Trial-based Heuristic Tree-search for Distributed Multi-agent Planning

Tim Schulte and **Bernhard Nebel**
Institut für Informatik, Albert-Ludwigs-Universität, Freiburg, Germany
{schultet, nebel}@informatik.uni-freiburg.de

Abstract

We present a novel search scheme for privacy-preserving multi-agent planning. Inspired by UCT search, the scheme is based on growing an asynchronous search tree by running repeated trials through the tree. We describe key differences to classical multi-agent forward search, discuss theoretical properties of the presented approach, and evaluate it based on benchmarks from the *CoDMAP* competition. Furthermore, we describe a technique that enhances search by performing explorative trials subsequent to each node expansion and show empirically that this technique has a strong positive impact on the number of problems solved.

Introduction

In collaborative multi-agent planning multiple agents attempt to achieve a common goal by planning and coordinating their actions appropriately. In this work, we consider a distributed form of collaborative multi-agent planning where agents cooperate with one another while keeping various information private. *MA-STRIPS* (Brafman and Domshlak 2013) is one of the most basic formalisms for this type of privacy-preserving multi-agent planning (or *privacy-preserving planning* for short), and several planning techniques have since been proposed to solve respective tasks (Nissim and Brafman 2013; 2014; Torreño, Onaindia, and Sapena 2014). The recent emergence of a dedicated competition on distributed and multi-agent planning (CoDMAP) (Štolba, Komenda, and Kovacs 2015) emphasizes the raising interest in this field.

In this paper, we introduce a novel search technique for privacy-preserving planning based on *trial-based heuristic tree-search* (THTS) (Keller and Helmert 2013), a general scalable framework for solving different types of planning tasks. Our main contribution is the definition and evaluation of the resulting search framework, which we call *distributed multi-agent trial-based heuristic tree-search* (DMT). We present two DMT algorithms. The first approach resembles best-first search, comparable to *multi-agent forward search* (MAFS) (Nissim and Brafman 2014), the second balances exploration and exploitation similar to UCT (Kocsis and Szepesvári 2006). We show that both algorithms are sound and complete, and evaluate them on a set of benchmark problems from the CoDMAP competition.

As a secondary contribution, we describe a technique that extends the regular search approach by small explorative trials which are performed subsequent to each node expansion. (Similar to the approaches successfully implemented by the ARVAND (Nakhost and Müller 2009) and PROBE (Lipovetzky and Geffner 2011) planning systems). We show that this technique significantly increases the number of problems solved for both DMT and MAFS.

Background

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

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

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

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

$a_k(\dots(a_1(s_0))\dots) \models s_\star$. *Privacy-preserving planning* extends multi-agent planning by a notion of privacy.

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

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

Definition 4 (Privacy-preserving planning task)**.** *A* privacy-preserving planning task *is a tuple $\Pi = \langle N, V, s_0, s_\star, \{A_j\}_{j\in N}, \mathcal{P}\rangle$, where $N, V, s_0, s_\star$ and $\{A_j\}_{j\in N}$ form a multi-agent planning task and $\mathcal{P}$ is an indexed family of sets called a* privacy partition*:*

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

For each variable $v \in V$, $\mathcal{P}$ contains the set of agents $P_v \subseteq N$ that have access to v. For the purpose of this paper, we assume that all sets $P_v, v \in V$, have a cardinality of either 1 or $|N|$. Then, $\mathcal{P}$ partitions the set of variables V into a set of public *variables V^{pub}, known to all agents, and $|N|$ sets of* private *variables V_j^{pri}, each known to a single agent $j \in N$ only. Likewise, actions are partitioned into a set of public actions A^{pub} and sets of private actions A_j^{pri}:*

$$V_j^{pri} = \{v \in V \mid P_v = \{j\}\}, \text{ for } j \in N$$
$$V^{pub} = \{v \in V \mid P_v = N\}$$
$$A_j^{pri} = \{a \in A_j \mid a = a|_{V_j^{pri}}\}, \text{ for } j \in N$$
$$A^{pub} = \textstyle\bigcup_{j\in N} (A_j \setminus A_j^{pri})$$

A privacy-preserving planning task is a multi-agent planning task with the addition of a privacy partition. The privacy partition describes, which agent can access which variables or actions at plan time. Every information of a privacy-preserving planning task that a single agent can access, is again a planning task: the *local view* of agent i.

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

$$\Pi^i = \langle N, V^i, s_0^i, s_\star, \{A_j^i\}_{j\in N}\rangle, \text{ where}$$
$$V^i = V^{pub} \cup V_i^{pri},$$
$$s_0^i = s_0|_{V^i}, \text{ and}$$
$$A_j^i = (A_j \setminus A_j^{pri})|_{V^i} \text{ for } j \neq i, \text{ and } A_i^i = A_i.$$

The local view only contains projections of other agents' public actions. These projections never contain preconditions or effects on variables that are private to another agent. Assume a_j to be an action of agent j. The local view of some other agent i includes a projection a_j^i of a_j. Since a_j^i does not contain preconditions or effects private to agent j, to agent i it might appear applicable in some state s, although a_j is not applicable in s (for some private precondition is not satisfied). This is why communication between the agents during the planning process is essential. A multi-agent planning algorithm is *weakly private* if, at plan time, each agent can only access its own *local view* on the planning task and the agents never exchange private information with one another. A multi-agent planning algorithm is *strongly private* if no agent can deduce private information, including knowledge about the existence or value of a variable or action private to another agent, from the course of conversation (message history) between the agents (Brafman 2015).

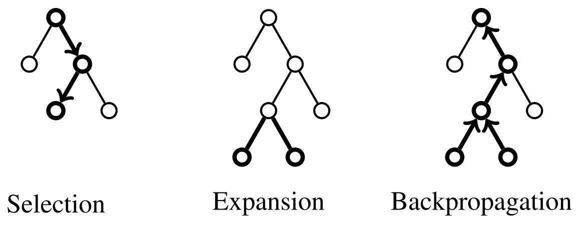

Figure 1: Phases of THTS.

Multi-Agent Forward Search (MAFS) (Nissim and Brafman 2014) is a general search scheme for privacy preserving multi-agent planning. Each agent conducts a best-first search, maintaining its own *open* and *closed* list. Successors of expanded states are generated by using the agents own actions only. Whenever an agent i expands a state for which the local view of agent i contains an applicable action projection of another agent j, a message is sent to that agent. The message contains the full state, heuristic score and g-value of the sending agent. Private fluents of the state are encrypted such that only the relevant agents can decrypt it. When agent i receives a message $m = \langle s, h_j(s), g_j(s)\rangle$ of some other agent j, it checks whether s is already in its open or closed list. If this is not the case, i puts s on its open list. If i generated state s previously with higher cost, it puts s on its open list again and assigns new costs $g_j(s)$ to it. When an agent generates a goal state, it initiates a distributed plan extraction procedure by broadcasting the goal state in a message to all agents.

Distributed Multi-agent Trial-based Heuristic Tree-search

While MAFS is locally based on best-first search, DMT is based on *trial-based heuristic tree-search* (THTS) (Keller and Helmert 2013; Schulte and Keller 2014). THTS algorithms repeatedly execute three phases. Each of these phases corresponds to a search component that must be specified in order to derive a concrete algorithm. In contrast to best-first search (BFS) approaches which expand nodes from an *open* list that is sorted by priority, THTS algorithms maintain a tree of nodes and select one of its leaf nodes for expansion in each search step. We now briefly sketch the three phases of THTS displayed in Figure 1.

1. *Selection* is the first phase of the algorithm with the objective to select one of the leaf nodes for expansion. Beginning from the root, a selection strategy recursively selects a child, until a leaf node is reached.
2. In the *expansion* phase, successor nodes of the previ-

ously selected leaf node are generated and integrated into the tree. While the algorithms presented in this paper use heuristic functions to compute state value estimates for the generated nodes, this is not a necessity. A non-systematic search approach could, for instance, use Monte Carlo rollouts to aggregate value estimates.

3. During *backpropagation* (or *backup*) phase new information, like state value estimates or the number of times a node has been visited during selection, is propagated through the tree.

After the backpropagation phase, the algorithm starts again with the first phase. This process is repeated until a goal state is generated, or some limit is reached.

We now present a complete and privacy-preserving scheme for the distributed application of trial-based heuristic tree-search. The concept is similar to MAFS, where forward-search is concurrently executed while state information is exchanged between the planning agents according to a specific message passing scheme. Each agent performs THTS locally, using its own actions only. Whenever agent i expands a state s in which an action projection of agent j is applicable, i will send a message to j containing s. Agent j then integrates s into its search tree, such that it can prospectively select s for expansion. To accomplish this, agent j identifies a suitable parent and adds s as a child to it. In principle, any node can be used as a parent without soundness or completeness being compromised. However, since the tree structure is crucial to the success of THTS algorithms, it is important where new states are integrated. Let s be the result of applying the sequence of actions $(a_1, \dots, a_k)$ in the initial state, i.e. $a_k(...(a_1(s_0))...) = s$, and let a_j be the last action of agent j in that sequence. If a_j exists, j adds s as a child to $s' = a_j(...(a_1(s_0))...)$. Otherwise, j adds s as a child to the root. Note that agent j is not aware of all actions in the sequence leading to s and hence cannot compute s'. We enable agent j to identify s' by using a special message type.

Definition 6 (State message)**.** *A* state message *from agent i to agent j for state s is a tuple* $m = \langle s, h_i, g_i, T\rangle$*, where*

- *s is a state; private components are encrypted, such that each agent can only decrypt its own private components.*
- *h_i is a value estimate of agent i for state s,*
- *g_i is the cost of agent i to establish state s,*
- *T is a set of* state tokens.

Each *state token* belongs to an agent k and contains a state identification number. This number references a node in the local search space of agent k and is meaningless to all other agents. Figure 2 illustrates how tokens are used to integrate states. Here, two agents i and j are planning concurrently. Numbers next to nodes depict state IDs that correspond to the local state represented by the node. Nodes associated with states for which an applicable action projection of the other agent exists are rendered in bold. When agent j expands the node with state ID 3, it transmits message $m_1 = \langle s, 7, 2, \{j \mapsto 3\}\rangle$ to agent i. m_1 contains a token that enables j to identify the node labelled with 3. When agent i receives m_1, it creates a new search node for s. Because m_1 contains no token for agent i, the new node is attached as a child to the root. Later on, i expands the node with state ID 5, for which j has an applicable public projection. The message $m_2 = \langle s', 5, 2, \{j \mapsto 3, i \mapsto 5\}\rangle$ is sent back, from i to j. Because state 5 was generated in a branch to which agent j contributed an *ancestor* state, the token $j \mapsto 3$ is attached to the message, along with the new token $i \mapsto 5$ of agent i. The latter token enables i to identify the state corresponding to state ID 5. When agent j receives m_2 it looks up its token $j \mapsto 3$, creates a new node for state s', and attaches it as a child to the node with state ID 3.

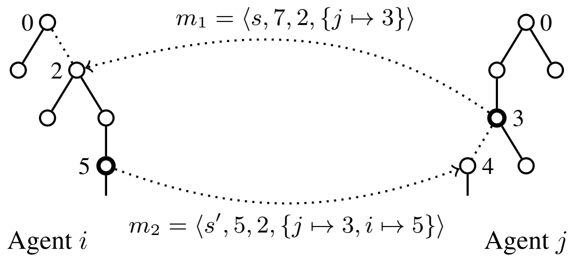
$m_1 = \langle s, 7, 2, \{j \mapsto 3\}\rangle$
$m_2 = \langle s', 5, 2, \{j \mapsto 3, i \mapsto 5\}\rangle$
0
2
5
0
3
4
Agent i
Agent j

Figure 2: State integration.

An overview of the resulting search scheme is depicted in Figure 3. The algorithms main routine is defined in Algorithm 1. Methods *process-messages, select, expand, send-messages* and *backup* correspond to *integration-, selection-, expansion-, distribution-* and *backup-phase* respectively. These components are described in detail below. For ease of exposition we define the following functions to access information stored with each search node σ:

- *state*(σ): associated search state
- *par*(σ): parent of σ
- *children*(σ): set of children of σ
- *action*(σ): action leading from *state*(*par*(σ)) to *state*(σ)
- $h(\sigma)$: value estimate for σ

We refer to a search node σ and its associated state s = *state*(σ) interchangeably where convenient.

Selection. A selection strategy is a function that maps from a set of search nodes Σ to a single node $\sigma \in \Sigma$. To ensure that the node selected last in the selection phase is an unexpanded leaf node, a special *locking mechanism* is used. The idea is to mark expanded nodes from which no unexpanded leaf node is reachable as *locked* and to ignore such nodes in the selection phase. Each expanded node σ^* without any non-locked children is locked in the backup phase by setting $l(\sigma^*) = true$. New nodes created in the expansion phase are non-locked by default. We use two selection strategies: *greedy* and *balanced*. *greedy* resembles a greedy best-first search policy, selecting the successor node σ with the lowest value estimate $h(\sigma)$. *balanced* aims to balance exploration and exploitation by using a selection formula similar to UCB1 (Auer, Cesa-Bianchi, and Fischer 2002) found

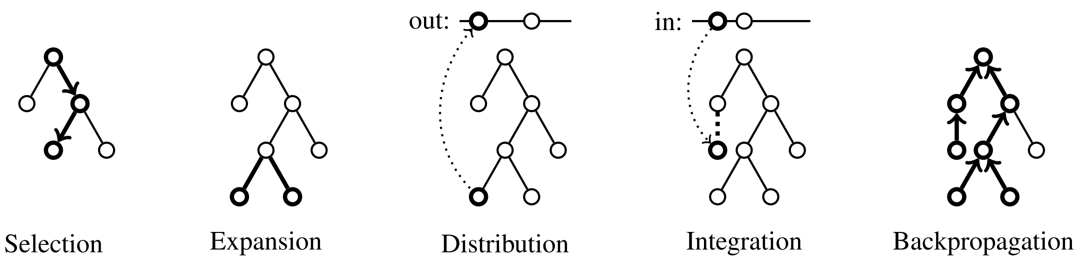

Figure 3: Phases of DMT.

Algorithm 1: DMT for agent i

```
Data: $\Pi^i = \langle N, V^i, s_0^i, s_\star, \{A_j^i\}_{j \in N}, \mathcal{P} \rangle$
Result: plan $\pi = \langle a_k \in A_i \rangle_{k=1}^K$
1  root ← new tree from $s_0$
2  while within computational budget do
3      $\sigma$ ← root
4      if $\neg l(\sigma)$ then
5          while children($\sigma$) $\neq \emptyset$ do
6              $\sigma$ ← select(children($\sigma$))
7          expand($\sigma$)                    // memorizes plans
8          send-messages($\sigma$, $N$)          // distribution
9          mark $\sigma$ for backup
10     process-messages()                    // integration
11     backup()
12 return best memorized plan
```

Algorithm 2: Expansion for agent i

```
Data: $\sigma, A_i = A_i^{int} \cup A_i^{pub}$
Result: modified tree node $\sigma$
1  $s$ ← state($\sigma$)
2  $h(\sigma)$ ← evaluate heuristic function for $s$
3  foreach action $a \in A_i$ applicable in $s$ do
4      $s'$ ← $a(s)$
5      if $s'$ is a goal state then
6          extract and memorize plan
7      if $s'$ is not in the tree then
8          $\sigma'$ ← new node
9          par($\sigma'$), action($\sigma'$), $h(\sigma')$ ← $\sigma, a, h(\sigma)$
10         state($\sigma'$), $v(\sigma')$, $l(\sigma')$ ← $s', 0, false$
11         children($\sigma$) ← children($\sigma$) $\cup \{\sigma'\}$
12     else
13         lookup $\sigma'$ where state($\sigma'$) $= s'$
14         if $g(\sigma) + c(a) < g(\sigma')$ and $\neg l(\sigma')$ then
15             mark par($\sigma'$) for backup
16             remove $\sigma'$ from children(par($\sigma'$))
17             par($\sigma'$), action($\sigma'$), $h(\sigma')$ ← $\sigma, a, h(\sigma)$
18             children($\sigma$) ← children($\sigma$) $\cup \{\sigma'\}$
```

in UCT algorithms (Kocsis and Szepesvári 2006). Formally:

$$\text{greedy}(\Sigma) = \underset{\sigma \in \Sigma, \neg l(\sigma)}{\arg\min}\ h(\sigma)$$

$$\text{balanced}(\Sigma) = \underset{\sigma \in \Sigma, \neg l(\sigma)}{\arg\min}\ \overline{h}(\sigma) - c \cdot \sqrt{\frac{\ln v(par(\sigma))}{v(\sigma)}}$$

Here, $\overline{h}(\sigma) \in [0, 1]$ is the normalized value estimate of σ, such that $\overline{h}(\sigma^\star) = 0$ for the node $\sigma^\star$ with the lowest value estimate from Σ and $\overline{h}(\sigma^-) = 1$ for the node σ^- with the highest value estimate from Σ. All other nodes $\sigma' \in \Sigma$ are interpolated accordingly. The number of times a node has been selected during selection phase is denoted by $v(\sigma)$ (visits). *balanced* selection favours nodes with fewer visits. Coefficient c is a weight bias to increase or decrease the desired amount of exploration. The higher c the higher the bias towards exploration. *greedy* and *balanced* are just two examples of selection strategies that can be used in line 6 of Algorithm 1.

Expansion. Algorithm 2 specifies how a node σ is expanded by an agent i. First, a heuristic value for *state*(σ) is computed and $h(\sigma)$ is set to that value. Then, all successor states s' are generated. For each successor state s' that is not already in the tree a new node σ' is created and added to *children*(σ); its values are set accordingly (Algorithm 2, line 9-11). If a successor state s' is already in the tree, the respective search node σ' with *state*$(\sigma') = s'$ is determined. If the new path to s' induces lower costs than the existing path, the subtree rooted at σ' is moved to *children*(σ) by adapting parent and child pointers of the involved nodes (Algorithm 2, line 16-18). Since the former parent of σ' lost a child, the value estimates of all nodes along the path from the former parent to the root are deprecated. Therefore, before σ' is moved to its new parent σ, *par*(σ') is marked to get updated in the next backup phase (line 15).

Distribution. Let σ be the node agent i expanded last. In the *distribution* phase i creates a *state message* $m = \langle state(\sigma), g(\sigma), h(\sigma), T \rangle$, such that T contains a token of i to identify σ. For each other agent the first token traceable on the path from σ to the root is attached to T. Then, agent i sends m to all agents that have a public action projection applicable in s.

Integration. Following the distribution phase agent i integrates each state s received in a message $m = \langle s, h_j, g_j, T \rangle$ into its local search tree. First, i identifies the new parent σ^* for s by looking up its token from T. If T contains no token for i, then σ^* is set to the tree's root node. If s is new to agent i, a new search node σ is created and added to *children*(σ^*).

If some node σ' representing s is already in the tree, it is moved to *children*(σ^*) in case s is reachable with lower cost that way. As in the expansion phase, when σ' is moved, its old parent is marked for backup.

Backpropagation. The backup function starts at the node σ that was expanded last and updates its values. The node's visits are increased by one, its value estimate is set to the minimum among its non-locked children, and the locked flag is set if the node itself has no non-locked child:

$$\begin{aligned} v(\sigma) &= v(\sigma) + 1 \\ h(\sigma) &= \min_{\sigma' \in children(\sigma), \neg l(\sigma)} h(\sigma') \\ l(\sigma) &= \textstyle\bigwedge_{\sigma' \in children(\sigma)} l(\sigma') \end{aligned}$$

Then backup continues with the node's parent *par*(σ) and updates it accordingly. This process is repeated until the root node is reached. In case other nodes have been marked for backup, during expansion or integration, the process is repeated for each marked node. This may lead to the same node getting updated multiple times, but can easily be avoided by using a backup queue.

Trial Length. When a node σ is expanded, all its successors are generated and associated state messages are sent. Before the agent continues with the integration phase, it can select one of the newly generated nodes and expand it as well. By alternatingly executing *selection*, *expansion* and *distribution* phase, multiple nodes can be expanded in each search step. The number of nodes to get expanded in a single search step is denoted as *trial length*. For simplicity we did not include it in Algorithm 1. It can easily be implemented by looping around lines 5-10.

An increased trial length induces additional exploration, which is supposed to help to overcome biases of the heuristic function and to exit local minima faster. The method, therefore, is most beneficial in the presence of inaccurate or misleading heuristic estimates, which is a known problem of locally computed heuristic functions in privacy-preserving planning. It turns out that the trial length is an important feature for making DMT search much more efficient. To see whether the advantages of the trial length are transferable to MAFS and for the sake of a fair comparison, we added this technique to MAFS as well.

Although the trial length is a native feature of DMT, transferring it to MAFS is straightforward: In each search step, MAFS selects the best node from its *open* list and generates all successor states. Then, the best successor (according to the heuristic function) is selected and all of its successor states are generated. Again, the best successor is selected and all of its successors are generated, and so on. For a trial length of t, this process is repeated $t-1$ times. Each node generated in the process is added to the *open* list and, if necessary, transmitted to relevant agents. Then, control is returned to the MAFS loop.

Plan Extraction

If some agent i generates a goal state a valid plan can be extracted and agent i initiates a distributed plan extraction process. First, it traces back all states of its local plan, until a state s^* is reached that was received in a state message from another agent j. Then, i sends a plan extraction request to j, including s^*. Agent j then continues to trace back its local plan, beginning from the state received in the state message. This process is repeated until some agent reaches the initial state, at which point plan extraction ends. If more planning time is available, DMT search progressively tries to generate better solutions. When a plan is extracted, its cost is computed and the plan with the best cost found so far is memorized as π. From then on each agent marks search nodes with a higher g-value than π as locked. Each time a new goal state is reached, its g-value is computed, and, if it is an improvement, π is updated. Once each agents root is locked, π is the optimal solution. If the time limit is exceeded earlier, π is returned.

Soundness and Completeness

Lemma 1. *Each state s in the search tree of an agent i is reachable.*

Proof sketch. The first state generated by DMT is the initial state. Each subsequently generated state is reached by an action applied in a previously generated state. Therefore, every state s in the search tree represents a valid sequence of actions that is applicable starting with the initial state, and that results in state s. Hence, if a state satisfies the goal, a valid plan can be extracted. □

Lemma 2. *If a goal is reachable by some sequence of actions then some agent will generate a goal.*

Proof sketch. We will only consider sequences in which a private action of an agent is followed by another action of that agent. In (Nissim and Brafman 2014) it was shown that it suffices to consider such sequences for any goal that involves public variables only. In the following we argue that the presented two selection functions (*greedy* and *balanced*), in combination with the other components presented, yield complete algorithms.

In every search step, each agent expands a new leaf node and generates all its successors. Nodes without children are locked, either because they are dead-ends or because all of their successor states can be reached on shorter paths and have been moved to other states in the tree. Therefore, all paths that do not lead to a solution will eventually be locked. Both selection functions solely select non-locked nodes and will eventually, for the lack of an alternative, select a node along a path that leads to a goal. Given sufficient time, all nodes along such a path will be selected until the goal is reached. If no such path exists in an agents local search space, the agent exhaustively generates all possible states, until its root node is locked.

We now regard sequences that involve actions of different agents and that lead to a goal state. It is easy to see that each agent transmits the last state s, established by a subsequence of its own actions, to the agent owning the next action in the sequence. If the next action is private, it is always followed by another action of the same agent, until one action is public. This actions public projection is applicable in state s, and hence sent to the agent in a state message. □

Relation to MAFS

MAFS and DMT are both schemes for distributed search algorithms that differ in the types of algorithms they support. MAFS supports forward search algorithms where nodes are expanded from an open list, while DMT supports THTS algorithms that use a search tree instead. In MAFS, states are inserted into an open list together with a static value estimate computed prior insertion. The value estimates of states in the open list never changes, hence, their relative order remains unchanged. DMT algorithms, by way of contrast, insert states into a tree together with value estimates that are continuously subject to change. Therefore, algorithms that depend on a dynamic node ordering, like UCT (Kocsis and Szepesvári 2006), can easily be expressed as DMT algorithms by defining appropriate selection, backup and expansion functions. It is not possible to implement these algorithms competitively with an open list, especially when a large number of nodes change their relative position in each search step. Another major difference between DMT and MAFS concerns the *reopening* of closed states. In MAFS, a newly generated state s is put on the open list only if it is not already on the closed list or if its new g-value is smaller than the registered g-value. In the latter case, states previously generated as successors to s will potentially be reopened in future search steps as well. In DMT, if s is already in the tree and its new g-value is smaller than the current g-value, the subtree of the existing node is moved to the node that is currently expanded. This is achieved by adapting parent and child pointers of the involved nodes (Algorithm 2, line 15-18). Successor states must not be generated all over again.

Evaluation

The presented algorithms were implemented in a distributed multi-agent planning system written in Go called GOA[1]. During the experiments, each problem instance was run on a single core of a 2.6 GHz Intel Xeon CPU and 4 GB of RAM. The assignment of processor time and memory per agent was left to the Go scheduler. Although supported by the planner, communication between the agents was realized by sharing memory instead of using a network layer. Privacy constraints were strictly met and no agent accessed memory containing private information of another agent. We experimented with the set of benchmarks from the CoDMAP competition (Štolba, Komenda, and Kovacs 2015) consisting of 12 domains with 20 problem instances each. Additionally, we included 40 problem instances of the *productionsite* domain, which is inspired by a (very abstract) smart factory. Planning time was limited to 30 minutes per planning task. In all cases, the FF heuristic (Hoffmann and Nebel 2001) was used to compute state value estimates. The heuristic function was applied to the agents' local view of the problem, containing the agents' private and public variables and actions together with the other agents' public action projections. Compared to jointly computed heuristics, this yields rather uninformed value estimates, but doesn't require extra communication. We conducted two experiments. The first experiment compares MAFS to the presented DMT algorithms: (1) DMT with *greedy* selection (DMTG) and (2) DMT with *balanced* selection (DMTB), each using a trial length of one. In a second experiment, we analyzed the impact of the trial length on the overall search performance, comparing MAFS to multiple DMT configurations that differ in the chosen trial length. Note that the results reported below differ from (but are generally consistent with) results reported in an earlier research abstract (Schulte and Nebel 2016). All configurations now use a lazy communication scheme, sending messages only when states are expanded as opposed to when they are generated. Other changes include general planner improvements and the much higher time limit of 30 minutes instead of 2.

[1]https://github.com/schultet/goa

Table 1 shows performance results for the tested base configurations: MAFS, DMTG and DMTB. We report coverage, as well as the number of messages that were sent (a message broadcast to k agents is counted as k messages), the time to completion and the number of expanded states. Except for the coverage, we report averages only over the problems that were solved by all algorithms. The numbers reflect that MAFS performs best in terms of coverage, solving 20 instances more than DMTG, and 67 instances more than DMTB. We expected greedy DMT to perform slightly worse than MAFS, because both approaches search the state space greedily, but the DMT approach is computationally more demanding (each node expansion requires to run a trial through the tree). Comparing the problems solved by all configurations, however, greedy DMT requires significantly fewer expansions in all but two domains and needs to send fewer messages in all but four domains. It also generally finds plans faster. The lower coverage is more than compensated for, when choosing a different trial length, as will be shown below. Interestingly, balanced DMT which overall performs worst, solves 15 instances of the *blocksworld* domain where neither MAFS nor greedy DMT solve a single instance. Another interesting observation, albeit not shown in Table 1, is that the plan quality vastly differs between MAFS and the two DMT variants, with MAFS having an average plan cost of 100.7, while DMTG and DMTB having an average plan cost of 53.4 and 51.8, respectively. The difference could be due to DMT breaking ties in favor of shallower nodes. This requires further investigation, though.

Next, we analyze the impact of the trial length. Figure 4 depicts the total number of problems solved by each of the three search approaches using different trial lengths. We consider all trial lengths from 10 to 400 in increments of 10, i.e. $t = 10, 20, \ldots, 400$, plus the baseline approaches ($t = 1$). For all configurations, increasing the trial length leads to a notable increase in coverage. The difference is especially pronounced in the case of greedy DMT, which solves 35 additional problem instances even when increasing the trial length only slightly by 10. The maximum coverage over all configurations is 211 (DMTG), 200 (MAFS) and 148 (DMTB), which is an increase over the baseline approaches by 30%, 11% and 29%, respectively. While the addition of explorative trials generally leads to an increase in the coverage, there are some domains where it has a detrimental effect on the coverage. To look further into this

	Coverage			Expansions			Messages			Time		
Domain	MAFS	DMTG	DMTB	MAFS	DMTG	DMTB	MAFS	DMTG	DMTB	MAFS	DMTG	DMTB
blocksworld	0	0	**15**	-	-	-	-	-	-	-	-	-
depot	3	**4**	1	4032	**2352**	46060	9904.0	**5540.0**	88704.0	**2.4**	**2.4**	20.0
driverlog	**18**	17	15	545	**416**	6612	384.9	**308.9**	2995.9	**0.3**	**0.3**	3.2
elevators	**13**	10	4	15274	**8789**	78898	8054.2	**5868.0**	56070.0	1.8	**1.5**	47.1
logistics	**20**	16	3	**11366**	14791	96648	**9542.7**	11700.7	84582.7	**1.1**	1.2	16.3
prod.site	**35**	29	20	8004	**4487**	106829	54.2	**30.4**	13548.6	**4.1**	**4.1**	211.6
rovers	**19**	13	7	**14374**	89782	208370	**5894.9**	30702.1	135844.9	**11.0**	130.9	179.5
satellites	**20**	12	3	2268	**1759**	24189	**265.5**	369.5	6336.0	1.8	**1.6**	22.4
sokoban	6	7	**11**	34069	**2048**	84927	90122.8	**2506.8**	80212.2	110.6	**3.1**	113.1
taxi	16	**19**	17	49862	**45447**	142137	66027.4	**65739.2**	226438.5	**9.0**	9.7	78.2
wireless	**2**	**2**	1	15586	**11774**	143804	27215.0	**19774.0**	195400.0	3.7	**3.4**	44.7
woodworking	10	**13**	2	5216	**3862**	3534	14609.5	10071.0	**9710.5**	**1.7**	**1.7**	1.8
zenotravel	**20**	**20**	16	11639	**2372**	24140	2925.3	**674.6**	13458.9	135.2	**21.0**	177.9
Sum/Ratio	**182**	162	115	1.0	**0.65**	5.69	1.0	**0.65**	7.64	1.0	**0.77**	7.84

Table 1: Comparing the performance of MAFS, DMTG and DMTB. For each domain the average over all problems is reported.

Domain	MAFS	DMTG	DMTB
blocksworld	**18**	16	4
depot	3	**4**	3
driverlog	**20**	**20**	17
elevators	**13**	11	6
logistics	**20**	19	8
prod.site	18	**27**	26
rovers	**20**	**20**	17
satellites	**20**	**20**	5
sokoban	9	9	**12**
taxi	18	**20**	14
wireless	2	2	**3**
woodworking	14	19	**20**
zenotravel	**20**	**20**	12
Sum	195	**207**	147

Table 2: Coverage with a trial length of 100 ($t = 100$).

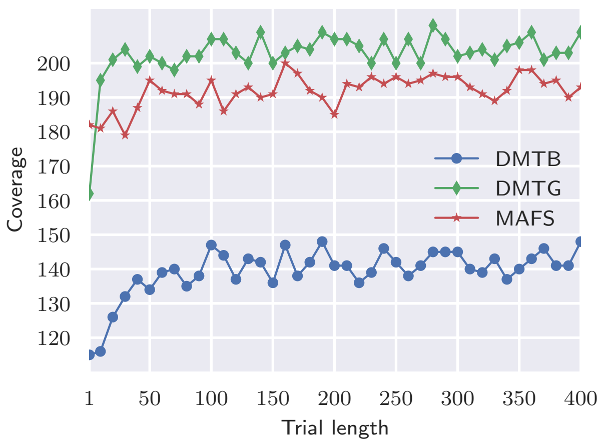

Figure 4: Total coverage (max 280) for MAFS and DMT using different trial length values.

behavior, Table 2 shows the domain coverage of the three search strategies using an exemplary trial length of 100.

The increase in coverage is most noticeable in the *blocksworld* domain where MAFS and greedy DMT previously ($t = 1$) solved zero instances, they now ($t = 100$) solve 18 and 16, respectively. However, it is also evident that the increased trial length negatively affects the balanced DMT variant which previously solved 15 instances and now solves merely 4. The opposite behaviour can be observed in the *productionsite* domain. Here, both MAFS and DMTG solve fewer problems (-17 and -2) with the higher trial length, while DMTB solves more ($+6$). To further understand the impact of the trial length, it is necessary to analyze the interplay of domain specific features, the search strategy used and the heuristic function used. The FF heuristic, for instance, yields comparatively uninformed estimates for *blocksworld* problems, resulting in huge local minima during the search. Here, the additional exploration of baseline ($t = 1$) DMTB pays off, while baseline MAFS and DMTG perform significantly worse. Increasing the trial length helps the latter two configurations to escape from local minima and therefore find plans much faster. Why the increase in trial length has a detrimental effect on the coverage of the balanced DMT version is less clear. However, it must be noted that the kind of exploration performed by DMTB is very different from the one resulting from an increased trial length. While the former converges to a breadth-first search when the exploration coefficient is set high enough, the latter behaves more like depth-first search when the explorative trials are long enough. To keep the discussion concise, we conclude that increasing the trial length causes regular search to perform additional exploration and encourages faster escape from local minima. Intuitively, this seems to be most beneficial in domains where many solution paths exist but search is misguided into local minima by inaccurate heuristic value estimates.

Figure 5 compares greedy DMT versions with different trial lengths against the baseline MAFS version. The brighter the color, the greater the trial length of the DMT configuration. We find that there is a minor impact on

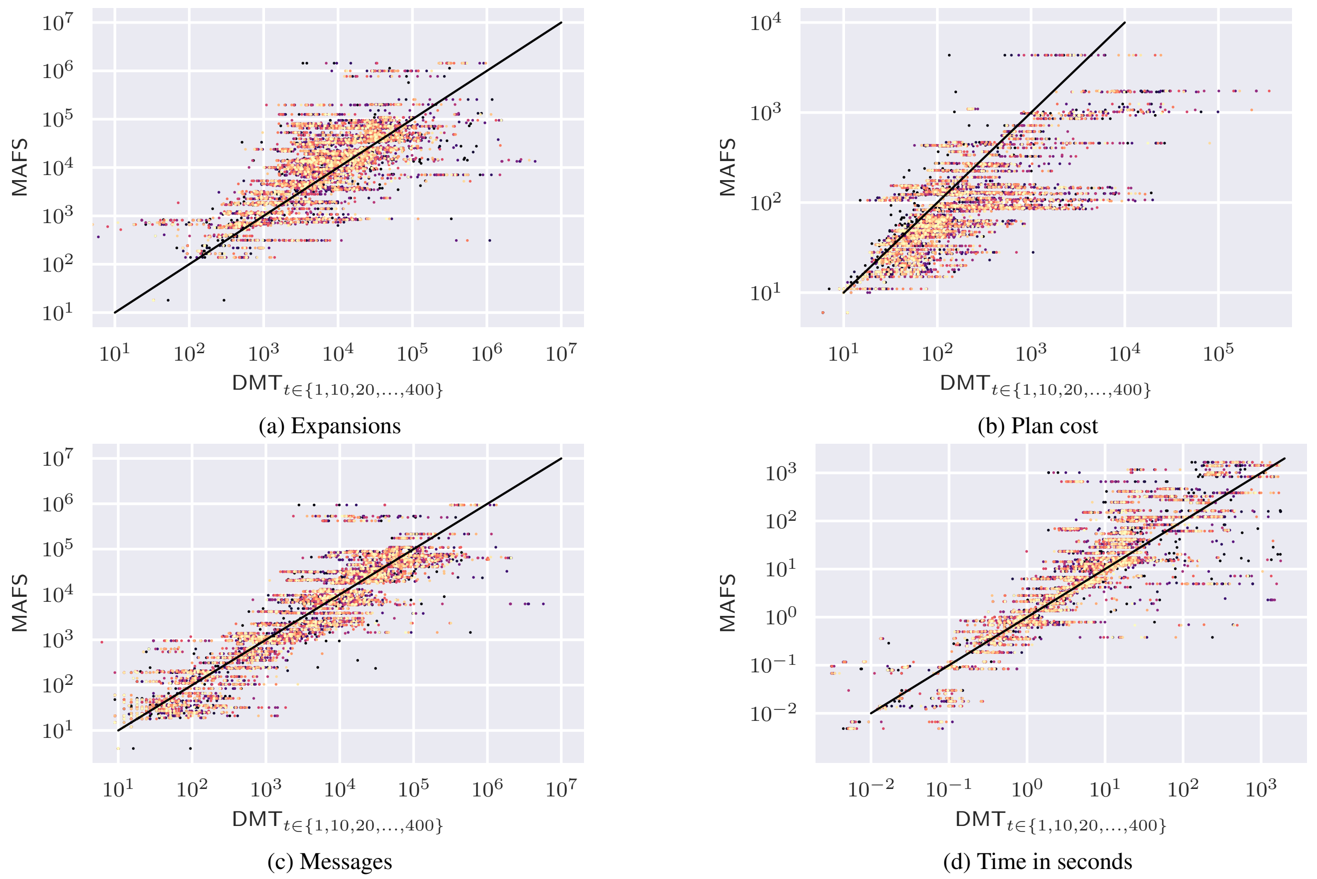

Figure 5: Expansions, time in seconds, messages and cost for baseline MAFS and DMT with trial length $1, 10, 20, \dots, 400$.

the number of expansions, slightly favoring the DMT approaches. On the other hand, increasing the trial length leads to generating plans of lower quality as Figure 5b shows. Regarding the number of messages sent and the time required to find a plan, we see no significant difference.

Lastly, we analyze how well MAFS and DMT complement each other. A portfolio planner running the three baseline ($t = 1$) versions of MAFS, DMTG and DMTB for 10 minutes each solves 205 instances. This already improves the coverage by 23 problems solved when compared to running the best of the three configurations (MAFS) on its own for 30 minutes. Combining higher trial length versions in the same fashion yields a coverage of 224. Excluding the *productionsite* domain, we have 192 problems solved. The best CoDMAP planners, PSM-VRD and MAPLAN, solved 180 and 177 instances, respectively.[2] The best planners evaluated in a setting almost identical to the setting used in this paper were GPPP, MAPLAN and DPP, with a coverage of 184, 197 and 224, respectively (Maliah, Shani, and Stern 2016). This shows that MAFS and DMT complement each other well, and that we can get comparable performance to state-of-the-art planners even when using much simpler, locally computed heuristics.

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

Conclusion

In this paper we presented DMT, a novel privacy-preserving planning scheme based on trial-based heuristic tree-search. We derived two concrete DMT algorithms and showed them to be sound and complete. Additionally, we discussed a search enhancement that extends regular search by small explorative trials which are executed subsequent to each node expansion. We showed empirically that this has a profound positive effect on the coverage for both DMT and MAFS. A comparison to classical multi-agent forward search revealed that DMT and MAFS have complementary strengths which can be exploited in a promising way. While the baseline version of MAFS had a higher coverage than the baseline version of DMT, DMT consistently outperformed MAFS when a higher trial length was chosen. The best performance and the highest coverage, however, achieved a portfolio planner that combines MAFS and DMT strategies. In future work we will create and analyze new DMT algorithms to further exploit complementary strengths. It remains interesting to see, how robust it is to enhance search by adding explorative trials across different heuristics. Furthermore, more sophisticated trial policies could increase the overall performance quite significantly.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2-3):235–256.

Brafman, R. I., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence* 198:52–71.

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR 2001)* 14:253–302.

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Proceedings of the Seventeenth European Conference on Machine Learning (ECML 2006)*, 282–293.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Twenty-First International Conference on Automated Planning and Scheduling*.

Maliah, S.; Shani, G.; and Stern, R. 2016. Stronger privacy preserving projections for multi-agent planning. In *Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*.

Nakhost, H., and Müller, M. 2009. Monte-carlo exploration for deterministic planning. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Nissim, R., and Brafman, R. I. 2013. Cost-optimal planning by self-interested agents. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*.

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

Schulte, T., and Keller, T. 2014. Balancing exploration and exploitation in classical planning. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*.

Schulte, T., and Nebel, B. 2016. Trial-based heuristic tree-search for distributed multi-agent planning. In *Ninth Annual Symposium on Combinatorial Search (SoCS 2016)*.

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

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