows: $Pos[i] = Alt[i+1] \setminus Alt[i]$ and $Neg[i] = Alt[i] \setminus Alt[i+1]$. Now, we can describe our method for choosing which public effects could be planned for together. We allow planning for $E_P[i]$ together with $E_P[j]$, for $j < i$, if the following conditions hold:

1. $E_P[i] \in Alt[j]$
2. $\forall k$ s.t. $j \leq k < i$: $E_P[k] \notin Neg[i]$
3. $E_P[i] \notin Neg[j]$
4. $\forall k$ s.t. $j \leq k < i$: $E_P[i]$ is not mutex with E_P[k]

The first condition ($E_P[i] \in Alt[j]$) ensures that achieving $E_P[i]$ is possible after $E_P[j$-1] is achieved. The second condition ($\forall k$ s.t. $j \leq k < i$: $E_P[k] \notin Neg[i]$) ensures that achieving $E_P[i]$ will not prevent achieving any action effects intended in the coordination scheme to be achieved before achieving $E_P[i]$. The third condition ($E_P[i] \notin Neg[j]$) ensures that achieving $E_P[j]$ will not prevent achieving $E_P[i]$. The fourth condition ($\forall k$ s.t. $j \leq k < i$: $E_P[i]$ is not mutex with E_P[k]) ensures that $E_P[i]$ will not be destroyed by achieving any of the action effects intended in the coordination scheme to be achieved after achieving $E_P[j]$ and before $E_P[i]$.

Our improved local planning considers each of the public effects in increasing order (starting from $E_P[0]$), trying to group every public effect $E_P[i]$ as early as possible in the sequence of public effects, where the above conditions hold. Whenever a public effect $E_P[i]$ was successfully grouped with $E_P[j]$, we update $E_P[j]$ to contain both public effects; and add $Pos[i]$ and remove $Neg[i]$ from $Alt[j]$.

The benefits of our improved local planning is two-fold. First, as we plan for several action effects together, the local planning phase involves solving fewer planning problems, potentially saving runtime. Also, the quality of the resulting plan is better, since agents can optimize their plans to achieve sets of facts together, instead of planning to achieve each public action preconditions one at a time.

Our improved local planning, however, is not always helpful. First, the public effects that were grouped together may not be extendable together. If this occurs, we revert back to the sequential local planning of the original GPPP. Second, the grouping mechanism also consumes runtime. Third, the planning problems solved in the improved local planning are potentially harder, as they require to achieve more facts than in the original GPPP local planning phase. The experimental results provided in Sect. 8 shed light on the benefit of our improved local planning in most domains.

5 Privacy preserving landmark heuristics

Some of the most successful heuristics in planning are based on identifying landmarks [28, 30, 39]. This section describes an adaptation of a landmark heuristic for multi-agent privacy preserving planning. First, we provide background on landmark heuristics in single agent planning. Then, we describe a simple projection-based method to detect landmarks in CPPP, followed by an improved collaborative algorithm for detecting better landmarks in a privacy preserving manner.

5.1 Landmarks in classical planning

A landmark Φ for a classical planning task $\Pi = \langle P, A, I, G\rangle$ is a logical formula over the facts P, which must be satisfied at some state along every solution of Π [28]. As in most literature dealing with landmarks, we will restrict our attention to landmarks that are facts or disjunctions over facts. Each planning task has the trivial landmark consisting of all the goal literals.

Although it is PSPACE-hard to even check whether a given literal is a landmark or not, there are several efficient algorithms that compute a set of landmarks and orderings of the identified landmarks [28,39]. Recent landmark detection algorithms work as follows. First, every literal that holds in the goal but not in the initial state is identified as a landmark. Then, for every landmark p, if all actions that achieve p (called the *achievers* of p) have some literal $q \in P$ as a precondition, then q is also a *fact landmark*.[2] If there are no common preconditions, a set of literals that is a hitting set over the preconditions of all achievers of p can be used as a *disjunctive landmark*, denoting that at least one of the literals in the disjunctive landmark must be achieved in every solution. We call the above process of identification of preconditions for a landmark the *development* of the landmark. This landmark development process continues until no new landmarks can be identified.

When landmark l_1 is contained in landmark l_2, we prefer the more focused landmark l_1. For example, the algorithm may identify that both p and $p \vee q$ are landmarks, in which case p is preferred over q. We say that p is more refined, or is a better landmark than $p \vee q$.

Originally, landmarks were used as subgoals [28], guiding a base planner inside a control loop. More recently, the number of landmarks which are yet to be achieved plus the number of landmarks that should be achieved again, was used as a heuristic function for very successful state space planners [39]. Note that this is an inadmissible heuristic estimate, because an action might achieve more than one landmark. There are also admissible heuristics that are based on landmarks [30], which are useful for optimal planning.

There are several methods to identify when a landmark that was previously achieved must be achieved again. This is done by defining various types of *orderings* between landmarks. The benefit of identifying landmarks that need to be achieved again is that we can add the cost of achieving them to the heuristic, resulting in a more accurate heuristic. In our experiments, we considered two types of orderings: *greedy-necessary* and *reasonable*. Briefly, a landmark l_1 is ordered before l_2 in a greedy-necessary order if l_1 needs to be true to achieve l_2 for the first time. A landmark l_1 is ordered after l_2 in a reasonable order if l_1 is needed at the same time or after l_2 is achieved, and achieving l_2 deletes l_1. The heuristic function we used in our experiments counts a landmark l_1 as a landmark that should be achieved again if there is another landmark (or goal predicate) l_2 that has not been achieved yet and one of the following conditions hold:

1. l_1 does not hold in the current state and there exists a *greedy-necessary* ordering $l_1 \rightarrow l_2$.
2. There exists a *reasonable* ordering $l_2 \rightarrow l_1$.

The first condition indicates that l_1 must be achieved again because l_2 hasn't been achieved yet and l_1 is required to achieve l_2. The second condition indicates that l_1 must be achieved again because l_2 hasn't been achieved yet and when it will be achieved it will delete l_1.

Richter [39] provided exact definitions of the different types of landmarks and landmark orders. The algorithms presented in this paper are independent of the actual landmark-based

[2] When computing the *achievers* of p, only actions that can be performed before obtaining p are considered. This is usually estimated using delete relaxation [39].

heuristic used during the search. We later describe a specific heuristic function used in our experiments, but replacing it with a different landmark based heuristic function is simple.

5.2 Projection-based landmark heuristic

As landmark detection algorithms were developed in the context of single agent planning, an obvious idea is for each agent to cast, or *project*, the multi-agent problem onto a single agent problem, and seek landmarks over this projection using standard algorithms. An agent can perform such a projection by considering the public actions and facts, as well as its own private actions and facts only. MAFS uses this projection approach such that each agent searches in its individual projected state space.

An agent may identify some landmarks by running an existing classical landmark detection algorithm on its projected state space. This projection-based heuristic ignores private actions and literals of other agents, including private pre-conditions and effects of public actions. There are, therefore, a few substantial problems with detecting and using landmarks in this projection of the full, multi-agent state space.

First, every agent only identifies landmarks related to goals it can achieve. A special case of this problem is when a goal is only achievable by a private action of some agent $agnt_1$. As a result, the other agents are not aware of any action that achieves the goal, and therefore cannot find any landmark for that goal. Moreover, if this is the only goal, then all other agents would simply search blindly towards achieving it.

As an example, consider the logistics setting in Fig. 3 and assume that the only goal is to move package p_1 to location g_1. The only action that can put a package in g_1 is a private action of agent $agnt_6$. Thus, all other agents would not be able to infer any landmark and would search blindly.

Projection-based landmarks can also produce low quality landmarks, i.e., disjunctive landmarks containing facts that could be removed. For example, assume that a goal can be achieved by several actions, each having a precondition. Assume also that only one of these preconditions is achievable from the start state. If this information depends on other agents and $agnt_i$ is not aware of it, then it would infer a disjunctive landmark. A better landmark would only consider the preconditions of the achievable action.

For example, assume that the landmark being developed is the goal $at(p2; [3, 1])$ in Fig. 3. Any agent applying a projection-based landmark detection algorithm by agents 1, 4, or 5, would detect $\{at(p2; A) \vee at(p2; B) \vee at(p2; C) \vee at(p2; D)\}$, as they are unaware of the initial location of $p2$. However, considering that $p2$ is initially at location [2, 1], $at(p2; B)$ is a better landmark (that would be detected by the PP-LM algorithm presented next). The coming section provides a possible remedy to these shortcomings, finding more and better (more refined) landmarks while still preserving privacy.

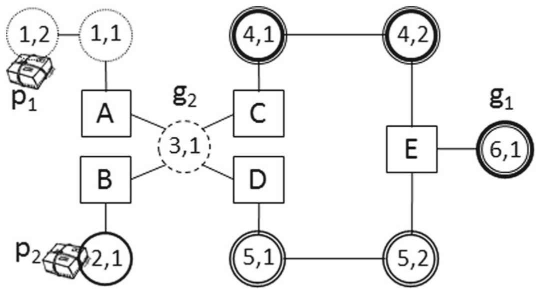


Fig. 3 A logistics example, with six agents. *Squares* mark public cities, while *circles* mark private cities. Private cities are labeled i, j where i is the agent, and j is the city index for that agent. *Different circle* types represent different agents. All actions move packages between nearby cities. There are two packages p_1, p_2 that must be delivered to g_1, g_2 respectively

5.3 Privacy preserving landmark detection

In this section we present a landmark detection algorithm called PP-LM (Privacy Preserving LandMark detection) that allows agents to collaboratively discover more informative landmarks and communicate the landmarks that they discover. This collaborative distributed landmark identification algorithm is performed jointly by the agents prior to a distributed search (e.g., before running GPPP or MAFS). A prominent benefit of PP-LM over the projection-based approach is that PP-LM enables an agent to consider landmarks that it cannot achieve, as well as finding more refined landmarks.

In each iteration of the algorithms the agents agree on a landmark to develop. Then, the agents collaborate to develop this landmark, until either it is satisfied by the initial state, or new landmarks are discovered. This process proceeds until all identified landmarks are satisfied by the initial state. We now explain this process in detail.

5.3.1 Initialization

We say that a landmark is a *public landmark* if it only contains public literals. A landmark is a *private landmark* if it contains at least one private literal. For example, in Fig. 3, $at(p_1, [1, 2])$ is a private landmark, while $at(p_1, E)$ is a public landmark. Each agent maintains a private list of private and public landmarks, initialized by the set of public and private goals it can achieve.

5.3.2 Identifying a landmark to develop

First, the agents agree on a single landmark to develop. We take a two step approach. First, prior to each selection the agents must agree on a current leading agent (say, by turn). Then, the leader picks a landmark to develop, either public or private from its landmark list. If an agent has no undeveloped landmarks in its list, it can forfeit the turn, or avoid competing for leadership.

If a public landmark is chosen, then the leading agent informs everybody about it. When a private landmark is chosen, the leading agent only informs other agents that it is working on some private landmark. We do not enforce any particular ordering on the landmarks to be developed. This can be done by any tie breaking method.

5.3.3 Identifying achievable literals

Once all agents agree on developing a landmark l, we begin the process of identifying which literals are achievable by the entire team of agents before the landmark is achieved. To do so, each agent maintains a set of public literals it knows are achievable by the team as well as the set of private literals it can achieve (including those it can achieve with the help of other team members). We call this set the agent's set of *achievable literals*. Initially this set contains only the literals in the initial state (the public ones and the private ones known to that agent).

Each agent then applies all private and public actions it can perform given its set of achievable literals, ignoring the delete effects of these actions. Whenever a public literal is achieved, the agent publishes it to all other agents. The other agents add this literal to their set of achievable literals, potentially allowing more actions to be performed, and more literals to be achieved. This process repeats until no more public literals can be achieved by any agent. To avoid circular reasoning, such that the achievement of l requires l to be achieved, the agents ignore all actions that achieve l (or any literal in l in the case of disjunctive landmarks).

When the reachability analysis phase is concluded, all agents have the same set of public literals in their set of achievable literals—the public literals that can be achieved without achieving l first. As we use delete relaxation, these sets of achievable literals are an approximation of the real set of literals that the team can achieve. Thus, not all literals in this set are indeed achievable. However, all the literals not in this set are guaranteed to be unachievable without achieving first the landmark l.

As an example consider the logistics problem in Fig. 3. Assume that the landmark being developed is the goal $at(p_2, [3, 1])$. Now, agent 3 must ignore all actions that move p_2 to [3, 1]. Restricting our attention to literals involving p_2, the set of agent 3's achievable literals contains only $at(p_2, B)$. Note that if we did not ignore actions that achieve $at(p_2, [3, 1])$, then agent 3's set of achievable literals would have been having p_2 at any public city.

The set of achievable literals must be recreated for every landmark we develop, that is, we can not reuse achievable sets that were computed for other landmarks. This is because when identifying the reachable literals, we ignore all actions which have the currently developed landmark as an effect.

5.3.4 Identifying satisfying agents

The next phase identifies which agents can satisfy the current landmark l. These are agents that have an action that achieves l, and the preconditions of this actions are in their set of achievable literals. Every agent that can satisfy l declares so to the entire team. Let $achievers(l)$ denote the set of actions that can achieve l.

If only one agent can satisfy l (e.g., when l is a private landmark) then that agent now identifies new landmarks, public and private, by considering the literals in the preconditions of the applicable actions in $achievers(l)$ that are not true in the initial state or an existing landmark. Literals existing in all preconditions are *fact landmarks*. If no such literal exists, then a *disjunctive landmark* is formed containing the set of facts that occur in all of the actions' preconditions. The agent then insert these new landmarks to its list of landmarks. For example, if $at(p_2, [3, 1])$ is the landmark being developed, and the only public achievable literal is $at(p_2, B)$, then only agent 3 can satisfy this landmark, using the action $move(p_2, B, [3, 1])$. Then, a new public landmark $at(p_2, B)$ is identified and added to the landmark list of agent 3.

When more than one agent can satisfy l, each such agent identifies landmarks required for achieving l using the same collaborative process. Then, each of these agents broadcasts the set of public landmarks it requires to achieve l. To preserve privacy, private landmarks are not published, and are used only by the corresponding agent when it computes a state's heuristic. All of these sets of public landmarks are joined together to form a public disjunctive landmark that is added to the list of landmarks of all the relevant agents. Note that when a

For example, when developing $at(p_1, E)$, the public achievable literals contains $at(p_1, C)$ and $at(p_1, D)$. Thus, two agents, 4 and 5, are able to satisfy the landmark. As they develop their private landmarks, we would eventually learn that $at(p_1, C) \vee at(p_1, D)$ is a public disjunctive landmark. Once a landmark has been satisfied by either generating more landmarks or by the initial state, the agents decide on the next landmark. This process repeats until no new landmarks can be identified.

Table 1 demonstrates the differences between the landmarks detected by the projection approach and our PP-LM method. For example, the projection-based landmark detection algorithm would not identify $at(p_1, [1, 1])$ as a landmark, because it assumes that actions moving p_1 to city E are being handled by other agents and have no preconditions (since the preconditions are private). The projection-based approach, which doesn't run the collabora-

Table 1 Differences in landmark detection between the projection method of MAFS and PP-LM, over the example in Fig. 3

Method	Private landmarks	Public landmarks
Projection landmarks	$\{at(p_2, [3, 1])\}$	$\{at(p_2, A) \vee at(p_2, B) \vee at(p_2, C) \vee at(p_2, D)\}$
	$\{at(p_1, [6, 1])\}$	$\{at(p_1, E)\}$
PP-LM	$\{at(p_2, [3, 1])\}$	$\{at(p_2, B)\}$
	$\{at(p_1, [6, 1])\}$	$\{at(p_1, A)\}$
	$\{at(p_1, [3, 1])\}$	$\{at(p_1, C) \vee at(p_1, D)\}$
	$\{at(p_1, [1, 1])\}$	$\{at(p_1, E)\}$

tive reachability analysis, would identify the landmark $at(p_2, A) \vee at(p_2, B) \vee at(p_2, C) \vee at(p_2, D)$, while PP-LM identifies the refined landmark $at(p_2, B)$.

5.4 Computing the landmark heuristic

The outcome of PP-LM is that all agents know the set of public landmarks and each agent knows its set of private landmarks. Now we must compute a heuristic value for a state using the discovered landmarks.

Given a state s, each agent publishes the number of private landmarks that still need to be achieved, i.e., that were not achieved thus far, or were achieved but need to be achieved again (due to the landmark orderings mentioned above). The value returned for this landmark heuristic is the sum of the number of private and public landmarks that need to be achieved (considering which landmarks were achieved and which landmarks need to be achieved again due to ordering).

Computing the landmark heuristic for a given state is a collaborative activity, as it considers all landmarks: private and public. In practice, however, computing the landmark heuristic for a generated state can be done incrementally by the agent that performed the action that generated the new state. This is because the action of one agent cannot satisfy any private landmark of another agent (otherwise, that action would not be public).

6 Privacy preserving pattern database heuristics

Another form of heuristic known to be very effective in a range of domains is a pattern database (PDB) [15–17,37]. We now provide brief background on PDBs for single agent planning, and then discuss how PDBs can be used in CPPP.

6.1 PDBs in single agent planning

PDBs are arguably the most successful *memory-based heuristic* in single agent planning. The high-level idea of memory-based heuristics is to compute, either before planning or during, a memory intensive data structure that will be used when computing the heuristic function. The PDB is a (usually very large) look-up table containing heuristic values. The PDB heuristic also defines a function that maps states to entries in this table, based on a homomorphic abstraction of the searched state space.

Algorithm 2: Construct the PDB for the 3PDB heuristic

```
1 PDB ← ∅
2 foreach Agent $agnt_i$ do
3     foreach $f_1 \in P_{pub}$ do
4         foreach $f_2 \in P_{pub}$ do
5             SearchForPlan($agnt_i, f_1, f_2, PDB$)
6 UpdateTransitiveClosure($PDB$)
```

The form of abstraction usually used in PDB heuristic is to consider a subset of the facts in the world. The PDB is then created by generating the entire abstract search space—the search space of the planning problem in which only the selected set of facts exists. Then, the optimal solution between every two states in this abstract search space is stored in the PDB, and used as a heuristic. We note that the PDB heuristic is a special case of the Merge & Shrink heuristic [25], where more flexibility is given in how the abstraction is defined. There are several Merge & Shrink heuristics, and PDBs, while somewhat restricted, are known to be very effective [37]. Next, we present a privacy preserving adaptation of the single agent PDB heuristic to CPPP. As for single agent PDBs, we explain how this PDB is created and how it is used during search.

6.2 Computing the PDB

The PDB we construct for computing the 3PDB heuristic has an entry for every pair of public facts f_1 and f_2 storing the cost of a plan that achieves f_2, starting from a state in which f_1 is true. For example, if $at(p_1, loc_1)$ and $at(p_1, loc_2)$ are two public facts, the PDB will contain an entry containing an estimated cost for moving p_1 from loc_1 to loc_2, and also an entry for moving p_1 from loc_2 to loc_1. Algorithm 2 shows the construction of this PDB.

Initially, the PDB is empty (line 1). Each agent iterates over all pairs of public facts. For every such pair of public facts f_1 and f_2, each agent searches for a plan from a state where f_1 holds to a state where f_2 holds using only its private and public actions. During the f_1 to f_2 planning phase, simplifying assumptions are made concerning the initial values of other private facts in the local initial state. For example, if $f_1 = at(p_1, loc_1)$ and $f_2 = at(p_1, loc_2)$, and the agent controls truck t_1, then the agent can assume that t_1 is at loc_1 to begin with. This is very similar to how regression search (searching from the goal backwards) is done in single agent planning [2,5]: applying an action adds facts to the state, and we allow required absent facts to be added to the initial state, except for f_2 and facts that are mutually exclusive with f_1.

This phase is denoted in the pseudo code as **SearchForPlan** (line 4), and was implemented in our experiments by running the FF planner.[3] If a plan from f_1 to f_2 was found, its cost is stored in the PDB. For clarity of presentation we assume that the same PDB is shared by all agents but such "shared memory" can be implemented in a distributed way using message passing—each agent maintains its own PDB, and broadcasts notifications after every modification to the PDB. The shared PDB does not compromise the weak privacy constraint we enforce, as the PDB only contains pairs of public facts and the cost of the

[3] We also experimented with using the A* [23] algorithm to find optimal plans for the PDB. Experimentally, we did not observe a substantial difference between the performance of the two algorithms for solving these simple single fact problems.

corresponding plan. Thus, this process does not explicitly reveal any private fact or action that any single agent can achieve.

In some cases, cooperation of multiple agents is needed to achieve f_2 from f_1. For example, perhaps no single agent can achieve f_2 from f_1, but agent $agnt_1$ can achieve f_3 from f_1 and agent $agnt_2$ can achieve f_2 from f_3. Thus, together $agnt_1$ and $agnt_2$ can achieve f_2 from f_1. Furthermore, it might be that $agnt_1$ requires 10 steps to achieve f_2 from f_1, while $agnt_2$ and $agnt_3$ can jointly achieve f_2 from f_1 in 5 steps, although none of them can achieve f_2 from f_1 alone. In this case, the entry for f_1, f_2 that was computed by $agnt_1$ needs to be updated to the new lower cost of 5.

To reflect such collaborative plans, the PDB construction process iteratively tries to improve the existing entries and add new entries to the PDB. This is done by computing the *transitive closure* of the PDB entries, denoted in the pseudo code as **UpdateTransitive-Closure** (line 2). For example, if there is an entry for $\langle f_1, f_2 \rangle$ with cost 10, and an entry for $\langle f_2, f_3 \rangle$ with cost 5 then computing the transitive closure will add an entry for $\langle f_1, f_3 \rangle$ with cost 10+5=15, unless a lower cost entry already exists in the PDB. Computing the transitive closure can be done either by exhaustively iterating through all pairs of facts until no improvements are achieved (similar to dynamic programming), or by considering each PDB entry as an action transitioning between facts, and searching the lowest cost transition between any two pairs using an optimal search algorithm. We use the latter option, which we found to be more efficient in our experiments.

6.2.1 Time and memory complexity

The time complexity of building the PDB as described in Algorithm 2 depends on the complexity of running a single agent planner to find a plan for an agent from one public fact to the other (*SearchForPlan* in line 4 of Algorithm 2).

Populating the PDB requires $|public(P)|^2 \cdot k$ calls to *SearchForPlan*, where k is the number of agents. Computing the transitive closure in this context is in fact an all pairs shortest path problem, and thus can be implemented in a runtime of $O(|public(P)|^3)$. Thus, if the cost of a call to *SearchForPlan* is C, the worst case complexity of building the PDB with Algorithm 2 is

$$|public(P)|^2 \cdot k \cdot C + O(|public(P)|^3)$$

If we assume that $k \cdot C$ is smaller than the number of public facts ($|public(P)|$), then the PDB construction time would be just $O(|public(P)|^3)$.

An alternative way to implement PDBs for privacy preserving MAP would be to run a privacy preserving multi-agent solver for every pair of public facts to set the PDB entries. In a preliminary study we observed that computing a PDB in this way was very time consuming—orders of magnitude slower than the method we describe above.

PDBs in single-agent planning are notorious for the amount of memory they require. Our version of PDB is very different in that aspect, as there is an entry for every pair of public facts. Thus, the size of the PDB is only $|public(P)|^2$, which in small many domains compared to the amount of memory stored while planning (which is some planners is related to the size of the search space).

6.3 Computing the 3PDB heuristic

After the PDB is computed, at planning time, we use the PDB entries to assign a heuristic value to states that are generated through the planning process. This is done while planning

Algorithm 3: Compute the 3PDB heuristic

```
  Input: $s$, the state for which to compute the heuristic
1 foreach Agent $agnt_i$ do
2     $Pubs_i \leftarrow$ all public facts $agnt_i$ can achieve
3     foreach $g$, a fact from the goal not satisfied in $s$ do
4         $cost(g, i) \leftarrow \min_{f \in Pubs_i}(PDB(f, g)+\text{cost}(f))$
5 $h \leftarrow 0$
6 foreach $g$, a fact from the goal not satisfied in $s$ do
7     $h \leftarrow h + \min_{agnt_i}(cost(g, i))$
8 return $h$
```

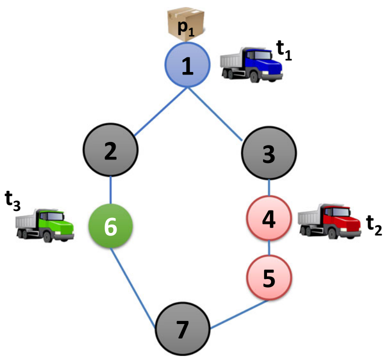


Fig. 4 Illustration of the scenario

for a specific problem, i.e., specific initial and goal states. The pseudo code describing the 3PDB heuristic computation for a given state s is presented in Algorithm 3.

Each agent $agnt_i$ computes all the public facts it can achieve from the current state in a delete relaxation problem (line 2), recording the cost of achieving each of these public facts. The set of resulting public facts for $agnt_i$ is denoted $Pubs_i$ and the cost of achieving a public fact $f \in Pubs_i$ in this relaxed planning problem is denoted by $cost(f)$.

For a given state s, each agent computes the cost of reaching g from s, for every goal g that does not hold in s. This is done by summing $cost(f)$ and the stored entry in the PDB for reaching g from f (denoted $PDB(f, g)$), for every fact $f \in Pubs_i$. The minimum over these costs is denoted by $cost(g, i)$, and represents an estimation of the cost it will take agent $agnt_i$ to achieve g (line 4). The agent $agnt_i$ computes and publishes $cost(g, i)$ to all agents. Agents that cannot achieve g publish the value ∞.

The resulting heuristic for a state s sums for every g the minimum $cost(g, i)$ over all agents (line 7). This approach, of adding the estimated cost of reaching each goal independently, is motivated by the h^{add} heuristic [5].

As an example of the 3PDB mechanics, consider the simple logistics scenario in Fig. 4. Each agent controls a single truck, grayed nodes (2, 3, and 7) are known to all agents, and the colored nodes (1, 4, 5, 6) are private locations, known only to a single agent. Location 1 is private for agent t_1, location 6 for for agent t_3, and locations 4 and 5 for agent t_2. Thus, every fact related to these specific locations would be private. The goal is to get package p_1 to location 7.

Table 2 shows the PDB that is precomputed offline for this scenario. For example, the PDB entry for moving package p_1 from location 2 to location 3 has a cost of 4. Indeed, the

Table 2 The generated PDB for Fig. 4

Fact 1	Fact 2	Cost
Package p_1 at 3	Package p_1 at 2	4
Package p_1 at 7	Package p_1 at 2	4
Package p_1 at 2	Package p_1 at 3	4
Package p_1 at 7	Package p_1 at 3	5
Package p_1 at 2	Package p_1 at 7	4
Package p_1 at 3	Package p_1 at 7	5

shortest plan would be to load p_1, move to location 1, then location 3, finally unloading p_1 at location 2. There is no entry in the PDB for getting from $at(p_1, 1)$ to the goal because $at(p_1, 1)$ is a private fact.

At planning time, given the initial state where p_1 is at location 1, we need to compute a heuristic value for achieving the goal $at(p_1, 7)$. In this setting, only agent t_1 can achieve public facts, p_1 at location 2 and p_1 at location 3. Achieving each of these public facts costs 3 (load p_1, move, unload p_1). Thus, the 3PDB heuristic will return $h = 3 + 4$, corresponding to the cost of achieving p_1 at location 2 (=3) plus the PDB entry for getting p_1 from location 2 to the goal (=4).

The scenario in Fig. 4 also demonstrates a case where our 3PDB heuristic is more informed than a landmark-based heuristic. Consider the two states generated by expanding the state where p_1 is at location 1. These are the states where p_1 is at location 2, denoted s_2, and the state where p_1 is at location 3, denoted s_3. Our 3PDB heuristic will return $h(s_2) = 4$ and $h(s_3) = 5$, thus expanding first s_2 and eventually reaching the goal with a lower cost plan. By contrast, landmark-based heuristic, such as the previously proposed privacy preserving landmark heuristic (PP-LM) [34], would give s_2 and s_3 the same heuristic value, because the disjunctive landmark p_1 at location 2 or location 3 is satisfied in both states, all else being equal.

The 3PDB heuristic can be tuned with some domain knowledge. Instead of having a PDB with all pairs of public facts, it is possible to construct the PDB only to a subset of all the public facts pairs, thus focusing on important facts only, avoiding the heuristic noise of non-important facts, as well as saving PDB construction time and lookup time during search. We have done this in the first set of experiments in Sect. 8.3.

6.3.1 Time complexity of the PDB heuristic computation

Algorithm 3 consists of computing $cost(g, i)$ for each agent i and goal fact g, and then computing h by summing $\min_{agnt_i}(cost(g, i))$ over all unsatisfied goal facts.

Computing $cost(g, i)$ requires computing which public facts each agent can achieve and then perform a PDB lookup for every pair of achievable fact and unsatisfied goal fact. The number of achievable facts and the number of unsatisfied goal facts are upper bounded by $|public(P)|$. The computation of which facts agent i can achieve is done on a delete relaxation problem, and thus can be done in $O(|\pi_i(P)| \cdot |A_i|)$. Hence, the total time complexity of computing $cost(g, i)$ for every unsatisfied goal fact g and agent i is

$$O\left(|public(P)| \cdot |public(G)| \cdot \left(\sum_{i=1}^{k} |\pi_i(P)| \cdot |A_i|\right)\right) \quad (1)$$

$$\leq O\left(|public(P)|^2 \cdot \left(\sum_{i=1}^{k} |\pi_i(P)| \cdot |A_i|\right)\right) \tag{2}$$

$$= O\left(|public(P)|^2 \cdot \left(\sum_{i=1}^{k} (|public(P)| + |private_i(P)|) \cdot |A_i|\right)\right) \tag{3}$$

$$= O\left(|public(P)|^2 \cdot \left(|public(P)| \cdot |A| + \sum_{i=1}^{k} |private_i(P)| \cdot |A_i|\right)\right) \tag{4}$$

$$\leq O\left(|public(P)|^2 \cdot (|public(P)| \cdot |A| + |private(P)| \cdot |A|)\right) \tag{5}$$

$$= O\left(|public(P)|^2 \cdot |P| \cdot |A|\right) \tag{6}$$

After $cost(g, i)$ is computed for every unsatisfied goal fact g and agent i we compute h by summing $\min_{agnt_i}(cost(g, i))$ over all unsatisfied goal facts, adding $O(|public(P)| \cdot k)$ to the complexity analysis. Therefore, the overall worst case complexity of computing the 3PDB heuristic, as defined in Algorithm 3, is:

$$O\left(|public(P)|^2 \cdot |P| \cdot |A|) + |public(P)| \cdot k\right) = O(|public(P)|^2 \cdot |P| \cdot |A|)$$

since k must be smaller than $|A|$.

7 Private goals

The exposition above and most previous research on privacy preserving planning often assumed (either explicitly or implicitly) that all goals are public facts. The case where some of the facts in the goal are private, however, was rarely addressed [35], except for the work by Luis and Borrajo on the PMR algorithm [33] (see Sect. 9), handling private goals explicitly. Private goals are a challenge for heuristic search algorithms in general, because agents must be able to know of goals in order to estimate the cost of reaching them. There exists however a simple compilation of private goals into public ones.

Given a private goal of agent $agnt_i$, the agent computes a set of public facts that are required for $agnt_i$ to achieve this private goal. This can be done by applying regression from the private goal in a way similar to our landmark detection algorithm. We then define a public artificial goal to replace the private goal, and create an artificial action for agent $agnt_i$ that takes as precondition the (possibly disjunctive) public facts that are needed for accomplishing the private goal, and its effect is the artificial goal. The new public goal and the public action are published to all other agents. The cost of the artificial action can be set to the cost of the plan that achieves the private goals given that the identified preconditions hold. If there are multiple plans that can achieve the private goals after performing that public action we can use the minimal cost plan. Following this compilation, GPPP can also handle private goals.

8 Experimental results

In this section we evaluate experimentally the proposed planner (GPPP with and without the improved local planning) and heuristics (PP-LM and 3PDB). The first part of the comparison is with respect to MAFS, a state-of-the-art planner, with a projection-based landmark heuristic. We focus on MAFS as MAFS and GPPP are similar in that both are based on a forward search scheme and use heuristic functions that evaluate states (as opposed to plans). Section 8.4 provides further comparison with other state-of-the-art planners, over the domains and infrastructure from the recent CoDMAP [45]. Unless stated otherwise, GPPP means GPPP with the improved local planning phase. All evaluated algorithms were run on a single machine, simulating a multi-agent distributed execution.

8.1 Domains

In our first set of experiments we used our own implementation of the satellite, logistics, elevators, rover, and zenotravel domains, which are multi-agent versions of known single agent domains used by prior work on privacy preserving MA-STRIPS planning [35]. Our implementation is somewhat different than the one used in the competition benchmarks.

The satellite domain involves scheduling observation tasks between multiple satellites. In the logistics domain trucks and planes are tasked to move packages to their destinations. Trucks are constrained to move within city boundaries while planes are constrained to move between cities. The elevators domain represents elevators tasked to move people between floors of a large building. The zenotravel domain models planes tasked to move people between destinations. In the rover domain rovers are sampling rocks and communicating information back to a lander. The agents in the satellite, logistics, elevators, zenotravel, and rover domains are the satellites, trucks and planes, elevators, planes, and rovers, respectively.

In addition, we created two new benchmark domains called MA-Blocksworld and MA-Logistics. These new domains were created for several reasons. First, as shown below, the existing benchmark domains were not challenging enough for our planners, solving all problem instances in a short amount of time. Second, we believe that (MAP) is perhaps most interesting when agents have complementing abilities, and when tasks that can be performed by several agents require different costs from different agents. Such domains make the high-level decision making process of assigning tasks to agents more challenging. Most existing benchmarks do not exhibit this behavior. For example, in the original logistics problem planes can fly between every pair of airports, and every truck can drive between locations within a city. This results in minor differences between the various paths of a package to its destination. Moreover, in the satellite domain each agent has different instruments but there is no fact in the goal that requires more than one agent to achieve it. Thus, the level of collaboration needed in this domain is limited to allocating the goals between the agents.

In the new domains we created—MA-Blocksworld and MA-Logistics—the planner must be smarter about task allocation. In MA-Blocksworld there is a set of stacks over which blocks can be piled. Each agent possess an arm that can reach only a part of the stacks. Several stacks are shared and provide the public collaboration locations. In MA-Logistics there is a graph of cities connected by roads, divided into areas. Each agent is responsible for one area, and neighboring areas share a city. In both cases, an object (package or block) can be moved through various paths to reach its goal location. As various agents control areas of different size and structure, it may be better to prefer one agent over the other to trasnfer the object to its goal.

The very recent CoDMAP competition [45] provides an additional set of MA-STRIPS problem instances and domains that are publicly available and intended to be used as a benchmark to compare MA-STRIPS planners. CoDMAP has not been established as an official track of the International Planning Competition (IPC) yet. Nonetheless, we hope that it will be added in future IPCs (gaining an “official” status as a benchmark) and have also evaluated the performance of GPPP on these set of problems. For a detailed description of these problem instances and domains, see the CoDMAP website[4].

8.2 Evaluation methods

Comparing the performance of algorithms that solve exponential problems is a challenge, because some problems cannot be solved in reasonable time. Following common practice in the planning community, we handle this by reporting *coverage* w.r.t a timeout, i.e., comparing the number of problem instances each algorithm solved under a predefined time threshold. In our experiments we used a 20 min timeout. For the instances solved by all configurations, we also compared runtime until a solution was found. The reported runtimes include all preprocessing times that are done per problem instance. This includes the landmark detection process, as this requires knowing the exact start and goal state, but does not include the PDB construction time, as that requires knowing the list of public facts, but is agnostic to the specific start and goal states. Thus, if we expect to have many problem instances with the same domain but different start and goal states, then the preprocessing required by the PDB construction can be amortized. Note that if the domain itself changed, e.g., having different waypoints in different rover problems, then the PDB construction would have to be accounted for in each problem.

Another comparison measure we consider is the cost of the resulting plan, by summing the costs of the executed actions (by all agents) until the goal is achieved. For example, a plan in which one agent performs two actions and the other agent performs three actions, all with a uniform cost of 1, would have a cost of 5. Note that there are other ways to measure the cost of a MAP, e.g., giving lower cost to actions that can be done by agents concurrently and incorporating fairness measures to balance the work load between the agents. Experimenting and analyzing such more sophisticated evaluation criteria is left for future work.

8.3 GPPP versus MAFS

The following set of experiments compare GPPP and MAFS using different heuristics and local planning algorithms.

8.3.1 Coverage results

Table 3 presents coverage results. The results clearly suggest that GPPP is preferable over MAFS, regardless of which heuristic is used. For example, while GPPP solves 32 and 34 instances of the MA-Logistics domain (32 using PP-LM and 34 with 3PDB), MAFS with every heuristic could only solve 20 instances. In terms of which heuristic was most effective, we do not see a clear trend: for GPPP, 3PDB is preferable over PP-LM, enabling solving all instances for the MA-Logistics domain. For MAFS, however, using PP-LM is actually better than 3PBD in the MA-Blocksworld and Rover domains. That being said, the only configuration able to solve all problem instances is GPPP with the 3PDB heuristic.

8.3.2 Runtime results

Table 4 shows the average runtime over the instances solved by all algorithms. The best configuration in every domain (in terms of average runtime) is marked in bold, where a

[4] http://agents.fel.cvut.cz/codmap.

Table 3 Coverage results: number of instances solved under 20 min

Algorithm	MAFS	MAFS		GPPP	
heuristic	LM-Proj	PP-LM	3PDB	PP-LM	3PDB
MA-Logistics	20/34	20/34	20/34	32/34	**34/34**
MA-Blocksworld	25/27	26/27	25/27	**27/27**	**27/27**
Satellite	11/15	11/15	11/15	**15/15**	**15/15**
Elevators	**15/15**	**15/15**	**15/15**	**15/15**	**15/15**
Rover	**13/13**	**13/13**	11/13	**13/13**	**13/13**
Zenotravel	**13/13**	**13/13**	**13/13**	**13/13**	**13/13**
Logistics	**12/12**	**12/12**	**12/12**	**12/12**	**12/12**

Table 4 Avg. runtime (in s) for instances solved by all algorithms

Algorithm	MAFS	MAFS		GPPP	
heuristic	LM-Proj	PP-LM	3PDB	PP-LM	3PDB
MA-Logistics	11.30	5.54	7.68	0.69	**0.68**
MA-Blocksworld	71.32	20.83	25.02	8.19	**0.55**
Satellite	124.92	124.92	47.58	**0.84**	2.13
Elevators	4.79	2.45	1.96	0.45	**0.39**
Rover	4.93	4.93	2.54	1.48	**0.67**
Zenotravel	30.67	30.52	25.05	1.23	**1.09**
Logistics	5.72	2.70	7.43	0.57	**0.47**

configuration is a pair of algorithms (GPPP or MAFS) and heuristic (LM-Proj, PP-LM, or 3PDB). Following the same trends observed in the coverage results (Table 3), we observe that GPPP with either PP-LM or 3PDB is substantially faster, on average, than MAFS with LM-Proj. For example, GPPP with PP-LM is two orders of magnitude faster, on average, than MAFS with LM-Proj in the Satellite domain. Using either PP-LM or 3PDB reduces the average runtime of MAFS in most cases, but it is always slower than GPPP with either heuristic. This clearly suggests that GPPP is faster than MAFS on average.

The second observation we make is that in most cases the 3PDB heuristic is superior to PP-LM. The advantage was most substantial when using GPPP in the MA-Blocksworld domain, where using 3PDB resulted in finding a solution more than 15 times faster than when using GPPP with PP-LM. In other domains, however, the improvement was more modest. Moreover, in a single domain—Satellite—the PP-LM heuristic was better than the 3PDB heuristic. Thus, we cannot conclude that any of these heuristics strictly dominates the other. This is also observed in single-agent planning—different heuristics perform best on different domains. This has lead to several techniques for combining different heuristics, such as dovetailing [50] and alternating [41]. We leave the exploration of combining several heuristics in privacy preserving MA-STRIPS to future work.

The third observation we make is that the privacy preserving landmark heuristic we propose (PP-LM) outperforms the simpler, projection-based landmark heuristic (LM-Proj). The benefit of PP-LM is most substantial in the Elevators, Logistics, MA-Logistics, and MA-Blocksworld domains, where MAFS with PP-LM requires less than half the runtime needed for MAFS with LM-Proj. As the heuristic is computed in the same way, the key to the advantage of PP-LM is in the collaborative way it detects landmarks. This also explains why the advantage is especially observed in the Elevators, Logistics, MA-Logistics, and MA-Blocksworld domains, as these are the domains in which collaboration between agents is required to achieve the goal.

8.3.3 Solution cost results

Table 5 presents the average plan cost over all instances solved by all algorithms. Here too, bold marks the best configuration (here in terms of average solution cost) per domain. In general, GPPP with 3PDB provides the most robust configuration, resulting in plans of cost that is either the lowest or very close to it. Moreover, the advantage of 3PDB over PP-LM, in terms of solution cost, is more substantial in the MA-Blocksworld and MA-Logistics domains. This is expected, as in these domains agents have several ways to achieve the goal, involving different agents which have different capabilities and resulting in plans with different costs. As discussed above in the context of Fig. 4, such cases are example cases in which 3PDB is superior over PP-LM. In terms of plan cost, in two domains (Satellite and Rover) MAFS with the 3PDB heuristic was the best performing algorithm. This highlights that while slower, MAFS can find lower cost plans. In fact, by choosing an appropriate heuristic function (an admissible one) MAFS can be tuned to find optimal solutions [35]. This is a limitation of GPPP, which does not find optimal solutions. However, even in the two domains where the average cost of the plans found by MAFS was lower than the average cost of the plans found by GPPP, the difference between the average costs was relatively small. This suggests that GPPP computes good, even if not optimal, plans.

To provide a fuller support for the conclusions so far, we repeated the above experiments for the Satellite, Elevators, Rover, Zenotravel, and Logistics domains on a larger set of problem instances, taken from CoDMAP. The results show the same trends, and can be observed in Appendix.

8.3.4 Improved versus basic local planning

In all the above experiments we used GPPP with the improved local planning phase described in Sect. 4. Now, we compare the improved local planning with the basic local planning phase. Table 6 shows the average plan cost (under the "Cost" column) and runtime of GPPP using the 3PDB heuristic with and without the improved local planning phase.

Table 5 Avg. cost (# actions) for instances solved by all algorithms

Algorithm	MAFS	MAFS		GPPP	
heuristic	LM-Proj	PP-LM	3PDB	PP-LM	3PDB
MA-Logistics	110.80	96.20	85.70	68.25	**67.25**
MA-Blocksworld	56.87	40.83	28.74	29.78	**27.48**
Satellite	32.50	32.50	**30.10**	30.40	30.40
Elevators	34.20	33.47	27.87	**27.00**	**27.00**
Rover	33.00	33.00	**32.55**	35.00	35.73
Zenotravel	32.08	32.08	23.00	**22.23**	**22.23**
Logistics	65.83	55.42	56.17	**43.92**	**43.92**

Table 6 Comparing plan cost and runtime of GPPP with and without the improved local planning phase

Local planning	Cost (# actions)		Runtime	
	Basic	Improved	Basic	Improved
MA-Logistics	125.15	**122.94**	5.63	**3.72**
MA-Blocksworld	**32.74**	32.81	4.08	**2.04**
Satellite	47.60	**36.07**	6.75	**5.79**
Elevators	29.75	**27.88**	0.64	**0.41**
Rover	39.23	**37.69**	1.42	**0.96**
Zenotravel	28.00	**22.23**	2.12	**1.09**
Logistics	55.08	**43.92**	1.00	**0.47**

All problems were solved under the 5 min timeout using GPPP and the 3PDB heuristic with either local planning methods. Table 6 shows the average runtime for solving all the instances when using each local planning method.[5]

Marked in bold are the lowest average plan cost and lowest average runtime per domain. As can be seen, the improved local planning phase was almost always superior to basic local planning in both plan cost and planning runtime. Basic local planning was only better in the MA-Blocksworld domain in terms of plan cost, and even there the advantage was negligible (0.07). By contrast, the advantage of improved local planning, while not huge, was significant. For example, the runtime of using improved local planning in Logistics was half of the runtime required when using the basic local planning, and in Zenotravel using improved local planning resulted in more than 20% reduction in plan cost. The same general trends were also observed when comparing GPPP with and without the improved local planning phase, when using PP-LM.

Thus, in conclusion, we observe that on most domains, GPPP with improved local planning and the 3PDB heuristic is the most robust configuration in terms of both runtime and plan cost. Note that PP-LM was better in one domain (and in more domains when using MAFS), and PP-LM also requires lower pre-processing time and is more general.

8.4 Performance in CoDMAP domains

Next, we run GPPP with the improved local planning on the CoDMAP problem instances and domains, comparing GPPP to the other planners in the CoDMAP competition. All planners were run on the CoDMAP servers, which were graciously made available by the CoDMAP organizers.

We present here results for GPPP with the PP-LM heuristic. The results for GPPP with the 3PDB heuristic in these domains were not good in many domains. The main reason for the lack of success of the 3PDB is the need for a problem specific PDB definition. These definitions state which pairs of facts to consider in the PDB. Constructing the PDB automatically with an entry for every pair of public facts, did not perform well on many domains. We leave smarter automated identification of PDB entries to future research.

Table 7 shows the number of instances solved under a 30 min timeout. The results for the other planners (not GPPP) were taken from the published results of CoDMAP [6] for the

[5] In the results in the previous tables, the averages were over a subset of these instances, as other planning algorithm/heuristic configurations did not solve all problem instances under the time limit.

[6] http://agents.fel.cvut.cz/codmap/results/presentation-RESULTS.

Table 7 Coverage results for a timeout of 30 min over the CoDMAP instances

Domain	MAPlan			MAFS	MAPR	MH-		PSM		
	FF+DTG	LM-Cut	MA-LM-Cut	PP-LM	-p	FMAP	PMR	-VR	VRD	GPPP
Blocksworld	**20**	2	2	13	**20**	0	**20**	13	**20**	12
Depot	13	5	4	16	0	2	0	2	**17**	11
Driverlog	17	15	11	13	**20**	16	19	15	**20**	14
Elevators08	11	2	2	18	19	9	19	13	12	**20**
Logistics00	18	6	8	**20**	19	4	0	17	18	**20**
Rovers	**20**	1	1	**20**	19	8	**20**	11	12	**20**
Satellites	**20**	3	6	14	20	17	19	16	18	**20**
Sokoban	**18**	13	5	4	0	4	6	7	**18**	9
Taxi	**20**	**20**	15	**20**	0	**20**	19	0	0	**20**
Wireless	4	4	2	2	2	0	**7**	0	0	3
Woodworking08	16	4	5	10	0	9	0	7	**19**	18
Zenotravel	**20**	6	8	**20**	**20**	13	18	12	13	**20**
Total	**197**	81	69	170	139	102	147	113	167	187

centralized track. Results are only shown for planners that maintain privacy in a similar manner to us. Thus, we excluded the planners ADP [13,14], ADP-legacy, MARC, CMAP-q, CMAP-t, and MADLA, Anytime-LAPKT, DFS+, and SIW+ -then-BFS(f). These algorithms either do not preserve privacy at all, or preserve privacy in a different way than our algorithm. The remaining planners are:

– *MAPlan/X* An extension of the MAFS algorithm [35] using different heuristics (the *X* being the heuristic used) [44].
– *MAPR-p* A planner that plans for each agent sequentially, where goals that were not achieved by one agent are assigned to the subsequent agents.
– *MH-FMAP* A planner which uses a partial order approach [48].
– *PMR* The plan merging and replan planner [33].
– *PSM-X* A planner which constructs a finite state machine for the plans of each agent [29,49], with various enhancements (the *X* indicates which enhancements were used).

In the related work section (Sect. 9) we provide a brief explanation of each of these planning families. For a more detailed description of the planners see the CoDMAP proceedings [45].

There were 20 problems in each domain, and we highlight in bold for every domain the planners that were able to solve the largest number of problems.

As can be seen, GPPP solves all problems in four domains and is the best performing planner in four domains. The best overall planner, in terms of coverage, is MAPlan with the FF+DTG heuristic but GPPP is the second best planner, solving only 13 problems less than MAPlan with FF+DTG. GPPP's poor performance on a few domains, such as depot and blocksworld, is due to a well known shortcoming of the landmark heuristic, which can push the planners to achieve goals early and to be reluctant to explore plans that ruin already achieved goals. For example, in blocksworld, once the two lower blocks in the goal structure are exposed, the planner places one over the other, even though the bottom one is not yet on the table. This shortcoming of the landmark heuristic is well known in single agent planning [40], and hence landmarks are often combined with another heuristic, such as helpful actions. We leave the combination of our landmark heuristic (PP-LM) with other heuristic to future research. Such a combination was shown to be effective in the MH-FMAP planner [48], and in the MA-Plan algorithm [44].

9 Related work

MAP is a fundamental problem in artificial intelligence in general, and in multi-agent systems (MAS) in particular. Core agent theories such as the well known belief-desire-intention (BDI) framework [8,18] and SharedPlans [22] refer to what we call *planning* as *means-end reasoning*. These assume that such a planner exists and attempt to capture the conditions required for agents to act and plan collaboratively.

The MAP problem has many variants and many formal models have been proposed to describe a broad range of MAP problems. This includes models that are much more expressive than MA-STRIPS, able to express knowledge about more than one possible action outcome (e.g., MMDP [6]), partial observability of the current state (e.g., Dec-POMDP [4]), and explicit reasoning about communication between the agents (e.g., COM-MTPD [38]). Our focus in this work was on the simpler MA-STRIPS model, but in Sect. 10 we discuss how the ideas in this work can be adapted in more complex MAP models.

To the best of our knowledge, collaborative privacy preserving planning (CPPP) has only been discussed in the context of simple models based on MA-STRIPS. However, a notion similar to CPPP was previously discussed in the distributed constraint optimization setting (DCOP), where an "agent" is responsible for the value of a single variable, and the overall task is to find an assignment of values to variables so as to maximize a joint reward [20]. DCOP can be viewed as an instance of CPPP, where the agents' actions are assignments of values to variables. DCOP would then be a special case of CPPP where agents may perform a single (assign a value to a variable), while in the general CPPP each agent can perform a sequence of actions.

9.1 MA-STRIPS planners

The GPPP algorithm and the proposed heuristics have been mostly discussed in this work with respect to the MAFS algorithm [35]. However, there are several other approaches to solve MA-STRIPS problems. We use here the term MA-STRIPS broadly, to include any multi agent planning model that does not consider stochastic action outcomes or partial observability.

One line of work constructs for each agent a *planning state machine* (PSM), which is a state machine that represents all possible state transitions that the agent can perform [49]. Coordination between agents is achieved by intersecting the PSMs of the different agents, and more recent work discusses the minimization of this intersection [29].

Crosby et al. [13,14] proposed a different approach to solving MA-STRIPS that is intended to handle MAP problems in which agents can apply their actions concurrently given constraints on the concurrent application of the agents' actions. An example of such a constraint are the move actions of two robots: the robots can move concurrently but cannot occupy the same location at the same time. They compile a multi-agent problem to a single agent planning problem, and run an off-the-shelf single agent planner. Then, they post-process the resulting plan to compress it by allowing actions that do not conflict to be applied concurrently. The resulting planner is called ADP, and it is not privacy preserving. Combining our approaches to have a planner that solves CPPP and is able to reason about concurrent actions is a promising direction for future work.

Luis and Borrajo [33] propose another approach for solving MA-STRIPS that considers concurrent actions and privacy constraints. In the *Plan Merging by Reuse* (PMR) planner, each agent plans independently. Later, agents concatenate their plans to a full multi-agent plan. Then, PMR attempts to compress the resulting multi-agent plan by parallelizing actions that can be executed concurrently—not unlike the compression mentioned above by Crosby et al.. [14]. The resulting, compressed, plan is checked for negative interactions between the agents. If such exists, then another algorithm replans to avoid these negative interactions, reusing the previous plans to speed up the search. Their definition of privacy is slightly more restrictive than ours, only allowing facts to be private and not actions.

Another line of work adapts techniques from partial-ordered planning to solve MA-STRIPS problems in a privacy preserving manner. Representatives of this line of work include the MAP-POP [46], FMAP [47], and more recent MH-FMAP [48] algorithms. Notably, these MA-STRIPS planners support privacy, and MH-FMAP uses a combination of multiple heuristics.

There is also some work on adapting heuristics from single agent planning to MA-STRIPS planning. Štolba and Komenda [43] showed how to distribute the computation of the classical delete relaxation heuristics h_{max} and h_{add}. They also proposed a distributed computation of the h_{ff} heuristic [43], which is used by the renown single agent planner FastForward (FF) [26]. For cost-optimal planning, Štolba et al. [44] proposed two admissible heuristics for MA-STRIPS based on the admissible single agent heuristics h_{max} [5] and LM-CUT [30]. Their MA-STRIPS version of LM-CUT, denoted h_{lmc}, is similar to our PP-LM, as both are adaptation of landmark-based single agent heuristics. A key difference is that PP-LM is not admissible, and can be viewed as an adaptation of LAMA's landmark heuristic [39] while their heuristic is admissible and adapts the LM-CUT landmark heuristic [30]. The admissibility requirements makes the LM-CUT-based heuristic weaker, using fewer and larger (disjunctive) landmarks. This is because they find landmarks from a justification graph in which every action has a single precondition, while PP-LM does not make this simplification (at the cost of admissibility).

Torreno et al. [48] proposed two heuristics designed for their MH-FMAP algorithm: an adaptation of the Context-Enhanced Additive heuristic [24] and another version of a landmark-based heuristic. Their landmark heuristic is an adaptation of our PP-LM landmark heuristic to their different privacy preserving MA-STRIP model, which supports subset privacy (see Sect. 10.2) and where facts are modeled through object fluents [48].

In general, although CPPP is a relatively recent model of MAP, there is great interest in this model in the community. This was also reflected in the recent CoDMAP[7] [45], which had 9 participating teams and 17 planners, some of which were not published in other venues.

9.2 Privacy preserving as partial observability

One can view the demand for privacy in CPPP as a special case of *partial observability*. Partial observability in planning means that the planning agent is not fully aware of its current state. After performing actions, the agent may receive an observation that provides information about the current state. Partially Observable Markov Decision Problem (POMDP) is a classical model [42] for single agent planning that explicitly reasons about partial observability. Contingent planning under partial observability is a similar model that avoids describing the probability of state transitions or observations [1]. A decentralized POMDP (Dec-POMDP) [4] is the multi-agent version of a POMDP. Recently, the QDec-POMDP [12] model was suggested, describing domains similar to a Dec-POMDP without transition and observation probabilities.

[7] http://agents.fel.cvut.cz/codmap.

Typically, in Dec-POMDP and QDec-POMDP the offline planning phase is centralized, with a single controlling agent planning for all, yet the execution is distributed, i.e., each agent observes different facts about the world. In that sense, agents do obtain private information throughout the policy execution. In order to share such information, agents need to execute explicit communication actions that disclose some information they have gathered. This also allows us to model the cost of sharing information, and produce optimal plans that reveal as little as possible in order to reach the goal.

In this paper, however, we focus on distributed planning in addition to the distributed execution. Standard Dec-POMDP or QDec-POMDP algorithms are hence inappropriate for this task.

Moreover, a privacy preserving planner according to our definition cannot share private information during planning. For example, an agent does not share which private action it is capable of when planning. There is no such restriction in Dec-POMDP or QDec-POMDP planners: the planning algorithm knows about the agents' capabilities. The exact relation between privacy preserving planning and planning under partial observabiliy deserves a deeper study.

10 Privacy preserving planning beyond MA-STRIPS

In this paper we addressed CPPP in the context of the MA-STRIPS model. MA-STRIPS's power is in its simplicity, but this simplicity comes at the price of limited expressiveness. Next, we discuss how the ideas of GPPP can transfer to CPPP in more complex MAP models.

10.1 Joint actions

The MA-STRIPS, MMDP, and Dec-POMDP models all assume that an action is performed by a single agent. In reality, however, there are cases where an action can only be performed by several agents together. For example, picking up some heavy load may require two robots lifting it at the same time.

Boutilier and Brafman showed how to extend STRIPS to allow reasoning about concurrency between action execution [7]. Moreover, they also proposed a partial-ordered planning algorithm for solving such modified STRIPS problem. Thus, we believe that extending GPPP to solve CPPP in MAP models with joint actions is possible, perhaps building on the compilation approach of Crosby et al. [14] (see Sect. 9).

10.2 Subset privacy

As mentioned earlier in Sect. 2.1, our definition of privacy is less general than the definition used in the recent CoDMAP [32] allowing for *"subset privacy"*—where a fact can be private to a subset of the agents.

A possible way to adapt GPPP to support subset privacy is by using more sophisticated state representation in both global planning and local planning phases. A state in GPPP is a set of public facts and a set of PSIs, one per agent. To support subset privacy, agents that share a private fact will have a joint PSI that is mapped to the state of the private facts that are known to these two agents. Thus, the state will consist of the public facts, a PSI per agent, and a PSI per subset of agents that share a private fact.

10.3 Non-deterministic action outcomes

Classical planning, as well as typical MAP, assumes deterministic action outcomes. In many cases, however, it is useful to allow actions to have multiple outcomes, such as success and failure.

A popular way to model multiple outcomes in the planning community is the contingent planning model [1], where there can be several non-deterministic outcomes to an action. In such problems, the solution is defined as a plan tree, rather than a sequence of actions. This poses a problem to the various agents, which are unaware as to which branch of the tree is executed by their associates.

Perhaps a possible way to overcome this in CPPP is by producing a policy, where agents respond when certain public facts become available. Augmenting GPPP to handle such domains is non-trivial, and we leave this for future research.

11 Conclusion

This paper studies the problem of collaborative planning for multiple agent while preserving agents' privacy. We introduce GPPP, a novel privacy preserving planning algorithm in which the agents collaboratively search for a global coordination plan and then each agent plans to perform its role in that global plan. GPPP can handle both private and public goals, and outperforms MAFS, a previous state-of-the-art planner. Both GPPP and MAFS require heuristics to guide the planning process, and in this work we propose two novel privacy preserving heuristics: PP-LM and 3PDB. Both heuristics are much more efficient than the baseline projection-based heuristics, and in conjunction with GPPP enable solving more planning problems substantially faster.

Developing privacy preserving heuristics has only been recently explored by the planning community, and there are many open questions to address. One question is the combination of several privacy preserving heuristics within the GPPP planner. Another, perhaps broader, research question is how to address a more flexible privacy constraint, where compromising some privacy is allowed but incurs a cost. This would raise interesting challenges related to balancing loss of privacy and improved heuristic accuracy.

Another interesting topic for future work is to improve private goal handling. A major limitation of the approach presented in Sect. 7 is in cases where the generated "artificial goals" are very large disjunctions. Then, the performance of the planner would degrade substantially. Finding ways to overcome such cases is also an open challenge for future research.

Acknowledgments We thank the reviewers for their useful comments. We also thank Antonin Komoda and Michal Stolba for their extensive help with running GPPP on the CoDMAP servers. 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-Gurion University of the Negev.

Appendix: GPPP versus MAFS on CoDMAP instances

In this appendix we present additional results to those presented in Sect. 8.3, on a different set of problems: the problem instances used in CoDMAP for the domains Satellite, Elevators, Rover, Zenotravel, and Logistics. The purpose of these results is to compare the performance of GPPP and MAFS using the landmark projection heuristic, PP-LM, and 3PDB.

Table 8 shows the coverage results, i.e, number of instances solved under 30 min. Here too, we see a clear advantage for GPPP over MAFS, and for PP-LM over the projection landmarks (LM-Proj). Note that there is no clear winner when comparing PP-LM and 3PDB, suggesting that a combination of the two would work well.

The next set of results compares the runtime and solution cost of the different configurations. To do so, we average only over the instances solved by all the compared configurations (GPPP/MAFS, LM-Proj/PP-LM/3PDB). Since MAFS with the projection LM is unable to solve many instances, we compared it only against MAFS with PP-LM, and provide separate results where we compare MAFS with PP-LM against the other configurations to allow an average over more instances.

Table 9 shows the average runtime over the instances solved by MAFS with the projection LM heuristic and by MAFS with PP-LM, showing a clear advantage for MAFS with PP-LM. Table 10 presents the average runtime of all configurations (except for MAFS with the projection LM heuristic), where the average is over all instances solved by all configurations. Here we observe a clear advantage to GPPP over MAFS, and to 3PDB over PP-LM.

Regarding solution cost, we observe only one clear trend, which is that the solution found by MAFS with the projection-based heuristic is poorer than that with PP-LM. We did not observe a clear trend regarding the solution cost when comparing the other configurations (Tables 11 and 12).

Table 8 Coverage results: number of CoDMAP instances solved under 30 min

Algorithm	MAFS	MAFS		GPPP	
Heuristic	LM-Proj	PP-LM	3PDB	PP-LM	3PDB
Satellites	12/20	14/20	**20/20**	**20/20**	**20/20**
Elevators08	16/20	18/20	19/20	**20/20**	**20/20**
Rovers	8/20	**20/20**	18/20	**20/20**	**20/20**
Zenotravel	13/20	**20/20**	19/20	**20/20**	**20/20**
Logistics00	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**

Table 9 Avg. runtime (in s) for solving all instances solved by MAFS using LM-Proj or PP-LM)

Algorithm	MAFS	
Heuristic	LM-Proj	PP-LM
Satellites	375.1	**320.83**
Elevators08	197.89	**13.62**
Rovers	44.03	**2.08**
Zenotravel	84.01	**5.65**
Logistics00	6.68	**1.18**

Table 10 Avg. runtime (in s) for solving all solved by MAFS and GPPP using PP-LM or 3PDB

Algorithm	MAFS		GPPP	
Heuristic	PP-LM	3PDB	PP-LM	3PDB
Satellites	413.30	120.4	54.62	**50.72**
Elevators08	18.34	8.88	2.88	**2.07**
Rovers	93.77	161.42	21.87	**20.02**
Zenotravel	135.40	285.20	17.46	**16.24**
Logistics00	1.18	1.74	0.64	**0.51**

Table 11 Avg. cost (# actions) over the CoDMAP instances solved by MAFS using LM-Proj or PP-LM

Algorithm	MAFS	
Heuristic	LM-Proj	PP-LM
Satellites	**44**	**44**
Elevators08	58	**53.6**
Rovers	38.25	**37.25**
Zenotravel	24.46	**21.69**
Logistics00	58	**51.41**

Table 12 Avg. cost (# actions) over the CoDMAP instances solved by MAFS and GPPP using PP-LM or 3PDB

Algorithm	MAFS		GPPP	
Heuristic	PP-LM	3PDB	PP-LM	3PDB
Satellites	56	**52.5**	53.42	54.85
Elevators08	53.71	**52.47**	54.17	55.47
Rovers	**54.94**	56.5	60.27	61.66
Zenotravel	**48.63**	49.63	49.57	49.57
Logistics00	51.41	51.08	**50.83**	**50.83**

References

1. Albore, A., Palacios, H., & Geffner, H. (2009). A translation-based approach to contingent planning. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, CA, USA, July 11-17, 2009* (pp. 1623–1628).
2. Alcázar, V., Borrajo, D., Fernández, S., & Fuentetaja, R. (2013). Revisiting regression in planning. In *International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 2254–2260).
3. Bäckström, C. (1998). Computational aspects of reordering plans. *Journal of Artificial Intelligence Research (JAIR)*, *9*(1), 99–137.
4. Bernstein, D. S., Givan, R., Zilberstein, S., & Immerman, N. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, *27*, 819–840.
5. Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1), 5–33.
6. Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *the conference on Theoretical aspects of rationality and knowledge* (pp. 195–210).
7. Boutilier, C., & Brafman, R. I. (2001). Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, *14*, 105–136.
8. Bratman, M. (1987). Intention, plans, and practical reason.
9. Brafman, R. I. (2015). A privacy preserving algorithm for multi-agent planning and search. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015* (pp. 1530–1536).

10. Brafman, R. I., & Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS* (pp. 28–35).
11. Brafman, R. I., & Domshlak, C. (2013). On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence*, *198*, 52–71.
12. Brafman, R. I., Shani, G., & Zilberstein, S. (2013). Qualitative planning under partial observability in multi-agent domains. In *Proceedings of the Conference on Artificial Intelligence (AAAI), Bellevue, WA, USA, July 14–18, 2013* (pp. 130–137).
13. Crosby, M., & Petrick, R.P. (2014). Temporal multiagent planning with concurrent action constraints. In *ICAPS workshop on Distributed and Multi-Agent Planning (DMAP).*
14. Crosby, M., Jonsson, A., & Rovatsos, M. (2014). A single-agent approach to multiagent planning. *European Conference on Artificial Intelligence (ECAI)*, *263*, 237.
15. Culberson, J. C., & Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, *14*(3), 318–334.
16. Edelkamp, S. (2001). Planning with pattern databases. In *the European Conference on Planning (ECP)* (pp. 13–34).
17. Felner, A., Korf, R. E., & Hanan, S. (2004). Additive pattern database heuristics. *Journal of Artificial Intelligence Research (JAIR)*, *22*, 279–318.
18. Georgeff, M., Pell, B., Pollack, M., Tambe, M., & Wooldridge, M. (1999). The belief-desire-intention model of agency. In *Intelligent Agents V: Agents Theories, Architectures, and Languages* (pp. 1–10). Berlin: Springer.
19. Greenstadt, R., Grosz, B. J., & Smith, M. D. (2007). SSDPOP: Improving the privacy of DCOP with secret sharing. In *AAMAS*.
20. Grinshpoun, T., & Tassa, T. (2014). A privacy-preserving algorithm for distributed constraint optimization. In *International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)* (pp. 909–916).
21. Grinshpoun, T., Grubshtein, A., Zivan, R., Netzer, A., & Meisels, A. (2013). Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research*, *47*, 613–647.
22. Grosz, B., & Kraus, S. (1996). Collaborative plans for complex group action. *Artificial Intelligence*, *86*(2), 269–357.
23. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.
24. Helmert, M., & Geffner, H. (2008). Unifying the causal graph and additive heuristics. In *ICAPS* (pp. 140–147).
25. Helmert, M., Haslum, P., Hoffmann, J., & Nissim, R. (2014). Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Alternative and Complementary Medicine*, *61*(3), 16.
26. Hoffmann, J. (2001). FF: The fast-forward planning system. *AI Magazine*, *22*(3), 57.
27. Hoffmann, J. (2005). Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, *24*, 685–758.
28. Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research (JAIR)*, *22*, 215–278.
29. Jakubuv, J., Tozicka, J., & Komenda, A. (2015). Multiagent planning by plan set intersection and plan verification. *ICAART 15*.
30. Karpas, E., & Domshlak, C. (2009). Cost-optimal planning with landmarks. In *IJCAI* (pp. 1728–1733).
31. Kovacs, D. L. (2012). A multi-agent extension of pddl3.1. In *Workshop on the International Planning Competition (IPC) in the International Conference on Automated Planning and Scheduling (ICAPS)* (pp. 19–27).
32. Kovacs, D. L. (2015). Complete BNF definition of MA-PDDL with privacy. http://agents.fel.cvut.cz/codmap/MA-PDDL-BNF-20150221.pdf.
33. Luis, N., & Borrajo, D. (2014). Plan merging by reuse for multi-agent planning. In *ICAPS workshop on Distributed and Multi-Agent Planning (DMAP).*
34. Maliah, S., Shani, G., & Stern, R. (2014). Privacy preserving landmark detection. In *the European Conference on Artificial Intelligence (ECAI)* (pp. 597–602).
35. Nissim, R., & Brafman, R. I. (2014). Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research (JAIR)*, *51*, 293–332.
36. Nissim, R., Brafman, R. I., & Domshlak, C. (2010). A general, fully distributed multi-agent planning algorithm. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (pp. 1323–1330).
37. Pommerening, F., Röger, G., & Helmert, M. (2013). Getting the most out of pattern databases for classical planning. In *the International Joint Conference on Artificial Intelligence, IJCAI* (pp. 2357–2364).
38. Pynadath, D. V., & Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, *16*(1), 389–423.

39. Richter, S., & Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)*, *39*(1), 127–177.
40. Richter, S., Helmert, M., & Westphal, M. (2008). Landmarks revisited. *Association for the Advancement of Artificial Intelligence*, *8*, 975–982.
41. Röger, G., & Helmert, M. (2010). The more, the merrier: Combining heuristic estimators for satisficing planning. In *ICAPS* (pp. 246–249).
42. Sondik, E. J. (1971). The optimal control of partially observable markov processes. Ph.D. Thesis, Stanford University, United States, California.
43. Štolba, M., & Komenda, A. (2014). Relaxation heuristics for multiagent planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*.
44. Štolba, M., Fišer, D., & Komenda, A. (2015). Admissible landmark heuristic for multi-agent planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*.
45. Štolba, M., Komenda, A., & Kovacs, D.L. (2015). Competition of distributed and multiagent planners (codmap). *The International Planning Competition (WIPC-15)* (p. 24).
46. Torreño, A., Onaindia, E., & Sapena, O. (2014). A flexible coupling approach to multi-agent planning under incomplete information. *Knowledge and Information Systems*, *38*(1), 141–178.
47. Torreño, A., Onaindia, E., & Sapena, Ó. (2014). Fmap: Distributed cooperative multi-agent planning. *Applied Intelligence*, *41*, 1–21.
48. Torreno, A., Sapena, O., & Onaindia, E. (2015). Global heuristics for distributed cooperative multi-agent planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*.
49. Tozicka, J., Jakubuv, J., & Komenda, A. (2014). Generating multi-agent plans by distributed intersection of finite state machines. In *ECAI* (pp. 1111–1112).
50. Valenzano, R. A., Sturtevant, N. R., Schaeffer, J., Buro, K., & Kishimoto, A. (2010). Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In *ICAPS* (pp. 177–184).
51. Yeoh, W., Felner, A., & Koenig, S. (2010). Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. *Journal of Artificial Intelligence Research*, *38*, 85–133.