Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling

Global Heuristics for Distributed Cooperative Multi-Agent Planning

Alejandro Torreño, Óscar Sapena, Eva Onaindia
Universitat Politècnica de València
Camino de Vera s/n
46022 Valencia, SPAIN

Abstract

Almost every planner needs good heuristics to be efficient. Heuristic planning has experienced an impressive progress over the last years thanks to the emergence of more and more powerful estimators. However, this progress has not been translated to multi-agent planning (MAP) due to the difficulty of applying classical heuristics in distributed environments. The application of local search heuristics in each agent has been the most widely adopted approach in MAP but there exist some recent attempts to use global heuristics. In this paper we show that the success of global heuristics in MAP depends on a proper selection of heuristics for a distributed environment as well as on their adequate combination.

Introduction

Cooperative Multi-Agent Planning (MAP) extends classical planning by introducing a set of individual entities or agents that plan together in a shared deterministic environment to solve a common set of goals. Agents in cooperative MAP address two basic tasks, synthesize individual plans and coordinate them to build a joint plan that solves the MAP task.

The various existing MAP approaches can be classified according to the planning and coordination models they use. Some approaches perform a pre-planning distribution of the MAP task. MAPR (Borrajo 2013) allocates the task goals to the participating agents, which in turn individually invoke LAMA (Richter and Westphal 2010) to solve their assigned subtasks. The work in (Crosby, Rovatsos, and Petrick 2013) automatically decomposes single-agent tasks into MAP problems, which are then locally solved through a centralized heuristic planner.

Other MAP techniques put the focus on plan merging. Planning First (Nissim, Brafman, and Domshlak 2010) is one of the first planners based on MA-STRIPS (Brafman and Domshlak 2008), a minimalistic multi-agent extension of the STRIPS model. Agents in Planning First individually synthesize plans through a state-based planner. The resulting local plans are then coordinated through a distributed Constraint Satisfaction Problem.

A third group of approaches directly apply multi-agent search, interleaving planning and coordination. MA-A* (Nissim and Brafman 2012) is also a MA-STRIPS-based approach that performs a distributed A* search, guiding the procedure through admissible local heuristic functions. The work in (Bonisoli et al. 2014) formulates a privacy-preserving MAP model by adapting MA-A*.

Most of the aforementioned MAP approaches resort to heuristic search at some point during the planning process, applying local heuristic search to each participating agent. Since agents usually have a limited knowledge of the task, the quality of local estimates diminish in comparison to the global heuristics applied in single-agent planning tasks.

A *global heuristic* in MAP is the application of a heuristic estimate to the MAP task carried out by several agents which have a different knowledge of the task and, possibly, privacy requirements. The design of global estimators for cooperative MAP is a challenging task (Nissim and Brafman 2012) which has been seldom studied. Exceptions are the work in (Štolba and Komenda 2014), which introduces a distributed version of some well-known relaxation-based heuristics, and the application of a landmark-based global heuristic in the GPP planner (Maliah, Shani, and Stern 2014).

The focus of the present work is to analyze the benefits of global heuristics in MAP and to study how the combination of these functions can noticeably improve the efficiency of cooperative MAP systems. For our purposes, we take FMAP as our framework (Torreño, Onaindia, and Sapena 2014). FMAP is a fully-distributed forward-chaining multi-agent POP approach that preserves agents' privacy. Specifically, this paper presents the following contributions:

- Formalization of two distributed heuristic functions: h_{DTG} (Torreño, Onaindia, and Sapena 2014), a variation of the Context-Enhanced Additive heuristic (Helmert and Geffner 2008) based on Domain Transition Graphs (Helmert 2004); and h_{Land}, a privacy-preserving version of the landmark extraction algorithm introduced in (Hoffmann, Porteous, and Sebastia 2004).
- MH-FMAP, a novel multi-heuristic MAP approach that combines h_{DTG} and h_{Land} orthogonally, notably improving the performance of FMAP.

This paper is organized as follows: after presenting some related work and the key notions of FMAP, we introduce the formalization of h_{DTG}, the design of h_{Land} and the combination of both heuristics into MH-FMAP. The experimen-

tal results evaluate the two heuristics and the multi-heuristic approach on various domains adapted from the International Planning Competition[1] (IPC) to a multi-agent context and compares the results with the ones obtained with GPPP.

Related Work

Many of the existing MAP frameworks apply some form of heuristic search to guide the planning process. The use of global heuristics in MAP is, however, less frequent due to the inherent features of MAP scenarios, which introduce additional requirements and make it an arduous task:

- The data of a MAP task are usually distributed across the agents; unlike single-agent planning, in MAP it does not exist an entity that centralizes the information of the task. Hence, a communication protocol among the agents is required to compute global heuristic estimates.
- Most MAP models deal with agents' privacy. The communication protocol must thus guarantee that agents are able to calculate heuristic estimates without revealing sensitive private information.

In some works, the features of the planning model force the application of a local heuristic search scheme, in which an agent calculates the heuristic value of a plan based on its local information. In (Borrajo 2013), goals are allocated to the agents, which then solve their problems iteratively, communicating the solution of an agent to the next agent. Thus, the heuristic functions defined in LAMA, namely h_{FF} (Hoffmann and Nebel 2001) and h_{Land} (Richter and Westphal 2010), are applied from a local standpoint.

Local search heuristics have also been used in other MAP approaches, even though their planning model is suitable to accommodate distributed functions. The work in (Nissim and Brafman 2012) presents MA-A*, a multi-agent design of the well-known A* algorithm. Authors test different configurations of the planner with two optimal heuristic functions, Merge&Shrink (Helmert, Haslum, and Hoffmann 2007) and LM-Cut (Helmert and Domshlak 2009). These functions are however applied locally by each agent.

Authors in (Štolba and Komenda 2013) introduce a multi-agent design of the h_{FF} heuristic. This adaptation, based on the use of *distributed Relaxed Planning Graphs* (dis-RPGs) (Zhang, Nguyen, and Kowalczyk 2007), yields the same results as the original single-agent design of h_{FF} (Hoffmann and Nebel 2001). However, the construction and exploration of a dis-RPG entails many communications between agents, resulting in a computationally expensive approach.

In (Štolba and Komenda 2014), authors present the distributed design of several relaxation heuristics, namely h_{add}, h_{max} and a relaxed version of h_{FF}. In this work, authors replace the dis-RPG by an *exploration queue*, a more compact structure that significantly reduces the need of communications among agents. The distributed version of h_{FF}, however, does not yield the same results as the original single-agent version.

Finally, in (Maliah, Shani, and Stern 2014), authors design a distributed version of a privacy-preserving landmarks extraction algorithm for MAP, resulting in a planner named GPPP. Authors show that the Landmarks Graph used in GPPP improves the performance of the MA-STRIPS-based planner MAFS (Nissim and Brafman 2014). In GPPP, the heuristic value of the plan is calculated as the sum of the local heuristic estimates computed by each agent.

[1] http://ipc.icaps-conference.org

Multi-Agent Planning Task Formalization

In this section we present the formalization of a MAP task as used in the FMAP framework (Torreño, Onaindia, and Sapena 2014). Agents have a limited knowledge of the planning task, and it is assumed that the information that is not represented in the agent's model is unknown to the agent. The states of the world are defined through a finite set of *state variables*, $\mathcal{V}$, each of which is associated to a finite domain, $\mathcal{D}_v$, of mutually exclusive values that refer to the objects in the world. Assigning a value d to a variable $v \in \mathcal{V}$ generates a *fluent*, a tuple of the form $\langle v, d\rangle$. A *state* S is defined as a finite set of fluents.

An *action* is of the form $\alpha = PRE(\alpha) \rightarrow EFF(\alpha)$, where $PRE(\alpha)$ and $EFF(\alpha)$ are finite set of fluents representing the preconditions and effects of α, respectively. Executing an action α in a world state S leads to a new world state S' as a result of applying $EFF(\alpha)$ over S. An effect of the form $\langle v, d\rangle$ updates S' w.r.t. S, replacing the fluent $\langle v, d'\rangle \in S$ by $\langle v, d\rangle$. Since values in $\mathcal{D}_v$ are mutually exclusive, the inclusion of $\langle v, d\rangle$ in S' implies that $\forall d' \in \mathcal{D}_v, d' \neq d, \langle v, d'\rangle \notin S'$.

Definition 1 *A **MAP task** is a tuple $\mathcal{T}_{MAP} = \langle \mathcal{AG}, \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{A}\rangle$. $\mathcal{AG} = \{1, \ldots, n\}$ is a finite non-empty set of agents. $\mathcal{V} = \bigcup_{i \in \mathcal{AG}} \mathcal{V}^i$, where $\mathcal{V}^i$ is the set of state variables known to an agent i. $\mathcal{I} = \bigcup_{i \in \mathcal{AG}} \mathcal{I}^i$ is a set of fluents that defines the initial state of $\mathcal{T}_{MAP}$. Since specialized agents are allowed, they may only know a subset of $\mathcal{I}$; the initial states of two agents never contradict each other. $\mathcal{G}$ is a set of fluents defining the goals of $\mathcal{T}_{MAP}$. Finally, $\mathcal{A} = \bigcup_{i \in \mathcal{AG}} \mathcal{A}^i$ is the set of planning actions of the agents. $\mathcal{A}^i$ and $\mathcal{A}^j$ of two specialized agents i and j will be typically disjoint sets; otherwise, $\mathcal{A}^i$ and $\mathcal{A}^j$ may overlap. $\mathcal{A}$ includes two fictitious actions α_0 and α_f that do not belong to any particular agent: α_0 represents the initial state of $\mathcal{T}_{MAP}$, while α_f represents the goal state.*

The *view* of an agent i on $\mathcal{T}_{MAP}$ is defined as $\mathcal{T}^i_{MAP} = \langle \mathcal{V}^i, \mathcal{A}^i, \mathcal{I}^i, \mathcal{G}\rangle$. $\mathcal{V}^i$ is the set of state variables known to agent i; $\mathcal{A}^i \subseteq \mathcal{A}$ is the set of its capabilities (planning actions); $\mathcal{I}^i$ is the subset of fluents of the initial state $\mathcal{I}$ that are known to agent i, and $\mathcal{G}$ is the set of goals, which are known to all the agents in $\mathcal{T}_{MAP}$. An agent i may also have a partial view on the domain $\mathcal{D}_v$ of a variable v. We define $\mathcal{D}^i_v \subseteq \mathcal{D}_v$ as the subset of values of v known to agent i.

Agents interact by sharing information about their state variables. For a pair of agents i and j, the information they share is defined as $\mathcal{V}^{ij} = \mathcal{V}^{ji} = \mathcal{V}^i \cap \mathcal{V}^j$. Additionally, the set of values of a variable v shared by agents i and j is defined as $\mathcal{D}^{ij}_v = \mathcal{D}^i_v \cap \mathcal{D}^j_v$.

FMAP follows a forward-chaining POP approach which has been adapted to a multi-agent context.

Definition 2 *A **partial-order plan** or partial plan is a tuple* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$. $\Delta = \{\alpha | \alpha \in \mathcal{A}\}$ *is the set of actions in* Π. $\mathcal{OR}$ *is a finite set of ordering constraints* ($\prec$) *on* Δ. $\mathcal{CL}$ *is a finite set of causal links of the form* $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$, *where* α *and* β *are actions in* Δ. *A causal link* $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ *enforces precondition* $\langle v, d \rangle \in PRE(\beta)$ *through an effect* $\langle v, d \rangle \in EFF(\alpha)$.

An *empty* partial plan is defined as $\Pi_0 = \langle \Delta_0, \mathcal{OR}_0, \mathcal{CL}_0 \rangle$, where $\mathcal{OR}_0$ and $\mathcal{CL}_0$ are empty sets, and Δ_0 contains only the fictitious initial action α_0.

The introduction of new actions in a partial plan may trigger the appearance of *flaws*: preconditions that are not yet supported in the plan, and threats. A *threat* over a causal link $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ is caused by an action γ not ordered w.r.t. α or β, where $(v = d') \in EFF(\gamma)$, $d' \neq d$. A *flaw-free* plan is a threat-free partial plan without unsupported preconditions.

Agents in FMAP jointly refine an initially empty plan until a solution is reached. We define a refinement plan as follows:

Definition 3 *A **refinement plan** $\Pi_r = \langle \Delta_r, \mathcal{OR}_r, \mathcal{CL}_r \rangle$ over a partial plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ is a flaw-free partial plan that extends Π by introducing an action α, resulting in $\Delta_r = \Delta \cup \alpha$. All the preconditions in $PRE(\alpha)$ are supported by existing actions in Π through causal links: $\forall p \in PRE(\alpha), \exists\, \beta \overset{p}{\rightarrow} \alpha \in \mathcal{CL}_r$, where $\beta \in \Delta$.*

For each refinement plan, FMAP computes the *frontier state* (Benton, Coles, and Coles 2012), that is, the state that results from executing the actions in the plan. Frontier states allow for the application of state-based heuristic functions.

Definition 4 *A **frontier state** $FS(\Pi)$ over a refinement plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ is the set of fluents $\langle v, d \rangle$ achieved by actions $\alpha \in \Delta \mid \langle v, d \rangle \in EFF(\alpha)$, such that any action $\alpha' \in \Delta$ that modifies the value of the variable v ($\langle v, d' \rangle \in EFF(\alpha') \mid d \neq d'$) is not applicable from α by following the orderings and causal links in Π.*

A *solution plan* for $\mathcal{T}_{MAP}$ is a refinement plan that achieves all the goals $\mathcal{G}$ of $\mathcal{T}_{MAP}$ by including the fictitious final action α_f and supporting all its preconditions, i.e., $\forall g \in PRE(\alpha_f), \exists\, \beta \overset{g}{\rightarrow} \alpha_f \in \mathcal{CL}, \beta \in \Delta$.

Privacy in partial plans Agents in FMAP carry out several distributed procedures that require communications. To keep privacy, only the information that is shared between the sender and receiver agents is transmitted. To do so, the sender *encodes* the information that is not in the view of the receiver agent. Each variable and value has an associated unique global identifier, a positive integer that is used to mask the original variable or value when necessary.

When an agent i refines a plan Π by adding an action $\alpha \in \mathcal{A}^i$, it communicates such refinement to the rest of agents. To preserve privacy, agent i will only communicate to agent j the fluents in α whose variables are common to both agents. The information of Π that agent j receives from i configures its view of that plan. More specifically, given a fluent $\langle v, d \rangle$, where $v \in \mathcal{V}^i$ and $d \in \mathcal{D}_v^i$, FMAP identifies three cases:

- **Public fluent**: if $v \in \mathcal{V}^{ij}$ and $d \in \mathcal{D}_v^{ij}$, the fluent $\langle v, d \rangle$ is public to both agents, and thus agent i will share with agent j all the information regarding $\langle v, d \rangle$.
- **Private fluent to agent** i: if $v \notin \mathcal{V}^{ij}$, $\langle v, d \rangle$ is private to agent i w.r.t. j, and hence agent i will send j $\langle gid(v), gid(d) \rangle$, thus replacing v and d by their global identifiers, $gid(v)$ and $gid(d)$, respectively.
- **Partially private fluent to agent** i: if $v \in \mathcal{V}^{ij}$ but $d \notin \mathcal{D}_v^{ij}$, $\langle v, d \rangle$ is partially private to agent i w.r.t. j. Instead of $\langle v, d \rangle$, agent i will send j a fluent $\langle v, gid(d) \rangle$, thus replacing the value d by its global identifier $gid(d)$.

As well as keeping privacy during planning, encoding variables and values eases the design of global heuristic functions and streamlines communications among agents.

FMAP: Multi-Agent Planning Framework

FMAP is a fully-configurable distributed search procedure, an appropriate testbed for the integration of global heuristic functions. This section summarizes some of the key aspects of this MAP framework.

Algorithm 1: FMAP search algorithm for an agent i

```
openList ← Π_0
while openList ≠ ∅ do
    Π_b ← extractPlan(openList)
    if isSolution(Π_b) then
        return Π_b
    RP ← refinePlan(Π_b)
    for all j ∈ AG, j ≠ i do
        sendRefinements(j)
        RP ← RP ∪ receiveRefinements(j)
    for all Π_r ∈ RP do
        distributedEvaluation(Π_r, heuristic)
        openList ← openList ∪ Π_r
return No solution
```

FMAP is a cooperative refinement planning procedure in which agents jointly explore a multi-agent, plan-space search tree (see Algorithm 1). Nodes of the tree are partial plans contributed by one or several agents. The process is led by an agent that plays the *coordinator* role (this role is rotated after each iteration of the procedure).

Agents keep a common $openList$ with the unexplored refinement plans prioritized according to a search criterion (by default, FMAP applies a weighted A* search, evaluating nodes through a function $f = g + 2 * h$). Agents jointly choose the best node of $openList$ and then each of them individually expands the selected plan through an embedded forward-chaining POP procedure, generating all the possible refinement plans. Afterwards, agents exchange the plans and apply a distributed heuristic evaluation of such plans, which are then inserted in the $openList$. The procedure ends up when a solution plan is found, or when $openList$ is empty.

As in most distributed frameworks, communications play a central role in FMAP. The system is built on top of the Ma-

gentix2[2] MAS platform, which provides the basic libraries to define the agents' behavior, as well as the communication infrastructure required by FMAP. Agents communicate by means of the FIPA Agent Communication Language (O'Brien and Nicol 1998), and communications are managed by the Apache QPid[3] message broker.

The communication broker acts as a post office, receiving the messages from the sender agents and forwarding them to the receivers. The use of a messaging broker offers some key advantages for the design of distributed systems since it allows agents to be launched in different machines, as long as the broker is accessible from the network. However, when the workload of messages is relatively high, the broker entails a bottleneck of the system. For this reason, the global estimators introduced in this paper have been designed and optimized to minimize the communications among agents.

Global Heuristic Functions

This section formalizes and details the distributed design of two different heuristic functions as well as a novel multi-heuristic approach to MAP that combines both functions, noticeably improving the performance of the FMAP system.

The first heuristic, h_{DTG}, is a variation of the Context-Enhanced Additive Heuristic (Helmert and Geffner 2008) that uses Domain Transition Graphs (DTGs) to estimate the cost of the state variables. The second one, h_{Land}, computes the Landmarks Graph (LG) of a MAP task, which is later used to calculate the number of landmarks of the partial plans. We designed a distributed version of the landmarks extraction algorithm introduced in (Hoffmann, Porteous, and Sebastia 2004).

The design of h_{DTG} and h_{Land} in FMAP aims to keep the number of messages exchanged among the agents as low as possible. Prior to the search process, we build data structures, like the DTGs or the LG, which remain immutable throughout the multi-agent search, thus reducing the communication overload during search. In contrast to other constructs, such as dis-RPGs (Zhang, Nguyen, and Kowalczyk 2007), the DTGs and the LG do not need to be re-calculated during search. The use of static structures makes h_{DTG} and h_{Land} be more suitable heuristics for fully-distributed systems than other well-known heuristic functions, such as h_{FF} (Štolba and Komenda 2013), that requires the generation of a dis-RPG at each search node.

Besides h_{DTG} and h_{Land}, we also introduce MH-FMAP, a multi-heuristic adaptation of the FMAP algorithm that alternates both heuristics, successfully improving the overall performance of the MAP system.

DTG heuristic This is a state-based additive heuristic calculated from the DTGs (Helmert 2004). A DTG is a graph in which nodes represent values of a particular variable, and transitions show the changes in the values of such variable through the actions of the agents. An action of the form $\langle v, d_0\rangle \rightarrow \langle v, d_n\rangle$ induces a transition $d_0 \rightarrow d_n$ in the DTG associated to v.

[2]http://www.gti-ia.upv.es/sma/tools/magentix2

[3]http://qpid.apache.org

Similarly to the Context-Enhanced Additive heuristic (h_{CEA}) (Helmert and Geffner 2008), h_{DTG} builds a relaxed plan and reuses the side effects of the actions in the relaxed plan as a basis to estimate the cost of the subsequent subgoals. A plan Π of FMAP is always evaluated from its frontier state, $FS(\Pi)$, but the cost of some of the subgoals can be estimated in a state different from $FS(\Pi)$.

Formally, the formulation of h_{DTG} is very close to h_{CEA}. Given a subgoal $g = \langle v, d\rangle$, a state S and an action $\alpha \in \mathcal{A}$, where $g \in EFF(\alpha)$, $g' = \langle v, d'\rangle \in PRE(\alpha)$, $d' \neq d$, and $z = PRE(\alpha) \setminus \{g'\}$, evaluating g in S with h_{DTG} is recursively defined as follows:

$$h_{DTG}(g|S) = \begin{cases} 0 \text{ if } g \in S \\ \min\limits_{\alpha:(g',z\rightarrow g)\in\mathcal{A}} (1 + h_{DTG}(g'|S) + \\ \sum\limits_{x\in z} h_{DTG}(x|S')) \text{ otherwise} \end{cases} \quad (1)$$

The recursive equation 1 expresses that the precondition g', related to the same variable v as the fluent g, is also evaluated in S, whereas the rest of preconditions, $x = \langle v', d'\rangle \in PRE(\alpha)$, $v' \neq v$ can be evaluated in a state S' different from S.

Following, we describe in detail the h_{DTG} algorithm to clarify aspects such as the evaluation of a subgoal g, the selection and insertion in the relaxed plan of the action α that minimizes equation 1 or the selection of the states S' from which the preconditions x of equation 1 are evaluated.

Instead of exploring the Causal Graph as h_{CEA} does, h_{DTG} explores the DTGs. The algorithm maintains a $subGoals$ list that stores the subgoals of the problem that are not yet evaluated (this list is initialized as $subGoals = \mathcal{G}$) and a $sideEffects$ list that maintains the side effects of the actions added to the relaxed plan (initially, $sideEffects = FS(\Pi)$). The heuristic h_{DTG} builds a relaxed plan by finding in the DTGs the shortest paths between the fluents in $sideEffects$ and $subGoals$ via the application of the Dijkstra algorithm.

We first introduce some notions that are needed for the h_{DTG} algorithm:

- $minPath(v, d_0, d_n) = \{d_0, \ldots, d_{n-1}, d_n\}$ is the shortest path between $\langle v, d_0\rangle \in sideEffects$ and $\langle v, d_n\rangle \in subGoals$, where d_0 is the initial value of the path and d_n is the final value of the variable or subgoal to be achieved.
- $getAction(v, d_{n-1}, d_n)$ is the minimum cost action that induces a value transition $d_{n-1} \rightarrow d_n$.

Subgoals are sorted according to their cost. We define the cost of a subgoal $g = \langle v, d_n\rangle$ as follows:

$$cost(g) = \arg\min_{\langle v,d_0\rangle\in sideEffects} |minPath(v, d_0, d_n)|$$

The cost of an action α is defined in terms of its preconditions:

$$cost(\alpha) = \sum_{p=\langle v,d_n\rangle\in PRE(\alpha)} cost(p)$$

The h_{DTG} algorithm extracts the subgoal $g = \langle v, d_n\rangle \in subGoals$ that maximizes $cost(g)$. Then, $minPath(v, d_0, d_n)$ is applied to all the values d_0 such that

$\langle v, d_0 \rangle \in sideEffects$. From all the obtained paths, the algorithm chooses the shortest one, $p = \{d_0, \ldots, d_{n-1}, d_n\}$.

Once the shortest path p is known, the algorithm introduces in the relaxed plan the minimum cost action α that induces each transition in p. That is, given, for instance, the last value transition in p, $d_{n-1} \rightarrow d_n$, the algorithm applies $getAction(v, d_{n-1}, d_n)$, obtaining an action α such that $\langle v, d_{n-1} \rangle \in PRE(\alpha)$ and $\langle v, d_n \rangle \in EFF(\alpha)$.

The effects of the action α for each value transition in p are inserted in the relaxed plan and stored in $sideEffects$, and the rest of preconditions of α, $\langle v', d' \rangle$, are inserted in the $subGoals$ list. Then, a new iteration of h_{DTG} starts with a new subgoal $g \in subGoals$.

Note that, as stated in equation 1, the cost of all preconditions related to the same variable v is estimated from the same state as $g = \langle v, d_n \rangle$ since they are solved in the same iteration of the process using the path p as a reference. The cost of the rest of preconditions $g' = \langle v', d' \rangle$, for variables $v' \neq v$, might be estimated from a state S' different from the state of g, depending on the fluent selected from $sideEffects$ to compute the cost of g'.

The process is completed when all the subgoals in $subGoals$ are processed. h_{DTG} returns the number of actions in the relaxed plan as an estimate of the cost of the plan.

To preserve privacy, each agent i stores its own version of the DTGs according to its knowledge of the planning task. Given a state variable v, agent i only keeps the DTG nodes and transitions that involve the values in $\mathcal{D}_v^i$. The rest of transitions are replaced by a reference to the agents that can realize such transition. For instance, given a transition $d_{n-1} \rightarrow d_n$, where $\mathcal{D}_v^i = \{d_{n-1}\}$ and $\mathcal{D}_v^j = \{d_{n-1}, d_n\}$, agent i maintains a transition $d_{n-1} \rightarrow j$, which indicates agent i that it must communicate with agent j in order to retrieve the cost of the transition. This way, the calculation of h_{DTG} preserves agents' privacy.

When $minPath$ is applied in a distributed context, agent i may have to resort to another agent j to find out the cost of a subpath that is not visible to i. In turn, agent j may also require the assistance of another agent k. To prevent an excessive number of messages among agents, the *recursion depth* of requests is limited during the application of h_{DTG}.

Landmarks heuristic This heuristic uses *landmarks*, fluents that must be satisfied in every solution plan of a MAP task, as the basis of its calculation.

Agents jointly generate the Landmarks Graph (LG). Formally, LG $= \{N, V\}$, where N is a set of nodes (landmarks) and V is a set of orderings between the nodes. Among the different types of orderings between landmarks, we use *necessary orderings*, which are directly inferred with the algorithm presented in (Hoffmann, Porteous, and Sebastia 2004). A necessary ordering of the form $l' \leq_n l$ indicates that the landmark l' should be achieved before l in all the solution plans for the task. *Single* landmarks contain only one fluent, while *disjunctive* landmarks are composed of a set of fluents, where one of them must be true in all the solution plans.

Algorithm 2 shows the distributed landmark extraction algorithm. This multi-agent procedure is described from the point of view of one agent i. In order to ensure privacy, all the fluents transmitted in Algorithm 2 are encoded as described in Subsection *Privacy in partial plans*. As a result of the execution of this algorithm, each agent i will obtain a version of the LG which includes only the landmarks that are public to i.

Algorithm 2: LG construction algorithm for an agent i

```
1  N ← ∅, V ← ∅, landmarks ← G
2  while landmarks ≠ ∅ do
3      l ← extractLandmark(landmarks)
4      producers^i ← α ∈ A^i | l ∈ EFF(α)
5      candidates^i ← ∩_{∀α∈producers^i} PRE(α)
6      disj^i ← groupNonCommonPrecs(producers^i)
7      if isCoordinator(i) then
8          lm ← candidates^i, disj ← {disj^i}
9          for all j ∈ AG, j ≠ i do
10             receive({disj^j, candidates^j}, j)
11             lm ← lm ∩ candidates^j
12             disj ← disj ∪ disj^j
13         lm ← lm ∪ groupDisjLandmarks(disj)
14         ∀j ∈ AG, j ≠ i, send(lm, j)
15     else
16         send({disj^i, candidates^i}, coordinator)
17         lm ← receive(lm, coordinator)
18     for all l' ∈ lm do
19         if isDisjunctive(l') ∨ verify(l') = true then
20             N ← N ∪ l'
21             V ← V ∪ l' ≤_n l
22             landmarks ← landmarks ∪ l'
23     Rotate coordinator role
24 for all l' ≤_n l ∈ V do
25     if verify(l' ≤_n l) = false then
26         V ← V \ {l' ≤_n l}
27 return LG = {N, V}
```

The algorithm is a backwards process that departs from the goals in $\mathcal{G}$. Given a landmark l, the process finds new landmarks as the preconditions that are common to all the actions that yield l as an effect. Once a landmark l' is inferred from l, a necessary ordering $l' \leq_n l$ is also established. Before their inclusion in the LG, all the single landmarks and necessary orderings must be *verified* to ensure their correctness.

An iteration of the Algorithm 2 is conducted by an agent that plays the role of coordinator (in the following, we reference in parenthesis the lines of Algorithm 2 in which each task is performed). Since actions are distributed across agents, the detection of single landmarks, from the viewpoint of an agent i, is described as follows:

- When a landmark l is extracted for its analysis (line 3), agent i calculates $candidates^i$ as the intersection of the preconditions of $producers^i$, the actions in $\mathcal{A}^i$ that yield

l as an effect (lines 4-5).

- Agent i masks the fluents in $candidates^i$ according to its level of privacy w.r.t. the coordinator agent. Then, i transmits $candidates^i$ to the coordinator agent (line 16), which applies the intersection of the sets of candidates received from all the agents in order to compute the actual set of landmark candidates called lm (line 11).

Agent i groups the preconditions of $producers^i$ that are not in $candidates^i$ according to its variable in order to generate disjunctive landmarks (line 6). For instance, let $producers^i = \{(\langle v, d_{n-1}\rangle, \langle v', d'\rangle) \rightarrow \langle v, d_n\rangle, (\langle v, d_{n-1}\rangle, \langle v', d''\rangle) \rightarrow \langle v, d_n\rangle\}$; then $candidates^i = \{\langle v, d_{n-1}\rangle\}$ and $disj^i = \{(\langle v', d'\rangle, \langle v', d''\rangle\}$ Agent i sends $disj^i$ along with $candidates^i$ to the coordinator agent, which groups together the disjunctive landmarks received from the agents, inserts them in the set lm (line 13) and sends lm back to the agents (lines 14 and 17).

In the next step, agents jointly verify the single landmark candidates $l' \in lm$ (line 19). The verification of l' entails solving a relaxed problem in which the actions α such that $l' \in EFF(\alpha)$ are excluded. If the goals in $\mathcal{G}$ are not satisfied then l' is verified as a landmark. If l' is verified, it is added to the LG along with a necessary order $l' \leq_n l$ (lines 20-21). For the verification of landmarks, agents are required to jointly generate a dis-RPG (Zhang, Nguyen, and Kowalczyk 2007).

Note that, in order to preserve privacy, agent i stores l' and the associated ordering $l' \leq_n l$ in its LG only if l' is public to i. This way, agents will keep different versions of the LG.

When the extraction and verification of landmarks is completed, the next step is the verification of the orderings in the LG (*forall* loop in lines 24-26). Given an ordering $l' \leq_n l$, agents create a dis-RPG excluding the actions $\alpha \in \mathcal{A} \mid l' \in PRE(\alpha) \wedge l \in EFF(\alpha)$ in order to validate it.

The LG created in Algorithm 2 is used to calculate the value of h_{Land} of a refinement plan in FMAP. Given a plan Π, $h_{Land}(\Pi)$ returns an estimate of the quality of Π, which is estimated as follows:

1. The agent i that generates Π checks which landmarks are satisfied in Π according to its LG (agent i coordinates the evaluation of Π). A refinement plan Π satisfies a landmark l iff $\exists \alpha \in \Delta(\Pi) \mid l \in EFF(\alpha)$, and $\forall l' \leq_n l \in N, l' \in EFF(\beta)$, where $\beta \in \Delta(\Pi)$ and $\exists \beta \prec \alpha \in \mathcal{OR}(\Pi)$; that is, a landmark l is not satisfied unless all its predecessors in the LG appear in Π as effects of the actions that precede the action α that has l in its effects.
2. Agent i communicates the verified landmarks to each agent j, $j \neq i$, masking the variables and values according to the level of privacy with agent j (see subsection *Privacy in partial plans*). Then, agent j verifies whether Π achieves any more landmarks that are not visible in the LG of the coordinator agent i.
3. Agents mask the new found landmarks and send them to the coordinator agent i, which computes the value of $h_{Land}(\Pi)$ as the number of landmarks that are not satisfied in Π.

The communication machinery required for the calculation of h_{Land} has been integrated into FMAP by reusing the messages of the original protocol, and thus, its distributed calculation does not increase the communication overhead.

Multi-heuristic approach Over the last years, one of the most successful research trends on single-agent state-based planning emphasizes the combination of heuristic functions. Recent studies conclude that the combination of multiple heuristics dramatically improves performance and scalability in planning (Röger and Helmert 2010). This conclusion is backed up by some well-known planning systems, such as Fast Downward (Helmert 2006) and LAMA (Richter and Westphal 2010), which successfully apply a multi-heuristic approach to state-based planning. Up to this date, however, the multi-heuristic approach has never been tested in MAP.

Algorithm 3: MH-FMAP algorithm for an agent i

```
openList ← Π_0, preferredList ← ∅
list ← true
while openList ≠ ∅ do
    if list = true then
        Π_b ← extractPlan(openList)
    else
        Π_b ← extractPlan(preferredList)
    list ← ¬list
    if isSolution(Π_b) then
        return Π_b
    RP ← refinePlan(Π_b)
    for all j ∈ 𝒜𝒢, j ≠ i do
        sendRefinements(j)
        RP ← RP ∪ receiveRefinements(j)
    for all Π_r ∈ RP do
        distributedEvaluation(Π_r, h_DTG)
        distributedEvaluation(Π_r, h_Land)
        openList ← openList ∪ Π_r
        if h_Land(Π_r) < h_Land(Π_b) then
            preferredList ← preferredList ∪ Π_r
return No solution
```

A basic question that arises when modeling a multi-heuristic approach is *how* to combine heuristics in order to maximize the performance of the resulting planner. The work in (Röger and Helmert 2010) experimentally compares different heuristic combination methods (sum, weighted sum, maximum, Pareto and alternation of heuristics), concluding that the *alternation* of heuristics is by far the most efficient method.

Our multi-heuristic MAP approach, MH-FMAP, is a heuristic alternation method. Rather than aggregating the heuristic values, alternation makes equal use of all the estimators, assuming that different heuristics might be useful in different parts of the search space. The most promising states are selected according to the currently used heuristic, completely ignoring all other heuristic estimates (Röger and Helmert 2010).

MH-FMAP is inspired by Fast Downward, which combines the FF and Causal Graph heuristics in an orthogonal way. Fast Downward maintains two open lists per heuristic: one list stores the open nodes and the other one keeps track of the *preferred successors*. While authors in (Helmert 2006) defined preferred successors as the ones generated by the so-called preferred operators, we define them by means of the landmark-based heuristic:

Definition 5 *A refinement plan* Π_r *is a* ***preferred successor*** *of a plan* Π *iff* $h_{Land}(\Pi_r) < h_{Land}(\Pi)$.

Algorithm 3 shows the FMAP basic search scheme adapted to our multi-heuristic approach, MH-FMAP. Agents now maintain two open lists: the $openList$ maintains the open nodes of the search tree, ordered by $f = g + 2 * h_{DTG}$, and the $preferredList$ keeps only the preferred successors, sorted by h_{Land}. If a plan is a preferred successor, it is introduced in both open lists. Agents extract a base plan from one of the lists alternatively; if a base plan is stored in both lists, it is removed from both of them.

The results of the next section prove that MH-FMAP yields notably superior results than the individual heuristics.

Experimental Results

We executed a wide range of experimental tests in order to assess the performance of the heuristic strategies presented in this paper[4]. Our benchmark includes the *STRIPS* suites of 10 different domains from the IPC[5], all of them adapted to a MAP context: *Depots*, *Driverlog*, *Elevators*, *Logistics*, *MA-Blocksworld*, *Openstacks*, *Rovers*, *Satellite*, *Woodworking* and *Zenotravel*. All the tasks were directly adapted from the *STRIPS* IPC suites, except for the *MA-Blocksworld* domain (Borrajo 2013), which introduces several arms that can simultaneously manipulate the blocks (4 agents per task).

The first experiment, shown in Table 1, compares the performance of FMAP with the h_{DTG} heuristic ($f = g + 2 * h_{DTG}$), the h_{Land} heuristic ($f = g + 2 * h_{Land}$) and MH-FMAP, our novel multi-heuristic approach based on the alternation of h_{DTG} and h_{Land}.

Due to the large amount of performed tests (244 planning tasks), we only display average results. More precisely, Table 1 summarizes, for each domain, the total number of solved tasks (*Sol* columns) and the average results of: search iterations (*#Iter* columns), execution time in seconds (*Time* columns), and plan quality results in terms of number of actions (*#Act* columns) and *makespan* (*MS* columns). The results of h_{DTG} and h_{Land} are relative to the results obtained with MH-FMAP, considering only the common tasks solved by both MH-FMAP and the respective single-heuristic approach. The nx values in Table 1 indicate "n times as much as the MH-FMAP result". Therefore, a value higher than 1x in *#Act*, *MS*, *Time* or *#Iter* is a better result for MH-FMAP.

[4]All the experimental tests were performed on a single machine with a quad-core Intel Core i7 processor and 8 GB RAM (2 GB RAM available for the Java VM).

[5]For more details on the MAP adaptation of the planning domains, please refer to (Torreño, Onaindia, and Sapena 2014).

The *Sol* columns of h_{DTG} and h_{Land} represent the number of problems solved by each heuristic, which happens to coincide, except for the *MA-Blocksworld* domain, with the number of common tasks solved by both MH-FMAP and h_{DTG} and h_{Land}, respectively. The last row of Table 1 displays the global average results.

MH-FMAP obtains the best coverage results in 9 out of the 10 tested domains, solving 215 out of 244 tasks (roughly 88% of the tasks). h_{DTG} solves one more problem than MH-FMAP in the *MA-Blocksworld* domain and it solves overall 191 tasks (78%). Using h_{Land} as a standalone estimator shows the worst performance, solving 117 tasks (48%).

It is worth noting that MH-FMAP tends to mimic the behaviour of the best-performing heuristic in most of the domains: for instance, in *Driverlog*, *Elevators*, *MA-Blocksworld* or *Zenotravel*, the results of coverage are much better with h_{DTG} than with h_{Land} and this is also reflected in MH-FMAP. However, h_{Land} solves more problems than h_{DTG} in the *Openstacks* domain and MH-FMAP equals the results obtained with h_{Land}. Interestingly, in the domains where h_{DTG} and h_{Land} offer a similar performance (namely, *Depots*, *Logistics* and *Woodworking*), the synergy of both estimators in MH-FMAP clearly outperforms the single-heuristic approaches, even resulting in twice as much the coverage in the *Logistics* domain.

h_{Land} takes much less time to evaluate a plan than the rest of approaches (33 ms per iteration in average, while MH-FMAP and h_{DTG} take around 200 ms).This is because, unlike h_{DTG}, the integration of h_{Land} in FMAP does not require any exchange of additional messages between agents apart from those already required by the FMAP search procedure. Nevertheless, h_{Land} requires the largest amount of iterations to find solutions in most domains; for instance, in *Driverlog* and *Elevators*, h_{Land} takes 350 and 585 times more iterations than MH-FMAP, respectively. In general, the accuracy of h_{Land} depends on the quality of the Landmarks Graph (LG). Particularly, in the *Openstacks* domain, the LG almost provides an skeleton for the solution plans, which explains the great performance of h_{Land} in this domain.

MH-FMAP requires more iterations and execution time than h_{DTG} in 6 out of the 10 tested domains. However, in general, MH-FMAP shows low execution times: less than 3 minutes in most domains, and around 6 minutes in *Openstacks*, the most time-consuming domain. Moreover, MH-FMAP performs admirably well in some domains, being around 5 times faster than h_{DTG} in *Depots* and *Logistics*.

Regarding plan quality (number of actions and makespan), MH-FMAP offers a good tradeoff between h_{DTG} and h_{Land}. According to the global results in Table 1, the quality results of the three approaches are very similar, being *#Act* slightly higher in h_{DTG}. As a whole, we can observe that the alternation of heuristics does not entail a loss of quality versus the standalone heuristics.

To sum up, h_{Land} turns out to be the fastest approach with the worst coverage. h_{DTG} is the slowest approach but it solves many more problems than h_{Land}. MH-FMAP, however, shows the potential of alternating global heuristics in MAP: it remarkably improves the coverage up to 88% of solved problems and, despite the overhead caused by the si-

Domain-Tasks	MH-FMAP					FMAP - h_{DTG}					FMAP - h_{Land}				
	Sol	#Iter	#Act	MS	Time	Sol	#Iter	#Act	MS	Time	Sol	#Iter	#Act	MS	Time
Depots-20	**12**	614,75	31,83	24,50	141,57	8	9,46x	1,23x	0,97x	5,55x	7	2,58x	0,96x	0,81x	0,57x
Driverlog-20	**15**	400,73	22,93	13,07	18,24	**15**	0,62x	1,04x	1,15x	0,60x	7	349,37x	0,90x	1,03x	51,78x
Elevators-30	**30**	53,93	24,67	13,40	9,43	**30**	0,66x	1,00x	0,96x	0,56x	13	585,47x	0,97x	0,92x	49,28x
Logistics-20	**20**	128,85	69,95	20,75	100,25	10	2,83x	1,05x	1,20x	5,43x	10	8,68x	1,00x	0,98x	1,75x
MA-Blocksworld-34	22	2542,36	18,18	14,64	45,71	**23**	0,96x	1,09x	1,06x	0,96x	16	11,44x	0,98x	0,91x	7,89x
Openstacks-30	**30**	707,80	63,10	52,90	353,73	25	0,75x	1,02x	0,95x	1,14x	**30**	0,18x	1,02x	1,02x	0,05x
Rovers-20	**20**	507,50	35,05	14,35	95,55	19	1,08x	1,01x	1,03x	1,50x	6	7,14x	0,99x	1,00x	1,21x
Satellite-20	**19**	72,74	32,58	19,95	115,05	18	0,92x	0,99x	0,97x	0,93x	4	22,01x	1,02x	1,00x	6,42x
Woodworking-30	**27**	1331,74	19,48	4,81	197,78	23	0,51x	1,01x	1,02x	0,40x	17	0,95x	0,97x	1,01x	0,13x
Zenotravel-20	**20**	96,65	32,35	18,35	115,68	**20**	0,94x	0,99x	0,97x	0,95x	7	155,19x	0,97x	1,03x	20,50x
Global results	**215**	670,56	35,59	20,37	128,51	191	0,95x	1,02x	0,99x	1,06x	117	11,32x	1,00x	0,99x	0,53x

Table 1: Comparison between MH-FMAP and FMAP (using h_{DTG} and h_{Land})

multaneous application of two heuristics, it offers competitive execution times. Finally, the combination of heuristics does not reduce the quality of the solution plans.

The second test compares MH-FMAP to another landmark-based approach to MAP, the Greedy Privacy Preserving Planner (GPPP). GPPP is the current best-performing MA-STRIPS planner and it introduces PP-LM, the first distributed version of a landmark-based heuristic (Maliah, Shani, and Stern 2014)[6].

Both PP-LM and h_{Land} build the LG and evaluate plans by counting the landmarks of the LG that are not reached yet. However, each heuristic is built upon a different planning framework (MH-FMAP and GPPP), presenting some key differences among them. PP-LM is designed for propositional MA-STRIPS domains, while h_{Land} supports tasks where facts are modeled through object fluents. In addition, the two heuristics are designed around a different notion of privacy: in GPPP, the private literals of an agent are occluded to the rest of agents, and the public literals are visible to all the participants. In contrast, MH-FMAP defines privacy between each pair of agents, masking the private information in preconditions and effects.

Table 2 compares the coverage, average execution time and plan quality of MH-FMAP and GPPP. Note that GPPP develops sequential plans, so the plan duration (makespan) equals the number of actions in this approach. Figures in Table 2 show average results for both approaches when running five IPC domains used in (Maliah, Shani, and Stern 2014).

Table 2 shows that GPPP is much faster than MH-FMAP (up to 50 times faster in some domains), mainly because, unlike MH-FMAP, GPPP does not use any communication infrastructure. As commented before, the use of a communication broker may entail a bottleneck when agents exchange a large amount of messages.

However, this superiority is not reflected in the coverage results. Despite being slower, MH-FMAP solves 109 out of 110 tasks, five more tasks than GPPP, which outnumbers MH-FMAP in only one task in the *Satellite* domain.

With respect to plan quality, MH-FMAP returns solution plans with fewer actions than GPPP in almost all the domains. For example, in *Zenotravel*, the solution plans of MH-FMAP contain 30% fewer actions than GPPP in average. GPPP only obtains slightly better results in the *Logistics* domain. Additionally, the POP-based approach of MH-FMAP allows us to obtain much shorter solutions (better makespan) than GPPP, which is limited to sequential plans.

[6]We want to thank the authors of GPPP for their kind support.

Domain-Tasks	MH-FMAP				GPPP		
	Sol	#Act	MS	Time	Sol	#Act	Time
Elevators-30	**30**	24,04	13,25	8,90	28	26,71	0,72
Logistics-20	**20**	69,95	20,75	100,25	**20**	69,25	2,02
Rovers-20	**20**	28,88	12,29	25,63	17	32,12	3,25
Satellite-20	19	32,58	19,95	115,05	**20**	38,32	3,44
Zenotravel-20	**20**	32,35	18,35	115,68	**20**	45,00	13,86

Table 2: Comparison between MH-FMAP and GPPP

In conclusion, MH-FMAP proves that the alternation of global heuristics is as effective in MAP as it is in classical planning. MH-FMAP not only performs much better than the single-heuristic FMAP setups, but also outperforms GPPP in terms of coverage and plan quality.

Conclusions

In this paper, we have presented MH-FMAP, a multi-agent planning system that draws upon the FMAP framework and incorporates a novel multi-heuristic search scheme that alternates two global heuristics: h_{DTG} and h_{Land}. We compared the performance of MH-FMAP against the standalone heuristics and GPPP, an MA-STRIPS-based planner, and the results throw a very positive balance in favor of MH-FMAP: a clearly superior coverage and a much better solution plan quality. In contrast, these excellent results come at the cost of a high number of message-passings between the agents.

The take-home lessons from this paper are: a) the use of global heuristics in MAP are actually worthy as long as the gain of the heuristic pays off the communication cost; b) the alternation of heuristics shows very beneficial for planning in general and also for MAP; c) using communication infrastructures is costly and affects the execution time but it is, however, necessary in order to implement heuristics in distributed environments with private information.

All in all, a proper combination of global heuristic estimators, well-defined communication protocols and a multi-heuristic search mechanism results in an ideal approach to cooperative MAP in distributed environments.

Acknowledgments

This work has been partly supported by the Spanish MICINN under projects Consolider Ingenio 2010 CSD2007-00022 and TIN2011-27652-C03-01, the Valencian Prometeo project II/2013/019, and the FPI-UPV scholarship granted to the first author by the Universitat Politècnica de València.

References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, 2–10.

Bonisoli, A.; Gerevini, A.; Saetti, A.; and Serina, I. 2014. A privacy-preserving model for the multi-agent propositional planning problem. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, 973–974.

Borrajo, D. 2013. Multi-agent planning by plan reuse. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1141–1142.

Brafman, R., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, 28–35.

Crosby, M.; Rovatsos, M.; and Petrick, R. 2013. Automated agent decomposition for classical planning. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, 46–54.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 162–169.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, 140–147.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 176–183.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)* 161–170.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26(1):191–246.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Maliah, S.; Shani, G.; and Stern, R. 2014. Privacy preserving landmark detection. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, 597–602.

Nissim, R., and Brafman, R. 2012. Multi-agent A* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1265–1266.

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

Nissim, R.; Brafman, R.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1323–1330.

O'Brien, P., and Nicol, R. 1998. FIPA - towards a standard for software agents. *BT Technology Journal* 16(3):51–59.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39(1):127–177.

Röger, R., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 246–249.

Štolba, M., and Komenda, A. 2013. Fast-forward heuristic for multiagent planning. In *Proceedings of the 1st Workshop on Distributed and Multi-Agent Planning (DMAP-ICAPS)*, 75–83.

Štolba, M., and Komenda, A. 2014. Relaxation heuristics for multiagent planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 298–306.

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

Zhang, J.; Nguyen, X.; and Kowalczyk, R. 2007. Graph-based multi-agent replanning algorithm. In *Proceedings of the 6th Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 798–805.