Artificial Intelligence 252 (2017) 175–210

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

The MADLA planner: Multi-agent planning by combination of distributed and local heuristic search

Michal Štolba *, Antonín Komenda

Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague, Karlovo náměstí 13, Praha 2, 121 35, Czech Republic

ARTICLE INFO

Article history:
Received 21 January 2015
Received in revised form 11 August 2017
Accepted 18 August 2017
Available online 30 August 2017

Keywords:
Multi-agent planning
Privacy-preserving
Automated planning
Multi-agent systems
State-space search
Multi-heuristic search
MA-STRIPS

ABSTRACT

Real world applications often require cooperation of multiple independent entities. Classical planning is a well established technique solving various challenging problems such as logistic planning, factory process planning, military mission planning and high-level planning for robots. Multi-agent planning aims at solving similar problems in the presence of multiple independent entities (agents). Even though such entities might want to cooperate in order to fulfill a common goal, they may want to keep their internal information and processes private. In such case, we talk about privacy-preserving multi-agent planning.

So far, multi-agent planners based on heuristic search used either a local heuristic estimating the particular agent's local subproblem or a distributed heuristic estimating the global problem as a whole. In this paper, we present the Multi-Agent Distributed and Local Asynchronous (MADLA) Planner, running a novel variant of a distributed state-space forward-chaining multi-heuristic search which combines the use of a local and a distributed heuristic in order to combine their benefits. In particular, the planner uses two variants of the well known Fast-Forward heuristic. We provide proofs of soundness and completeness of the search algorithm and show how much and what type of privacy it preserves. We also provide an improved privacy-preserving distribution scheme for the Fast-Forward heuristic.

We experimentally compare the newly proposed multi-heuristic scheme and the two used heuristics separately. The results show that the proposed solution outperforms classical (single-heuristic) distributed search with either one of the heuristics used separately. In the detailed experimental analysis, we show limits of the planner and of the used heuristics based on particular properties of the benchmark domains. In a comprehensive set of multi-agent planning domains and problems, we show that the MADLA Planner outperforms all contemporary state-of-the-art privacy-preserving multi-agent planners using a compatible planning model.

1. Introduction

If planning is to be used in large-scale personal, corporate or military applications, multiple independent entities will need to cooperate during the plan synthesis process. As witnessed in many other applications, such independent entities

* Corresponding author.
E-mail addresses: michal.stolba@agents.fel.cvut.cz (M. Štolba), antonin.komenda@agents.fel.cvut.cz (A. Komenda).

http://dx.doi.org/10.1016/j.artint.2017.08.007
0004-3702/

may have serious concerns in protecting the privacy of their input data. Privacy-preserving multi-agent planning allows the definition of factors of the global planning problem private to the respective entities (i.e., agents). The recently most prevalent model for Multi-Agent Planning (MAP) goes back to the roots of research in planning itself—to STRIPS. The STRIPS model [1] formalized planning by a propositional description of the world and the actions, together forming a transition system representing the planning problem. An extension of STRIPS towards MAP denoted as MA-STRIPS [2] generalizes the model by allowing more than one finite set of actions, each characterizing capabilities of one agent. Since not all agents must necessarily be capable of influencing the whole environment, some parts of the information about it can be private to a single agent, alongside the common information public to all agents.

In contrast to previous models, MA-STRIPS targets cooperative domain-independent multi-agent planning for discrete and deterministic environments. In this setting, each agent is planning for itself, communicating with other agents in order to ensure agreement on common coordination points. Although execution is not explicitly considered (we are talking about off-line planning), it is assumed that the agents will execute the plans in parallel, interacting at the mentioned coordination points. The privacy in MA-STRIPS can be also understood as a special case of partial observability, where the agents should not *know*, *observe* and/or *use* private knowledge of other agents. Although the privacy in MA-STRIPS is induced (i.e., implied by the partitioning of the problem), the principles described in this paper are similarly applicable when the privacy is explicitly defined as a part of the input.

The real-world motivation for MA-STRIPS spans over a wide variety of problems [3], as in classical planning [4]. The examples can be a consortium of cooperating logistic companies with common transportation tasks, but private know-how about local transport possibilities; a team of spatially separated gas station inspectors with a common goal to analyze quality of gasoline in the whole country, but with private knowledge about the quality of particular gas stations; or a heterogeneous fleet of satellites and rovers surveying a distant planet, for which keeping local information private is the only feasible way not to overload the communication network. In these motivational cases, MA-STRIPS partitioning of actions would be defined over trucks, inspectors, rovers and satellites. The know-how of mentioned corporations, knowledge of inspectors and local information of robots would define private knowledge in MA-STRIPS and prevent sharing all of the information freely among all the agents.

As in classical planning, forward-chaining state-space search guided by an automatically derived heuristic is a well established technique for multi-agent planning. A multi-agent variant of the well-known A* search algorithm was proposed as Multi-Agent Distributed A* (MAD-A*) in [5] and the multi-agent variant of general best-first search as Multi-Agent Best-First Search (MA-BFS) in [3,6].

The MAD-A* planner uses classical heuristics restricted to *projected* problems, that is each particular agent is using the heuristics only on the part of the MAP problem it has access to. Such projection can substantially underestimate the cost as it does not consider private actions of any other agent. A seeming remedy is to *distribute* the process of heuristic estimation among all agents such that each agent preserves its privacy by not communicating anything else than its partial heuristic estimations.

One of the most prominent classical planning heuristics still in use is the Fast-Forward (FF) [7] heuristic. A distributed estimator for the FF heuristic, proven to return the same estimations as centralized FF, is MA-FF [8] (based on building a distributed form of a Relaxed Planning Graph [9]). MA-FF is, however, practically inefficient, more precisely, the results indicate that there is a trade-off between the quality of the heuristic estimation and the efficiency of its computation. This is similar to classical planning, where the computation of more informed heuristic is typically more time consuming. In multi-agent planning, the communication overhead of the distributed heuristic adds to the complexity of more informed heuristic and must be considered. Two distributed variants of MA-FF focusing on practical improvement are lazily computed lazyFF [8,6] and rdFF using recursive distributed computation [6]. Although both heuristics improve efficiency of the underlying search in contrast to MA-FF, the projected variant of FF still performs better in a non-negligible number of planning problems, because of its computational ease and lack of communication requirements [6]. A next step forward we take in this article is to combine the projected and distributed variants of FF.

The principle of combining multiple heuristic estimators is well known in classical planning as multi-heuristic search [10, 11], yet, it was never analyzed in the context of MAP. Specifically, the way to combine projected and distributed heuristic estimators in the multi-agent distributed state-space search resulting in an efficient planning approach, is an open problem. This is because the methods used in classical planning are clearly not suitable for this class of heuristics. As the main contribution of this article, we address this issue by designing a new search scheme, proving its soundness and completeness and evaluating it experimentally.

In practice, most of the MA-STRIPS-based multi-agent planners operate similarly to the classical planners. They are off-line, in the sense that the planner is presented with an input describing the planning problem, runs for some given time and either returns a solution, or reports that no solution has been found. The common input description language in classical planning is PDDL [12], but there is no standard in multi-agent planning (the recently introduced MA-PDDL [13] attempts to become one).

Similarly to many other multi-agent planners, the MADLA planner starts with a classical PDDL input and a description of which PDDL objects are actually agents. After grounding the PDDL input to an internal representation of the global planning problem, it is partitioned to the agents and private and public parts are determined automatically, based on the properties of the problem and the MA-STRIPS definition.

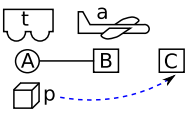

Fig. 1. Logistics example with two agents.

In particular, each action is assigned to the agent which (its respective PDDL object) first appears in the action parameters. Each action is assigned to exactly one agent (the first agent in the parameters), but the action may refer to other agents (their respective PDDL objects) as objects of the action. Imagine a construction crane and a truck, each operated by an agent, the process of loading a concrete slab by the crane into the truck is an action done solely by the crane, although the truck is involved in the action only as a target (object), therefore the action is owned by the crane or more precisely the agent representing the crane. There are no actions without an assigned agent and no joint actions requiring co-operation of two or more agents and no actions without an agent. This places some requirements on the domain designer and also the domain design influences which objects are agents and which actions belong to which agent. Based on this factorization, the public facts are determined. The public facts are those shared among at least two agents (according to the MA-STRIPS definition) or appearing in the common goal. All other facts are private. Subsequently, actions which operate with public facts are public.

The planner is run so that each agent uses its own thread and memory space, communicating with each other only through message passing via a TCP-IP connection (possibly on a network). An alternative way to provide an input to a multi-agent planner is such each agent receive its own separate (possibly PDDL) input, with factorization and privacy defined explicitly (possibly not adhering to the MA-STRIPS definition). This option is currently not supported by the MADLA Planner, but the techniques and algorithms presented in this paper are well suited for such approach (and the used formalism even suggests such use).

Throughout the paper, we use a small logistics example depicted in Fig. 1. In this example, there are two agents, a truck t and a plane a (naturally, the agents could be the drivers, the logistic companies, etc. here we adhere to the benchmark domains commonly used in multi-agent planning literature). They are transporting a package p from city A to city C, while the truck can move between A and B and the plane between B and C.

The paper is structured as follows. First, we define a formalism based on MA-STRIPS and define privacy in MAP. Next, we describe the novel planning system, MADLA Planner, with detailed focus on the new variant of the distributed FF heuristic and the novel distributed multi-heuristic search scheme. We prove the soundness and completeness of the search algorithm and provide analysis of privacy of both the search and the heuristics. In the last section, we evaluate the planner by analyzing the performance of the particular building blocks, comparing it with a centralized solution and evaluating it against other comparable state-of-the-art multi-agent planners.

2. Multi-agent planning

The global Ma-STRIPS definition becomes confusing when describing more complex distributed algorithms, therefore we use a more suitable definition. To describe a multi-agent planning problem, where each agent is planning for itself, we explicitly define a set of classical STRIPS problems, one for each agent. Otherwise, we keep the MA-STRIPS treatment of privacy mentioned above. We also explicitly define that if a fact is a goal fact, it is public, resulting in one common public goal state definition. This is a simplification[1] assumed by most MA-STRIPS planners, but not explicitly stated in the MA-STRIPS definition. The formal definition is as follows.

Definition 1. Let $\mathcal{A} = \{\alpha_i\}_{i=1}^{n}$ be a set of n agents and P be a set of propositions describing the world, where $s \subseteq P$ is a state of the world. Then $\mathcal{M} = \{\Pi_i\}_{i=1}^{n}$ is a *multi-agent planning problem* where for each agent $\alpha_i \in \mathcal{A}$, an *α_i-agent planning problem* is a quadruple $\Pi_i = \langle P_i \subseteq P, A_i, s_I \cap P_i, G \rangle$, where:

- P_i is a finite set of propositions describing facts about the world relevant to agent α_i.
  - A fact $p \in P_i$ is public if $p \in P_j$ for some $j \neq i$. Let $P_i^{\mathsf{pub}} \subseteq P_i$ be the set of public facts and $P_i^{\mathsf{priv}} \subseteq P_i$ be the set of private facts, then $P_i^{\mathsf{pub}} \cup P_i^{\mathsf{priv}} = P_i$ and $P_i^{\mathsf{pub}} \cap P_i^{\mathsf{priv}} = \emptyset$.
  - To comply with MA-STRIPS, we assume that each public fact p is known to all agents, that is if $p \in P_i^{\mathsf{pub}}$, then also $p \in P_j^{\mathsf{pub}}$ for all $j \in 1..n$. Thus we can ignore the agent index and state $P^{\mathsf{pub}} = P_i^{\mathsf{pub}}$ for all $i \in 1..n$.
- A_i is a finite set of actions agent α_i can perform.
  - Each action $a \in A_i$ is a tuple $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a), \mathsf{lbl}(a), \alpha_i \rangle$, where $\mathsf{pre}(a) \subseteq P_i, \mathsf{add}(a) \subseteq P_i, \mathsf{del}(a) \subseteq P_i$ represent preconditions, add effects, and delete effects respectively according to the standard STRIPS syntax and semantics, $\mathsf{lbl}(a)$ is a unique (across all agents) label of the action a and the agent α_i is the owner of the action a. Each action is owned exactly by one agent exclusively, that is, there are no joint actions and no actions without an assigned agent.

[1] Without loss of generality, a problem with a set of private goals can be converted to an equivalent problem with a single public goal by adding a new public goal fact and adding an action which has private preconditions on all the private goal facts and a single effect which is the new public goal fact.

  - An action a is public if either $\mathsf{pre}(a) \cap P^{\mathsf{pub}} \neq \emptyset$, $\mathsf{add}(a) \cap P^{\mathsf{pub}} \neq \emptyset$ or $\mathsf{del}(a) \cap P^{\mathsf{pub}} \neq \emptyset$. Otherwise a is private to α_i.
  - Let $A_i^{\mathsf{pub}} \subseteq A_i$ be the set of public actions and $A_i^{\mathsf{priv}} = A_i \backslash A_i^{\mathsf{pub}}$ the set of private actions of agent α_i.
  - The sets of actions of agents are pairwise disjoint, that is $\forall i \neq j : A_i \cap A_j = \emptyset$.
  - Let $A = \bigcup_{i=1}^{n} A_i$ be the set of all actions in $\mathcal{M}$.
  - An action a is applicable in state s if $\mathsf{pre}(a) \subseteq s$. The application of a in s is represented as $s[a] \mapsto s'$, where the resulting state is $s' = (s \backslash \mathsf{del}(a)) \cup \mathsf{add}(a)$.
- $s_I \subseteq P$ is the initial state of the world, each agent observes only its part $s_I \cap P_i$,
- $G \subseteq P^{\mathsf{pub}}$ is the common goal condition defining the goal (final) states of the problem; a state s is a goal state iff $G \subseteq s$.

For the example from Section 1, the truck problem Π_{t} consists of the following (public facts and actions are bold):

- $P_{\mathsf{t}} = \{$truck-at-A, truck-at-B, package-at-A, **package-at-B**, package-in-t, **package-at-C**$\}$
- $A_{\mathsf{t}} = \{$move-t-A-B, move-t-B-A, load-t-A, **load-t-B**, unload-t-A, **unload-t-B**$\}$
  move-t-A-B $= \langle \{$truck-at-A$\}, \{$truck-at-B$\}, \{$truck-at-A$\}$, move-t-A-B, t$\rangle$
  move-t-B-A $= \langle \{$truck-at-B$\}, \{$truck-at-A$\}, \{$truck-at-B$\}$, move-t-B-A, t$\rangle$
  load-t-A $= \langle \{$truck-at-A, package-at-A$\}, \{$package-in-t$\}, \{$package-at-A$\}$, load-t-A, t$\rangle$
  load-t-B $= \langle \{$truck-at-B, **package-at-B**$\}, \{$package-in-t$\}, \{$**package-at-B**$\}$, **load-t-B**, t$\rangle$
  unload-t-A $= \langle \{$truck-at-A, package-in-t$\}, \{$package-at-A$\}, \{$package-in-t$\}$, unload-t-A, t$\rangle$
  unload-t-B $= \langle \{$truck-at-B, package-in-t$\}, \{$**package-at-B**$\}, \{$package-in-t$\}$, **unload-t-B**, t$\rangle$
- $s_I \cap P_{\mathsf{t}} = \{$truck-at-A, package-at-A$\}$
- $G = \{$**package-at-C**$\}$

and the plane problem Π_{a}:

- $P_{\mathsf{a}} = \{$plane-at-B, plane-at-C, **package-at-B**, package-in-a, **package-at-C**$\}$
- $A_{\mathsf{a}} = \{$move-a-B-C, move-a-C-B, **load-a-C**, **load-a-B**, **unload-a-C**, **unload-a-B**$\}$
  move-a-B-C $= \langle \{$plane-at-B$\}, \{$plane-at-C$\}, \{$plane-at-B$\}$, move-a-B-C, a$\rangle$
  move-a-C-B $= \langle \{$plane-at-C$\}, \{$plane-at-B$\}, \{$plane-at-C$\}$, move-a-C-B, a$\rangle$
  load-a-C $= \langle \{$plane-at-C, **package-at-C**$\}, \{$package-in-a$\}, \{$**package-at-C**$\}$, **load-a-C**, a$\rangle$
  load-a-B $= \langle \{$plane-at-B, **package-at-B**$\}, \{$package-in-a$\}, \{$**package-at-B**$\}$, **load-a-B**, a$\rangle$
  unload-a-C $= \langle \{$plane-at-C, package-in-a$\}, \{$**package-at-C**$\}, \{$package-in-a$\}$, **unload-a-C**, a$\rangle$
  unload-a-B $= \langle \{$plane-at-B, package-in-a$\}, \{$**package-at-B**$\}, \{$package-in-a$\}$, **unload-a-B**, a$\rangle$
- $s_I \cap P_{\mathsf{a}} = \{$plane-at-B$\}$
- $G = \{$**package-at-C**$\}$

The locations of the truck and plane are private to the respective agents as well as the location of the package unless it is in the cities B or C. The location of the package in B is public because it is shared by the two agents, whereas the package in location C is public because **package-at-C** is a public goal. Because of that, also the **unload-t-B**, **unload-a-C** and the respective load actions are public.

The agents are not restricted only to their Π_i problems, as each agent is aware of other agents' public actions in the form of projections. An i-projection of an action $a \in A_j$ is defined as

$$a^{\triangleright i} = \left\langle \mathsf{pre}(a) \cap P_i, \mathsf{add}(a) \cap P_i, \mathsf{del}(a) \cap P_i, \mathsf{lbl}(a), \alpha_j \right\rangle.$$

The i-projected action $a^{\triangleright i}$ where $a \in A_j$ is restricted only to the public facts P^{pub}, as no other facts than P^{pub} are shared among agents α_i and α_j, thus we can omit the agent index i and write simply

$$a^{\triangleright} = \left\langle \mathsf{pre}(a) \cap P^{\mathsf{pub}}, \mathsf{add}(a) \cap P^{\mathsf{pub}}, \mathsf{del}(a) \cap P^{\mathsf{pub}}, \mathsf{lbl}(a), \alpha_j \right\rangle.$$

Note also that the label $\mathsf{lbl}(a)$ and the agent α_j owning the original action a is retained in the projection.

An i-projected problem is defined as

$$\Pi_i^{\triangleright} = \left\langle P_i, A_i^{\triangleright} = A_i \cup \{a^{\triangleright} | a \in \bigcup_{j \in 1 \ldots n \wedge j \neq i} A_j^{\mathsf{pub}}\}, s_I^{\triangleright i} = s_I \cap P_i, G \right\rangle.$$

An i-projected state $s^{\triangleright i}$ is a global state $s \subseteq P$ restricted to P_i, formally $s^{\triangleright i} = s \cap P_i$. Analogously a publicly projected state $s^{\triangleright \mathsf{pub}}$ is a global state $s \subseteq P$ restricted to P^{pub}, formally $s^{\triangleright \mathsf{pub}} = s \cap P^{\mathsf{pub}}$.

Although the agent α_i cannot plan using $\Pi_i^{\triangleright}$, as the resulting plan might not be executable in the multi-agent problem $\mathcal{M}$, it can be used for heuristic computation (the projected problem is in fact an abstraction of $\mathcal{M}$).

In the example, the problem $\Pi_{\mathsf{t}}^{\triangleright}$ projected to agent t differs from Π_{t} in that it contains the additional projected actions, that is

- **load-a-C**$^{\triangleright} = \langle \{$**package-at-C**$\}, \emptyset, \{$**package-at-C**$\}$, **load-a-C**, a$\rangle$
- **load-a-B**$^{\triangleright} = \langle \{$**package-at-B**$\}, \emptyset, \{$**package-at-B**$\}$, **load-a-B**, a$\rangle$

- **unload-a-C**$^{\triangleright} = \langle \emptyset, \{\textbf{package-at-C}\}, \emptyset, \textbf{unload-a-C}, \text{a}\rangle$
- **unload-a-B**$^{\triangleright} = \langle \emptyset, \{\textbf{package-at-B}\}, \emptyset, \textbf{unload-a-B}, \text{a}\rangle$

Notice the empty preconditions of the actions **unload-a-B**$^{\triangleright}$ and **unload-a-C**$^{\triangleright}$ as all the facts plane-at-B, plane-at-C and package-in-a are private to the agent a. The projected problem for agent a, that is $\Pi^{\triangleright}_{\text{a}}$ is defined analogously.

Let us now formally define the notion of a multi-agent plan. Let $\pi = (a_1, ..., a_m)$ be a sequence of actions from A (the actions may be owned by different agents). The sequence π is applicable in state s_0 if there are states $s_1, ..., s_m$ s.t. for $1 \leq k \leq m$, action a_k is applicable in s_{k-1} and $s_k = s_{k-1}[a_k]$. The sequence of actions π will be referred to as a s_0–s_m-plan and $s_0[\pi]$ denotes the resulting state s_m. For a state s, an s-plan is a sequence of actions π from A applicable in s, which results in a goal state, that is $G \subseteq s[\pi]$. An s_I-plan is a *multi-agent plan* solving the multi-agent problem $\mathcal{M}$. Clearly, the multi-agent plan may contain public and private actions of different agents. An $s^{\triangleright i}$-plan solving $\Pi^{\triangleright}_i$ is defined analogously, but using only the actions from $A^{\triangleright}_i$. Any multi-agent s-plan can be expressed in a *distributed multi-agent plan* form $\{\pi_i\}^n_{i=1}$, where π_i retains all actions in $\pi \cap A_i$ and all actions from $A \backslash A_i$ are replaced with an empty (no-op) action $\epsilon = \langle \emptyset, \emptyset, \emptyset, -, -\rangle$. In other words, a_k in π_i is a_k from π if $a_k \in A_i$. Otherwise, a_k in π_i is ϵ, that is a no-op action with no preconditions and effects.

A multi-agent plan solving the example problem consists of the following 6 actions:

$$\langle \text{load-t-A}, \text{move-t-A-B}, \textbf{unload-t-B}, \textbf{load-a-B}, \text{move-a-B-C}, \textbf{unload-a-C}\rangle$$

and can be reformulated as a distributed multi-agent plan

$$\{\langle \text{load-t-A}, \text{move-t-A-B}, \textbf{unload-t-B}, \epsilon, \epsilon, \epsilon\rangle,$$
$$\langle \epsilon, \epsilon, \epsilon, \textbf{load-a-B}, \text{move-a-B-C}, \textbf{unload-a-C}\rangle\}$$

Notice, that an optimal solution to $\Pi^{\triangleright}_{\text{t}}$ is a plan consisting of a single action $\langle \textbf{unload-a-C}^{\triangleright}\rangle$. The projected problem $\Pi^{\triangleright}_{\text{a}}$ is analogous and can be solved using four actions, in particular $\langle \textbf{unload-t-B}, \textbf{load-a-B}, \text{move-a-B-C}, \textbf{unload-a-C}\rangle$.

In the rest of the article, we will use the term *plan* for multi-agent plans. We will also say that a state s is a *dead end* (or dead-end state), if no s-plan exists. We assume the quality criterion for a plan π to be its length $|\pi|$, that is a plan π is optimal iff it contains the minimal number of actions among all plans solving $\mathcal{M}$. Nevertheless, all techniques presented in this article are easily modified for a quality criterion based on the (non-negative) cost of actions.

Definition 2. A (distributed) algorithm is a *multi-agent planner* iff it accepts a multi-agent planning problem $\mathcal{M} = \{\Pi_i\}^n_{i=1}$ as an input and produces a set of sequences $\{\pi_i\}^n_{i=1}$ of actions from $A \cup \{\epsilon\}$ s.t. each π_i contains only actions from $A_i \cup \{\epsilon\}$. Such a planner is

sound iff every set of sequences $\{\pi_i\}^n_{i=1}$ produced is a distributed multi-agent plan for $\mathcal{M}$,
complete iff the multi-agent planner produces a distributed multi-agent plan for any multi-agent problem $\mathcal{M}$ for which a plan exists.

Notice that according to Definition 2, any classical planner accepting the defined input (e.g., by converting it to the global STRIPS problem) is a multi-agent planner. In the next section we will specify some more interesting properties a multi-agent planner should satisfy to set it apart from the classical planners, namely the property of being *privacy-preserving*.

A heuristic function is a function estimating the length of the shortest s-plan for a given state s. A *heuristic estimator* is the actual algorithm computing the heuristic function. We formalize two variants:

Definition 3. A function $h^{\mathcal{M}} : 2^P \to \mathbb{Z}^+_0$ is a *global heuristic function* for a multi-agent planning problem $\mathcal{M}$. If the function $h^{\mathcal{M}}$ is computed by a distributed heuristic estimator, we say it is a *distributed (global) heuristic*.

Definition 4. For an agent α_i, a function $h^{\triangleright}_i : 2^{P_i} \to \mathbb{Z}^+_0$ is an i-projected heuristic function.

The term *local heuristic* denotes the i-projected heuristic function for an unspecified agent. For brevity, we refer to a heuristic function simply as a heuristic.

2.1. Privacy in multi-agent planning

As already emphasized, privacy is an important aspect of multi-agent planning. Privacy has been thoroughly studied in the context of distributed constraint satisfaction problems [14,15], but sparsely in the context of (multi-agent) planning. A significant contribution is the concept of *privacy-preserving multi-agent planning* (PP-MAP) described in [3].

The idea of privacy-preserving planning is based mainly on the research in secure multi-party computation [16], which is a subfield of cryptography. In secure multi-party computation, multiple agents are jointly computing a function, each agent having a portion of the function input as private data. The goal is to compute the function without revealing the private input data (except for what can already be reasonably deduced from the resulting value of the function).

In secure multi-party computation, various assumptions are placed on the adversary model (i.e., the other agents trying to violate the privacy), the network model and the security guarantees. Here, we assume the agents to be *honest, but curious*. This means that the agents will not purposefully deviate from the defined protocol, but may try to deduce some additional information during the computation. This is consistent with the idea of cooperative planning—the agents are cooperating to achieve a common goal and not trying to exploit each other. Concerning the network model, as with other algorithms in this paper, we assume asynchronous communication without information loss and with in-order delivery.

To summarize the assumptions:

a) Agents are honest, but curious.
b) Agents may use polynomial time and memory in order to deduce additional information from the algorithm.
c) The communication channels are asynchronous and safe, that is, no information is lost.

The most comprehensive definition of privacy for MA-STRIPS planning up to date has been presented in [3], together with discussion of privacy of the distributed multi-agent heuristic search we are building upon in this work. The privacy in MA-STRIPS based multi-agent planning based on [3] can be formally defined as follows. For each agent $\alpha_i \in \mathcal{A}$, the private part of its problem Π_i is

i) the set of private facts P_i^{priv} and its size,
ii) the set of private actions A_i^{priv}, i.e., the number of actions, the facts in $\mathsf{pre}(a)$, $\mathsf{add}(a)$ and $\mathsf{del}(a)$,
iii) the private parts of the public actions in A_i^{pub}, i.e., the private facts in $\mathsf{pre}(a)$, $\mathsf{add}(a)$ and $\mathsf{del}(a)$.
iv) In addition, private actions in a multi-agent plan π must be kept private to their respective agents.

The public parts of actions in A_i^{pub} can be shared in the form of projections. The algorithms may fall in two categories.

Definition 5. A *(weak) privacy-preserving* algorithm is a distributed algorithm that does not directly communicate any private part of the agent problems.

Definition 6. A *strong privacy-preserving* algorithm is a distributed algorithm wherein no agent α_i can deduce existence of a private fact or an isomorphic (that is differing only in renaming) model of a private action or private part of a public action belonging to some other agent α_j, except for what can already be deduced from the projected problem $\Pi_j^{\triangleright}$ and the resulting distributed multi-agent plan $\{\pi_i\}_{i=1}^n$.

This definition of privacy is a reasonable baseline, even though covering only extreme cases. It is straightforward to design a weak-privacy preserving algorithm, but the guarantees are weak (it may be possible to deduce a complete isomorphic or equivalent problem formulation). On the other hand, designing a strong-privacy preserving algorithm is extremely hard. It is often not possible to make the algorithm strong privacy-preserving without incurring significant (or even forbidding) increase in the computation complexity. An example of such algorithm is the treatment of the shortest path problem in [17], where it is solved using a cryptographic technique polynomial in the size of the input graph. Since in planning the implicit transition graph is already exponential in the size of the input problem, such approach is clearly intractable.

We are interested in algorithms which fall somewhere in-between the weak and strong privacy definition.

Definition 7. A *privacy-preserving multi-agent planner* is a multi-agent planner (Definition 2) which is privacy-preserving (Definition 5).

The MAD-A* was shown in [3] to be a privacy-preserving multi-agent planner, but is not strong privacy-preserving. It does not communicate the private parts, but some information can be deduced. More in-depth discussion is provided later in the description of the MADLA Search.

3. The MADLA planner

The MADLA (Multi-Agent Distributed and Local Asynchronous) Planner proposed in this article is a sound and complete MA-STRIPS privacy-preserving multi-agent planner (Definition 7). In order to find a solution (multi-agent plan) to a multi-agent planning problem $\mathcal{M}$ the MADLA Planner employs a distributed search through the state space induced by the problem. If a solution exists, it returns a distributed multi-agent plan $\{\pi_i\}_{i=1}^n$ solving $\mathcal{M}$. The novel aspect of the search is that it is guided by a combination of two heuristics: h_{L}^i and h_{D}^i. The first heuristic h_{L}^i is a local projected heuristic (Definition 4), evaluating the single agent's view (projection) of the problem, that is, $\Pi_i^{\triangleright}$. The other one is a distributed heuristic h_{D}^i (Definition 3) evaluating the multi-agent problem $\mathcal{M}$ in a distributed manner. In order to increase the efficiency of the planning process, MADLA Planner exploits properties often present in the distributed and projected heuristics.

A distributed heuristic estimator can require computation of parts of the heuristic estimate by other agents. The heuristic estimator may require all (or some) agents to compute the heuristic synchronously and wait for the replies of other agents. On the contrary, if the heuristic estimator does not wait for the responses from other agents (i.e., is non-blocking), it allows the agent to run asynchronously and perform some other computations in the meantime. We say that such distributed heuristic estimator is *asynchronous*.

A projected heuristic evaluates states based on the projected problem $\Pi_i^{\triangleright}$. As we have seen in the example in Section 1, a projected problem often misses crucial information (such as the preconditions of **unload-a-C**$^{\triangleright}$) and the estimate is lower than the estimate for the global problem $\mathcal{M}$ (the optimal plan for $\Pi_t^{\triangleright}$ has the length of 1 although 6 is the length of the optimal solution for $\mathcal{M}$). Often, a distributed heuristic dominates its projected counterpart, which means that for each state s, $h^i_{\mathsf{D}}(s) \geq h^i_{\mathsf{L}}(s)$ generally holds.

Although a distributed heuristic does not necessarily dominate a projected heuristic (or not in all states), dominance is a desirable property, because otherwise time is wasted on distributed computation without any benefit. In sub-optimal planning, dominance of heuristics does not automatically mean better informativeness as the heuristics are not admissible. Thus we are rather looking for better informed heuristics. Informally, a heuristic function h_1 is better informed than heuristic h_2 for a search algorithm A if that search algorithm guided by h_1 expands fewer states than the same algorithm guided by h_2. A distributed heuristic is typically better informed than a projected heuristic as it takes other agents private part of the problem into account.

It is also obvious that a distributed heuristic will typically take longer time to compute than the projected one as a) it is computed on a larger problem and b) it involves communication, which is more time-consuming than local computation.

With the exception of the asynchronous estimator of the distributed heuristic, the properties described above are not crucial for the usability of heuristics in the proposed search scheme, but are important to increase the efficiency of the planning process. In the next sections, we present the MADLA Search scheme, the heuristics used in the MADLA Planner and show the soundness and completeness of the search. We also assess how much privacy is preserved in all described algorithms.

3.1. Search

In order to find a plan, the MADLA Planner searches through a state-space (where states are $s \subseteq P$) induced by the input multi-agent planning problem, driven by a pair of a projected (local) heuristic and a distributed (global) heuristic. From the literature [6,18], we know that the performance of local and distributed heuristics differ over various multi-agent planning problems. The novel search scheme we are proposing combines the local and distributed heuristics in a manner inspired by the classical multi-heuristic search [10], but modified to be suitable for similar (or as in our case the same) heuristics which differ only in that one is computed on the projected problem and the other one on the global problem using a distributed algorithm.

Since MADLA is a distributed multi-agent planner, the search is a distributed computation spread over the agents. We build on the idea of distributed state-space search [5,6] and extend it to a distributed multi-heuristic search. We base the algorithm on the Greedy Best-First Search (GBFS) principle (the states are ordered solely according to the heuristic value) and also use the technique of deferred heuristic evaluation. Deferred heuristic evaluation means that unlike in A*, the heuristic estimate of a state is computed when it is extracted from the open list and its children are inserted into the open list according to the heuristic value of their parent state. This is useful especially for a time-consuming heuristic which is typically the case when it is computed distributedly.

In this section, we first briefly introduce the techniques we are building on—the distributed heuristic search and the classical multi-heuristic search and later on describe the novel MADLA Search algorithm.

3.1.1. Multi-agent single-heuristic search

A multi-agent variant of heuristic search was proposed in [5] as a multi-agent optimal search (MAD-A*) and in [6,3] as Multi-Agent Greedy Best-First Search (MA-BFS). The search in MADLA adheres to the latter.

The main principle of multi-agent heuristic search is that all agents explore their portions of the search space asynchronously and in parallel, each agent using only its actions from A_i. In order to manage the coordination, states expanded using a public action are sent to all other agents. Thus the other agents are informed about the new reached state and can expand it. A simplified principle is illustrated in Fig. 2. The figure is simplified in that the states are expanded in synchronous steps, whereas in reality both agents proceed asynchronously.

In *multi-agent single-heuristic search*, each agent α_i has its own separate open list O^i, closed list C^i and a heuristic function h^i. The search begins with $O^i = \{s_I^{\triangleright i}\}$ and $C^i = \emptyset$ for each agent. In parallel, each agent α_i extracts a state $s^{\triangleright i}_{min} = \arg\min_{s^{\triangleright i} \in O^i} h^i(s^{\triangleright i})$ from O^i, adds $s^{\triangleright i}_{min}$ into C^i, computes a new heuristic value $h^i(s^{\triangleright i}_{min})$ and expands $s^{\triangleright i}_{min}$. All $s'^{\triangleright i} \in S$, where S is the set of new expanded states

$$S = \{s'^{\triangleright i} | s'^{\triangleright i} = s^{\triangleright i}[a], a \in A_i \text{ s.t. } \mathsf{pre}(a) \subseteq s^{\triangleright i}\} \setminus C^i$$

are added into the open list O^i (and communicated to other agents if the expanding action was public, as described later). The search terminates if any agent α_i finds $s^{\triangleright i}$ s.t. $G \subseteq s^{\triangleright i}$, or if all open lists are empty and no communication is waiting to be processed. Recall that the goal G is public and thus it is the same for all agents.

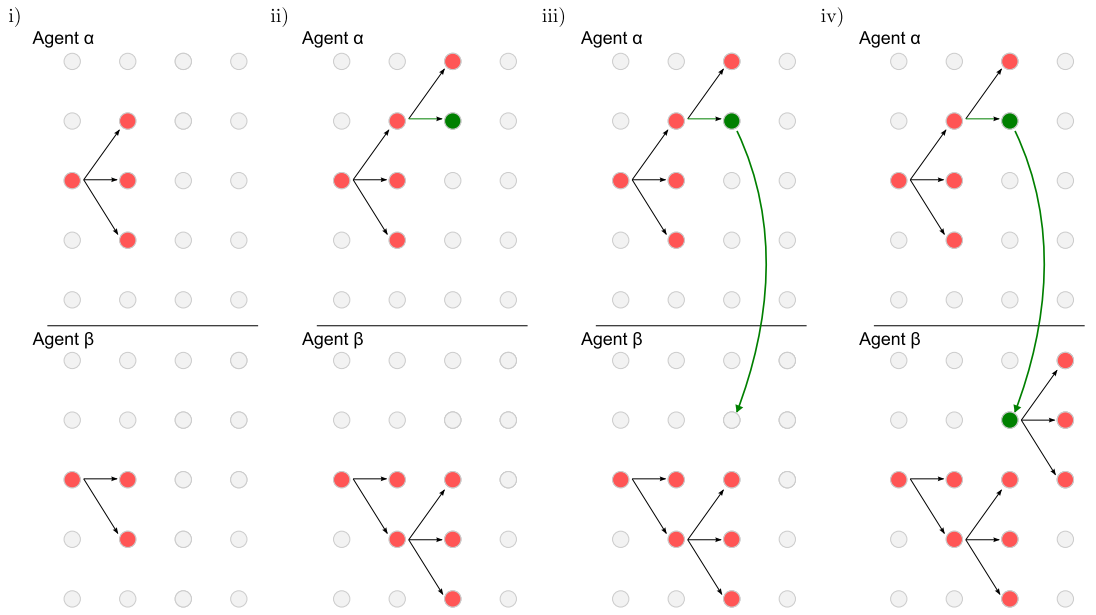

Fig. 2. Simplified example of four steps in a multi-agent heuristic search for two agents α and β. In i both agents expand the initial state by private actions (black arrows), in ii agent α expands one state by a public action (green arrow and circle) and in iii it is sent to agent β. In iv) agent β expands the received state. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

As each agent uses only its own actions for the state expansions, it is necessary to communicate reached states to other agents in order to ensure completeness. In MA-BFS, the states are communicated via message broadcasts, as exemplified later. If a state $s^{\triangleright i}$ is expanded by an agent α_i using a public action $a \in A_i^{\text{pub}}$, the state is sent to all other agents $\alpha_{j \neq i}$ and added to their open lists O^j. In order to hide the private facts $P_i^s = P_i^{\text{priv}} \cap s$ of a sent state s, the agent α_i can obfuscate the facts in P_i^s (as proposed in [19]) or replace the facts in set P_i^s with a private unique identifier (or a hash value) $\delta_i(s)$ known only to α_i (this private part of the state cannot be modified by other agents). If a modified state amended by such identifier returns by one of the later broadcasts back to the agent α_i, it can use the identifier $\delta_i(s)$ and restore the private facts P_i^s back as if they were always part of the state. The initial state is treated as a special case. As already said, each agent α_i starts only with its projection of the initial state, that is $s_I^{\triangleright i}$. As all agents start in fact from the same initial state s_I (although not completely observable by any of them), the unique identifier can use some specific value (the same for all agents), such as $\delta_i(s_I) = 0$ for all i. When an agent α_j receives a state from α_i with its private part P_j^s replaced by the identifier $\delta_j(s_I) = 0$, agent α_j knows that it must restore the values in P_j^s from the initial state, that is, $s_I^{\triangleright j}$.

When a goal state s_g s.t. $G \subseteq s_g$ is found, the solution plan needs to be also reconstructed in a distributed way. This can be done by modifying the parent of a state to contain either the previous state (and the respective action) as in classical search, or a reference to the agent from which the state was received. The backward reconstruction of the plan then proceeds as usual, except for when instead of a state, the parent is an agent α_j, in which case a message is sent to α_j to continue the plan extraction process.

We illustrate the process on the running example in Fig. 3. Each agent starts with its projection of the initial state $s_I = s_0 = \{\text{truck-at-A}, \text{plane-at-B}, \text{package-at-A}\}$. The truck starts expanding $s_0^{\triangleright \text{t}} = \{\text{truck-at-A}, \text{package-at-A}\}, \delta_{\text{a}}(s_0)$ using the load-t-A and move-t-A-B actions sequentially. Further on in the process, when the truck expands the state $s_2^{\triangleright \text{t}} = \{\text{truck-at-B}, \textbf{package-at-B}\}, \delta_{\text{a}}(s_0)$ using the action **unload-t-B**, the resulting state $s_3 = \{\text{truck-at-B}, \text{plane-at-B}, \textbf{package-at-B}\}$, seen by the truck as $s_3^{\triangleright \text{t}} = \{\text{truck-at-B}, \textbf{package-at-B}\}, \delta_{\text{a}}(s_0)$, is sent to the plane as $\{\textbf{package-at-B}\}, \delta_{\text{t}}(s_3), \delta_{\text{a}}(s_0)$. Here $\delta_{\text{t}}(s_3)$ encodes the private part of the truck in the state s_3. When received by plane, it reconstructs the state s_3 as $s_3^{\triangleright \text{a}} = \{\text{plane-at-B}, \textbf{package-at-B}\}, \delta_{\text{t}}(s_3)$ by using the private part of the state with the identifier $\delta_{\text{a}}(s_0) = 0$ which is s_0.

3.1.2. Classical multi-heuristic search

In classical planning, multi-heuristic search was pioneered by the Fast Downward planning system [10] as a way to combine different heuristic estimators without the need to combine the heuristic values, and was also one of the main mechanisms behind the success of the LAMA planner [20].

The principle of multi-heuristic search is simple. Instead of a single open list O, multi-heuristic search uses a set of open lists $\{O_1, \ldots, O_m\}$, one for each used heuristic $h_1, \ldots, h_m$. In each search step, a state is extracted from one open list according to a open list selection function. Then the state is evaluated by each heuristic and its successors placed in the respective open list (if using the deferred evaluation scheme). This means that if a state s is evaluated by a heuristic h_k, its successors are placed in O_k.

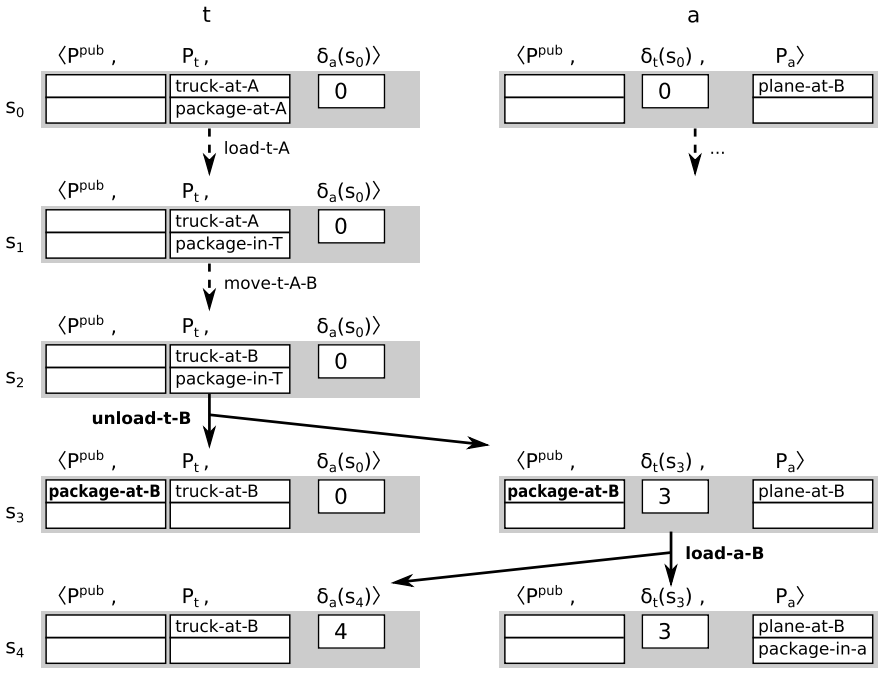

Fig. 3. State communication with private values encrypting and restoring.

The choice of an appropriate open list selection function for classical planning was thoroughly examined in [11] with a conclusion that the simple alternation mechanism, where the open lists are chosen in turns, appears to be the best one (this mechanism was used in both FD and LAMA planners).

The idea of multi-heuristic search has not yet been used in multi-agent planning. A straightforward utilization of the multi-heuristic search in multi-agent planning is to extend the MA-BFS scheme with multiple heuristics (and thus multiple open lists) for each agent. If we aim to use such a search scheme not with significantly different heuristics but with a projected and distributed variant of the same heuristic, we encounter several limitations of this simple approach. Mainly if the states evaluated by the projected heuristic, which is arguably less informed and thus gives worse estimates, are placed in the open list of the distributed heuristic, they would skip better states evaluated by the more informative distributed heuristic. Nevertheless, we propose this simple approach as a baseline and present a significantly better one in the following section.

3.1.3. MADLA search: multi-agent distributed and local asynchronous search

The main contribution of this paper is the MADLA Search, which is a variant of the multi-agent multi-heuristic search mentioned in Section 3.1.2. As in MA-BFS (Section 3.1.1), the search is performed in parallel by all agents, each searching using its own set of actions and communicating states expanded by public actions, in order to reach a common public goal. The plan is then extracted in a distributed manner, so that each agent knows only its respective part of the plan.

The main distinctive feature of the MADLA search is its use of two open lists per agent, where the first one is associated with a local projected heuristic and the second one with a distributed global heuristic. The open list selection function is tailored to handle this special case. The main high-level principles of the search are the following:

a) Evaluate a state only using a single heuristic.
b) Prefer the distributed heuristic, and only if the distributed heuristic is waiting for replies from other agents (i.e. is busy), use the projected heuristic instead.

An overview of the principle is shown in Fig. 4 and it is elaborated on in the next paragraphs. In the main search loop, after processing the communication (i.e. receiving and sending queued messages), a state is extracted from an open list. Which open list is used for the extraction is determined based on whether the distributed estimator of h^i_{D} is busy and the open lists are empty or not (as shown in Fig. 4). As we use deferred heuristic evaluation, the extracted state is evaluated by a heuristic (after it was checked for being in the closed list or being a solution), again depending on the state of the distributed heuristic estimator. If the extracted state was created using a public action, it is sent to all other agents. Then, the state is expanded using all applicable actions of the agent and its successors are added to the respective open list(s) as shown in Fig. 4 with the heuristic estimate of the parent state (again because of the deferred heuristic evaluation used).

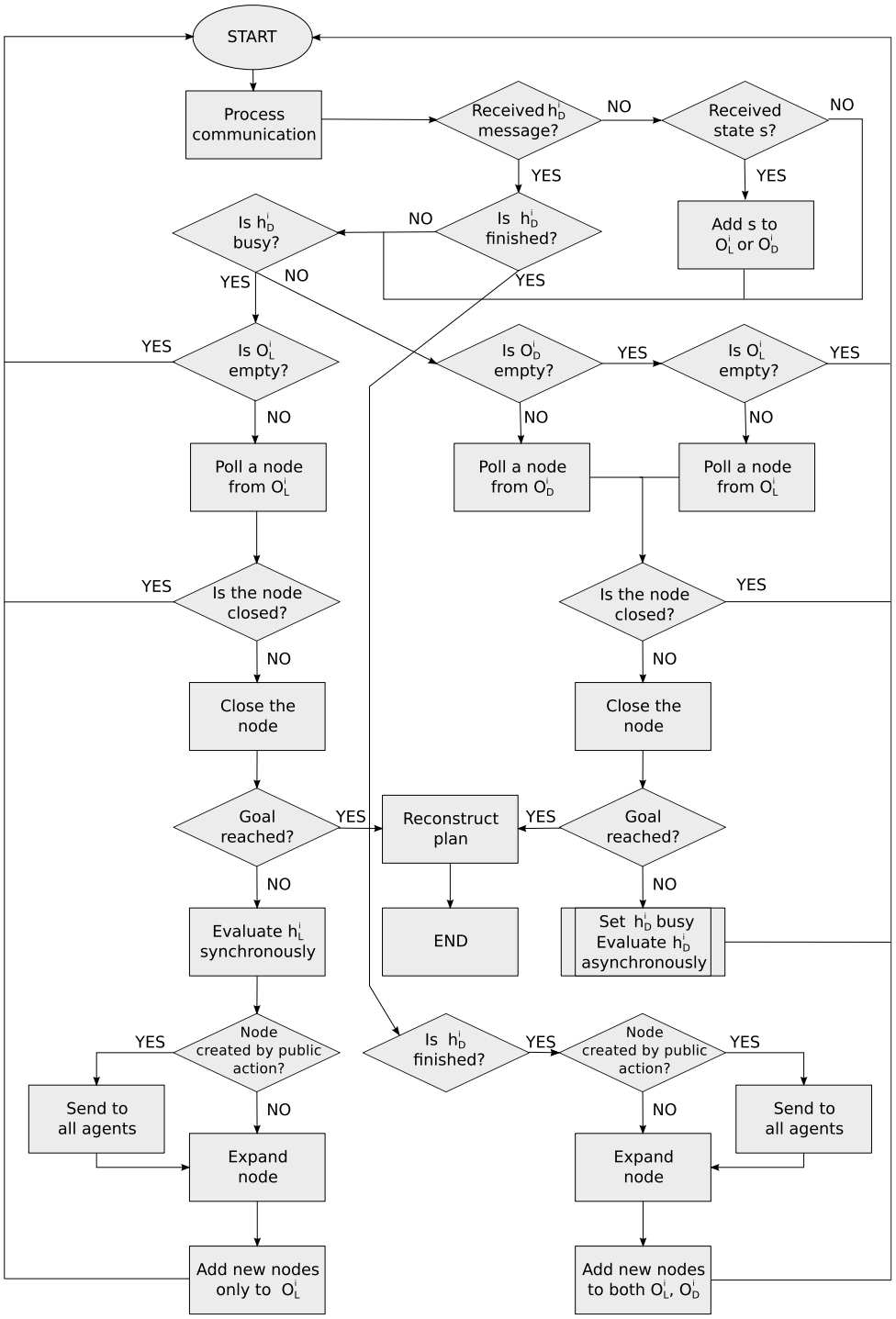

Fig. 4. Flow chart showing one iteration of the main search loop for a single agent.

The implementation of the distribution in the proposed search follows the principles of MA-BFS, i.e., broadcasts are used to inform other agents about states reached by public actions. Additionally, information as to which open list the state should be added to is included (whether it is the local or the distributed one). The overall principles of the MADLA Search will now be presented and the algorithm itself will be described in detail later on.

The MADLA Search uses two heuristics for each agent α_i, a local projected h^i_{L} and a distributed h^i_{D}. The search uses an open list for each of the heuristics $O^i_{\mathrm{L}}, O^i_{\mathrm{D}}$ for each agent α_i. The open list selection function prioritizes expansion of

states in the open list O^i_{D} respective to the distributed heuristic h^i_{D}, if the heuristic estimator of h^i_{D} is not in the process of computing an heuristic estimate (i.e. waiting for some replies from other agents).

Unlike the classical multi-heuristic search (Section 3.1.2), in MADLA Search, the extracted states are not evaluated using both heuristics. The heuristic used depends on the state of the distributed heuristic h^i_{D} estimator, represented in the algorithms as a boolean variable $\mathsf{busy}^i_{\mathsf{D}}$. The variable is set by the heuristic estimator to true if it is currently evaluating a state and false if not. If $\mathsf{busy}^i_{\mathsf{D}} = \mathsf{false}$, the state is evaluated by h^i_{D}. If $\mathsf{busy}^i_{\mathsf{D}} = \mathsf{true}$, the state is evaluated by h^i_{L}, that is the distributed heuristic is preferred if possible. This approach is most reasonable if h^i_{D} dominates h^i_{L} for most states, which is typically the case for a projected and distributed variant of the same heuristic such as FF.

As the MADLA Search is running in a single process (except for the communication) for each agent, the local heuristic search is performed only when the distributed heuristic search is waiting for the distributed heuristic estimation to finish. This principle makes sense only if finishing an estimation of h^i_{D} takes longer than that of h^i_{L} and if computation of the h^i_{D} estimator does not block the search process (incl. h^i_{L} estimations). These two requirements often hold for distributed and projected versions of one heuristic and hold for the two variants of the FF heuristic we use as well (described in detail in Section 3.2.2).

Using two separate open lists has the benefit of using two heuristics independently, but if some information between the two searches could be shared,[2] most importantly the heuristically best state found so far, it could boost the efficiency of the planner. The direction $O^i_{\mathsf{D}} \rightarrow O^i_{\mathsf{L}}$ is straightforward as most of the time, h^i_{D} dominates h^i_{L}. Thus, we can add all states evaluated by h^i_{D} also to O^i_{L} without ever skipping a better state evaluated by h^i_{L} with a worse state evaluated by h^i_{D}.

If a state s is taken from the local open list, its successor is inserted only in the local open list. If the state s is taken from the distributed open list, its successor is inserted into both local and distributed open lists. If state s' was obtained by application of a private action, it is inserted in the respective open lists of agent α_i. If the action is public, it is also inserted in open lists of all agents $\alpha_{j\neq i}$ via a message sent to them by α_i. The messages include additional information whether the state should be added to O^j_{D} (if it was evaluated by h^i_{D}) or to O^j_{L} (if it was evaluated by h^i_{L}). In the latter case, the heuristic estimate is recomputed using h^j_{L}, because the local heuristic estimate from the agent α_i may significantly differ from that of agent α_j.

The other direction $O^i_{\mathsf{L}} \rightarrow O^i_{\mathsf{D}}$ is trickier. If we added a state s evaluated by h^i_{L} to O^i_{D} the search would skip many states which are actually closer to the goal only because the local, less informative heuristic, gives a lower estimate. The way at least some information can be shared in this direction is that whenever the open list O^i_{D} is empty and the heuristic estimator h^i_{D} is not computing any heuristic, the best state s is extracted from the local open list O^i_{L} and evaluated by the distributed heuristic h^i_{D} and its successors are added to both open lists. This way, the states placed in O^i_{D} are evaluated only by h^i_{D}, but sometimes, the best state from O^i_{L} is taken to be evaluated by h^i_{D}.

The search is terminated when a goal state is reached by one of the agents, followed by a distributed plan extraction, or if there is no solution. Detection of solution nonexistence in the multi-agent setting is more complicated than in classical planning. Even if both open lists are empty, the agent cannot be sure that some other agent is not going to find a solution or broadcast some new state in the future. Therefore, the agents need to check that all open lists are empty and also that there are no pending messages to be delivered (that is no state is "in the air"). By that we close the high-level description of the MADLA Search and the MADLA Planner and continue with a detailed formal description.

3.1.4. Details of the MADLA search

To talk about the actual algorithm formally, we first need to distinguish the state $s \subseteq P$ and its representation in the search algorithm. Note, that due to privacy concerns (as explained later), each agent must have its own set of search nodes with its own representation of the actual state. When a state is to be sent to another agent, only its public projection is sent together with a tuple of ids representing the private parts of each agent. Thus, the search node of agent α_i is represented as follows.

Definition 8. A *search node* representing a state $s \in P$ is a tuple $u = \langle s^{\triangleright i}, p, a_{\mathsf{par}}, h, g, \alpha_i, \langle \delta_1, ..., \delta_n \rangle \rangle$ where $s^{\triangleright i} \subseteq P_i$ is the i-projected state, p is the parent of the search node u. Action $a_{\mathsf{par}} \in A_i \cup \{\epsilon\}$ is the action used to create s, h is the heuristic value, g the distance of s from s_I, α_i is the agent u belongs to and $\langle \delta_1, ..., \delta_n \rangle$ is the n-tuple of private unique identifiers representing the private parts $s \cap P^{\mathsf{priv}}_j$ of s for all agents $\alpha_j \in \mathcal{A}$ including α_i.

The parent p is determined as follows. If the search node u is created from a predecessor search node u' by the application of an action a, then $p = u'$ and $a_{\mathsf{par}} = a$. In the case the search node is created from a state received from another agent $\alpha_{j\neq i}$, then $p = \alpha_j$ and $a_{\mathsf{par}} = \epsilon$.

Each search node is agent-dependent, that is, it belongs to the agent α_i, which is part of the search node definition, and thus indexing a node u as u_i would be superfluous and is omitted.

We extend the action application to search nodes so that for a search node $u_k = \left\langle s^{\triangleright i}_k, p, a_k, h_k, g_k, \alpha_i, \langle \delta_1, ..., \delta_i, ..., \delta_n \rangle \right\rangle$ and an action $a_{k+1} \in A_i$ we define $u_k[a_{k+1}] = u_{k+1} = \left\langle s^{\triangleright i}_k[a_{k+1}], u_k, a_{k+1}, h_k, g_{k+1} = g_k + 1, \alpha_i, \langle \delta_1, ..., \delta'_i, ..., \delta_n \rangle \right\rangle$, where p

[2] The two searches are both run by a single agent, therefore the question of privacy is irrelevant here.

can either be a search node or an agent. As we use deferred heuristic evaluation, the heuristic estimate h_k is in fact the heuristic estimate of the state represented by the parent node. Thus, the heuristic estimate is not changed by the action application, but is updated later after the new search node is extracted from the open list and evaluated.

We define $\mathcal{U}^i$ as the set of all possible search nodes of agent α_i.

A *public search node* is a tuple $u^{\triangleright\text{pub}} = \langle s^{\triangleright\text{pub}}, p, a_{\text{par}}, h, g, \langle \delta_1, ..., \delta_n \rangle \rangle$ where $s^{\triangleright\text{pub}} \subseteq P^{\text{pub}}$ is the public projection of state s, i.e., $s^{\triangleright\text{pub}} = s \cap P^{\text{pub}}$.

For use in the algorithm (and consequent proofs) we define $u.\text{state} = s^{\triangleright i}$, $u.\text{action} = a_{\text{par}}$, $u.\text{parent} = p$ (that is $u.\text{parent} = u'$ or $u.\text{parent} = \alpha_j$), $u.h = h$, $u.g = g$, $u.\text{agent} = \alpha_i$, $u.\text{uids} = \langle \delta_1, ..., \delta_n \rangle$ and $u.\text{uid}_i = \delta_i$. A search node is agent-dependent, it is always created by an agent and never sent to another agent. When sending a state represented by a search node u, it is first converted to a public search node $u^{\triangleright\text{pub}}$ and then only the public projection $s^{\triangleright\text{pub}}$ of the state is sent together with the private unique identifiers $\langle \delta_1, ..., \delta_n \rangle$.

Each agent α_i starts with an initial search node u_I created from the initial state s_I as the following $u_I = \left\langle s_I^{\triangleright i}, \text{null}, \text{null}, h_0, 0, \alpha_i, \langle 0, ..., 0 \rangle \right\rangle$. By assigning 0 to each δ_j we represent that the private part of other agents represent the initial state (of course any value can be used as long as it is agreed upon by all agents).

Definition 9. The agent data structure is defined as

Object `Agent`(α_i)

- O^i_{D}; // distributed heuristic open list
- O^i_{L}; // local heuristic open list
- C^i; // closed list
- busy^i_{D}; // determines if the distributed heuristic is being computed
- search^i; // determines if the search should continue
- h^i_{D}; // distributed heuristic estimator
- h^i_{L}; // local heuristic estimator
- μ^i; // mapping of the sent states to the search nodes
- $\text{plans}^i = \{\left\langle \alpha_j, \pi^i_j \right\rangle, ...\}$ // currently reconstructed plans

The data structures related to each agent α_i are grouped in an object-like description in Definition 9. In each algorithm or procedure, the agent object is given as the first parameter (similarly to "this" in object-oriented languages) and the data structures are accessed directly. The structures O^i_{D}, O^i_{L} and C^i denote the open lists and the closed list of agent α_i and busy^i_{D} is a boolean variable denoting whether the distributed heuristic estimator of agent α_i is busy or not, similarly search^i denotes whether the search loop should continue or not. Moreover, h^i_{D} and h^i_{L} denote the distributed and local heuristic evaluators of agent α_i. Finally, the definition includes the state reconstruction function μ^i, and a set plans^i of all currently reconstructed plans in the form of a tuple $\left\langle \alpha_j, \pi^i_j \right\rangle$ consisting of an agent (the plan reconstruction originator) and the (partially) reconstructed plan π^i_j, initially empty, prospectively a part of the multi-agent plan $\{\pi^i\}^n_{i=1}$. The set plans^i is initially empty.

Now, let us have a detailed look at the pseudo-code and some implementation details. First, we start with Algorithm 1 which outlines the main loop of the search together with initialization. The algorithm (and all subsequent algorithms) is seen from the perspective of agent α_i, that is each agent runs a copy of Algorithm 1 in parallel with all other agents. All data structures are local to agent α_i and all search nodes contain i-projected states.

The initialization starts with the open lists containing the initial search node $u_I = \left\langle s_I^{\triangleright i}, \text{null}, \text{null}, h_0, 0, \alpha_i, \langle 0, ..., 0 \rangle \right\rangle$ consisting of the i-projected initial state $s_I^{\triangleright i}$, no parent, no action and the agent α_i. The closed list C^i is initialized as empty and the distance of the initial search node is set to $u_I.g \leftarrow 0$. Although the open lists are ordered only by the heuristic value, we need the g value for the later plan reconstruction to know the length of the plan being reconstructed upfront in order to be able to insert the appropriate number of no-op actions in the place of actions of other agents. The heuristic value of the initial node is set to some initial value $u_I.h = h_0$ which is only a mock value for later state selection.

The main loop terminates when a solution is found, or both open lists are empty (for all agents) and there are no pending messages, which means that there is no solution. Such synchronization can be straightforwardly achieved using the textbook distributed snapshot algorithm [21], but we leave it out of the pseudo-code for simplicity.

The distributed snapshot algorithm is a general technique to determine the state of a distributed system and roughly works as follows. The agent performing the snapshot saves its own local state and sends a snapshot request message with a snapshot token to all other agents. When any agent receives this particular snapshot token for the first time, it sends the initiator agent its own saved state (e.g., all open lists are empty) and attaches the snapshot token to all subsequent messages. When an agent that has already received the snapshot token receives a message that does not have the snapshot token, the agent needs to inform the snapshot initiator about the message (e.g., if it was a state message, its open lists are no longer empty). This message was sent before the snapshot started and needs to be included in the snapshot. This means

Algorithm 1: MADLA Search for agent α_i and multi-agent planning problem $\mathcal{M}$. The plan is not returned directly by the algorithm as the reconstruction is asynchronous and thus the plan is returned by the Procedure 3.

```
1  Algorithm MADLA-Search(α_i, Π_i)
2      O^i_D ← {u_I}; O^i_L ← {u_I};
3      C^i ← ∅; u_I.g ← 0; u_I.h ← h_0;
4      busy^i_D ← false;
5      search^i ← true; // search begins
6      while search^i do
7          processComm(α_i); // see Algorithm 3
8          if busy^i_D = false then
9              if O^i_D ≠ ∅ then
10                 u ← arg min_{u∈O^i_D} u.h;
11             if O^i_L ≠ ∅ then
12                 u ← arg min_{u∈O^i_L} u.h;
13             else
14                 goto Line 6;
15             processNode(α_i, u, true);
16         // local search loop
17         while (O^i_D = ∅ or busy^i_D = true) and O^i_L ≠ ∅ do
18             u ← arg min_{u∈O^i_L} u.h;
19             processNode(α_i, u, false);
20             //process communication also in the local search loop
21             processComm(α_i);
22 Procedure processNode(α_i, u, d)
23     if u ∉ C^i then
24         C^i ← C^i ∪ {u}; // close search node
25         if G ⊆ u.state then
26             search^i ← false; //end search
27             //inform agents
28             send M_PLANFOUND to all α_j ∈ A;
29             //(asynchronously) reconstruct and return the plan
30             reconstructPlan(α_i, u, u.g, α_i); // see Algorithm 4
31         expand(α_i, u, d); // see Algorithm 2
```

that all messages need to have the possibility to include the snapshot token, which we have also not included in the formal description in order to keep it simpler.

The main loop follows the classical heuristic search with deferred evaluation scheme with some modifications. From line 8 to line 15, an open list is determined based on the state of the distributed heuristic and the node u with minimal heuristic value $u.h$ is extracted from the selected open list (if both open lists are empty, the loop continues with checking the messages). Node u is processed as in classical search (added to open list, checked for goal), but if u contains a goal state, distributed plan reconstruction is initiated (see Algorithm 4). If u does not contain a goal state, u is expanded (see Algorithm 2). Since we use the deferred heuristic evaluation, the heuristic is evaluated before the actual expansion.

The expansion procedure is detailed in Algorithm 2. The node u is evaluated using either the local projected heuristic h^i_{L}, which proceeds as usual, or using the distributed global heuristic h^i_{D}. The distributed heuristic evaluation is represented as a function call for simplicity. In reality, the procedure evaluating the heuristic function is asynchronous. This means that a callback is passed to the procedure and the `expand`(α_i, u, d) procedure exits. Only when the heuristic evaluation is finished, the rest of the procedure continues. This means the following steps are performed either directly after the local heuristic evaluation, or in the callback of the distributed heuristic evaluation.

If node u was expanded by a public action, a message M_{STATE} is sent to all other agents, containing the public projection $s^{\triangleright\mathsf{pub}}$ of state s together with the private unique identifiers representing the private parts of all agents, its $u.g$, its heuristic value $u.h$ and a parameter d determining whether it was evaluated using the local or distributed heuristic in order to determine to which open list it should be added.

The states sent to other agents are stored by a mapping function $\mu^i : 2^{P^{\mathsf{pub}}} \times \mathbb{N} \to \mathcal{U}^i$, which assigns to a public projection of a state $s^{\triangleright\mathsf{pub}}$ and a private unique identifier δ_i the respective search node. The function μ^i is used both to be able to reconstruct the path later and to reconstruct the private part of a received state. The storing does not cause significant memory overheads as the states are stored in the closed list anyway.

Next, the node u is expanded using all applicable actions from A_i, the new nodes u' are created according to the Definition 8 (so that $u'.\mathsf{state} = s^{\triangleright i}[a]$, $u'.\mathsf{parent} \leftarrow u$, $u'.\mathsf{action} \leftarrow a$, $u.h \leftarrow u'.h$, $u.g \leftarrow u'.g + 1$, $u'.\mathsf{agent} \leftarrow \alpha_i$, $u'.\mathsf{uid}_j \leftarrow u.\mathsf{uid}_j$ for all $j \neq i$ and $u'.\mathsf{uid}_i$ is assigned a new private unique identifier) and the new search nodes are added to the open list(s) based on which heuristic was used for evaluation.

Algorithm 2: Procedure expand(α_i, u, d).

```
1  Procedure expand(α_i, u, d)
2      if d = false then
3          u.h ← h^i_L(u.state) // local
4      else
5          u.h ← h^i_D(u.state); // distributed, asynchronous
6          set busy^i_D ← true when h^i_D starts evaluation
7          set busy^i_D ← false when h^i_D finishes evaluation (asynchronously)
8      if u.action ∈ A^pub_i then
9          send M_STATE = ⟨u^▷pub.state, u^▷pub.uids, u.h, d⟩;
10         μ(u^▷pub.state, u^▷pub.uid_i) ← u;
11     E ← ∅
12     for all a ∈ A_i s.t. pre(a) ⊆ u.state do
13         u' ← u[a];
14         u'.g ← u.g + 1;
15         u'.h ← u.h; // deferred heuristic
16         E ← E ∪ {u'};
17     O^i_L ← O^i_L ∪ {u' | u' ∈ E ∧ u' ∉ C^i};
18     if d = true then
19         O^i_D ← O^i_D ∪ {u' | u' ∈ E ∧ u' ∉ C^i};
```

Algorithm 3: processComm(α_i).

```
1  Procedure processComm(α_i)
2      for each message M in message queue, received from α_j do
3          switch M do
4              case M_STATE = ⟨s^▷pub, ⟨δ_1, ..., δ_n⟩, h, g, d⟩
5                  u_sent ← μ^i(s^▷pub, δ_i);
6                  s^▷i ← u_sent.state;
7                  u ← ⟨s^▷i, α_j, ε, h, g, α_i, ⟨δ_1, ..., δ_n⟩⟩;
8                  if d = true then
9                      O^i_D ← O^i_D ∪ {u};
10                 else
11                     u.h ← h^i_L(u.state) // local heuristic recomputed
12                 O^i_L ← O^i_L ∪ {u};
13             case M_PLANFOUND
14                 search^i ← false;
15                 plans^i ← plans^i ∪ {⟨α_j, ∅⟩};
16             case M_RECONSTRUCT = ⟨s^▷pub, δ_i, t, α_k⟩
17                 //continue reconstruction of the best plan so far
18                 u ← μ^i(s^▷pub, δ_i); //reconstruct search node for the sent state s^▷pub
19                 reconstructPlan(α_i, u, t, α_k);
20             case M_TERMINATE = ⟨α_k⟩
21                 if received M_TERMINATE = ⟨α_j⟩ for all α_j s.t. ⟨α_j, π^i_j⟩ ∈ plans^i then
22                     select π^i_min of minimal length from plans^i;
23                     report solution π^i_min and terminate;
24             case else
25                 forward M to h^i_D;
```

In addition to the main loop in Algorithm 1, there is an inner loop on lines 17–21, which is performed when the distributed open list is empty or busy with a heuristic evaluation (this happens when the distributed heuristic is being asynchronously evaluated). This inner loop is again the classical heuristic search loop with deferred evaluation, this time taking into account only the local open list and local heuristic. This inner local search loop also needs to process the communication, that is to send and to receive messages (otherwise the distributed heuristic computation could not be finished).

The part of the communication taking care of message receiving is shown in Algorithm 3 (message sending is trivial). There are four types of messages sent among the agents for the distributed search (more messages are sent for the distributed heuristic computation). A state message $M_{\text{STATE}} = \langle s^{\triangleright\text{pub}}, \langle \delta_1, \ldots, \delta_n \rangle, h, g, d \rangle$ contains a public projection $s^{\triangleright\text{pub}}$ of a state s, the private unique identifiers, the state's heuristic value h, g and the parameter d. When received, the i-projection

$s^{\triangleright i}$ is reconstructed from $s^{\triangleright \mathsf{pub}}$ and the private unique identifier δ_i using the function μ^i, which encodes the node and respective i-projected state from which the private part should be copied. Thus the previously anonymized private part of agent i is restored, and a new search node u is created, its parent is set to the sending agent α_j. The heuristic is re-computed to reflect the local problem of the receiving agent, although not recomputing the heuristic does not have significant effect in most domains. As the heuristics are not admissible, taking the maximum as in MAD-A* is not reasonable. Finally, the node u is added to the open list(s) selected based on the received parameter d.

The "plan found" message $M_{\mathsf{PLANFOUND}}$ (with no parameters, except for the implicit sender α_j) is sent by an agent α_j when it reaches a goal state to inform other agents that it has found a plan and it is starting the plan reconstruction process. When received by the agent α_i, it knows that it can exit the search loop (the communication still has to be processed). A new tuple $\left\langle \alpha_j, \pi_j^i \right\rangle$ is added to plans^i as the reconstruction of plan initiated by α_j will take place. Note that before receiving the $M_{\mathsf{PLANFOUND}}$ message, the agent α_i could have already started the plan reconstruction itself, thus multiple plans can be reconstructed in parallel.

The plan reconstruction message $M_{\mathsf{RECONSTRUCT}} = \left\langle s^{\triangleright \mathsf{pub}}, \delta_i, t, \alpha_k \right\rangle$ contains the state s (represented as $s^{\triangleright \mathsf{pub}}$ and the private unique identifier δ_i of the receiving agent), from which the reconstruction should continue, a discrete time-point t, initially set to $u.g$ of the last search node u (that is, the search node which satisfied the goal condition) and the agent α_k which started the plan reconstruction. The identifier of the agent α_k is used to determine which partial plan $\langle \alpha_k, \pi_k^i \rangle \in \mathsf{plans}^i$ the reconstruction message belongs to. To reconstruct the plan, first the respective search node needs to be retrieved, which is the search node returned by $\mu^i(s^{\triangleright \mathsf{pub}}, \delta_i)$ based on the public part $s^{\triangleright \mathsf{pub}}$ and private part δ_i of the state $s^{\triangleright i}$, which was sent to α_j on line 9 in Algorithm 2. Next, the plan reconstruction procedure is invoked (see Algorithm 4).

Finally, the termination message $M_{\mathsf{TERMINATE}} = \langle \alpha_k \rangle$ announces that the reconstruction of the plan initiated by the agent α_k has been finished by the sender agent α_j. Only when a message $M_{\mathsf{TERMINATE}} = \langle \alpha_k \rangle$ is received for all α_k such that $M_{\mathsf{PLANFOUND}}$ was received from α_k before and thus there is $\langle \alpha_k, \pi_k^i \rangle \in \mathsf{plans}^i$, the plan reconstruction of all plans is finished and the shortest plan can be selected (ties are broken based on the ordering of the agent indexes). Of course, it may happen that the agent α_i was not involved in the plan reconstruction at all and thus its plan π_i^i consists of l no-op actions ϵ, where l is the length of the shortest plan.

Note that messages for the distributed heuristic h_{D}^i are also handled in Algorithm 3 and are forwarded to the heuristic estimator.

Algorithm 4: reconstructPlan(α_i, u, t, α_k).

```
1  Procedure reconstructPlan(α_i, u, t, α_k)
2      π_k^i ← ⟨α_k, π_k^i⟩ ∈ plans^i
3      π_k^i[t] ← u.action;
4      π_k^i[t'] ← ϵ for all t' > t s.t. π_k^i[t'] is empty;
5      t ← t − 1;
6      if u = u_I then
7          send M_TERMINATE = ⟨α_k⟩ to all α_j ∈ A;
8      if u.action = ϵ then
9          //node received from u.parent
10         send M_RECONSTRUCT = ⟨u^▷pub.state, u.uid_j, t, α_k⟩ to α_j ← u.parent
11     else
12         reconstructPlan(α_i, u.parent, t, α_k);
```

The distributed plan reconstruction procedure is shown in Algorithm 4. Recall that the plan is stored in a list $\langle \alpha_k, \pi_k^i \rangle$ of actions for each agent, together forming a multi-agent plan $\{\pi_k^i\}_{i=1}^n$ for each such k. The reconstruction process is started by an agent α_k in a search node u s.t. $G \subseteq u.\mathsf{state}$ (see Algorithm 1 line 25) and with $t = u.g$ which is the distance from u_I to u. Every time an action is added to the position t of π_k^i (line 3), t is decreased by 1, which represents backward reconstruction of the plan. All positions between the last added action and t are padded with the no-op action ϵ (line 4).

If the initial node is reached (line 6), the terminate message is sent to all other agents. Otherwise, the parent $u.\mathsf{parent}$ of node u is retrieved, which is either the predecessor node, or an agent $\alpha_{j \neq i}$. If $u.\mathsf{parent} = \alpha_{j \neq i}$ a reconstruct message is sent to α_j prompting it to continue with the reconstruction from a state s represented by the search node u.

As multiple agents may start plan reconstruction independently, any agent that found a solution reports to all other agents by sending a $M_{\mathsf{PLANFOUND}}$ message. This way, each agent α_i receiving the message terminates the search and adds $\left\langle \alpha_j, \pi_j^i \right\rangle$ to the plans^i set (see Algorithm 3 line 15). This means that a plan reconstruction was started by agent α_j. When the plan reconstruction started by the agent α_k is finished by an agent α_j by reaching the initial search node u_I (see Algorithm 4 line 6) the $M_{\mathsf{TERMINATE}} = \langle \alpha_k \rangle$ message is sent to all agents. When $M_{\mathsf{TERMINATE}} = \langle \alpha_k \rangle$ sent by α_j is received by α_i, the agent α_i knows that the reconstruction of $\langle \alpha_k, \pi_k^i \rangle$ has been finished. When received for all such $\langle \alpha_k, \pi_k^i \rangle \in \mathsf{plans}^i$, the shortest plan from plans^i is chosen (see Algorithm 3 line 22) and reported as the solution for agent α_i (see Algorithm 3 line 23). To ensure that each agent α_i chooses the corresponding plan π_k^i together forming the distributed multi-agent plan

$\{\pi_k^i\}_{i=1}^n$, the ties are broken based on the ordering of the indexes k of the agents which started the reconstruction of the respective plan.

Description of details of the distributed plan reconstruction concludes the proposed MADLA Search.

3.2. Heuristics

The key principle behind the MADLA Planner is a favorable combination of both local (Definition 4) and distributed (Definition 3) heuristics. A heuristic pair complying with the search used in the MADLA Planner combines a light single-agent and a heavy distributed multi-agent heuristic which in this article are both forms of the classical Fast-Forward (FF) heuristic [7].

The local heuristic is the classical FF applied to each agent's projected problem $\Pi_i^{\triangleright}$, whereas the distributed variant is a novel privacy-preserving modification of previously published distributed FF [8,6]. Here, we describe both variants and show what properties important for the MADLA Search are satisfied and how privacy is treated.

3.2.1. Local heuristic

One of the most successful and most studied heuristics for satisficing planning is the FF heuristic. The FF heuristic belongs to the delete relaxation heuristic family. The idea behind delete relaxation heuristics is to simplify the problem by ignoring negative effects of actions. In the STRIPS formalism, this means that an action $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)\rangle$ is transformed to a relaxed form $a^+ = \langle \mathsf{pre}(a), \mathsf{add}(a), \emptyset\rangle$. A set of relaxed actions $A^+ = \{a^+ | a \in A)$ is used in the definition of a classical relaxed planning problem $\Pi^+ = \left\langle P, A^+, s_I, G\right\rangle$ respective to the original planning problem $\Pi = \langle P, A, s_I, G\rangle$. By relaxation, the whole problem becomes additive, meaning that whenever a fact is added it is never deleted again and as both preconditions and goal are positive in STRIPS, any action that becomes applicable remains applicable in all subsequent relaxed states. The solution of a relaxed problem Π^+ is a relaxed plan π^+. Notice that thanks to the additivity of the relaxed problem, the relaxed plan can be represented as an unordered set of relaxed actions. From such set, the relaxed plan can be reconstructed by iteratively applying all actions as soon as they are applicable. This way, all facts achievable by any permutation of the actions in π^+ are achieved. This also means that in an optimal relaxed plan, no action is used more than once.

An optimal relaxed heuristic h^+ is defined as the length of an optimal relaxed plan π^+. In contrast to STRIPS planning, which is PSPACE-complete, finding an optimal relaxed plan π^+ is NP-Complete [22], which is still impractical as a heuristic. In order to lower the complexity even more, approximations of h^+ are used in classical planning. The most commonly used approximation in satisficing planning is to use the length of a sub-optimal relaxed plan (RP) instead of an optimal one. Finding a sub-optimal RP can have as low as polynomial complexity and therefore can be fast enough in practice.

The main idea behind the FF heuristic is to find a sub-optimal relaxed plan by analyzing which facts are successively reachable by applied relaxed actions (reachability analysis). From this analysis the relaxed plan is determined in a backward fashion. The principle is based on a notion of *supporter action a of fact p* which is an action a s.t. $p \in \mathsf{add}(a)$. Let $\Pi^+ = \left\langle P, A^+, s_I, G\right\rangle$ be a relaxed planning problem, then the principle of relaxed plan extraction is the following:

1. Initialize a set of unsupported facts U to contain all goal facts and a set of supported facts S to contain all initial state facts: $U \leftarrow G, S \leftarrow s_I$.
2. Move an unsupported fact p from U to a set of supported facts S and determine its supporter a.
3. Mark all preconditions of a as unsupported if not supported already: $U \leftarrow U \cup (\mathsf{pre}(a) \setminus S)$.
4. Loop 1–3 until all facts in U are supported: until $U \setminus S = \emptyset$.

There are many ways of implementing this high-level scheme (which differ mainly in the way the supporters are chosen) and many methods to perform the reachability analysis. In our work, we have used one of the most prevalent, based on an Exploration Queue algorithm as implemented for example in the Fast-Downward planning system [10]. The details of the reachability analysis algorithm are not relevant for this work as any standard technique can be used, therefore we will remain focused on the relaxed plan extraction part.

In the MADLA Planner, the FF heuristic is used as the local heuristic h^i_{L}. As such it is computed by each agent α_i on its respective i-projected problem $\Pi_i^{\triangleright}$. The i-projected (Definition 4) FF heuristic is computed on a relaxed i-projected problem, formally

$$\Pi_i^{\triangleright +} = \left\langle P_i, A_i^{\triangleright +} = \{a^+ | a \in A_i^{\triangleright}\}, s_I^{\triangleright i} = s_I \cap P_i, G\right\rangle. \tag{1}$$

Although projected heuristics have been used previously in [3], privacy preservation of the projected heuristics was not explicitly discussed. Indeed, it is trivial to see that when using only a projected heuristic the privacy is not compromised in any way. Each agent computes the heuristic estimate separately, using only the information present in the projected problem. How the heuristic is used depends on the search scheme, but even if the heuristic estimate is shared among agents, other agents have no means to decode and exploit the heuristic value (it can possibly be computed using an algorithm completely different to their own).

3.2.2. Distributed heuristic

In addition to a local heuristic, the MADLA Planner uses a distributed multi-agent heuristic (Definition 3). In contrast to the computation of the local heuristic by a single agent, a coordinated computation by multiple agents is necessary for evaluation of the distributed heuristic estimates. Computation of such a heuristic incurs a communication overhead, but the overhead is typically outweighed by heuristic estimates significantly better than the estimates of a projected heuristic.

Although the benefits of using a global heuristic estimate computed in a distributed way are clear, computing such heuristic efficiently whilst preserving privacy is a major challenge. So far, the distribution has been tackled by a couple of approaches. In [23], the authors present a plan-space multi-agent planner FMAP, which is guided by a heuristic similar to the high-level principle of the FF heuristic described in Section 3.2.1, but computed on the non-relaxed problem using (distributed) Domain Transition Graphs (DTGs) [10] for the reachability analysis. The benefit of DTGs is that they are not state-dependent, allowing to cache the results and thus lower the communication load.

The FF heuristic was first distributed in [8], where the aim was to compute provably the same heuristic estimates as the centralized FF would give on the global problem. The approach was based on synchronized reachability analysis computed by all agents, which was shown to be rather impractical. To improve the computation speed, a lazy alternative was used, where the relaxed plan extraction process was distributed instead of the reachability analysis, which was kept local and computed only when necessary (lazily).

In [6], a general scheme for computing distributed relaxation heuristics was proposed. The FF heuristic is again present in two variants, one based on the distribution of the reachability analysis, and the other one based on the distribution of the relaxed plan extraction.

In the following section, we present an improved distributed FF heuristic algorithm as a part of the MADLA Planner. The novel distributed FF heuristic is based on the idea of distributing the relaxed plan extraction process, while performing the reachability analysis lazily, that is only when requested. The novel aspect of the algorithm is the use of a set-additive principle [24] to overcome over-counting and also an adaptation towards more privacy.

3.2.3. Privacy-preserving set-additive FF

The two leading ideas of the novel distribution of the FF heuristic (ppsaFF) are the use of the lazy approach and the idea of the set-additive heuristic [24]. The lazy principle states that the distributed heuristic first starts as a projected FF, computing the reachability analysis and relaxed plan as in Section 3.2.1. Such relaxed plan π_i^+ computed on an i-projected relaxed problem $\Pi_i^{\triangleright+}$ may contain projected actions, which may have private preconditions. The private preconditions are not satisfied in π_i^+ as α_i is not aware of them. Let $a^{\triangleright+}$ be such a projected action and let $a^+ \in A_j^+$ for some $j \neq i$. In that case, α_i requests α_j to provide a relaxed plan that satisfies the private part of precondition of a^+ (the public part is already satisfied in the projected RP). Here the set-additive principle comes into play. In the original lazy FF variant, agent α_j would report just the cost of achieving private preconditions of a, which leads to significant over-counting. In the new variant, the agent α_j provides the actual relaxed plan π_a^+, which satisfies the private precondition of a^+, that is $\mathsf{pre}(a^+) \cap P_j^{\mathsf{priv}}$, which can be merged with the original relaxed plan $\pi_i^+ \leftarrow \pi_i^+ \cup \pi_a^+$ as both relaxed plans are represented as sets of actions. This request-reply protocol is performed for all projected actions in π_i^+, even those newly received from α_j and even for public actions of α_i itself received in π_a^+ (in which case the request can be handled by an internal call).

At this moment we have possibly violated privacy by sharing a relaxed plan π_a^+ which may contain private actions of agent α_j, even though we share only a unique identifier of that private action and no preconditions or effects. In order to treat the privacy correctly, the algorithm has to be further modified.

Instead of sending back the relaxed plan π_a^+, agent α_j builds a local relaxed reply plan $\pi_{j,i}^{\mathsf{RE}+}$ for the requesting agent α_i (we always write the index of the agent owning the variable/data structure first and the other agent it relates to second), which is updated for every requested action a^+ as $\pi_{j,i}^{\mathsf{RE}+} = \pi_{j,i}^{\mathsf{RE}+} \cup \pi_a^+$. To maintain privacy, α_j keeps the private part of the plan locally, that is $\pi_{j,i}^{\mathsf{priv}+} = \pi_{j,i}^{\mathsf{RE}+} \cap A_j^{\mathsf{priv}+}$ and sends only its public part $\pi_{j,i}^{\mathsf{pub}+} = \pi_{j,i}^{\mathsf{RE}+} \cap A_j^{\mathsf{pub}+}$ together with the length of the private part $l_{j,i} = |\pi_{j,i}^{\mathsf{priv}+}|$. The relaxed plan $\pi_{j,i}^{\mathsf{RE}+}$ is maintained by α_j throughout the whole computation of the heuristic estimate for the agent α_i and a single state s, so that each action of α_j is counted at most once. This is made easier by the fact that each agent is computing the distributed heuristic for at most one state, thus the agent has to keep track of at most $|\mathcal{A}| - 1$ relaxed plans of other agents. Meanwhile, agent α_i builds a single relaxed plan π_i^+ containing actions from $A_i^{\triangleright+}$ and a value $l_{i,j}^{\mathsf{priv}}$ for each agent $\alpha_{j\neq i}$ representing the length of the private part of the relaxed plan of agent α_j. After all projected actions in π_i^+ are processed (that is all replies received), the resulting heuristic value is computed as:

$$h_{\mathrm{ppsaFF}}(s^{\triangleright i}) = |\pi_i^+| + \sum_{j \in 1..n \wedge j \neq i} l_{i,j}^{\mathsf{priv}} \tag{2}$$

For clarity, the algorithm is transcribed into pseudo-code in Algorithm 5. The algorithm is split into three procedures, each has the agent which is performing the procedure (i.e., is written from its perspective) as its first parameter. The procedures work as follows.

Algorithm 5: Procedures for computing the Privacy-Preserving Set-Additive Lazy FF.

1 **Procedure** `computeDistributedFF` $(\alpha_i, s^{\triangleright i}, \langle \delta_1, ..., \delta_n \rangle, \Pi_i^{\triangleright +} = \left\langle P_i, A_i^{\triangleright +}, s_I^{\triangleright i}, G \right\rangle)$
2 $\quad l_{i,j}^{\text{priv}} \leftarrow 0$ for each $j \neq i$;
3 $\quad \pi_i^+ \leftarrow$ `computeRelaxedPlan` $(s^{\triangleright i}, G, A_i^{\triangleright +})$;
4 $\quad$ **if** $\pi_i^+ =$ fail **then**
5 $\quad\quad$ **return** ∞;
6 $\quad A_{\text{DONE}} \leftarrow \pi_i^+ \cap A_i^{\text{pub}+}$; $A_{\text{WAITING}} \leftarrow \emptyset$;
7 $\quad$ **while** $\exists a^{\triangleright +} \in \pi_i^+ \setminus A_{\text{DONE}}$ *s.t.* $a^+ \in A_j^+ \wedge j \neq i$ **or** $A_{\text{WAITING}} \neq \emptyset$ **do**
8 $\quad\quad A_{\text{WAITING}} \leftarrow A_{\text{WAITING}} \cup \{a^{\triangleright +}\}$;
9 $\quad\quad A_{\text{DONE}} \leftarrow A_{\text{DONE}} \cup \{a^{\triangleright +}\}$;
10 $\quad\quad$ **send** $M_{\text{REQUEST}} = \langle \alpha_i, s^{\triangleright \text{pub}}, \delta_j, a^{\triangleright +} \rangle$ **to** α_j;
11 $\quad$ **return** $|\pi_i^+| + \sum_{j \in 1..n \wedge j \neq i} l_{i,j}^{\text{priv}}$;
12 **Procedure** `processRequest` $(\alpha_j, M_{\text{REQUEST}} = \langle \alpha_i, s^{\triangleright \text{pub}}, \delta_j, a^{\triangleright +} \rangle)$
13 $\quad$ // α_j can use $a^+ \in A_j^{\text{pub}+}$ instead of $a^{\triangleright +}$ for $\text{lbl}(a^+) = \text{lbl}(a^{\triangleright +})$
14 $\quad s^{\triangleright j} \leftarrow \mu^j(s^{\triangleright \text{pub}}, \delta_j)$.state; // reconstruct the state
15 $\quad \pi_a^+ \leftarrow$ `computeRelaxedPlan` $(s^{\triangleright j}, \text{pre}(a^+) \cap P_j^{\text{priv}}, A_j^{\triangleright +})$;
16 $\quad$ **if** M_{REQUEST} *is a first request for* $s^{\triangleright j}$ **then**
17 $\quad\quad \pi_{j,i}^{\text{RE}+} \leftarrow \emptyset$;
18 $\quad \pi_{j,i}^{\text{RE}+} \leftarrow \pi_{j,i}^{\text{RE}+} \cup \pi_a^+$;
19 $\quad l_{j,i} \leftarrow |\pi_{j,i}^{\text{RE}+} \cap A_j^{\text{priv}+}|$;
20 $\quad$ // all except for the private part of π_a^+ is sent
21 $\quad \pi_{j,i}^{\text{pub}+} \leftarrow \pi_a^+ \cap (A_j^{\triangleright +} \setminus A_j^{\text{priv}+})$;
22 $\quad$ **send** $M_{\text{REPLY}} = \left\langle \pi_{j,i}^{\text{pub}+}, l_{j,i}, a^{\triangleright +} \right\rangle$ **to** α_i;
23 **Procedure** `processReply` $(\alpha_i, M_{\text{REPLY}} = \left\langle \pi_{j,i}^{\text{pub}+}, l_{j,i}, a^{\triangleright +} \right\rangle)$
24 $\quad \pi_i^+ \leftarrow \pi_i^+ \cup \pi_{j,i}^{\text{pub}+}$;
25 $\quad l_{i,j}^{\text{priv}} \leftarrow l_{j,i}$;
26 $\quad A_{\text{WAITING}} \leftarrow A_{\text{WAITING}} \setminus \{a^{\triangleright +}\}$;

The main procedure `computeDistributedFF` $(\alpha_i, s^{\triangleright i}, \langle \delta_1, ..., \delta_n \rangle, \Pi_i^{\triangleright +})$ is called by the search to evaluate a state $s^{\triangleright i}$ by agent α_i (shown as a call to h_{D}^i at line 5 in Algorithm 2). After the initialization steps, the relaxed plan π_i^+ is computed such that π_i^+ achieves the goal G in the projected relaxed problem (Equation (1)), using actions from $A_i^{\triangleright +}$. Then, while there is some projected action $a^{\triangleright +}$ in π_i^+ which has not been processed yet (i.e., is not in A_{DONE}), process it by sending a request $M_{\text{REQUEST}} = \langle \alpha_i, s^{\triangleright \text{pub}}, \delta_j, a^{\triangleright +} \rangle$ to the agent α_j, the owner of a^+ (the owner of an action is known by definition). The loop also does not terminate while there are some actions in A_{WAITING}, which means a reply has not been received for them (a request is not sent if there is no unprocessed action in π_i^+, this condition has been omitted for simplicity). When all projected actions and replies are processed, the heuristic value (Equation (2)) is returned. Note that the actual implementation differs from the pseudocode in that the loop is implemented as an asynchronous event-based message processing (i.e., waiting for replies from other agents is non-blocking).

When the agent α_j receives a request from the agent α_i to evaluate private preconditions of some action $a^+ \in A_j^{\text{pub}+}$, the procedure `processRequest` $(\alpha_j, M_{\text{REQUEST}} = \langle \alpha_i, s^{\triangleright \text{pub}}, \delta_j, a^{\triangleright +} \rangle)$ is called. Agent α_j first reconstructs the state $s^{\triangleright j}$ and then computes a relaxed plan π_a^+, which solves the j-projected relaxed problem of α_j starting in $s^{\triangleright j}$ with goal being the private preconditions of a^+, formally $\text{pre}(a^+) \cap P_j^{\text{priv}}$. The computed relaxed plan (RP) is then used to update the reply RP $\pi_{j,i}^{\text{RE}+}$, whose private length $l_{j,i} \leftarrow |\pi_{j,i}^{\text{RE}+} \cap A_j^{\text{priv}+}|$ is sent back to α_i together with the public part of $\pi_{j,i}^{\text{RE}+}$. The public part of $\pi_{j,i}^{\text{RE}+}$ consists of public actions of α_j and all projected actions of $\alpha_{k \neq j}$ including projected actions of α_i, which are in π_a^+. Note that the reply RP $\pi_{j,i}^{\text{RE}+}$ is kept for each agent α_i over all requests regarding one particular state $s^{\triangleright j}$. When the agent α_j receives a request for another state s' from the agent α_i, the reply RP $\pi_{j,i}^{\text{RE}+}$ is initialized to $\pi_{j,i}^{\text{RE}+} = \emptyset$ first. This works thanks to the fact that each agent α_k computes the heuristic estimate for at most one state at any moment.

When a reply message $M_{\text{REPLY}} = \left\langle \pi_{j,i}^{\text{pub}+}, l_{j,i}, a^{\triangleright +} \right\rangle$ is received from the agent α_j for the request $M_{\text{REQUEST}} = \langle \alpha_i, s^{\triangleright \text{pub}}, \delta_j, a^{\triangleright +} \rangle$ by agent α_i, the procedure `processReply` $(\alpha_i, M_{\text{REPLY}} = \left\langle \pi_{j,i}^{\text{pub}+}, l_{j,i}, a^{\triangleright +} \right\rangle)$ is called. The relaxed plan π_i^+ is updated with the received public part and the estimate of the private part for agent α_j is replaced with the new received value. Action $a^{\triangleright +}$ is removed from A_{WAITING} as its processing has been finished.

Note that the search and heuristic estimation is all running in a single thread and all messages are managed through a message queue and the calls on the line 6 and line 21 in Algorithm 1. This means that the procedure `computeDis-`

`tributedFF()` is called once (line 5 of Algorithm 2) and the loop on line 7 is in fact managed through callbacks and thus the call to the heuristic computation is asynchronous. Meanwhile, the `processRequest()` and `processReply()` procedures are called in response to the messages received on line 25 of Algorithm 3, sequentially, one at a time.

We illustrate the process on the running example. Let us start with a situation, where the truck is computing the heuristic estimate for the initial state. First, the projected relaxed plan π_t^+ is computed

$$\pi_t^+ = \{\textbf{unload-a-C}\}.$$

In the t-projected problem, the unload action of plane has no preconditions and it fulfills the goal. Next, the truck sends a request to the plane, which computes a relaxed plan from the initial state to the private precondition of **unload-a-C**, which is {package-in-a, plane-at-C}. The plane comes up with the following relaxed plan

$$\pi_{a,t}^{RE+} = \{\textbf{unload-t-B}, \textbf{load-a-B}, \text{move-a-B-C}\}$$

and sends back a reply containing only the public actions and the number of private actions which is 1. The truck accordingly updates its relaxed plan to

$$\pi_t^+ = \{\textbf{unload-t-B}, \textbf{load-a-B}, \textbf{unload-a-C}\}$$

and proceeds by sending requests for the newly added actions. The request for **load-a-B** does not have to be sent as it was received from the plane (this optimization is ignored in the algorithm for simplicity). The request for **unload-t-B** has to be sent, but as the receiver is the truck itself, it can be forwarded via a direct call. Also the private actions of the truck are directly included in the relaxed plan π_t^+ by the local computation. The resulting relaxed plan is

$$\pi_t^+ = \{\text{load-t-A}, \text{move-t-A-B}, \textbf{unload-t-B}, \textbf{load-a-B}, \textbf{unload-a-C}\}$$

with the additional number of private actions of the plane being 1, thus the complete heuristic estimate is $h_{\text{ppsaFF}}(s_I) = 5 + 1 = 6$.

This wraps up the description of the novel distributed variant of the FF heuristic. We did not pay attention to the actual process of finding the relaxed plan, as the distribution is general so that any approach can be used and the reachability analysis is kept local (computed on the projected relaxed problem).

3.2.4. Termination of the privacy-preserving set-additive FF

Now we formally show that the Privacy-Preserving Set-Additive FF heuristic always terminates.

Theorem 10. *Assuming liveness of the communication, the heuristic* ppsaFF *shown in Algorithm 5 always terminates.*

Proof. Let s be the state the ppsaFF heuristic is computed for by agent α_i. If the goal is not reachable from s in $\Pi_i^{\triangleright+}$ then the computation of `computeRelaxedPlan(`$s, G, A_i^{\triangleright+}$`)` will fail and ∞ is returned. Otherwise, π_i^+ contains a finite number of (relaxed) actions. For each action $a^{\triangleright+} \in \pi_i^+ \setminus A_{\text{DONE}}$ such that $a^+ \in A_j^+ \wedge j \neq i$ a request is sent to the action owner α_j. The computation of the reply, that is `computeRelaxedPlan(`$s, \text{pre}(a^+) \cap P_j^{\text{priv}}, A_j^{\triangleright+}$`)` always finishes, with either a finite non-empty or an empty plan π_a^+. When the reply is received, the action a is added to A_{DONE} and the public actions in π_a^+ are added to π_i^+. In this step, a finite (but possibly zero) number of actions is added to π_i^+ and the number of actions in A_{DONE} increases by 1 as a is added. As the number of actions in A is finite (and so is the number of public actions), and in each iteration, the number of actions in A_{DONE} increases, eventually, the set of actions $a^{\triangleright+} \in \pi_i^+ \setminus A_{\text{DONE}}$ such that $a^+ \in A_j^+ \wedge j \neq i$ becomes empty and the computation terminates. □

3.2.5. Suitability of the projected and privacy-preserving set-additive FF for the MADLA search

The MADLA Search places a number of assumptions on the properties of the pair of heuristics it operates with. Here, we examine, how these assumptions hold for projected FF and Privacy-Preserving Set-Additive FF. Let us recall and expand upon the assumptions, where i is a required property and ii and iii are desirable properties:

i) The distributed heuristic h_D^i is non-blocking. This is a required property, meaning that the distributed heuristic allows the agent computing it to run other computations meanwhile (all in a single computational process). The motivation behind this is the expectation that a distributed heuristic will communicate with other agents and while waiting for replies, the search will continue using the local heuristic.

ii) The distributed heuristic h_D^i is better informed than the local heuristic h_L^i, i.e., a single-heuristic search using h_D^i expands fewer states than the same search using h_L^i. This assumption is not strict, but using a less informed distributed heuristic gives no advantage. Although it cannot be guaranteed in general, it can be expected that for most states the dominance holds.

iii) The local heuristic h^i_{L} is easier to compute than the distributed heuristic h^i_{D}. Again, this assumption is not strict. Let us assume unit-time atomic computational steps used by both heuristic estimators (i.e., procedures computing the heuristic functions), in that sense, h^i_{L} is assumed to take less steps than h^i_{D} for a given state s. As the assumption is not strict, it suffices to hold for a significant number of states. The reasoning behind this assumption is that in conjunction with assumption ii it does not make sense to use a less informed local heuristic which takes longer time to compute.

Let us have a look how the proposed projected FF and distributed FF algorithms adhere to assumptions i–iii. As addressed in the description of Algorithm 5, the loop on lines 7–10 is actually implemented as an asynchronous event-based message processing, which means that after all messages are sent, another computation can proceed until some reply message is received. This can be utilized by the MADLA Search as shown in Algorithm 1, where on line 15 the `processNode(`α_i, u, d`)` is called with the parameter $d = \mathsf{true}$ and thus Algorithm 2 proceeds with the asynchronous call to the distributed heuristic on line 5 and the search can continue with the local search loop on lines 17–21 in Algorithm 1. Notice that the communication is processed also inside the local search loop, thus if a reply message is received, the local search loop is exited as the $\mathsf{busy}_{\mathsf{D}}$ flag is set to true. This behavior assures that the ppsaFF heuristic adheres to Assumption i.

To assess the assumptions ii and iii we first observe that the initial phase of the distributed heuristic computation as shown in Algorithm 5 is to compute the projected relaxed plan (line 3), which is exactly how the projected FF is computed. After that, some additional steps are performed, in order to improve the quality of the relaxed plan estimation. This means that the number of computational steps performed by the distributed FF is always at least as high as the number of steps performed by the projected FF or higher, thus confirming the assumption iii.

For similar reasons as above, it can be expected that the informativeness of the distributed heuristic would be higher as information is only added, nevertheless, the informativeness of a heuristic is best assessed by an experimental evaluation. Here we refer to the Section 4 where the number of expanded states is compared.

3.3. Soundness and completeness

Here we present the proofs of soundness and completeness of the MADLA Search algorithm. The proofs are based on proofs for classical single-agent single-heuristic search.

Before delving into the proof, we state the assumptions on the used heuristics. First, the heuristics always terminate. Second, the heuristics are *safe*, i.e., if $h(s) = \infty$ then $h^*(s) = \infty$ where h^* is the perfect heuristic. In other words, if the heuristic reports a dead-end, it truly is a dead-end (note that the other direction of the implication does not have to hold, a heuristic (e.g., FF) can report a finite heuristic value for a state which actually is a dead-end).

Throughout the section, we use the notion of a search node from Definition 8. Note that the search node is agent-specific, that is a search node $u = \langle s^{\triangleright i}, p, a_p, h, g, \alpha_i, \langle \delta_1, ..., \delta_n \rangle \rangle$ is known to and manipulated by agent α_i only. Subsequently, a search node represents a global state s as an i-projection $s^{\triangleright i}$ and a tuple $\Delta = \langle \delta_1, ..., \delta_n \rangle$ encoding the private parts of all agents in $\mathcal{A}$. We use the dot notation $u.\mathsf{state} = s^{\triangleright i}$, $u.\mathsf{action} = a$, $u.\mathsf{parent} = p$ (that is $u.\mathsf{parent} = u'$ or $u.\mathsf{parent} = \alpha_j$), $u.h = h$, $u.g = g$, $u.\mathsf{agent} = \alpha_i$, $u.\mathsf{uids} = \Delta = \langle \delta_1, ..., \delta_n \rangle$ and $u.\mathsf{uid}_i = \delta_i$.

We define a global equality relation for search nodes as follows.

Definition 11. Let u, u' be search nodes such that $u.\mathsf{agent} = \alpha_i$ and $u'.\mathsf{agent} = \alpha_j$ and let s, s' be global states reconstructed from $u.\mathsf{state}$ and $u'.\mathsf{state}$ respectively using all private parts defined in $u.\mathsf{uids}$ and $u'.\mathsf{uids}$ respectively. Then the search nodes u and u' are globally equal iff the states s and s' are equal, formally $u \overset{\mathsf{G}}{=} u' \Leftrightarrow s = s'$.

In other words, two search nodes are globally equal if the global states represented by them are equal.

3.3.1. Soundness

First, we define the sequence of search nodes corresponding to a particular run of the algorithm.

Definition 12. A *path* with a *resulting search node* u_l is a sequence of search nodes $\mathsf{path}(u_l) = (u_0, ..., u_l)$, possibly of different agents (different k s.t. $1 \leq k < l$ can exist for which $u_k.\mathsf{agent} \neq u_{k+1}.\mathsf{agent}$). The search nodes do not repeat, although the respective states may repeat.

A path is a *valid path* iff $u_0 = u_I$ is the initial node for some agent α_I and for each k s.t. $1 \leq k \leq l$: $u_{k-1} = u_k.\mathsf{parent}$ and $a_k = u_k.\mathsf{action}$ is applicable in $u_{k-1}.\mathsf{state}$, or $u_k.\mathsf{agent} = \alpha_i$ and $u_k.\mathsf{parent} = \alpha_j$ s.t. $i \neq j$, in which case $u_{k-1}.\mathsf{agent} = \alpha_j$ and $u_k \overset{\mathsf{G}}{=} u_{k-1}$.

Informally, a path represents one particular trace of the exploration of the search space. In the case a state was reached through expansions of multiple agents, agent α_j sends a state s_{k-1} to agent α_i which creates a new search node u_k from it. This results in a sequence of search nodes $(u_0, ..., u_{k-1}, u_k, ..., u_l)$ where the search nodes u_{k-1}, u_k represent the same state s_{k-1}, but u_{k-1} was created by α_j (using a public action) and u_k was created subsequently (after receiving the message) by α_i, setting $u_k.\mathsf{parent}$ to α_j (see Algorithm 3, line 7). Next, we define the corresponding sequence of actions.

Definition 13. For a path $\text{path}(u_l) = (u_0, u_1, ..., u_l)$, we say $\text{path}(u_l)$-*plan* is a sequence of actions $(a_1, ..., a_l)$ s.t. $a_k = u_k.\text{action}$ and $a_k \in A \cup \{\epsilon\}$ for all $1 \leq k \leq l$. For all a_k it holds that either $a_k \in A_i$ where $\alpha_i = u_k.\text{agent}$ or $a_k = \epsilon$ if u_k was received form another agent (that is $u_{k-1}.\text{agent} \neq u_k.\text{agent}$). The $\text{path}(u_l)$-plan is a *valid* $\text{path}(u_l)$-*plan* iff $\text{path}(u_l)$ is a valid path.

A trivial consequence of the above definitions and Definition 1 is that a valid $\text{path}(u_l)$-plan $(a_1, ..., a_l)$ is a multi-agent plan solving $\mathcal{M}$ iff $u_l.\text{state}$ is a goal state, that is $G \subseteq u_l.\text{state}$ (remember that the goal is public, i.e. $G \subseteq P^{\text{pub}}$).

To prove the soundness, the following lemma will be shown first:

Lemma 14. ***(Invariant)*** *At any given step of the MADLA Search, for any search node $u_{\text{L}} \in O^i_{\text{L}}$ and any search node $u_{\text{D}} \in O^i_{\text{D}}$ for any agent α_i, $\text{path}(u_{\text{L}})$ and $\text{path}(u_{\text{D}})$ are valid paths.*

Proof. In the initial step for every agent, $O^i_{\text{L}} = O^i_{\text{D}} = \{u_I\}$, where $\text{path}(u_I) = (u_I)$, which is a valid path. There are two possibilities where new nodes are added to any of the open lists. In `expand(u,d)`, regardless of the d parameter, for each action applicable in $u.\text{state}$, a new search node $u' = \langle s'^{\triangleright i}, u, a, h', g+1, \alpha_i, \Delta \rangle$ is created such that $s'^{\triangleright i} = u.\text{state}[a]$. If we assume that $\text{path}(u) = (u_I, ..., u)$ is a valid path, then after expansion, $\text{path}(u') = (u_I, ..., u, u')$ is also valid for each new u'.

A node may be received from another agent in `processComm()`. Assume that for some node $u_k = \langle s^{\triangleright j}_k, u_{k-1}, a_k, h_k, g_k, \alpha_j, \Delta_k \rangle$, $\text{path}(u_k)$ is a valid path and u_k is a first node in $\text{path}(u_k)$ expanded using a public action $a_k \in A^{\text{pub}}_j$. According to Algorithm 2, line 9, u_k is sent to α_i as a message $M_{\text{STATE}} = \langle s^{\triangleright \text{pub}}, \Delta_k, h, g, d \rangle$. The message is received by α_i and a new search node $u_{k+1} \leftarrow \langle s^{\triangleright i}_k, \alpha_j, \epsilon, h_k, g_k, \alpha_i, \Delta_k \rangle$ is created (based on Δ_k and the state reconstruction function μ^i) and added to either open list based on the parameter d. According to Definition 12, $\text{path}(u_{k+1})$ is a valid path. There is no other possible way a node could be added to any of the open lists. □

Theorem 15. *When the MADLA Search terminates and returns a solution, it is a distributed multi-agent plan solving $\mathcal{M}$.*

Proof. The algorithm terminates on line 23 of Algorithm 3. In each step, either an action $a_k = u_k.\text{action}$ is added to the solution π^i_m for the solution initiated by agent α_m and continues the recursion on $u_{k-1} = u_k.\text{parent}$ (if $u_k.\text{action} \neq \epsilon$), or a $M_{\text{RECONSTRUCT}} = \langle u^{\triangleright \text{pub}}_k.\text{state}, u_k.\text{uid}_j, t, \alpha_m \rangle$ message is sent to $\alpha_j = u_k.\text{agent}$. When received, the `reconstructPlan()` procedure is called on a node u'_k s.t. $u'_k \overset{\text{G}}{=} u_k$ and $u'_k.\text{action}$ is public (u'_k is obtained from μ^i). Thanks to the condition on line 6 of Algorithm 4, upon termination, the last action added to the returned solution π^i_m is the action that was applied on the initial node u_I. This ensures that the recursion proceeds along the path $\text{path}(u_t)$, where u_t is the node on which `reconstructPlan()` was first called.

Apart from the recursive call, the procedure `reconstructPlan()` is called only from line 30 of Algorithm 1, where u_t was extracted from O^i_{L} or O^i_{D}. From Lemma 14 it follows that $\text{path}(u_t)$ is a valid path. Because on line 30 of Algorithm 1, $G \subseteq u_t.\text{state}$ always holds, the $\text{path}(u_t)$-plan corresponding to $\text{path}(u_t)$ is a multi-agent plan π solving $\mathcal{M}$. As each agent α_i adds to π^i_m only actions from A_i (and ϵ actions as padding, see Algorithm 4 line 4), the resulting set $\{\pi^i_m\}^n_{i=0}$ is a distributed multi-agent plan solving $\mathcal{M}$ for each m as the reconstructions are independent if started by different agents and each agent can start the reconstruction of at most one plan. The final plan is chosen consistently and uniquely as it is the shortest plan, ties broken based on unique agent indices (ordering). □

3.3.2. Completeness

To show completeness, we first consider a modification of the algorithm such that any reachable state is expanded eventually. The modified algorithm is named MADLA$^+$ Search in which the condition on lines 25–30 in Algorithm 1 are ignored and the algorithm terminates only when both O^i_{L} and O^i_{D} are empty for all agents α_i and no messages are pending, which can be detected using the distributed snapshot algorithm [21].

Lemma 16. *In the MADLA$^+$ Search, each state s is added to O^i_{D} and to O^i_{L} of any agent α_i at most finite times, each time represented by a different search node.*

Proof. Because the number of possible search nodes (with respect to the state $s^{\triangleright i}$ and the set of private unique identifiers Δ) is finite as each search node represents one state of the finite sets of states ($2^{|P|}$ since $s \subseteq P$) and the number of actions (of each agent) is also finite, each expansion produces a finite number of search nodes consequently added to O^i_{D}, O^i_{L} or both (line 17 and 19 of Algorithm 2 respectively). If a search node u is extracted from O^i_{L} or O^i_{D}, it is added to the closed list C^i and no search node u' s.t. $u \overset{\text{G}}{=} u'$ is ever added to any of the open lists of agent α_i again.

Another possibility of adding a search node to an open list is when it is received from another agent. A state s is sent by the agent α_i, only if a search node u' (representing a different state s') was extracted from either O^i_{L} or O^i_{D} and a public action $a \in A^{\text{pub}}_i$ was applied. Since there is a finite number of public actions and a finite number of agents and each action

is applicable only in finite number of states (which are then placed into C^i and never expanded again), state s can be sent and received only a finite number of times. □

Lemma 17. *In the MADLA$^+$ Search, each search node in O^i_{L} and in O^i_{D} of all agents α_i is eventually extracted.*

Proof. In each step of the outer search cycle (lines 6–21 of Algorithm 1), a node is extracted from O^i_{D}, if h^i_{D} is not busy. Since h^i_{D} always terminates, any finite number of nodes can be extracted in finite time. From Lemma 16 follows that only a finite number of nodes may be added to O^i_{D}, therefore O^i_{D} eventually becomes empty. When O^i_{D} is empty, in each step of the inner search cycle a node is extracted from O^i_{L}. Following the same reasoning as before, O^i_{L} eventually becomes empty as well. □

Theorem 18. *The MADLA$^+$ Search terminates.*

Proof. Follows directly from Lemma 16 and Lemma 17. □

Recall that for states s_0 and s_m a s_0–s_m-plan is a sequence of actions $\pi = (a_1, ..., a_m)$ from A (the actions may be from different agents and may repeat) if there are states $s_1, ..., s_m$ s.t. for all k in $1 \leq k \leq m$, action a_k is applicable in s_{k-1} and $s_k = s_{k-1}[a_k]$. For short, $s_0[\pi]$ denotes the resulting state s_m. We extend this notion to the search nodes the same way action application was extended (i.e., the actions are applied on the respective nodes).

We define the notion of reachability for the multi-agent planning problem $\mathcal{M}$ as follows.

Definition 19. A state $s \subseteq P$ is reachable in $\mathcal{M}$ iff a s_I–s-plan $\pi = (a_1, ..., a_m)$ exists such that s_I is the initial state. We say that s is reachable by agent α_i iff $a_m \in A_i$.

Now we show equality of reachability and the existence of a valid path.

Lemma 20. *A state s is reachable in $\mathcal{M}$ by agent α_i iff a valid path $\mathsf{path}(u) = (u_I, ..., u)$ exists such that $u_I.\mathsf{state} = s_I^{\triangleright j}$ for some agent α_j, $u.\mathsf{state} = s^{\triangleright i}$, $u.\mathsf{uids}$ represent private parts of s and $u.\mathsf{agent} = \alpha_i$.*

Proof. If a valid path exists, the corresponding $\mathsf{path}(u)$-plan proves the reachability. If a state s is reachable in $\mathcal{M}$ by α_i, there exists the sequence $\pi = (a_0, ..., a_l)$ of actions from Definition 19 (and $a_l \in A_i$). Let $a_0 \in A_j$. Since a_0 is applicable in $s_I^{\triangleright j} = u_I.\mathsf{state}$, it will be applied by the agent α_j and the resulting search node u_1 will be added to its open lists. Additionally, if $a_0 \in A_j^{\mathsf{pub}}$, $s_1^{\triangleright \mathsf{pub}} = u_1.\mathsf{state}$ together with $u_1.\mathsf{uids}$ will be sent to other agents. According to the Definition 19, a_1 is applicable in $s_1^{\triangleright j}$. If $a_1 \in A_j$ the process is repeated. If $a_1 \in A_{k \neq j}$, a_1 is applicable in the received state $s_1^{\triangleright k}$ and thus will be applied by agent α_k. By induction we conclude that $\mathsf{path}(u) = (u_I, ..., u)$ is a valid path. □

Now, we provide an alternative definition of reachability in $\mathcal{M}$, with the focus shifted on the agents. We use the bracketed index (k) to annotate the k-th agent in the sequence which is not the same as the agent $\alpha_k \in \mathcal{A}$.

Definition 21. A state s is reachable in $\mathcal{M}$ by a sequence of agents $\varpi = (\alpha_{(1)}, ..., \alpha_{(m)})$ of length $m + 1$ (agents in ϖ can repeat and an agent can perform zero actions or a no-op action ϵ) if a sequence of $u_1^{(i)}$–$u_{k_i}^{(i)}$-plans $(\bar{\pi}_1, ..., \bar{\pi}_m)$ exists such that each $\bar{\pi}_i$ contains only actions from $A_{(i)}$, $\bar{\pi}_1$ is applicable in u_I and for each i s.t. $1 \leq i \leq m$, $u_{k_{i-1}}^{(i-1)} \overset{\mathsf{G}}{=} u_1^{(i)}$.

Informally, in the sequence of agents, each agent performs a sequence of actions, such that the final resulting state is s.

The following lemma uses either the assumption that the used heuristics are safe, or requires the states evaluated as dead ends to be placed in the open-list nonetheless, with the heuristic value of ∞. This is necessary to make sure that a reachable state is never unreached only because of the heuristic evaluation.

Lemma 22. *If a state s is reachable in $\mathcal{M}$ by the sequence of agents $\varpi = (\alpha_{(1)}, ..., \alpha_{(m)})$ of length m, it is placed into the closed list $C^{(m)}$ after a finite number of steps.*

Proof. We will show the proof by induction in the number of agents m.

If a state s is reachable by a sequence containing single agent $\varpi = (\alpha_i)$, then assume that no search node u such that $s = u.\mathsf{state}$ is ever placed in C^i. Since s is reachable, based on Lemma 20 there exists a search node u s.t. $u.\mathsf{state} = s$ and $\mathsf{path}(u) = (u_I, ..., u)$. Let u_m be the first search node in $\mathsf{path}(u)$, s.t. u_m is not added to C^i. Note that there exists an action $a \in A_i$, such that $u_m = u_{m-1}[a]$. Since at some point $u_{m-1} \in C^i$, u_{m-1} must have been taken from O^i_{L} or O^i_{D}. At that

point, u_{m-1} was also expanded and because a is applicable in u_{m-1}.state it must have been applied. The resulting node $u_m = u_{m-1}[a]$ was added to either O^i_{L} or O^i_{D} and because of Lemma 17, u_m was eventually extracted and added to C^i. This contradicts the assumption that u_m is not added to C^i.

Let us now assume that for all k s.t. $k \leq m$ if a node is reachable by a sequence of agents $\varpi = (\alpha_{(1)}, ..., \alpha_{(k)})$ of length k, it is added to $C^{(k)}$ after finite many steps. We will show that the same holds if a node is reachable by a sequence of agents $\varpi' = (\alpha_{(1)}, ..., \alpha_{(m)}, \alpha_{(m+1)})$ of length $m+1$. We will show the induction step by a contradiction. For the contradiction let us assume that s is a state reachable by sequence of agents ϖ' of length $m+1$, but no search node u such that $s = u$.state is ever added to $C^{(m+1)}$. Let $\mathsf{path}(u) = (u_1, ..., u)$ and let u_l be the first node that is never added to $C^{(m+1)}$. One of the following holds:

i) u_l is reachable by agent α_{m+1}. If so, the same reasoning used in the first part of the proof can be used to obtain a contradiction.
ii) u_l is reachable by some $k' < m+1$ agents $\varpi'' = (\alpha_{(1)}, ..., \alpha_{(k')})$. In that case, we have a contradiction with the assumption of the induction. □

Now we can conclude the proof by the following theorem.

Theorem 23. *The MADLA Search is complete.*

Proof. In the MADLA$^+$ Search, for every state s reachable in $\mathcal{M}$ by an agent α_i such that $s \subseteq G$, a search node u such that $s = u$.state is eventually placed into C^i (Lemma 22). The first such search node is, after adding it to the closed list C^i, given to the `reconstructPlan()` procedure which reports a valid multi-agent plan (as MADLA Search is sound, Theorem 15). □

3.4. Privacy of the algorithms

In this section, we analyze the privacy of the MADLA Search, the distributed heuristic from Section 3.2.2 and the combination of both according to Section 2.1.

3.4.1. Privacy of the MADLA Search

To discuss the privacy of the MADLA Search, it is important to notice that in terms of privacy, MADLA Search is no different from MAD-A* or MA-BFS. This is due to the fact that MADLA Search uses exactly the same mechanism for sharing states expanded by a public action (see Section 3.1.1). This leads us to the analysis of what information is compromised by the search.

According to [3], MA-BFS and thus also the MADLA Search is at least weak privacy-preserving, as no private information is explicitly sent to other agents. Similarly, both MA-BFS and thus also the MADLA Search are not strong privacy-preserving as some additional information can be deduced from the search. Here we analyze what this information is.

Recall the definition of a valid path $\mathsf{path}(u) = (u_0, ..., u)$ (Definition 12), a sequence of search nodes leading from the initial search node $u_0 = u_I$ to the search node u, such that each consecutive search node (except for u_I) is a result of application of an action on the previous one (or a result of receiving a message in which case the represented state is the same as the state represented by the previous node). If $\alpha_j = u_k.\mathsf{agent} \neq u_{k+1}.\mathsf{agent} = \alpha_i$ holds for two nodes, then u_{k-1} was expanded by the agent α_j using a public action, resulting in a search node u_k which was via its state sent to α_i, resulting in u_{k+1}. In order to reconstruct the part of u_{k+1} private to α_i, the agent is able to determine the last search node u_l in $\mathsf{path}(u)$ s.t. $l < k-1$ and $u_l.\mathsf{agent} = \alpha_i$ (in an extreme, this may be u_I which is known to all agents). If $u_l \neq u_I$, the search node was expanded from u_{l-1} using a public action and sent to all agents, including α_j. We say that u_l is an i-parent of u_{k+1}. Informally, an i-parent of a search node of agent α_i is the last predecessor of the search node which is also of agent α_i. All states in between are of other agents. We also say that u_{k+1} is an i-successor of u_l and i-succ(u_l) is a set of all i-successors of u_l.

An important observation an agent may make is that for two search nodes u, u' s.t. the i-projections of their respective states are equal it holds that i-succ$(u) \neq i$-succ(u'). This indicates that for each search node $u_{\mathsf{dif}} \in i$-succ$(u) \setminus i$-succ(u'), there is some action a in the u–u_{dif}-plan which is not applicable in the sequence starting with u (analogously for i-succ$(u') \setminus i$-succ(u)), which can be only due to an unknown private proposition(s) with a different value in u and u'. The agent can now deduce that u and u' are in fact two different nodes. Although a may be private, the final effect of u, u' being different is demonstrated by the public action $a' = u'_{\mathsf{dif}}.\mathsf{action}$, where u'_{dif} is the search node preceding u_{dif}. Notice that based on the difference of public parts of u and u_{dif}, agent α_i can determine the transition in the public projection induced by the action a'. We refer to such actions as *state-discerning actions*.

Let us demonstrate the behavior of state-discerning actions on the example from Section 1. Imagine that the truck has performed the public action **unload-t-B** on two nodes, resulting in two states which are the same in t-projection, namely $s^{\triangleright \mathsf{t}} = s'^{\triangleright \mathsf{t}} = \{\mathsf{truck\text{-}at\text{-}B}, \textbf{package-at-B}\}$. Since the states were results of a public action, they are both sent to the plane. Throughout the search, the truck receives only the t-successor of $s^{\triangleright \mathsf{t}}$, but no t-successor of state $s'^{\triangleright \mathsf{t}}$. This indicates that

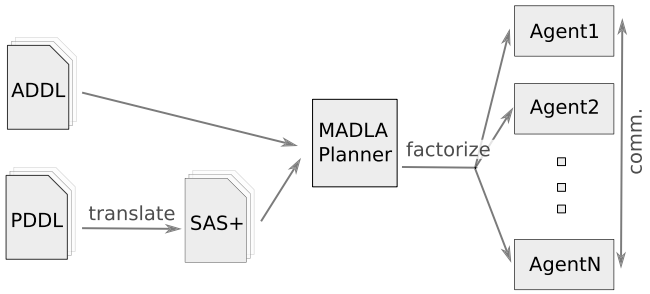

Fig. 5. Stages of the MADLA Planner.

$s^{\triangleright t}$ and $s'^{\triangleright t}$ were not the same and there is some private fact which discerns them (it was plane-at-B which holds in s, but not in s'). Moreover, as the t-successor of $s^{\triangleright t}$ is {truck-at-B}, that is **package-at-B** is no longer present, the state-discerning (public) action must have **package-at-B** in its delete effects. In this problem, **load-a-B** is the only such action.

Clearly, the existence of state-discerning actions enables agents to determine some non-trivial information about the other agent's private parts of the problem, such as existence of a private variable. Combination of multiple state-discerning actions can reveal even the existence of multiple private variables (e.g., if a is applicable in s and s', but not in s'' and a' is applicable in s but not in s', it means that there are at least two private variables). Thus we can conclude that MAD-A* and thus also MADLA Search is not strong privacy-preserving.

3.4.2. Privacy of the distributed heuristic

The distributed heuristic hides the private information by communicating only public actions and heuristic estimates. This satisfies the requirements for a weak privacy-preserving algorithm. But is there some information that can be deduced from the heuristic computation?

It turns out that there is a very similar principle of revealed information as in the search. In the heuristic computation, agent α_i directly asks agent α_j about applicability of actions. More precisely, if agent α_i is computing a heuristic for state s, it computes a projected relaxed plan π_i^+. If such projected relaxed plan contains a projection of some action $a^+ \in A_j^{\text{pub}}$, α_i requests α_j to provide the heuristic estimate of reaching the private preconditions of a^+ from s. Agent α_j sends a reply containing public actions used to reach the private preconditions and the number of private actions used to reach the private preconditions.

Again, agent α_i can observe that although from its projection the two states s and s' are the same, and thus the respective projected relaxed plans π_i^+ and $\pi_i'^+$ are the same, the replies from α_j differ in either the public actions or the number of private actions sent in the reply. This simply means that the states s, s' are different in the problem of α_j. Here, a similar situation to the state-discerning actions in search occurs in the heuristic computation, thus again revealing some non-trivial private information. Again, we conclude that the privacy-preserving set-additive FF heuristic is not strong privacy-preserving.

In the example problem and in a similar situation as in the previous section, the replies to a request for **unload-a-C** would yield 1 as the number of private actions in one case (that is the plane first needs to use move-a-C-B) and 0 in the other case (plane is already at B as plane-at-B holds).

Obviously, the combination of weak privacy-preserving search and heuristic is also weak privacy-preserving and as both the search and heuristic are not strong privacy-preserving, also their combination is not strong privacy-preserving.

4. Implementation and evaluation

After the formal verification of the proposed MADLA Search, we analyze its practical properties in an experimental evaluation. First, we describe the implementation in more technical detail. Second, we discuss the related work. Third, we compare the building blocks of the planner, namely projected FF and distributed ppsaFF heuristics in a multi-agent single-heuristic search, the pair of the heuristics in a multi-agent multi-heuristic search and finally in the MADLA Search. Fourth, we discuss the properties of the planner on particular planning domains and on various metrics in detail. Finally, we compare the planner with the state-of-the-art distributed multi-agent planners on a multi-agent benchmark set.

4.1. Implementation

The MADLA Planner proceeds in a number of stages (similarly to many classical planners), shown in Fig. 5. The input of the planner is a description of the domain and problem in classical PDDL, accompanied with a file in a novel format Agent Domain Description Language (ADDL). The ADDL file lists which objects in the PDDL represent agents.

The first stage is the translation of the PDDL input into SAS+ format, which is used internally by the planner. The translation is performed by the standard tool in the Fast-Downward Planning System [10] and thus is centralized (we leave

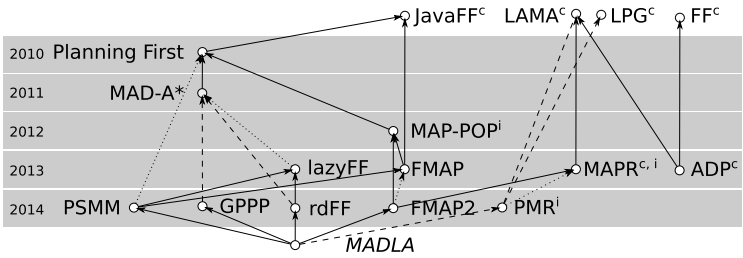

Fig. 6. Subset of related work to the MADLA Planner. The solid arrows represent comparison and improvement (in most cases) as better → worse; dashed lines represent informal comparison not over coverage (number of solved problems) or runtime (e.g. theoretical comparison); and dotted lines represent inspiration of the approaches as inspired by → inspiration. Additionally, superscripts *c*, *i* stand for centralized and incomplete algorithms respectively. MAD-A* is the only optimal multi-agent planner in the figure.

the distribution of the translation process for future work). Next, the bootstrapping part of the MADLA Planner takes the SAS+ and ADDL inputs and partitions the problem for the specified agents.

The partitioning process starts with partitioning of (grounded) actions. A grounded action a is assigned to an agent α_i if α_i (its respective PDDL object) is the first parameter of a mentioned in the ADDL input file representing an agent. Next step is partitioning of the variables and values and determining which of them are public (i.e. those shared among multiple agents).

After the partitioning step, each agent is initialized and run in a separate process. From now on, the planning is distributed, each agent is running on its own thread and communicating with other agents via message passing over the TCP/IP protocol. Detecting the nonexistence of a solution in the distributed setting is nontrivial (using the distributed snapshot algorithm) and introduces communication overheads and thus we currently resort to a time limit instead, which is not a problem in practice, as the planner is typically run within a time limit anyway. When a solution is found, we use a variant of the plan reconstruction process slightly more efficient than the described one. The main difference is that if a plan of length l is being reconstructed, the reconstruction process of any plan with length l' such that $l' > l$ is terminated as soon as it is detected (e.g. when the reconstruction message is received).

Also, in the implementation, we use separate closed lists C^i_{D} and C^i_{L} for the states evaluated using the distributed and the local heuristic respectively, but we are using only a single closed list C^i for the simplicity of the presentation. The use of the separate closed lists improves the coverage of MADLA Search by 2.9% and the use of local heuristic re-computation when state is received by 3.2% (see the next sections). All the theoretical results hold also when using two closed lists as eventually, all states will be in both closed lists.

The last implementation detail we present here is related to the non-blocking property of the distributed heuristic (see Section 3.2.2 and Section 3.2.3). The heuristic computation is not ideally non-blocking, as parts of the heuristic are computed locally in the agent's thread and thus do not allow other computation to run at the same time. Such situation is namely when the heuristic is initiated and computes local relaxed plan. Another such situation occurs when the initiator agent needs to satisfy private preconditions of its own public action. The experimental evaluation shows, that in some domains, this violation of the non-blocking property degrades the efficiency of the MADLA Planner.

The MADLA Planner is written in Java, its code, the data sets and processing scripts and used benchmarks are available at http://github.com/stolba/MADLAPlanner.

4.2. Related work

The planners for MA-STRIPS and related multi-agent models range in the literature from centralized[3] single-core planners [25,19], parallel multi-core ones [5,26], to fully distributed implementations with communication within one physical computer or over the network [5,6,23]. In Fig. 6, we summarize the related planners relevant to the MADLA Planner (i.e., compatible with the MA-STRIPS model) with visualization of which planners were compared and how.

The first MA-STRIPS planner called Planning First [27] was based on the principles proposed in the MA-STRIPS paper [2]. Planning First coordinates local plans resulting from agents' forward-search planners by a distributed solver of Constraint Satisfaction Problems (DisCSP). Planning First was the first representative of a *plan-based (top-down) coordination* multi-agent planner, whereas the MADLA Planner is a multi-agent planner based on *state-based (bottom-up) coordination*. Planning First outperformed a Java implementation of the FF planner JavaFF [28] in the coverage metrics.

In the plan-based coordination planners, the process towards the resulting global plan works with complete or partial local plans. The plans are usually locally sound and finding a valid combination of such local plans is the coordinating part of the planner. Provided that the coordination of the local planning processes is based on states, not plans, we talk about state-based coordination of the planning process. This is not the only distinction of multi-agent planners possible. Moreover,

[3] Centralized in the sense that although the algorithms utilize decomposition on agents, the algorithm is running on a single core of a single machine without the use of parallelization or distribution.

one can come up with a planner which uses a mixture of the described coordination techniques. For the context of this article it helps us, however, to explain how our contribution fits in the set of related planners, where are similarities and how the techniques proposed here can help other multi-agent planners. The most notable benefit of state-based coordination planners is the possibility to use state-based heuristic functions not only for local planning, but also for the global (coordination) planning as it uses states both locally and globally. This is also the key motivation for the MADLA Planner being designed as a state-based coordination planner. The key benefit of the plan-based coordination planners is the ability to represent the local (partial) plans compactly.

Besides Planning First, other representatives of plan-based coordination planners are Distributed Planning by Graph Merging (DPGM) [9,29], μ-SAT-PLAN [30] and Best Response Planning (BRP) [31], each using a different method to find the coordinated set of local plans and representation of the plans. Similar to Planning First, DPGM uses a Constraint Satisfaction Problem solver for coordination of the local plans, which are however not planned by a forward-search planner, but represented and extracted from distributed planning graphs. μ-SATPLAN limited to at most two agents represents the plans and their coordination as a SAT problem. BRP planner uses the best-response principle, i.e., it iteratively amends a plan by compatible local plans (not containing mutually exclusive actions) of other agents eventually getting a global multi-agent plan.

The plan-based coordination planner Distoplan [32] pioneered an idea of planning by intersection of Finite State Machines (FSM) representing the local plans of the agents. This idea was extended and practically refined in a satisficing planner based on nondeterministic FSMs representing local agents' plans and their merging—Planning State Machine Merging (PSMM) in [26].

Highly efficient incomplete planners from the plan-based coordination family, able to outperform the state-of-the-art centralized planners such as LAMA [20], FF [7], or LPG [33], are Multi-Agent Planning by Plan Reuse (MAPR) [19] and its variation called Plan Merging by Reuse (PMR) [34], which additionally proposes a distribution scheme. MAPR and PMR are based on the best-response principle similarly to BRP, sequentially using a plan repairing algorithm (particularly LPG-adapt [35]) to converge to a globally sound plan.

The multi-agent state-space search was first used in a planner using distributed variant of the A* algorithm called MAD-A* [5] with admissible projected landmark heuristic LM-cut [36] and Merge&Shrink [37]. MADLA Search described in the previous sections is a Multi-Agent Greedy Best-First Search (MA-BFS) variant of MAD-A* with inadmissible heuristics. The MA-BFS with projected FF heuristic evaluated in Section 4.3.2 represents the same search algorithm and projected heuristic as used in the MAFS [3], therefore we do not evaluate it explicitly again. Similarly rdFF [6] and lazyFF with MAFS [8] presented in our previous work use the same principle of state-space search distribution. Although the state-space search in MAD-A* is distributed, it does not treat the local private states differently than the public states with exception of communication of the public states to other agents providing the state-based coordination. Therefore planners related to MAD-A* search fit the state-based coordination group. A recent distributed multi-agent state-space search planner inspired by MAD-A* called Greedy Privacy Preserving Planner (GPPP) [18] uses global landmark extraction and is based on an iterative deepening backtrack search both in the relaxed private subproblems and in the non-relaxed public subproblem. The landmarks are used for the computation of the heuristic function as well as the public search.

Another distributed search using A* was proposed in [38] as A# planner. A# does not explicitly communicate public states, instead, it transforms the coordination problem into multiple goal variants. When solving the planning problem by a state-space search, the planner implicitly solves the coordination problem by searching for particular goals describing an appropriate solution of the coordination sub-problem.

The Agent Decomposition-based Planner (ADP) [25] uses Relaxed Planning Graphs for repeated extraction of subgoals of a multi-agent planning problem and the FF heuristic to steer the search toward the extracted subgoals. The extracted subgoals help to escape plateaus of the FF heuristic towards states at the overlapping boundaries of the agent's problem factors.

Multi-agent planners using partial-order plans were firstly studied in a multi-agent temporal planner TFPOP [39], which required an extended model in contrast to MA-STRIPS because of an additional requirement on temporal constraints. A series of partial-order planners based on a model compatible with MA-STRIPS and fitting the state-based coordination class begun with incomplete MAP-POP [40] and two versions of complete planners FMAP [41,23]. FMAP does not use state-space search similarly to the planners in the MAD-A* family, but as different local and global searches. First, it uses local forward searches run by the agents to successively complete partial ordered plans. Second, the coordination subproblem is driven by a synchronized Best First Search (all agents search in a synchronized manner the same tree for efficiency reasons) over a space of the partial ordered plans. The synchronized Best First Search utilizes a distributed state heuristic based on Domain Transition Graphs (DTGs) approximating relaxed plans [10], the length of which is used as an evaluation of the partial ordered plans. The state heuristic is computed for frontier states of agent's local plans similarly to e.g., in OPTIC a centralized partial-ordered planner [42].

4.3. Evaluation

In the classical planning literature, the typical method for comparing planners or heuristics is to test them on a set of benchmark domains, each having a number of problem instances. The domains (or problems) vary in many properties, such as size, combinatorial hardness, structural traits and others. The same methodology was adopted in multi-agent planning.

A notable difference is in the set of benchmark domains, which is much less "standardized" in multi-agent planning than in classical planning, as there is no track at International Planning Competition (IPC) [43] for multi-agent planning yet. Therefore the set of benchmarks we are using in this article is a union of domains used for comparison of the four planners we are comparing MADLA to in the last section. All these domains and problems originate in single-agent domains and problems from the IPC series. Since the IPC domains are designed to both examine efficiency of the planners against various (theoretically) interesting properties and relate to real world problems, the same motivated our benchmark selection. Not many properties specific only to the multi-agent planners (as most of the multi-agent planners are variations on distribution of principles used in classical planning) were studied so far, therefore no special multi-agent domains and problems are usually used. The most notable property studied already by [2] is coupling of the agents in the problems which correlates (to some extent) with the ratio of private and public actions in the planning problems. In the detailed analysis section, we will discuss influence of this and other more subtle traits of the problems on the efficiency of the used search and heuristics.

As the core comparison metrics, we use coverage, that is the number of solved problems under 20 minutes with 8 GB total memory limit (the memory is shared among all agents, assigned on an as needed basis until exhausted). In the detailed analysis, we use the number of expanded states (and their ratios for the used heuristics), computation time of each particular heuristic and the number of public actions requiring a (private) action supporter for a private precondition fact.

The MADLA Planner is a distributed multi-agent system, where the agents communicate over a TCP/IP connection, even if running on a single physical machine. The 20 minute time limit used for coverage timeout represents a make-span of the distributed process (that is, the time between the earliest time any agent starts computing and the latest time any agent finished) and thus we used wall-clock time, as it is not viable to deduce make-span including communication based on CPU time of the individual processes. For the same reasons, we have used wall-clock time also for all other time-related measurements. We are aware, that wall-clock time can be influenced by external sources, such as other processes running on the system. We attempted to mitigate such effects by running the experiments on dedicated machines. Moreover, the non-determinism of the planning process is inherent and impossible to be synchronized under the assumption of unknown ordering in message delivery from two different agents to one recipient. Considering the non-determinism and the unlikely, but possible interference of other system processes, every measurement was repeated 10 times[4] and the results were averaged. Each machine was equipped with 8 hyper-threading[5] i7 cores (i.e., 16 threads) at 2.6 GHz. Each agent was running on two threads. One thread is receiving messages and filling in content data structures in appropriate collections (e.g., states into the open lists O^i_{L}, O^i_{D}) and the other thread is searching and evaluating the heuristics.

4.3.1. Selected domains

The set of benchmarks we are using in this article is a union of domains used in the literature on the state-of-the-art MA-STRIPS based planners [3,5,6,23,19,18]. The domains are based on the classical IPC domains converted to multi-agent domains by choosing some objects to be treated as agents. This choice is arbitrary and we adhere to the conventions used in the cited papers. All agents are chosen so that each action is assigned to exactly one agent (this is required by the MA-STRIPS formalism), sometimes necessitating minor changes in the domain descriptions. Privacy is determined by the MA-STRIPS privacy definition—a fact is public if used by actions of multiple agents, an action is public if it uses some public fact. Here, we describe the domains and their complexities in more detail:

blocksworld This domain is the same as the classical blocksworld domain except for having multiple hands as agents, the holding and free facts being private. Each agent can solve the problem on its own, which makes it hard for MAP as for the projected heuristic it seems that the solution by other agents is cheaper (ignoring the private preconditions). All actions in the domain are public but have some private precondition, which means there are some dependencies among the actions not known to all agents. As an example, agents do not know that other agents must use pick-up before put-down, it seems to them that other agents can simply move blocks by the put-down action without picking them up first, that is ignoring the necessary preconditions of the pick-up action such as that the block is free (on top).

depot In depot, the trucks, depots, and distributors are agents. Most of the actions are public, nearly a quarter of them have private preconditions. Only the drive-truck actions are completely private. Truck locations and truck loads (crates) are always private. The position of a crate is specified by a fact on. A crate can be either on a hoist at a depot or on a truck. When it is on a truck, the fact is private.

driverlog In driverlog, the drivers are agents (the problem is modified so that each action has the driver as a parameter). Their locations, walk action and the fact that a driver is driving a truck are private, everything else is public. Most of the problems can be solved by a single agent, but unlike the blocksworld domain, this does not cause agents

[4] Although 10 samples is not enough for a reasonable statistical confidence, it helps to identify cases with extreme variance. This phenomenon did not manifest in any of our experiments, therefore we concluded the planner is deterministic enough for the used metrics. Deeper analysis of statistical properties of a multi-agent (multi-heuristic) search is an interesting topic left for future work.

[5] Although hyper-threading may affect the measurement of CPU-time, as we use solely wall-clock time, the experiments were not affected by hyper-threading.

to significantly underestimate costs of other agents (the ignored costs are basically only those of walking actions, which are not that significant).

elevators08 In the elevators domain, the elevators are agents. The locations of passengers are public only if shared among multiple elevators (i.e., changing floors). Public actions are only those board and leave actions involving a floor accessible to multiple elevators. Most of the problems can be solved by a subset of agents (typically the slow elevators). All public actions have private preconditions on the state of the lift, its capacity, etc. which makes the agents significantly underestimate the costs of other agents.

logistics00 The logistics domain contains two types of agents: trucks and planes, transporting packages between cities. The goal specifies only the locations of packages. The transporting task often requires cooperation of several agents (that is so in the classical benchmark domains as well). Locations of packages accessible to only one truck are private to that truck. The loading and the unloading actions at these locations are private as well. The multi-agent logistics domain, similarly to the classical variant, is suitable for the relaxation heuristics.

openstacks The openstacks problems contain two agents: a manager and a manufacturer, which are added atop of the classical IPC domain. The goal of the problems is to produce and ship several orders. The manufacturer has a number of orders. The orders are started and shipped by the manager agents. Each order is for a combination of different products, and the manufacturer can only make one product at a time. The stacks are temporary storage spaces for open orders. The information about made products is private. The rest of the information is public and the two agents have to coordinate finishing the orders. Shipment is public as it is in the goal.

rovers The domain models Mars exploration rovers, each represented by one agent. The goal is to collect samples and communicate acquired data. Rovers problems can be well decomposed as each agent has its own private set of targets and reachable locations (even if a location is shared, the rovers do not interfere), but the communication channel is public, shared and imposes coordination constraints. If a sample can be analyzed only by one rover, the location of this sample is the agent's private fact. Rovers is another domain suitable for relaxation heuristics, in contrast to logistics, the number of required interactions is lower, however the private plans are longer.

satellites The problems of the domain contain agents representing satellites independently taking images in space by various instruments, which have to be powered from a limited on-board power source. The state of the instruments is private to the particular satellites. Pointing directions of each satellite are private unless they appear in the goal. The domain is almost completely decomposed to agents as each satellite is practically independent, sharing only the global goal.

woodworking08 In the woodworking domain, each tool is an agent. All facts and actions in this domain are public, except for the fact stating that a high-speed saw is empty or loaded. Subsequently, loading and unloading the high-speed saw are the only public actions with private preconditions.

zenotravel The zenotravel problems contain agents representing planes with limited fuel. The goal is to transport passengers between cities and park some of the planes at designated airports. Only the planes are represented by agents. Positions of planes are private and positions of passengers are public. Fly and zoom (fast fly) actions are private. The fuel levels and the positions of passengers in cities reachable by only one plane are also private.

4.3.2. Comparison of the building blocks

The building blocks of the MADLA Planner are the projected and distributed FF heuristics and the scheme that combines these in a search. A baseline approach (as we presented in Section 3.1.2) is to adapt a classical multi-heuristic (MH) search for multi-agent planning without the non-blocking MADLA principle with a simple alternation mechanism of the open lists respective to the projected and distributed FF heuristics. However, this approach is not viable as the heuristics are not "orthogonal". The proposed MADLA Search utilizes the requirement for a non-blocking distributed heuristic estimator (particularly implemented in the form of the Privacy-Preserving Set-Additive FF heuristic) by running projected FF in the spare time, therefore in contrast to the MH search utilizing the waiting times for computation of the projected heuristic. Table 1 summarizes the coverage of the projected FF (projFF) and Privacy-Preserving Set-Additive FF (ppsaFF) heuristics in separate multi-agent single-heuristic searches, in the classical multi-heuristic search (Section 3.1.1) and in the MADLA Search (Section 3.1.3) with both heuristics.

The results clearly indicate (as expected) that the baseline multi-heuristic approach is not suitable for the pair of projected and distributed FF heuristics. The summed up coverage results are similar for the single-heuristic searches following the results in our previous work [6]. The price for better estimates of the distributed variant of FF than projected FF is its slower computation. Driverlog, woodworking08, and zenotravel are domains where projected FF is performing substantially better than the distributed variant. In such cases, where the FF heuristic is not appropriate or the overhead of its distributed computation overweights the fact the heuristic is more informative, the search using only the projected FF performs better.

The openstacks domain is the only case where even the classical multi-heuristic search outperforms both single-heuristic searches. The efficiency of the FF heuristic in the openstacks problems differs even between the projected and distributed heuristics, therefore their combination outperforms each one separately (see Section 4.3.3 for more details) even with the alternating scheme. Elevators08 and woodworking08 are domains where one of the single-heuristic searches outperformed the MADLA Search (with reasonable difference). In the former case, as the implementation of the MADLA scheme is not completely non-blocking (see Section 4.1 for details), the computation of the distributed heuristic can be slowed down by the computation of the local heuristic. In the latter case, as the FF heuristic itself is not guiding the search well, a large

Table 1
Average coverage of the building blocks. Number of problems in a domain are in the brackets, $|\mathcal{A}|$ denotes the number (interval) of agents in the problems. The best results are emphasized.

Domain	$\|\mathcal{A}\|$	projFF	ppsaFF	MH search	MADLA search
blocksworld (35)	4	32.9	**35**	15	34.1
depot (20)	5–12	10.3	10.5	7	**10.8**
driverlog (20)	2–8	**17.2**	14	13	**17.2**
elevators08 (30)	4–5	17.5	**29**	2	27.7
logistics00 (20)	3–7	**20**	**20**	3	**20**
openstacks (30)	2	13.6	14.9	15.5	**20**
rovers (20)	1–8	**20**	**20**	6.6	**20**
satellites (20)	1–5	**20**	**20**	6	19.8
woodworking08 (30)	7	**8.8**	4.8	5.6	7.5
zenotravel (20)	1–5	**19**	16.9	8	**19**
total (245)		179.3	185.1	81.7	**196.1**

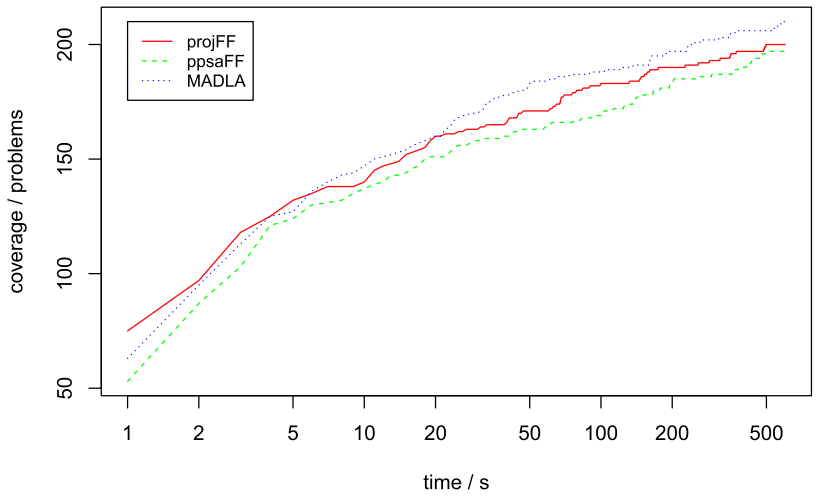

Fig. 7. Variation of coverage depending on time.

number of states needs to be evaluated. The distributed heuristic does not achieve better guidance, only slows down the whole search process and thus the projected heuristic performs slightly better on its own.

Fig. 7 shows the coverage of MA-BFS using a single heuristic compared with the MADLA Search using both. The results were obtained by first averaging the runtime over all 10 runs per problem, which results in slightly different final coverage than in Table 1 as each problem is counted as 1 even if not solved in some runs. The plot shows very interesting properties. First, it shows that the projected heuristic (projFF) solves more problems in the same time than the distributed heuristic (ppsaFF), this holds for all time limits. Even more interesting result is that the MADLA Search solves less problems than pure projFF in very short time limits (approx. less than 10 s), because the MADLA Search is slowed down by the distributed heuristic. For longer time limits, the MADLA Search solves more problems than projFF as it is able to harness the benefits of the distributed heuristic.

Table 2 lists results of the two heuristics run separately in MA-BFS and their MADLA combination evaluated using additional metrics of expanded states, plan length, bytes communicated among the agents and total planning time, all in form of the IPC score. For a particular problem, the IPC score is computed as a ratio of the optimal value (or the best value of all configurations if the optimum is not known) V^* and the particular value V, formally V^*/V. This means that the best configuration for a given problem gets 1, worse configurations get a value < 1. The number in the table is the sum of the IPC scores over all problems in all domains for each configuration. The table shows that the distributed heuristic ppsaFF expands the least states (highest score) indicating the heuristic is most informed in comparison to projFF and MADLA which both compensate for the informativeness by the speed of computation of projFF, as the total planning time shows. MADLA slightly improves over projFF in the metric of plan length as the distributed more informed heuristic can shorten some parts of the plans. Notice, that on its own, the distributed heuristic leads to longer plans, as it overestimates the true cost more often. This illustrates one of the benefits of the heuristic combination in the MADLA Search. The results of communicated bytes demonstrate the fact that the projected heuristic does not communicate at all (only the search messages are counted). On the contrary, the distributed heuristic is communication intensive and MADLA is balancing both.

The proposed MADLA Search scheme improves the overall coverage over both single-heuristic searches and doubles the coverage of the classical multi-heuristic scheme with the same heuristics. It also provides the best quality plans.

Table 2
Total sums of IPC Scores over all domains and problems for additional metrics. IPC Score is calculated per problem as V^*/V, where V^* is the best and V the particular value for a configuration.

	projFF	ppsaFF	MADLA
Expanded states	109	151	96
Plan length	171	154	184
Bytes communicated	161	49	84
Total time	169	136	160

4.3.3. Detailed analysis

In this section, we analyze the performance of presented building blocks and MADLA Search[6] in detail. A view on comparison of the multi-agent single-heuristic searches with projected FF and distributed Privacy-Preserving Set-Additive FF is presented in Fig. 8. The top graph shows the heuristic values for the initial state of all problems for which the value was computed. It is clear that for most of the domains, as the complexity of the problem grows, also the difference between the distributed and projected heuristic grows (note this does not say anything about the heuristic quality). Bottom is the number of expanded states.

Together the two plots show some interesting properties. First, the elevators08 domain is an example of a domain where the distributed heuristic gives much larger heuristic estimates, which also seems to be significantly more informed, as suggested by the number of expanded states. As the heuristic difference grows, also the difference of the number of expanded states grows in favor of the distributed heuristic. Similar behavior, only not as prominent, can be observed in the blocksworld domain. The depot domain paints a completely different picture, where the distributed heuristic also gives significantly larger estimates, but as shown in the plot of the expanded states, the heuristic guidance degrades and for larger problems, the projected heuristic is better informed for the search. The driverlog domain also fits into this category, where the larger distributed heuristic estimates do not necessarily lead the search better. On the other hand, in the woodworking08 domain, we can observe that, although the heuristic estimates are pretty much the same for both heuristics, the number of states expanded by the projected heuristic grows in comparison with the distributed FF, which suggests that even slight differences in the heuristic may have a significant impact on the heuristic quality and its ability to lead the search.

Table 3 shows a comparison of various metrics measured using the multi-agent single-heuristic search with projected FF, distributed Privacy-Preserving Set-Additive FF and MADLA Search using both. The first two columns are ratios of coverage and the number of expanded states of the single-heuristic search with projected FF and distributed FF respectively. The next two columns show the percentage of states in MADLA Search expanded using projected FF and the ratio of states expanded using projected FF and distributed FF. The next three columns show the time per state (in milliseconds) the MADLA Search spends on computing the projected and distributed heuristics and the distributed/projected ratio. The last two columns show the average percentage of public and state discerning (SD) actions (see Section 3.4.1) in the domain. A SD action is public and has some private preconditions, which are hidden for agents other than the owner of the SD actions and may cause dependences between SD actions. Ignoring such dependencies in the projected heuristic may significantly influence the quality of estimates.

Now, we analyze the results shown in the Table 3 for each of the domains in detail.

blocksworld All actions in the domain are SD actions. This results in better heuristic guidance of the distributed heuristic, best-first search expanding over 6× more states with the projected heuristic than with the distributed one (column *exp* in Table 3). In the MADLA Search, only 6% of states are expanded using distributed heuristic (column *MADLA exp*), which is, nevertheless, enough to slightly reduce the coverage of MADLA compared to MA-BFS with only the distributed heuristic.

depot The distributed heuristic takes approx. 8× longer to evaluate on average (column *MADLA* t_h right in Table 3), but its better heuristic guidance (11×more expanded states using the projected heuristic) results in almost equal coverage (Table 1). In MADLA, due to the time demanding distributed heuristic computation, about 96% of states are expanded using the projected heuristic (column *MADLA exp* left). But the small number of states expanded using the distributed heuristic improves the coverage of MADLA in comparison to the single-heuristic search using either of the heuristics.

driverlog The distributed heuristic seems to lead the search slightly better (approx. 2× less expanded states, column *exp* in Table 3) and takes approx. 1.3× more time per state. Nonetheless, the coverage of MA-BFS using the distributed heuristic is worse than using the projected one, which is likely caused by the higher chance of finding the goal with more expanded states (although less informed), especially in the harder problems (column *MADLA* t_h right, column *cvg* and Table 1). In MADLA, this is improved by approx 80% of states expanded using the projection (column *MADLA exp* left), which is enough to reach the coverage score of the projected heuristic on its own.

[6] In this section, the MADLA Search does not re-compute the local heuristic on received states. The results slightly differ from the coverage results in Section 4.3.2, but the total difference in coverage is 0.6 problems and thus negligible.

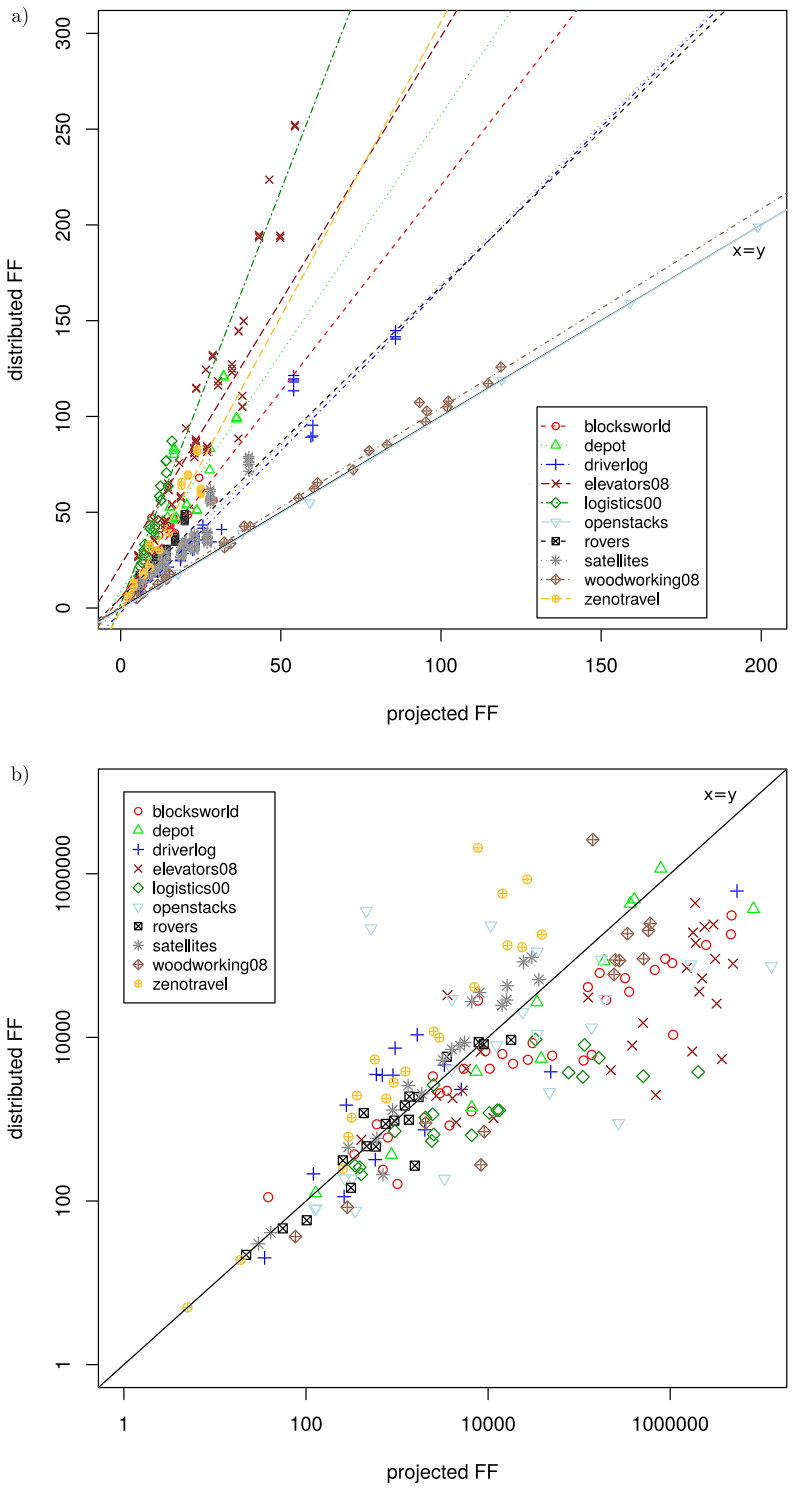

Fig. 8. Heuristic values for initial states (a) and the number of expanded states (b).

elevators08 The single-heuristic search with a projected heuristic expands over 50× more states than with the distributed one (column *exp* in Table 3) and the distributed heuristic takes on average only approx. 1.5× longer to compute (column *MADLA* t_h right). This results in a difference in problem coverage of over 10 problems in favor of the distributed heuristic (Table 1). In MADLA, even though only 6% of states are expanded using the distributed heuristic

Table 3
Comparison of various metrics. Ratios of coverage and expanded states of MA-BFS running projFF and ppsaFF, ratio of states expanded in MADLA using projFF to all expanded states, time per state in MADLA spent on evaluation of projFF and ppsaFF and their ratios and ratios of public action to all actions and state discerning (SD) actions to all actions in each domain. All values are averages per domain.

Domain	$\frac{\text{ppsaFF}}{\text{projFF}}$		MADLA exp	MADLA t_h [ms] per state			Action ratio	
	coverage	expanded	$\frac{\text{projFF}}{\text{all}}$	projFF	ppsaFF	$\frac{\text{ppsaFF}}{\text{projFF}}$	$\frac{\text{public}}{\text{all}}$ %	$\frac{\text{SD}}{\text{all}}$ %
blocksworld	0.94	6.4	0.94	0.4	0.7	1.7	100	100
depot	0.98	11.1	0.96	0.7	9.6	8.0	95.7	23
driverlog	1.23	2.4	0.80	0.9	1.1	1.3	91.9	26.7
elevators08	0.60	50.7	0.94	0.3	0.4	1.5	66	66
logistics00	1.00	47.4	0.95	0.2	0.8	6.3	67.4	33.7
openstacks	0.91	12.9	0.67	2.3	1.6	0.9	100	0
rovers	1.00	1.5	0.75	0.4	0.5	1.4	26.1	11.5
satellites	1.00	0.8	0.77	0.3	2.4	9.8	7.2	2.7
woodwork.	1.83	7.2	0.90	2.2	12.2	7.4	99.9	13
zenotravel	1.12	0.3	0.77	0.5	0.9	1.6	20.7	14.1

(column *MADLA exp* left), it is enough to reduce the performance of the single-heuristic search with the distributed heuristic.

openstacks The openstacks domain is the only case, where even the classical multi-heuristic search outperforms both single-heuristic searches. The projected heuristic is in several instances not informed enough, as the two agents need to strongly coordinate the orders. On the other hand, the distributed estimation is in the other instances computationally hard and not leading the search well (see Fig. 8-bottom). There are no state discerning actions. The favorable combination of distributed and projected heuristic (over 30% of states evaluated with the distributed one) improves significantly the coverage of MADLA.

woodworking08 In the single-heuristic search, the distributed heuristic expands approx. 7× fewer states and takes over 7× longer to evaluate per state (columns *exp* and *MADLA* t_h right in Table 3). The MA-BFS using projected heuristic solves nearly twice as many problems as MA-BFS using the distributed heuristic. This is probably caused by the dominant effect of the number of expanded states, where a higher number of expanded (albeit worse guidance) leads to a solution more often. The MADLA Search, however, is able to take advantage of projected heuristic by expanding only 10% of states using the distributed one. Combined, MADLA performs nearly as well as MA-BFS using only the projected heuristic.

zenotravel The zenotravel domain is one of a few domains where the projected heuristic actually offers better guidance resulting also in better coverage of MA-BFS using only the projected heuristic. Again, MADLA is able to take advantage of that and match the coverage.

Based on the measured values and also on the understanding of the domain, the logistics00 domain is similar to the elevators08 domain, although much easier to solve. The rovers and satellites domains are very loosely coupled domains with only a small portion of public actions and are also easy to solve.

In summary, the distributed heuristic is useful in domains with a high number of public actions with private preconditions, which is a sign of necessary interaction among the agents, not visible to the projected heuristic, or with some crucial information being private (as in woodworking). On the contrary, the projected heuristic performs better on domains where the agents are interchangeable (driverlog, zenotravel). In most cases, the MADLA Search is able to let the better heuristic dominate the search and thus on most domains, MADLA closely matches the better one of the single heuristic approaches. Notice that on some domains, MADLA even improves the coverage over each heuristic used separately (namely depot and openstacks). It is also important to note that in woodworking, both projected and distributed FF heuristics ignore a significant number of dead-ends (as described in Section 3.2.3), thus slowing the search and solving fewer problems.

4.3.4. Comparison with a centralized planner

To report on the effects of the multi-agent partitioning (sometimes referred to as factorization for its resemblance with factored planning), we run the benchmark problems on a centralized Greedy Best-First Search with the FF heuristic. In Table 4 the results are compared with the MADLA Search with the projected and distributed FF heuristics. The centralized planner is a configuration of the MADLA Planner using the same codebase but no agent factorization in order to have a fair comparison of the techniques. The original FF planner or Fast Downward (FD) would most probably perform better as they are more optimized and use additional techniques such as preferred operators. In order to account for the MADLA Planner running on 8 cores, the centralized planner was allowed an 8× longer time limit.

The results show substantial differences in the depot, driverlog, elevator08, and openstacks domains. The multi-agent search doubles the efficiency in depot and 1.5× in elevators08, as the agent subproblems are loosely coupled (the same holds for logistics00, rovers, satellites, and zenotravel). Loose coupling results in beneficial decomposition, making the agents' problems significantly smaller but with not much overhead in coordination. The strong efficiency improvement in openstacks is caused by better informed combination of heuristics and beneficial partitioning of the problem. In driverlog, the most

Table 4
Comparison of the MADLA Search with projected and distributed FF heuristics and a centralized search without multi-agent partitioning with centralized FF heuristic. Average coverage is used for MADLA and one coverage value is used for the (deterministic) centralized search.

Domain	$\|\mathcal{A}\|$	Centralized	MADLA planner
blocksworld (35)	4	34	**34.1**
depot (20)	5–12	6	**10.8**
driverlog (20)	2–8	**19**	17.2
elevators08 (30)	4–5	17	**27.7**
logistics00 (20)	3–7	**20**	**20**
openstacks (30)	2	9	**20**
rovers (20)	1–8	**20**	**20**
satellites (20)	1–5	**20**	19.8
woodworking08 (30)	7	**8**	7.5
zenotravel (20)	1–5	**19**	**19**
total (245)		172	**196.1**

plausible explanation of the coverage drop is the relaxation principle of the FF heuristic provides better estimates without the partitioning. In general, the results follow the results of similar experiments performed with MAD-A* [3].

4.3.5. Comparison with the state of the art

The MADLA Planner, as all its predecessors MAFS with rdFF and lazyFF and MA-BFS, fits in the MAD-A* family—a distributed forward-chaining heuristic search with no distinction of privately or publicly expanded states during the local search (with exception of informing the other agents), therefore MADLA follows the state-based coordination paradigm. In contrast to MAD-A*, MADLA adopts a distributively computed heuristic, similarly to FMAP and GPPP do. The use of projected actions is another similarity of MAD-A* and MADLA. Both planners do not use projections of other agents during the search, however they use such actions within the heuristic evaluation. In the case of MAD-A*, the LM-cut and Merge&Shrink use projected public actions of other agents, which holds for both the projected and distributed heuristics of MADLA too, however in the latter case all projected public actions are additionally evaluated by requesting the owning agent for additional costs caused by private supporting actions. The projected heuristic estimation is nothing more than classical computation of the heuristic on a projected planning problem of one agent. The distributed computation of the heuristic is principally the same as computation of the distributed DTG heuristic in FMAP and landmark heuristic in GPPP planners.

In contrast to ADP and GPPP, MADLA does not extract sub-goals, thus does not use them neither as a heuristic (ADP) nor in search for the public subproblem (GPPP). Coordination of the planning process in MADLA is driven by communicating public states, therefore MADLA does not need to adjust (MAPR, PMR), validate (PSMM) or amend (FMAP) partial plans, or partially ordered plans. As in MAD-A*, MADLA represents prefixes of the plans for all reached states in the form of references to parent search nodes. MADLA's distributed heuristic computation can be seen as a process using plan-based coordination on the relaxed planning problem. It uses projected actions of other agents, merges partial results of other agents which are requested if the agent needs them and it uses distributed extraction of the plan in the Privacy-preserving Set-Additive FF (ppsaFF) heuristic.

Distributed computation of the ppsaFF heuristic results in an approximation of a relaxed plan to the goal from the current state. The principle MADLA is using is similar to PSMM and Planning First planners, but in a backtracking fashion: the PSMM process can be viewed as one central initiator agent, which generates public plans (in a compact form of the Finite State Machines), asks the other agents if they can fill in the required supporting private actions (gaps) caused by the projection of their actions. In the non-relaxed case, this process requires backtracking, whereas in the relaxed case, the process is monotonous and can be easily done in the backward manner from the goals. In the Planning First case the DisCSP represents the coordination problem which is simultaneously solved by the agents, however the used unary Internal Planning Constraint [2] represents planning of the local private plans of the agents. Therefore, similarly to PSMM and the distributed FF, the planner fills the gaps locally by the respective agents. As a matter of fact, MADLA's distributed process to evaluate the FF heuristic could be used as another plan-based coordination planning approach. In the non-monotonic case, however, it would have to search, i.e., adopt distributed backtracking. A naive implementation would need synchronized stacks representing the distributed state of the recursion and would require substantial communication among the agents.

Using the state-based heuristic is troublesome in the coordination-centric approaches as such planners usually do not have a complete state the heuristic could evaluate. Therefore, straightforward adoption of the MADLA distributed heuristic is not possible, however the heuristic can be used to improve efficiency of the public planning sub-problem (e.g., PSMM proposes to use distributed relaxed heuristic in initial state) or can be used if the partial plan can generate a frontier state as the local search in FMAP. Moreover, hybrid approaches such as FMAP and GPPP, which work with full states as subgoals, could use MADLA heuristics to evaluate the subgoal states if needed, which is a similar principle as proposed by GPPP, but using the landmark heuristic.

In Table 5, we show a comparison of problems solved by MADLA and four complete and distributed multi-agent planners. The results show that MADLA loses considerably in woodworking08 and openstacks against all planners supporting action

Table 5
Comparison MADLA and state-of-the-art planners. †In GPPP experiments a version of the domain without action costs was used, consisting of 16 problems. ‡GPPP does not support action costs.

Domain	$\lvert\mathcal{A}\rvert$	rdFF	GPPP	PSMM	FMAP	MADLA
blocksworld (35)	4	6.8	3	25	19	**34.1**
depot (20)	5–12	6.2	8	0	6	**10.8**
driverlog (20)	2–8	14	9	13	15	**17.2**
elevators08 (30)	4–5	2.9	16†	4	**30**	27.7
logistics00 (20)	3–7	5.8	**20**	9	10	**20**
openstacks (30)	2	11.7	0‡	**30**	23	20
rovers (20)	1–8	14.7	10	14	19	**20**
satellites (20)	1–5	10.8	16	8	16	**19.8**
woodworking08 (30)	7	5.6	0‡	**25**	22	7.5
zenotravel (20)	1–5	6.1	**20**	17	18	19
total (245)		84.6	102	145	178	**196.1**

costs. These domains contain substantial number of dead-ends of which the FF heuristic (especially in the projected form) is oblivious.

We argue that MADLA is not fairly comparable to (a) planners which do not consider multi-agent privacy (ADP), (b) planners incompatible with MA-STRIPS (μ-SATPLAN, BRP, Distoplan, A#, TFPOP), or (c) optimal planners (MAD-A*). Additionally, we present comparison only with the most efficient planners using a particular paradigm. We do not present detailed comparisons with ADP planner, however on the benchmark set present, ADP outperforms MADLA by more than 28% solved problems. By definition ADP cannot preserve privacy in the same sense as MADLA in general, as it does not obey the definition of the agents by which MA-STRIPS defines the privacy. Moreover MADLA has to use one partitioning of the planning problem defined in the input PDDL and ADDL, but ADP targets classical planning benchmarks and is free to partition the problem as it sees fit.

Although the result table does not contain the PMR planner, MADLA outperforms it on the presented benchmark set as well, just because PMR is an incomplete planner as stated in [19,34]. PMR solves only problems where each goal fact is solvable by a single agent. Thus it does not solve problems of depot, logistics00, openstacks, and woodworking08 domains. Even if PMR solved all problems of all other domains, MADLA would outperform it by 38%.

Against MAFS running rdFF [6], our recent multi-agent heuristic search using different distribution schemes and implementations of FF, MADLA shows more than 2× improvement over all domains with an exception of driverlog and woodworking08, where the improvement is about 20%. Similarly, MADLA outperforms GPPP nearly 2× over many domains and PSMM by 36%. Finally, MADLA solves 16 more problems of the benchmark set in contrast to the top performing multi-agent planner FMAP, which correspond to nearly 11% improvement. In comparison to FMAP, MADLA is considerably better on four domains (blocksworld, depot, logistics, satellites) and significantly worse only on two (elevators08, woodworking08).

The MADLA Planner also took part in the recent Competition of Distributed and Multi-Agent Planners (CoDMAP). See the on-line results[7] or [44] for more details. Note, that although the MADLA Planner itself was not significantly modified for the competition, one of the best performing planners, MAPlan [45] is built on the same principles. In particular, it uses distributed heuristic search with the privacy-preserving set-additive FF heuristic (Section 3.2.3), but without the MADLA Search (Section 3.1.3).

5. Conclusion

As for classical planning, heuristic state-space search is a viable technique for multi-agent planning. However, in contrast to the classical heuristic search, the multi-agent setup raises its own challenges. The dilemma of highly informed but slow versus less informed but fast heuristic estimators is manifested in the dichotomy of projected heuristic restricted to the agent's local view of the problem versus distributed heuristic estimating the global heuristic value at the cost of significant communicational or computational overhead. The MADLA Planner combines both fast local projection of the FF heuristic with a global distributed FF heuristic in an attempt to combine their benefits and mitigate their negative effects.

There are two contributions of this paper. The main contribution is a novel distributed multi-heuristic multi-agent search scheme and its practical implementation. A minor contribution is an improved approach to computing privacy-preserving distributed FF heuristic, while reducing the negative effect of overcounting.

The novel technique used in the MADLA Planner to combine the heuristic estimators is based on the classical multi-heuristic search, but it does not evaluate all states by both heuristics. Instead, the local heuristic is used only while the distributed heuristic is busy (that is, waiting for replies from other agents). The projected heuristic is used to fully utilize the computational resources of the agent, even for less-informed, but faster search of the state space of the individual agent. When the estimation of the more-informed heuristic is finished, the evaluated state is used in both searches. This principle

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

was theoretically analyzed and practically evaluated in the form of a MA-STRIPS based planner, which on many domains outperforms all current multi-agent planners in the coverage metric over a common benchmark set.

Some research directions are left for future work. First, whether the principle of MADLA Search can be used for optimal multi-agent planning with similarly promising results. Second, as the results show, the planner does not perform well on domains with dead-ends, which relaxation heuristics are oblivious to. We can ask what efficiency boost could be achieved by the combination not only of one heuristic as projected and distributed estimators, but also with other heuristics possibly orthogonal to the first heuristic pair; landmarks are an obvious choice. Moreover, the open list selection scheme is by no means perfect and could be possibly improved, e.g., by occasionally evaluating the best state from O^i_{L} by h^i_{L} even if O^i_{D} is not empty. Also, the use of preferred operators might significantly improve the overall performance.

Generally, utilizing the principle of combining fast and less accurate and slow but more informed heuristics in an asynchronous manner may be also an interesting research direction in classical planning (especially on multicore machines) and in search in general.

Acknowledgements

This research was supported by the Czech Science Foundation (grants no. 13-22125S and 15-20433Y), by the Air Force Office of Scientific Research, USAF (grant USAF (grant no. FA8655-12-1-2096), and by the Grant Agency of the CTU in Prague (grant no. SGS14/202/OHK3/3T/13). Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the program "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

References

[1] R. Fikes, N. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, in: Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI'71), 1971, pp. 608–620.
[2] R.I. Brafman, C. Domshlak, From one to many: planning for loosely coupled multi-agent systems, in: Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08), 2008, pp. 28–35.
[3] R. Nissim, R. Brafman, Distributed heuristic forward search for multi-agent planning, J. Artif. Intell. Res. 51 (2014) 293–332.
[4] D. Nau, M. Ghallab, P. Traverso, Automated Planning: Theory & Practice, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
[5] R. Nissim, R.I. Brafman, Multi-agent A* for parallel and distributed systems, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12), 2012, pp. 1265–1266.
[6] M. Štolba, A. Komenda, Relaxation heuristics for multiagent planning, in: Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14), 2014, pp. 298–306.
[7] J. Hoffmann, B. Nebel, The FF planning system: fast plan generation through heuristic search, J. Artif. Intell. Res. 14 (2001) 253–302, http://dx.doi.org/10.1613/jair.855.
[8] M. Štolba, A. Komenda, Fast-forward heuristic for multiagent planning, in: Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13), 2013, pp. 75–83.
[9] D. Pellier, Distributed planning through graph merging, in: Proceedings of the 2nd International Conference on Agents and Artificial Intelligence (ICAART 2010), 2010, pp. 128–134.
[10] M. Helmert, The fast downward planning system, J. Artif. Intell. Res. 26 (2006) 191–246.
[11] G. Röger, M. Helmert, The more, the merrier: combining heuristic estimators for satisficing planning, in: Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10), 2010, pp. 246–249.
[12] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL – the Planning Domain Definition Language, Tech. Rep. TR-98-003, Yale Center for Computational Vision and Control, 1998.
[13] D.L. Kovacs, A multi-agent extension of PDDL3.1, in: Proceedings of the 3rd Workshop on the International Planning Competition (IPC), 2012, pp. 19–27.
[14] S. Conry, K. Kuwabara, V. Lesser, R.A. Meyer, Multistage negotiation for distributed constraint satisfaction, IEEE Trans. Syst. Man Cybern. 21 (6) (1991) 1462–1477.
[15] M. Yokoo, K. Suzuki, K. Hirayama, Secure distributed constraint satisfaction: reaching agreement without revealing private information, in: P. Van Hentenryck (Ed.), Principles and Practice of Constraint Programming – CP 2002, in: Lecture Notes in Computer Science, vol. 2470, Springer, Berlin/Heidelberg, 2002, pp. 387–401.
[16] A. Yao, How to generate and exchange secrets, in: 27th Annual Symposium on Foundations of Computer Science, 1986, IEEE, 1986, pp. 162–167.
[17] D. Gupta, A. Segal, A. Panda, G. Segev, M. Schapira, J. Feigenbaum, J. Rexford, S. Shenker, A new approach to interdomain routing based on secure multi-party computation, in: Proceedings of the 11th ACM Workshop on Hot Topics in Networks, ACM, 2012, pp. 37–42.
[18] S. Maliah, G. Shani, R. Stern, Privacy preserving landmark detection, in: Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14), 2014, pp. 597–602.
[19] D. Borrajo, Plan sharing for multi-agent planning, in: Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13), 2013, pp. 57–65.
[20] S. Richter, M. Westphal, The LAMA planner: guiding cost-based anytime planning with landmarks, J. Artif. Intell. Res. 39 (1) (2010) 127–177.
[21] K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Trans. Comput. Syst. 3 (1) (1985) 63–75.
[22] T. Bylander, The computational complexity of propositional STRIPS planning, Artif. Intell. 69 (1–2) (1994) 165–204.
[23] A. Torreño, E. Onaindia, O. Sapena, FMAP: distributed cooperative multi-agent planning, Appl. Intell. 41 (2) (2014) 606–626, http://dx.doi.org/10.1007/s10489-014-0540-2.
[24] E. Keyder, H. Geffner, Heuristics for planning with action costs revisited, in: Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08), 2008, pp. 588–592.
[25] M. Crosby, M. Rovatsos, R. Petrick, Automated agent decomposition for classical planning, in: Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13), 2013, pp. 46–54.
[26] J. Tozicka, J. Jakubuv, A. Komenda, Generating multi-agent plans by distributed intersection of finite state machines, in: Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14), 2014, pp. 1111–1112.
[27] R. Nissim, R.I. Brafman, C. Domshlak, A general, fully distributed multi-agent planning algorithm, in: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10), 2010, pp. 1323–1330.

[28] A. Coles, M. Fox, D. Long, A. Smith, Teaching forward-chaining planning with JavaFF, in: Colloquium on AI Education, Twenty-Third AAAI Conference on Artificial Intelligence, 2008.
[29] K. Durkota, A. Komenda, Deterministic multiagent planning techniques: experimental comparison (short paper), in: Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13), 2013, pp. 43–47.
[30] Y. Dimopoulos, M.A. Hashmi, P. Moraitis, Extending SATPLAN to multiple agents, in: Proceedings of the 30th International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI'10), 2010, pp. 137–150.
[31] A. Jonsson, M. Rovatsos, Scaling up multiagent planning: a best-response approach, in: Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11), 2011, pp. 114–121.
[32] E. Fabre, L. Jezequel, P. Haslum, S. Thiébaux, Cost-optimal factored planning: promises and pitfalls, in: Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10), 2010, pp. 65–72.
[33] A. Gerevini, I. Serina, LPG: a planner based on local search for planning graphs with action costs, in: Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS'02), 2002, pp. 13–22.
[34] N. Luis, D. Borrajo, Plan merging by reuse for multi-agent planning, in: Proceedings of the 2nd ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'14), 2014, pp. 38–44.
[35] M. Fox, A. Gerevini, D. Long, I. Serina, Plan stability: replanning versus plan repair, in: Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06), 2006, pp. 212–221.
[36] M. Helmert, C. Domshlak, Landmarks, critical paths and abstractions: what's the difference anyway?, in: Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09), 2009, pp. 162–169.
[37] M. Helmert, P. Haslum, J. Hoffmann, Flexible abstraction heuristics for optimal sequential planning, in: Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07), 2007, pp. 176–183.
[38] L. Jezequel, E. Fabre, A#: a distributed version of A* for factored planning, in: Proceedings of the 51st IEEE Conference on Decision and Control (CDC'12), 2012, pp. 7377–7382.
[39] J. Kvarnström, Planning for loosely coupled agents using partial order forward-chaining, in: Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11), 2011, pp. 138–145.
[40] A. Torreño, E. Onaindia, O. Sapena, An approach to multi-agent planning with incomplete information, in: Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12), 2012, pp. 762–767.
[41] A. Torreño, E. Onaindia, O. Sapena, FMAP: a heuristic approach to cooperative multi-agent planning, in: Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13), 2013, pp. 84–92.
[42] J. Benton, A.J. Coles, A. Coles, Temporal planning with preferences and time-dependent continuous costs, in: L. McCluskey, B. Williams, J.R. Silva, B. Bonet (Eds.), Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25–19, 2012, AAAI, 2012.
[43] International planning competition, http://ipc.icaps-conference.org/, 2014.
[44] A. Komenda, M. Stolba, D.L. Kovacs, The international competition of distributed and multiagent planners (codmap), AI Mag. 37 (3) (2016) 109–115.
[45] D. Fišer, M. Štolba, A. Komenda MAPlan, in: Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15), 2015, pp. 8–10.