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

Tim Schulte
Institut für Informatik
Albert-Ludwigs-Universität
Freiburg, Germany
schultet@cs.uni-freiburg.de

Bernhard Nebel
Institut für Informatik
Albert-Ludwigs-Universität
Freiburg, Germany
nebel@cs.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.

Introduction

In multi-agent planning multiple agents attempt to satisfy a given objective by interacting appropriately. Many tasks require collaboration among agents, either because they cannot solve the problem on their own, or because they cannot do so in a cost effective way. Planning algorithms generating solutions to such problems can, in principle, implement one of two different concepts. First, *centralized* multi-agent planning algorithms grant a single agent access to the full description of the planning task. This agent then devises plans for the coordinated execution of all agents. Therefore, centralized multi-agent planning can be described as single-agent planning for multiple agents. Second, *distributed* multi-agent planning (DMAP) algorithms implement local planning by each of the agents. In contrast to centralized multi-agent planning, no trusted center is required. Each agent utilizes its own planning system that needs to exploit only those parts of the search space which are relevant to it. The agents inform each other about world states relevant to one another, therefore communication and coordination during the planning process are essential.

In this work, we consider a form of distributed 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 cooperative multi-agent planning, 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 distributed multi-agent planning. The approach is based on *trial-based heuristic tree-search* (THTS) (Keller and Helmert 2013); a general scalable framework for solving different types of planning tasks. Though originating from the field of probabilistic planning, THTS has recently been applied to classical planning (Schulte and Keller 2014). If we want to integrate THTS in a multi-agent planning context, the challenging part is to incorporate communication between the agents in such a way that the resulting algorithm preserves privacy and completeness. To achieve this, we define a suitable message passing scheme and explain how the agents can integrate states from other agents into their local search tree. Our main contribution is the definition of the resulting search framework, which we call *distributed multi-agent trial-based heuristic tree-search* (DMT). This framework extends the way of how distributed plans can be generated and so might be useful for portfolio approaches to multi-agent planning. We exemplify two DMT algorithms. The first approach resembles best-first search, comparable to MAFS, the second balances exploitation and exploration 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.

Background

We consider the problem of classical planning for multiple cooperative agents that maintain private information on their capabilities and internal states. The following definitions are based on MA-STRIPS (Brafman and Domshlak 2013) but use a multi-valued variable representation. Furthermore, privacy is not implied by the definition of the agents actions as in MA-STRIPS, but declared explicitly.

Privacy-Preserving Multi-Agent Planning

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

- N *is a finite set of agents* φ_i*, indexed* $1, \dots, |N|$*,*
- V *is a finite set of* finite-domain state variables. *Each* $v \in V$ *is associated with a domain* D_v*. A* partial variable assignment *over* V *is a function* s *on some subset of* V *such that* $s(v) \in D_v$ *wherever* $s(v)$ *is defined. A* partial variable assignment *defined for all variables in* V *is called* state.

- s_0 *is the* initial state.
- $s_\star$ *is a partial variable assignment over* V *called the goal.*
- A_i *is a finite set of actions available to agent* φ_i*. Each* action $a = \langle pre(a), \mathit{eff}(a), c(a)\rangle \in A_i$ *consists of two partial variable assignments over* V *called* precondition *and* effect*; and a* cost $c(a) \in \mathbb{R}_0^+$.

An action a is *applicable* in state s if its precondition *holds* in that state, i.e. s is identical to $pre(a)$ wherever $pre(a)$ is defined. Application of action a in state s, denoted by $a(s)$, yields *successor state* s' which is identical to $\mathit{eff}(a)$ where $\mathit{eff}(a)$ is defined, and identical to s, elsewhere. The solution to a MMPT is a sequence of actions $\pi = (a_1, \ldots, a_k)$ such that a_1 is applicable in s_0, every subsequent action is applicable in the state generated by its preceding action, and the goal holds in $a_k(\ldots(a_1(s_0))\ldots)$. Such a sequence is called *plan*. A plan is *optimal* if its incurred cost $\sum_{i=1}^{k} c(a_i)$ is minimal among all plans.

In privacy preserving domains, the set of variables V is partitioned into sets of private variables V_i^{int} containing those variables proprietary to agent φ_i, and a set of public variables V^{pub} containing the remaining variables which are common to all agents. Private variables can only be observed and be affected by actions of the agent to which the variables are private. The agents are mutually unaware of variables private to another agent. In principle, it is possible to define goals on public and private variables. For a simpler exposition of the algorithms presented below, we assume that goals are only defined for public variables $v \in V^{pub}$. In the same manner as the set of variables is partitioned into sets of private and public variables, each agents' set of actions is partitioned into a set of private actions A_i^{int} and a set of public actions A_i^{pub}. Private actions are only known to the agent to which they are private and only depend on and affect its private variables. Public actions can affect or depend on both public and private variables of the agent. During planning, the agents use both their private and public variables and actions, but restrict information exchange to the set of public variables and actions. To hide private preconditions or effects of public actions, the agents create and solely exchange public projections of their actions.

Definition 2. *A* public projection $a|_{pub}$ *of an action* a *of agent* φ_i *consists of the actions' precondition and effect restricted to the set of public variables* V_i^{pub}*:*

$$a|_{pub} = \langle pre(a)|_{pub}, \mathit{eff}(a)|_{pub}, c(a)\rangle$$

where $pre(a)|_{pub}$ *and* $\mathit{eff}(a)|_{pub}$ *are partial variable assignments over* V^{pub}*, such that* $pre(a)|_{pub} = pre(a)$ *for all variables* $v \in V^{pub}$ *for which* $pre(a)$ *is defined and* $\mathit{eff}(a)|_{pub} = \mathit{eff}(a)$ *for all variables* $v \in V^{pub}$ *for which* $\mathit{eff}(a)$ *is defined.*

The set of public projections of φ_i's public actions is $A_i|_{pub}$, the set of all agents public projections is $A|_{pub} = \bigcup_{i \in N} A_i|_{pub}$. Note that an MMPT planning task is a MA-STRIPS task when (1) the domain of each state variable is binary, (2) variables only affected or required by agent φ_i's actions are private to φ_i, and (3) actions that solely affect or depend on variables private φ_i are private to φ_i. In other words, MMPT is a generalization of MA-STRIPS.

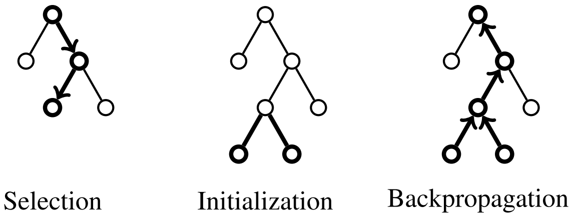


Figure 1: Phases of THTS.

Multi-Agent Forward Search

Multi-Agent Forward Search (MAFS) (Nissim and Brafman 2014) is a general search scheme for privacy preserving multi-agent planning. Each agent conducts a best-first search, maintaining its own *open* and *closed* list. Successors of expanded states are generated by using the agents own actions only. Whenever a state is generated for which another agent has an applicable public action, a message is sent to that agent. The message contains the full state, heuristic score and g-value of the sending agent. Private fluents of the state are encrypted such that only the relevant agents can decrypt it. When agent φ_i receives a message $m = \langle s, h_j(s), g_j(s)\rangle$ of some other agent φ_j, it checks whether s is already in its open or closed list. If this is not the case, φ_i puts s on its open list. If φ_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.

Trial-based Heuristic Tree-search

In the same way as MAFS is locally based on best-first search, DMT is based on trial-based heuristic tree-search. *Trial-based Heuristic Tree-Search* (Keller and Helmert 2013) is a generic search framework for probabilistic planning that was recently applied to classical planning (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 will briefly sketch the three phases of THTS using the examples 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 *initialization* phase, the previously selected leaf node is initialized. Successor Nodes are generated and integrated into the tree.
3. During *backpropagation* (or *backup*) phase new information, like value estimates or the number of times a node has been visited during selection, is propagated through the tree.

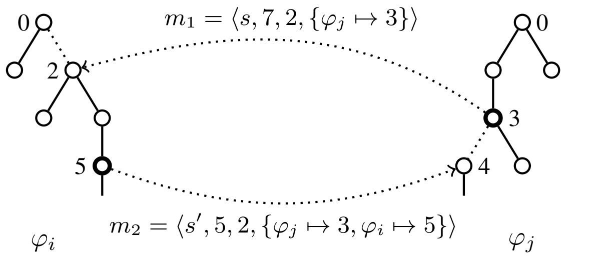
After the backpropagation phase, the algorithm starts again with the first phase. This process is repeated until a goal state is generated, or some limit is reached.

Distributed Multi-Agent THTS

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 φ_i expands a state s in which a public projection of an action of φ_j is applicable, φ_i will send a message to φ_j containing s. φ_j then integrates s into its search tree, such that it can prospectively select s for expansion. To accomplish this, φ_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 φ_j in that sequence. If a_j exists, φ_j adds s as a child to $s' = a_j(...(a_1(s_0))...)$. Otherwise, φ_j adds s as a child to the root. Note that φ_j is not aware of all actions in the sequence leading to s and hence cannot compute s'. We enable φ_j to identify s' by using a special message type.

Definition 3 (State message)**.** *A state message from φ_i to φ_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 φ_i for state s,*
- *g_i is the cost of φ_i to establish state s,*
- *T is a set of* state tokens.

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



$m_1 = \langle s, 7, 2, \{\varphi_j \mapsto 3\}\rangle$
0
2
5
0
3
4
$m_2 = \langle s', 5, 2, \{\varphi_j \mapsto 3, \varphi_i \mapsto 5\}\rangle$
φ_i
φ_j

Figure 2: State integration.

Algorithm 1: DMT for φ_i

```
Data: $\langle N, V_i^{int}, V^{pub}, s_0, s_\star, A_i^{int}, A_i^{pub}, A|_{pub}\rangle$
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          initialize($\sigma$)                 // memorizes plans
8          send-messages($\sigma$, $N$)          // distribution
9          mark $\sigma$ for backup
10     process-messages()                      // integration
11     backup()
12 return best memorized plan
```

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, initialize, send-messages* and *backup* correspond to *integration-, selection-, initialization-, 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 uninitialized leaf node, a special *locking mechanism* is used. The idea is to mark initialized nodes from which no uninitialized leaf node is reachable as *locked* and to ignore such nodes in the selection phase. Each initialized node σ^* without any non-locked children is locked in the backup phase by setting $l(\sigma^*) =$ *true*. New nodes created in the initialization phase are non-locked by default. We use the following two

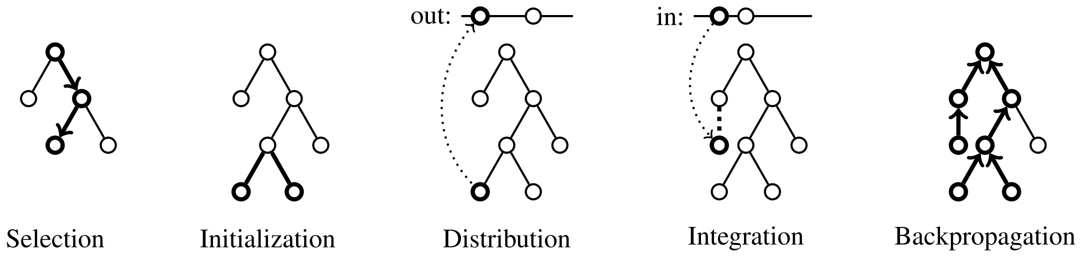


Figure 3: Phases of DMT.

selection strategies.

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

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

gbfs constitutes a greedy best-first search variant, selecting the successor node σ with the best (minimum) value estimate $h(\sigma)$.

ucb aims to balance exploration and exploitation by using a selection formula similar to UCB1 (Auer, Cesa-Bianchi, and Fischer 2002) found in UCT algorithms (Kocsis and Szepesvári 2006). 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 best (minimum) value estimate from Σ and $\overline{h}(\sigma^-) = 1$ for the node σ^- with the worst (maximum) 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). *ucb* 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. *gbfs* and *ucb* are just two examples of selection strategies that can be used in line 6 of Algorithm 1.

Initialization Algorithm 2 specifies how a node σ is initialized by an agent φ_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 φ_i initialized last. In the *distribution* phase φ_i creates a *state message* $m = \langle state(\sigma), g(\sigma), h(\sigma), T\rangle$, such that T contains a token of φ_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, φ_i sends m to all agents that have a public action projection applicable in s.

Algorithm 2: Initialization for φ_i

```
Data: $\sigma, A_i = A_i^{int} \cup A_i^{pub}$
Result: modified tree node $\sigma$
1  $s \leftarrow state(\sigma)$
2  $h(\sigma) \leftarrow$ evaluate heuristic function for $s$
3  foreach action $a \in A_i$ applicable in $s$ do
4      $s' \leftarrow a(s)$
5      if $s'$ is a goal state then
6          extract and memorize plan
7      if $s'$ is not in the tree then
8          $\sigma' \leftarrow$ new node
9          $par(\sigma'), action(\sigma'), h(\sigma') \leftarrow \sigma, a, h(\sigma)$
10         $state(\sigma'), v(\sigma'), l(\sigma') \leftarrow s', 0, false$
11         $children(\sigma) \leftarrow children(\sigma) \cup \{\sigma'\}$
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             $par(\sigma'), action(\sigma'), h(\sigma') \leftarrow \sigma, a, h(\sigma)$
18             $children(\sigma) \leftarrow children(\sigma) \cup \{\sigma'\}$
```

Integration Following the distribution phase φ_i integrates each state s received in a message $m = \langle s, h_j, g_j, T\rangle$ into its local search tree. First, φ_i identifies the new parent σ^* for s by looking up its token from T. If T contains no token for φ_i, then σ^* is set to the tree's root node. If s is new to φ_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 initialization phase, when σ' is moved, its old parent is marked for backup.

Backpropagation The backup function starts at the node σ that was initialized last and updates its values. The nodes 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:

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

Then backup continues with the nodes 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 initialization 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 initialized, 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 initialize it as well. By alternatingly executing selection-, initialization- and distribution phase, multiple nodes can be initialized in each search step. The number of nodes to get initialized 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.

Plan Extraction

If an agent φ_i generates a state that satisfies the goal a valid plan can be extracted. φ_i informs all other agents about the goal state and 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 φ_j. Then, φ_i sends a plan extraction request to φ_j, including s^*. φ_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. The solution plan is sequential but can often be parallelized.

The first solution found is not necessarily the optimal solution. Therefore, if more planning time is available, DMT search can easily be extended to progressively search for 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 φ_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. Completeness must be decided individually for each concrete DMT algorithm, because it depends on the components used. In the following we argue that the presented two selection functions (*gbfs* and *ucb*), in combination with the other components presented, yield complete algorithms.

In every search step, each agent initializes 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 distributing search algorithms, such that completeness and privacy is preserved. They differ in the types of algorithms that 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 initialization 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 the two approaches concerns the *reopening* of closed states. In MAFS, a newly

	$t = 1$			$t = 100$		
Domain	*mafs*	*dmt-bfs*	*dmt-gus*	*mafs*	*dmt-bfs*	*dmt-gus*
blocksworld	-	-	-	3	-	2
depot	1	1	-	2	-	4
driverlog	16	16	15	17	16	16
elevators	-	-	-	1	-	-
logistics	8	5	1	9	5	2
rovers	10	6	1	19	19	18
satellites	3	2	3	6	11	9
sokoban	4	8	8	3	9	8
taxi	17	14	11	10	14	6
wireless	2	2	-	1	1	-
woodworking	6	3	1	5	4	6
zenotravel	13	12	12	13	13	13
Total (240)	80	69	52	89	92	84

Table 1: Coverage

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 initialized. 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 DMAP algorithms were implemented in a distributed multi-agent planning system written in Go. Experiments were run on a PC with an Intel 3.2 Ghz quad-core CPU and 4 GB of RAM. The four cores were shared among all agents; assignment of processor time was left to the Linux process scheduler. For communication between processes a TCP connection was used. 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. Planning time was limited to two minutes per planning task. Table 1 shows coverage results for the tested configurations: Multi-agent forward search (*mafs*), DMT with greedy selection (*dmt-bfs*) and DMT with ucb selection (*dmt-ucb*). The configurations were tested with a trial length of either 1 or 100. In all cases FF heuristic (Hoffmann and Nebel 2001) was used to compute state value estimates. The heuristic function was applied to the agents local problem projection, containing the agents private and public variables and actions together with the other agents public actions projections.

Regarding configurations with a trial length of 1 ($t = 1$), the numbers reflect that *mafs* performs best, solving 11 instances more than *dmt-bfs*, and 28 instances more than *dmt-gus*. The only domain in which *dmt-bfs* and *dmt-gus* solve more instances than *mafs* is *sokoban*. We expected *dmt-bfs* to perform slightly worse than *mafs*, because both approaches search the state space in a greedy manner, but the DMT approach is computationally more expensive. Due to the brief time limit of 2 minutes, this also affects coverage. When the trial length is set to 100 ($t = 100$) all configurations improve in coverage. *dmt-gus* records the biggest gain solving 32 additional instances, followed by *dmt-bfs* with 23 and *mafs* with 9 additional instances solved. The increase in coverage is most noticable in the *rovers* domain where *mafs*, *dmt-bfs* and *dmt-gus* increase their coverage by factor 1.9, 3.17 and 18.0 respectively. Increasing the trial length causes regular search to perform additional exploration and encourages faster escape from local minima. This is most beneficial in domains where many solution paths exist but search is misguided into local minima by inaccurate heuristic values. When combining the solutions solved between configurations, we find that the two MAFS configurations solve 102 problems combined, while the DMT configurations solve 110 problems combined. A portfolio planner running *dmt-gus*, *dmt-bfs* and *mafs* with $t = 100$ for 2 minutes each solves 117 instances, which shows that MAFS and DMT complement each other well.

Conclusion

In this paper we presented DMT, a novel and privacy preserving scheme for distributing THTS algorithms. Based on DMT, we derived two concrete algorithms and showed them to be sound and complete. The algorithms were evaluated on a set of benchmark instances from the CoDMAP competition and compared to classical multi-agent forward search. Overall, DMT and MAFS approaches performed equally well, complementing each other in a promising way. In future work we will create and analyze new DMT algorithms to further exploit such complementary strengths. Additionally, we would like to use DMT in settings where goals are also defined for private variables.

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.

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.

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)*.

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.