Fast-Forward Heuristic for Multiagent Planning

Michal Štolba and **Antonín Komenda**
{stolba|komenda}@agents.fel.cvut.cz
Department of Computer Science and Engineering,
Faculty of Electrical Engineering, Czech Technical University in Prague

Abstract

Use of heuristics in search-based domain-independent deterministic multiagent planning is as important as in classical planning. In this work we propose a formal and an algorithmic adaptation of a well-known heuristic Fast-Forward into multiagent planning. Such treatment is important as it solves challenges in decentralization of this and other heuristics based on relaxation of the original planning problem. Such decentralization enables global heuristic estimates to be computed without exposing local information. Additionally, since Fast-Forward heuristic is based on relaxed planning, we propose a multiagent approach for building factored relaxed planning graphs among the agents. We sketch proofs that the results of the distributed version of the algorithm gives the same results as the centralized version. Finally, we experimentally validate different distribution strategies of the heuristic estimate.

Introduction

In recent years the landscape of multiagent planning research has changed by Brafman and Domshlak's formal treatment and promising complexity results of domain-independent deterministic multiagent planning (DMAP) (Brafman and Domshlak 2008) represented as an extension of STRIPS for more agents. An important piece of the puzzle was a decomposition of a planning problem common for all the agents. In principle, the ideas behind relate to the research of planning problem factorization and utilization of such for more efficient solving of classical planning problems. Therefore even for cooperative agents, it is reasonable to hide parts of the information used during planning from other agents as this helps (in loosely coupled problems) the agents to focus only on their parts of the problem.

After this publication, the community started to design and implement first planners using the principles of DMAP described in the Brafman and Domshlak's paper. The first one from Nissim et al. (Nissim, Brafman, and Domshlak 2010) was built on distributed constraint satisfaction problem solver and a forward chaining planner. This approach precisely followed the ideas in (Brafman and Domshlak 2008), however exposed a couple of issues making the approach incomparable in efficiency with current state-of-the-art implementations of classical planners. One of the issues was bad scalability with growing length of the coordination part of the resulting plans. Improvement of scalability was proposed in (Nissim and Brafman 2012) by leaving the DisCSP+Planning approach and moving to a principle which is currently the most successful in classical planning—A* or variations on Best First Search (BFS) with highly informed automatically derived heuristics.

Since the motivation of (Nissim and Brafman 2012) was to propose an optimal planner (MA-A*), the heuristics used were LM-cut (Helmert and Domshlak 2009) with pathmax equation and merge-and-shrink (Helmert, Haslum, and Hoffmann 2007). In the distributed search approach, the heuristics were used only with local information of the respective agent, i.e., with its internal actions, its public actions and projections of other agents' public actions. In discussion of (Nissim and Brafman 2012), the authors state that *"the greatest practical challenge [...] is that of computing a global heuristic by a distributed system"*, which is precisely our focus in this work. According to our knowledge, there is no work proposing efficient planners for DMAP not focused on optimality of the resulting plans. In the field of classical planning, on the other hand, the best performing planners as Fast Downward and LAMA incorporate a fast, but suboptimal search algorithm using non-admissible heuristics.

In this work we propose a formal and algorithmic adaptation of a well-known relaxation heuristic Fast-Forward h^{FF} (Hoffmann and Nebel 2001) into multiagent planning. We argue that such treatment is important as it demonstrates algorithmic challenges in decentralization of computation of h^{FF} and other related heuristics. Additionally, since the h^{FF} heuristic is based on relaxed planning, we propose a multiagent (MA) approach for building factored relaxed planning graphs among the agents. We sketch proofs that the results of the distributed version of the algorithm gives the same results as the centralized version. Finally, we experimentally validate two distribution strategies of the heuristic estimate against the local estimate.

Multiagent Planning

We consider a number of *cooperative* and *coordinated* agents featuring distinct sets of capabilities (actions), which concurrently plan and execute their local plans in order to achieve a joint goal. The world wherein the agents act is *classical* and the actions are *deterministic*. The following formal preliminaries compactly restate the MA-STRIPS problem (Brafman and Domshlak 2008) required for the following sections.

A MA-STRIPS planning problem is a quadruple $\Pi = \langle \mathcal{L}, \mathcal{A}, s_0, S_g \rangle$, where $\mathcal{L}$ is a set of propositions, $\mathcal{A}$ is a set of *agents* $\alpha_1, \ldots, \alpha_{|\mathcal{A}|}$, s_0 is an initial state and S_g is a set of goal states. A *state* $s \subseteq \mathcal{L}$ is a set of atoms from a finite set of propositions $\mathcal{L} = \{p_1, \ldots, p_m\}$ which holds in s. An *action* an agent can perform is a tuple $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$, where a is a unique action label and $\mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)$ respectively denote the sets of preconditions, add effects and delete effects of a, taken from $\mathcal{L}$. *Act* denotes the set of all actions in the multiagent planning problem Π, i.e., $Act = \bigcup_{\alpha \in \mathcal{A}} \alpha$.

An *agent* $\alpha = \{a_1, \ldots, a_n\}$ is characterized precisely by its capabilities, a finite repertoire of actions $a_i \in Act$ it can preform in the environment. MA-STRIPS problems distinguish between the *public* and *internal* facts and actions. Let $\mathsf{atoms}(a) = \mathsf{pre}(a) \cup \mathsf{add}(a) \cup \mathsf{del}(a)$ and similarly $\mathsf{atoms}(\alpha) = \bigcup_{a \in \alpha} \mathsf{atoms}(a)$. An α-internal and public subset of all facts $\mathcal{L}$ will be denoted as $\mathcal{L}^{\alpha-\text{int}}$ and $\mathcal{L}^{\text{pub}}$ respectively, where $\mathcal{L}^{\alpha-\text{int}} = \mathsf{atoms}(\alpha) \setminus \bigcup_{\beta \in \mathcal{A} \setminus \alpha} \mathsf{atoms}(\beta)$ and $\mathcal{L}^{\text{pub}} = \mathsf{atoms}(\alpha) \setminus \mathcal{L}^{\alpha-\text{int}}$. Facts relevant only for one agent α are denoted as $\mathcal{L}^{\alpha} = \mathcal{L}^{\alpha-\text{int}} \cup \mathcal{L}^{\text{pub}}$ and a *projection* of a state s^{α} to an agent α is a subset of a global state s containing only public facts and α-internal facts, formally $s^{\alpha} = s \cap \mathcal{L}^{\alpha}$. The set of *public actions* of agent α is defined as $\alpha^{\text{pub}} = \{a \mid a \in \alpha, \mathsf{atoms}(a) \cap \mathcal{L}^{\text{pub}} \neq \emptyset\}$ and *internal actions* as $\alpha^{\text{int}} = \alpha \setminus \alpha^{\text{pub}}$. The symbol a^{α} will denote a projection of action $a \in \beta, \beta \neq \alpha$ for agent α, i.e., action stripped of all other agents' propositions, formally $\mathsf{atoms}(a^{\alpha}) = \mathsf{atoms}(a) \cap \mathcal{L}^{\alpha}$.

Note that all actions of an agent α uses only agent's facts, formally $\forall a \in \alpha : \mathsf{atoms}(a) \subseteq \mathcal{L}^{\alpha}$ by definition in (Brafman and Domshlak 2008). The goal set S_G of a multiagent planning problem will be treated as public (Nissim and Brafman 2012), therefore all goal-achieving actions are public. In the following sections, as an algorithm for multiagent planning, we will assume the MA-A* from (Nissim and Brafman 2012), but with a novel distribution of the h^{FF} heuristic.

As a running example, we will use a simple logistics problem (see Figure 1) in a multiagent setting. There are two cities each with two locations A, B and C, D and one package p. A and D represent depots and B,C airports. Three agents represent two cargo trucks t_1,t_2 (moving only within the cities) and one airplane a (moving only between airports B and C). The goal is to transport the package from depot A to the other depot D.

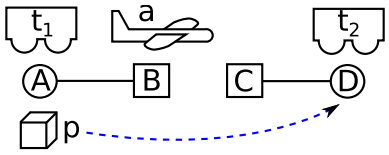

Figure 1: A running example is an instance of LOGISTICS problem with three agents and one package.

Agent Relaxed Planning Graph

Relaxation is a way of simplifying a problem by removing some constraints. In planning, a relaxation is typically obtained by removing delete effect of actions. Solution of such relaxed planning problem is a relaxed plan, which can be used to estimate the cost of a plan in the original problem, e.g., the Fast-Forward heuristic estimation is based on the length of the relaxed plan. A classical technique for finding the relaxed plan is to build a Relaxed Planning Graph (RPG). RPG is a graph representing the reachability of facts and applicability of actions in the relaxed problem.

Building distributed planning graphs (not relaxed) was studied by (Pellier 2010), focusing on distribution of the Graphplan algorithm. Relaxed MA Planning Graphs were recently studied by (Torreño, Onaindia, and Sapena 2012), but in the area of planning with incomplete information and fluent cost estimation.

To obtain a more informed global heuristic estimate in a MA planning problem using the estimation based on a RPG, the RPG has to be decentralized. In this work, we propose a distributed global RPG in form of a set of distinct Agent RPGs. Such Agent RPG (ARPG) contains only facts of its owner agent. The initial state is projection for that agent and since the goals are treated as public, all agents have complete goals in their ARPGs. The usage of actions is straightforward in case of owner agent's internal and public actions which are used equally as in a classical RPG. Additionally, the Agent RPGs are extended by projections of other agents' public actions which were reachable by their particular owners. This extension enables the agents to take other agents' capabilities into account, but only at the time points, where their owners are able to reach them. Similarly to relaxed problems in STRIPS, we define a relaxed multiagent planning problem in MA-STRIPS as a problem stripped of delete effects in all actions of all agents:

Definition 1. An *agent relaxed planning graph (ARPG)* is a directed, labeled and layered graph $\mathcal{R}'^{\alpha} = (P' \cup A', E')$ of one particular agent α for a relaxed multiagent planning task. Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_G)$ be a MA planning task, then a relaxed MA planning task $\Pi' = \langle \mathcal{L}, \mathcal{A}', s_0, S_G \rangle$ contains an altered set of agents $\mathcal{A}'$, s.t., $\forall \alpha \in \mathcal{A}$ and $\alpha = \{a_1, \ldots, a_{|\mathcal{A}|}\}$ there exist a relaxed agent $\alpha' = \{a'_1, \ldots, a'_{|\mathcal{A}|}\}$ and all its actions are relaxed versions of the regular actions $a'_i = \langle \mathsf{pre}(a_i), \mathsf{add}(a_i), \emptyset \rangle$. As in RPG, the nodes of the graph represent proposi-

tions P' and actions A'. The arcs E' represent linkup of propositions and actions.

In the rest of the paper, the discussion will be only about relaxed structures, therefore we will omit the prime signs, which are by convention used to denote relaxed structures.

ARPGs stem from the classical RPGs, therefore an i-th proposition layer and action layer will be denoted as P_i and A_i respectively. The layers alternate, so that $(P_0, A_0, P_1, A_1, \ldots, A_{n-1}, P_n)$ and all layers $P_i \subseteq P$ and all layers $A_i \subseteq A$. The first proposition layer P_0 contains nodes labeled by propositions of the agent's projection of the initial state, formally

$$P_0 = \{p | p \in s_0^\alpha\}.$$

Each action layer contains action nodes for all applicable relaxed actions of the agent α in a state represented by the previous fact layer and external projections of other agents' public actions reachable in the same layer

$$A_i \quad = \quad \{a | a \in \alpha, \mathsf{pre}(a) \subseteq P_i\} \cup \bigcup_{\beta \in \mathcal{A}, \beta \neq \alpha} \{b^\alpha | b \in P_i^\beta\}.$$

In all successive fact layers, the nodes copy the previous fact layer according to the frame axiom and transforms the facts by actions in the previous action layer, since for all relaxed actions $\mathsf{del}(a) = \emptyset$, we can write

$$P_i = P_{i-1} \cup \{p | p \in \mathsf{add}(a), a \in A_{i-1}\}.$$

At least one of the following terminating conditions has to hold for the last fact layer P_n:

- the last fact layer fulfills the goal $S_G \subseteq P_n$,
- or $P_n = P_{n-1}$, meaning there are no additional actions which can extend further fact layers (a fixed-point).

The arcs in ARPG represent applicability and application of actions in relaxed states. We can split the arcs between two fact layers P_i and P_{i+1} into three groups. The first one contains arcs among facts of layer P_i and preconditions of actions in a layer A_i. The second one contains relation between effects of actions and next induced fact layer P_{i+1}. Additionally, there are arcs for all facts from a previous layer effectively representing the frame axioms of the closed world assumption. Formally,

$$\begin{aligned} E_i^{\mathrm{pre}} &= \{(p_i, a_i) | p_i \in \mathsf{pre}(a_i), a_i \in A_i\}, \\ E_i^{\mathrm{add}} &= \{(a_i, p_{i+1}) | a_i \in A_i, p_{i+1} \in \mathsf{add}(a_i)\}, \\ E_i^{\mathrm{frm}} &= \{(p_i, p_{i+1}) | p_i \in P_i, p_{i+1} \in P_{i+1}, p_i = p_{i+1}\} \end{aligned}$$

and $E_i = E_i^{\mathrm{pre}} \cup E_i^{\mathrm{add}} \cup E_i^{\mathrm{frm}}$. Now we will provide an algorithm for distributed building of ARPGs.

Algorithm The algorithm starts with each agent building an ARPG using only its own internal and public actions. An iterative process is then initiated, in which the agents exchange information about their *public* actions and extends their ARPGs with projected public actions of other agents. The algorithm terminates when the goal (or a fixed-point) is globally reached and there are no more messages to process.

The pseudo-code of the algorithm is given in Algorithm 1 in an event-driven fashion. The events can be caused either by receiving a message or internally by the algorithm itself. We assume that messages sent from one agent arrive in the same order as they were

Algorithm 1 Distributed build of Agent Relaxed Planning Graphs

Input: An agent's factor of the relaxed MA planning problem $\Pi^\alpha = \langle \mathcal{L}^\alpha, \alpha, s_0^\alpha, S_G \rangle$.
Output: Agent Relaxed Planning Graph $\mathcal{R}$ for α.

```
1: init():
2: R ←P_0 = {p|p ∈ s_0^α}
3: R ←build-RPG(R, α, S_G )
4: S ←map[α^pub, integer]
5: ack-count←idle-count←0
6: check()
---
7: check():
8: for all a ∈ A_{n-1} s.t. a is public do
9:    if a ∉ S or S[a] > earliest layer of appearance
      of a in R then
10:      S[a] ←earliest appearance of a in R
11:      ack-count←ack-count+|A|
12:      ∀β ∈ A \ α : send(ext-a[a^β, S[a]], to β)
13:   end if
14: end for
---
15: receive(ext-a[a^α, i ∈ N], from β ∈ A \ α):
16: ∀α ∈ A : send(not-idle, to α)
17: send(ack, to β)
18: R ←extend-RPG(R, [a^α, i])
19: R ←build-RPG(R, α, S_G )
20: check()
21: goal-reached()
---
22: receive(ack):
23: ack-count ← ack-count - 1
24: goal-reached()
---
25: receive(idle):
26: idle-count ← idle-count + 1
27: if idle-count = |A| then
28:    return R
29: end if
---
30: receive(not-idle):
31: idle-count ← idle-count - 1
---
32: goal-reached():
33: if S_G ∈ R and ack-count = 0 and message queue
    is empty then
34:    ∀α ∈ A : send(idle, to α)
35: end if
```

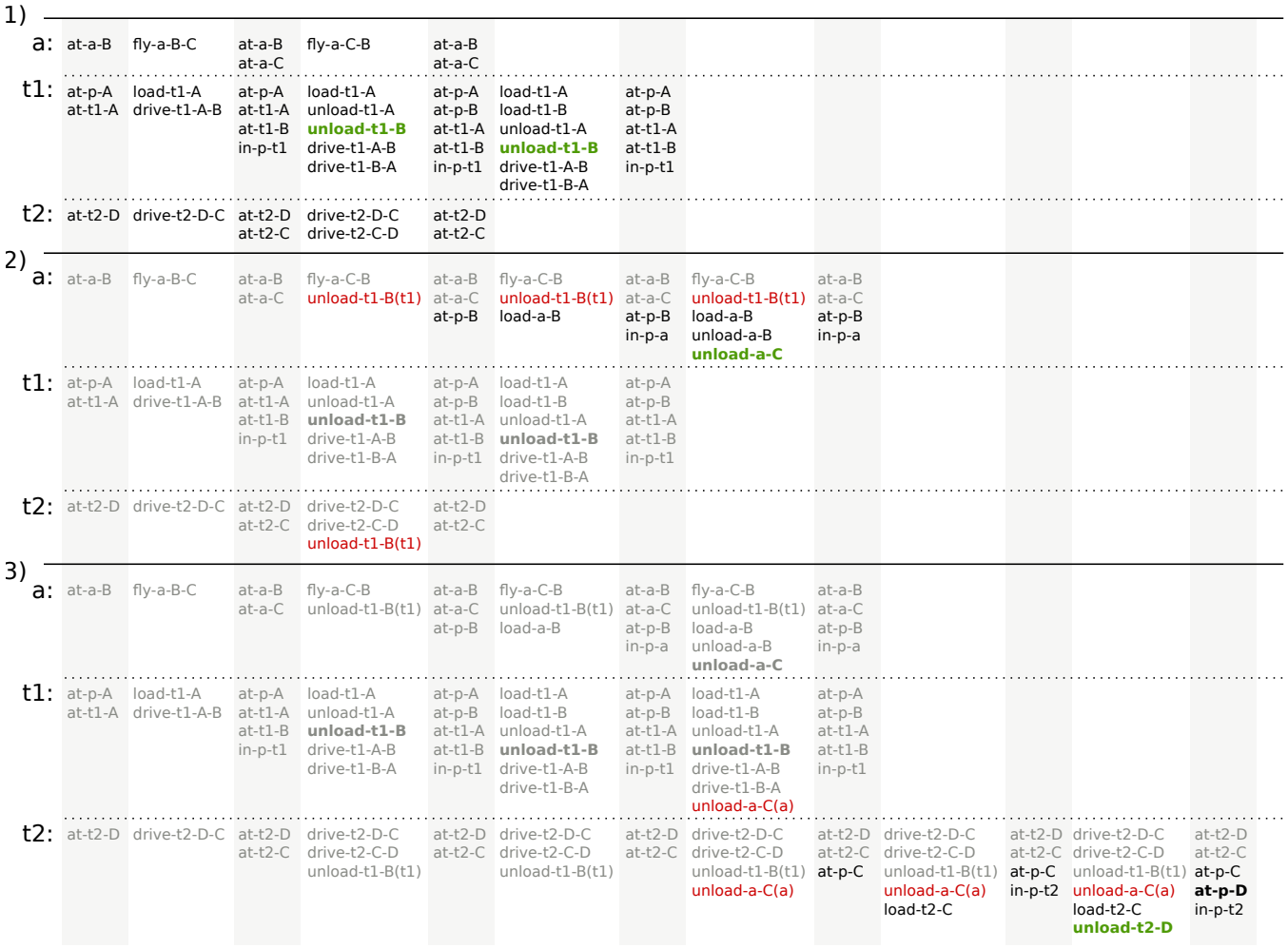

Figure 2: Distributed building of Agent Relaxed Planning Graphs decomposed into iterations.

sent, but we assume no ordering between messages sent from different agents. We will now explain each event handling routine.

In the **init** phase a Relaxed Planning Graph $\mathcal{R}$ is built using only agent's own actions by build-RPG method from the initial state projection s_0^α. A map $\mathcal{S}$ used to store the *earliest layer of appearance*[1] of the agent's public actions is initialized along with other supporting data structures used for synchronized termination of the algorithm. After the initialization phase, reaching of the goal (or a fixed-point) is checked, and if positive, all agents are informed that the agent is idle now. Next, the executed **check** procedure is responsible for checking whether $\mathcal{R}$ contains any public actions. If so, each action is sent to all other agents $\beta \in \mathcal{A} \setminus \alpha$ as a projection a^β with its earliest layer of appearance, unless it was already broadcasted with equal or lower number of layer (this can happen in future **check** calls).

[1]Earliest layer of appearance of an action a in (A)RPG is the first action layer, where a is applicable.

In the next part of the algorithm, there are four message handling **receive** procedures. The first one *ext-a* is executed when a projection of other agent's public action is received. After sending control messages, the action is integrated into $\mathcal{R}$ on the i-th layer by extend-RPG method and the change is propagated by build-RPG, so that all actions newly applicable in the following layers are applied accordingly. Then the built ARPG is checked, whether new public actions (and public actions newly applicable on earlier layers) are reachable and whether the goal or the fixed-point was reached. The last three **receive** procedures maintain the control information needed for distributed termination detection (Mattern 1987). The *acks* counter keeps track of number of sent external actions and postpones termination until all sent actions are processed. If an *idle* message is received, there are no pending *acks* and the number of idle agents is equal to $|\mathcal{A}|$, the algorithm terminates and the resulting ARPG $\mathcal{R}$ is returned. Since *not-idle* and *ack* messages are sent in this particular order (lines 17 and 18) and the messages from one agent

are presumed to keep ordering, the algorithm terminates synchronously when all external actions are processed and no messages are pending.

In Figure 2, the Algorithm 1 is applied on the running example depicted in Figure 1. Although the algorithm is running asynchronously, we can decompose it for clarity into several iterations. In the first iteration, the ARPGs are built using only the actions of the respective agents a, t_1 and t_2 (airplane and two trucks). Notice the bold green action unload-t1-B, which is a public action of the truck t_1, can be applied thanks to the initial position of the package. In the next iteration, projection of the public action is broadcasted and received by other agents. Upon receiving, their ARPGs are updated, which for the airplane means that the ARPG is expanded with further layers. Another public action unload-a-C is applied and therefore broadcasted. In the third iteration, the projection of the airplane's unload action is added to the ARPGs of the trucks. For truck t_1 it has no effect, but it allows truck t_2 to expand the ARPG and reach goal at-p-D. Notice, that when the projected unload-a-C(a) was received by truck t_2, its ARPG was first extended to have enough layers for the action to be added to the correct layer.

Although not shown in Figure 2, the algorithm would continue with one more iteration after broadcasting the public action reached by truck t_2, resulting in all agents having ARPGs with the same number of layers and all having reached the goal. Additionally, the algorithm does not have to terminate when the goal is reached, but can continue until the fixed-point, which can be desirable in some situations and which is also the case when the goal is not reachable.

Proof sketch In this section, we will sketch a proof showing, that the Agent Relaxed Planning Graphs built by Algorithm 1 are *compatible* with a global RPG, meaning that they contain the same actions (with respect to projections) and that the actions are in the same layers. We will use this proven theorem further in a proof of equality of the centralized FF and multiagent FF (MAFF) heuristics.

Firstly, we will formally define the concept of compatibility, then we will show that single iteration of the algorithm does not violate the compatibility, and finally we will show that the algorithm terminates, i.e., the resulting ARPGs are compatible with a centrally built RPG and that no actions are missing or are superfluous.

Let $\Pi = \langle \mathcal{L}, \mathcal{A}, s_0, S_G \rangle$ be a relaxed MA planning task, $\mathcal{R}^\alpha$ be an Agent Relaxed Planning Graph for agent α built from Π using Algorithm 1, having alternating layers $(P_0^\alpha, A_0^\alpha, P_1^\alpha, A_1^\alpha, \ldots, A_{n-1}^\alpha, P_n^\alpha)$ and let $A^\alpha = \bigcup_{i \in \langle 0, n-1 \rangle} A_i^\alpha$ and $A^{\mathcal{A}} = \bigcup_{\alpha \in \mathcal{A}} A^\alpha$. Let $\hat{\Pi} = \langle \mathcal{L}, Act, s_0, S_G \rangle$ be a classical relaxed planning task, $\hat{\mathcal{R}}$ be a classical Relaxed Planning Graph built from $\hat{\Pi}$ having alternating layers $(\hat{P}_0, \hat{A}_0, \hat{P}_1, \hat{A}_1, \ldots, \hat{A}_{n-1}, \hat{P}_n)$ and let $\hat{A} = \bigcup_{i \in \langle 0, n-1 \rangle} \hat{A}_i$. Note that from the monotonicity of (A)RPGs follows $\forall i < j : P_i \subseteq P_j$ and $\forall i < j : A_i \subseteq A_j$.

Definition 2. Let $a \rhd A_i$ denote that an action a is *first applicable* in layer A_i (formally $\mathsf{pre}(a) \subseteq P_i \wedge \mathsf{pre}(a) \nsubseteq P_{i-1}$) regardless of whether the underlying structure is RPG or ARPG.

Definition 3. We define that a set of ARPGs $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ is *compatible* with a RPG $\hat{\mathcal{R}}$ iff for each action $a \in A^{\mathcal{A}}$ for which $a \rhd \hat{A}_i$ holds the following:

1) If a is an *internal action* of agent α, then $a \rhd A_i^\alpha$.

2) If a is a *public action* of agent α, then $a \rhd A_i^\alpha$ and $\forall \beta \in \mathcal{A} \setminus \alpha : a^\beta \rhd A_i^\beta$, where a^β is the projection of action a for agent β.

Lemma 4. *A set of ARPGs $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ compatible with a RPG $\hat{\mathcal{R}}$ stays compatible with $\hat{\mathcal{R}}$ after application of build-RPG by agent α and successive extend-RPG by all other agents.*

Proof. Let us have a RPG $\hat{\mathcal{R}}$ built from a relaxed planning task $\hat{\Pi}$ and a set of ARPGs $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ being built from relaxed MA planning task Π using Algorithm 1. The symbol $A^{\mathcal{A}}$ denotes all actions applied in the algorithm so far and let α^{proj} be the set of projected actions received by agent α so far. Now, agent α applies build-RPG, so that $\mathcal{R}^\alpha$ is updated by $A_i^\alpha = A_i^\alpha \cup \{a | a \in \alpha \cup \alpha^{\text{proj}} : \mathsf{pre}(a) \subseteq P_i^\alpha\}$ (and accordingly P_{i+1}^α), for each layer A_i^α. Let us assume, there exists an extra action $a \in \alpha$ which was newly applied ($a \notin A^{\mathcal{A}}$) and for which $a \rhd \hat{A}_i \wedge a \not\rhd A_i^\alpha$. We can also assume WLOG, that a is first such action (in terms of layer of appearance).

From definition of $a \rhd \hat{A}_i$, where $\mathsf{pre}(a) \subseteq \hat{P}_i \wedge \mathsf{pre}(a) \nsubseteq \hat{P}_{i-1}$ follows that $\mathsf{pre}(a) \subseteq \hat{P}_0 \cup \{p | b \in \hat{A}_{i-1}, p \in \mathsf{add}(b)\}$ and $\mathsf{pre}(a) \nsubseteq \hat{P}_0 \cup \{p | b \in \hat{A}_{i-2}, p \in \mathsf{add}(b)\}$. Because a is first action for which $a \rhd \hat{A}_i \wedge a \not\rhd A_i^\alpha$, for all actions $b \in \alpha$, for which holds $b \rhd \hat{A}_k$ where $k < i$, holds also $b \rhd A_k^\alpha$. Therefore $A_k^\alpha = \hat{A}_k \cap \alpha$ and $P_k^\alpha = \hat{P}_k \cap \mathsf{atoms}(\alpha)$ for all $k < i$ and therefore $\mathsf{pre}(a) \subseteq P_i^\alpha \wedge \mathsf{pre}(a) \nsubseteq P_{i-1}^\alpha$, which means $a \rhd A_i^\alpha$ and that is a contradiction. Now, we can assign $A^{\mathcal{A}} \leftarrow A^{\mathcal{A}} \cup \{a\}$ and repeat the former step.

After broadcasting projections of the newly applied public actions and calling extend-RPG by all other agents, we can show that the second part of Definition 3 also holds. Let a^α be the projection of an action $a \in \beta$ which is broadcasted first. If there exists some i for which $a \rhd \hat{A}_i$ then $\mathsf{pre}(a) \subseteq \hat{P}_i$ and for the projection a^α holds $\mathsf{pre}(a) \subseteq \hat{P}_i \cap \mathcal{L}^{\text{pub}}$. Because for all actions $b \in \hat{A}_{i-1}$ the lemma holds, if b is public, $b^\alpha \rhd A_{i-1}^\alpha$. Because a^α is a projection, $\mathsf{pre}(a^\alpha) \subseteq P_0^\alpha \cup \bigcup_{b^\alpha \in A_{i-1}^\alpha} \mathsf{add}(b)$ and therefore a^α is applicable in layer i (and subsequent layers), which means that the extend-RPG ensures that $a^\alpha \rhd A_i^\alpha$. □

Theorem 5. *When Algorithm 1 terminates, resulting set of ARPGs* $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ *built from* Π *is compatible with the RPG* $\hat{\mathcal{R}}$ *built from* $\hat{\Pi}$ *and there are no additional actions, i.e.,* $\hat{A} = A^{\mathcal{A}}$. *Each public action in* $\hat{A}$ *has its projected counterparts in* $A^{\mathcal{A}}$ *and vice versa, i.e., for each public action* $a \in \hat{A}$ *such that* $a \in \alpha$ *for some agent* α *exists projected action* a^β *for each agent* $\beta \in \mathcal{A} \setminus \alpha$ *and* $a^\beta \in A^{\mathcal{A}}$. *There is no projected action* $a^\beta \in A^{\mathcal{A}}$ *for which there is no original action* $a \in \hat{A}$.

Proof. We will now sketch an induction which shows the compatibility in Theorem 5, based on the Lemma 4. For the initial step of the induction we take all ARPGs containing only the first fact layer P_0^α which is trivially compatible because $A^{\mathcal{A}} = \emptyset$. The induction step is covered by Lemma 4, because each step of the Algorithm 1 can be decomposed as an application of build-RPG and, if there are any applied public actions, broadcasting their projections and application of extend-RPG by all other agents. Even though the algorithm is running asynchronously, Lemma 4 holds because of the monotonicity of (A)RPGs.

Termination of the algorithm follows from the termination of classical RPG, either the algorithm reaches goal or a fixed-point, where no more actions are added. Similarly to classical RPG, building of ARPGs is monotonic, which means that the facts and actions can only be added and because the set of actions is finite, there must be a point where no more actions can be added and the algorithm terminates. The detection of such situation is more complicated in the distributed setting and is described thoroughly in the algorithm section.

The last statement we are about to show is that there are no additional actions, i.e., $\hat{A} = A^{\mathcal{A}}$ (irrespective of the projections of public actions) and that each public action in $\hat{A}$ has its projection in $A^{\mathcal{A}}$ and vice versa. Let us assume, that $\exists a \in \hat{A}$ such that $a \notin A^{\mathcal{A}}$, let us also assume, WLOG, that a is such action appearing in the earliest layer in $\hat{\mathcal{R}}$, say $\hat{A}_i$, and that $a \in \alpha$ for some agent α. We know, that exists minimal $A_{\text{pre}} \subseteq \hat{A}_{i-1}$ such that $\mathsf{pre}(a) \subseteq \{p | b \in A_{\text{pre}}, p \in \mathsf{add}(b)\}$, because $\mathsf{pre}(a) \subseteq \mathcal{L}^{\alpha-\text{int}} \cup \mathcal{L}^{\text{pub}}$, for all actions $b \in A_{pre}$ either $b \in \alpha$ or b is public. Because we assumed, that $A_{\text{pre}} \subseteq \hat{A}$, for each $b \in A_{\text{pre}}$, if $b \in \alpha$ then $b \in A_{i-1}^\alpha$ and if $b \notin \alpha$ then b is public, therefore $b^\alpha \in A_{i-1}^\alpha$. From the said $\mathsf{pre}(a) \subseteq \{p | b \in A_{i-1}^\alpha, p \in \mathsf{add}(b)\}$, which means that a is applicable in A_i^α and therefore a must be applied by the algorithm. If we continue with next such action we end up with $\hat{A} \subseteq A^{\mathcal{A}}$.

Now, we will show that $\hat{A} \supseteq A^{\mathcal{A}}$. Let us assume, that $\exists a \in A^{\mathcal{A}}$ such that $a \notin \hat{A}$ and that a is first such action. Similarly to the previous situation, if a is applicable in some layer A_i^α then there is some set of actions $A_{\text{pre}} \subseteq A_{i-1}^\alpha$, which contains all actions providing preconditions of a. Since all actions $b \in A_{\text{pre}}$ are also in $\hat{A}$, a is applicable in $\hat{A}_i$ and therefore must be applied.

From $\hat{A} \subseteq A^{\mathcal{A}}$ and $\hat{A} \supseteq A^{\mathcal{A}}$ follows that $\hat{A} = A^{\mathcal{A}}$. We have already shown that public actions have their projected counterparts. The only remaining part to show is that there are no projected actions in $A^{\mathcal{A}}$ without their respective original actions in $\hat{A}$. This clearly follows from the algorithm itself, because all projected actions are created only when a public action is added to some A_i^α by agent α and as shown before, such action would also be added to $\hat{A}_i$. □

Multiagent FF Heuristic

With the help of ARPGs, the Fast-Forward heuristic estimate can be straightforwardly adapted to a multiagent setting. We will denoted such heuristic as h^{MAFF}. The multiagent (MA) relaxed plan backing the h^{MAFF} estimate can be in general spread over all ARPGs of the agents in the team as illustrated in Figure 3. The most left achieving actions has to be considered from all agents. In the case of projected public actions, the owner agent has to define part of the the relaxed plan, possibly using his internal actions, to achieve the internal facts of the provided public action. Additionally, the relaxed plan has to share public actions which are required by more agents at the same layers. The private parts of the relaxed plan provided by the other agents can be described by place-holding actions and therefore no private information of the other agents has to be revealed. The final heuristic estimate is the count of actions of the MA relaxed plan.

Definition. Let a MA relaxed plan π be a solution of a MA relaxed problem $\Pi = \langle \mathcal{L}, \mathcal{A}, s, S_G \rangle$, where s is the state, we are estimating the cost for, then $|\pi| = h(s)$ is the *multiagent relaxation heuristic estimate*.

Similarly to the relaxation heuristic estimate h^{FF}, we restrain π for h^{MAFF} according to h^{FF}. A particular π is defined using ARPGs $\mathcal{R}^\alpha = (P \cup A, E)$ of all agents $\alpha \in \mathcal{A}$ built for the state s. From the right (meaning as in Figure 3), the relaxed plan π contains minimal set of actions $A_m^* \subseteq A_m$, achieving the goal facts. The action layer A_m contain actions of all agents in layer m (ignoring projections of actions, since the respective original actions are also included in the same layer). If there is a frame arc $(p_{m-1}, p_m) \in E_m^{\text{frm}}$ of such facts, i.e., $p_m \in S_G$, the fact p_m does not need an explicit achieving action from this particular layer as it will be achieved by an action from an earlier (more left) layer. This principle effectively selects the most-left achievers of a fact as proposed by FF heuristic. The action set A_m^* induces next set of facts across all agents

$$S_m = \{p | p \in \mathsf{pre}(a) : a \in A_m^*\},$$

which has to be achieved by actions from previous action layer A_{m-1} and so on until the action layer A_0 is reached, where all the actions have their preconditions satisfied by the initial state in P_0.

Notice that since this definition works across all ARPGs of all agents, the resulting π may contain actions of different agents.

Figure 3: Multiagent Relaxed Plan

We can compute $h^{\mathrm{MAFF}}(s)$ by first building the set of ARPGs $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ for relaxed MA planning problem $\Pi = \langle \mathcal{L}, \mathcal{A}, s, S_G \rangle$ using Algorithm 1, then simultaneously extracting relaxed plans π_α for each agent using Algorithm 2 and finally summing the lengths of the resulting relaxed plans, excluding projections of other agent's public actions.

Theorem 6. *Let $\pi_\alpha \cap \alpha$ be the computed relaxed plan of agent α restricted only to the agent's actions (excluding all projections of other agent's public actions), then $h^{\mathrm{MAFF}}(s) = \sum_{\alpha \in \mathcal{A}} |\pi_\alpha \cap \alpha| = h^{\mathrm{FF}}(s)$.*

Proof. The fact that $h^{\mathrm{MAFF}}(s) = h^{\mathrm{FF}}(s)$ follows from the previously shown compatibility of the RPG $\hat{\mathcal{R}}$ built for relaxed planning problem $\hat{\Pi}$ and the set of ARPGs built from $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ for relaxed MA planning problem Π. We first extract relaxed plans $\hat{\pi}$ for $\hat{\Pi}$ and $\{\pi_\alpha | \alpha \in \mathcal{A}\}$ for Π by effectively choosing first achievers of goal facts and of preconditions of previously chosen achievers. It is clear that for some fact p we choose an action a s.t. $p \in \mathsf{add}(a)$ only if $a \rhd A_i$ for some layer A_i and there is no action b s.t. $p \in \mathsf{add}(b)$ and $b \rhd A_j$ for some $j < i$. Because of the compatibility of the RPG and ARPGs, we choose exactly the same actions (and their projections, which are then omitted) for $\hat{\pi}'$ and for $\{\pi'_\alpha | \alpha \in \mathcal{A}'\}$, which means that $|\hat{\pi}'| = \sum_{\alpha \in \mathcal{A}'} |\pi'_\alpha \cap \alpha|$ and therefore $h^{\mathrm{MAFF}}(s) = h^{\mathrm{FF}}(s)$. □

Experiments

The experiments were conducted on an implementation of satisficing version of MA-A* (Nissim and Brafman 2012) with various relaxation heuristic estimates[2]. The algorithm begins with a centralized factorization and a reachability analysis of a centralized planning problem. After the factorization, the agents are started receiving its factors of the problem as an input. The agents run in parallel, each one in its own thread and the messages are delivered by an additional asynchronous messaging thread. Each agent uses an event queue to serialize the computation and reactions to incoming messages. If an agent finds a sound plan (with parts from other agents) it prints it and stops the distributed process.

Since the algorithm is asynchronous, the runs are non-deterministic. Therefore we conducted each experimental run as ten measurements. Each measurement was limited to 8GB of memory for the Java Virtual Machine and to 10 minutes of runtime. Each measurement was run on 8-core processor at 3.6GHz separately. The results from the measurements were averaged.

[2]Technically the implementation is not A* as the heuristics are not admissible, therefore the used algorithm is precisely MA-BestFirstSearch.

Algorithm 2 Distributed extraction of h^{MAFF}

Input: ARPG $\mathcal{R}$ for state s, having layers $(P_0, A_0, P_1, A_1, \ldots, A_{n-1}, P_n)$ and goal $S_G \subseteq \mathcal{L}$.
Output: Relaxed plan R from s to S_G.

```
1:  P ← S_G
2:  R ← ∅
3:  for i = n − 1; i > 0; i ← i − 1 do
4:    P' ← ∅
5:    for p ∈ P do
6:      if p ∉ P_i then
7:        a ← a ∈ A_i, such that p ∈ add(a)
8:        R ← R ∪ {a}
9:        P' ← P' ∪ pre(a), P ← P\{p}
10:       if a is projected action of agent α then
11:         request R^α for s, goal pre(a^orig) from α
12:         R ← R ∪ R^α
13:       end if
14:     end if
15:   end for
16:   P ← P ∪ P'
17: end for
18: return R
```

We used five planning domains, four originating in the single-agent IPC planning benchmarks. Similarly to the evaluation of the algorithms in (Nissim, Brafman, and Domshlak 2010), we chose domains which are straightforwardly modifiable to the multiagent setting: LOGISTICS (similar to the running example, but with more agents), LINEAR LOGISTICS (one package has to be transported stepwise by all agents), ROVERS, and SATELLITES. As in (Komenda, Novák, and Pěchouček 2013), we have extended the set of IPC-based domains by a coordination domain COOPERATIVE PATHFINDING, in which robots on a grid are tasked to switch their positions not colliding with each other. We tested following distribution strategies of FF heuristic estimation:

- h^{FF}_α using only locally built RPGs (including projections of public actions) and local estimation of Fast-Forward heuristic, as proposed in (Nissim and Brafman 2012).
- h^{MAFF} using distributed ARPGs based on Algorithm 1 and distributed extraction of FF heuristic as described in Algorithm 2.
- Lazy h^{MAFF} using only locally built RPGs (including projections of public actions) and distributed extrac-

			h^{FF}_{α}				h^{MAFF}						Lazy h^{MAFF}					
	$\lvert\mathcal{A}\rvert$	l^*	$t[s]$	v	c_{s}	l	$t[s]$	v	c_{s}	c_{r}	c_{h}	l	$t[s]$	v	c_{s}	c_{r}	c_{h}	l
CP	3	6	0.4	99	97	6	6.3	168	166	6.6k	39	6	0.7	117	115	72	40	6
	5	16	88	25k	25k	16	–	–	–	–	–	–	7.6	1.8k	1.8k	120	145	18
	7	24	–	–	–	–	–	–	–	–	–	–	323	52k	52k	168	254	30,9
LOG	4	14	0.6	1.5k	847	14	2.5	505	272	10k	50	14	0.4	365	223	32	45	14
	6	20	6.2	17k	7.4k	20,6	–	–	–	–	–	–	1.3	732	341	52	185	20
LLG	6	18	0.2	134	61	18	4.9	118	54	2.0k	35	18	0.5	130	61	22	21	18
	8	24	0.6	241	113	24	11	216	102	4.6k	64	24	1.2	231	108	30	34	24
	10	30	0.9	381	181	30	27	337	161	8.7k	99	30	2.8	357	170	38	46	30
ROV	2	22	–	–	–	–	88	378	4	25k	4	22	19	482	4	72	4	22
	3	33	–	–	–	–	–	–	–	–	–	–	261	2.0k	14	108	11	33
SAT	4	14	32	32k	369	14	16	941	27	9.8k	22	14,3	0.8	536	29	4	23	14,3
	6	21	–	–	–	–	–	–	–	–	–	–	6.2	1.7k	67	6	54	21,3
	8	–	–	–	–	–	–	–	–	–	–	–	45	4.5k	147	8	104	28,1

Table 1: Experimental results for the heuristics. $|\mathcal{A}|$ is number of agents, l is sequential length of the plan (l^* is optimal), t is duration of the search in seconds, v is a number of visited states, c_{s} is a number of search messages (each of size of a state), c_{r} is a number of messages building ARPGs (each of size of a projected public action) and c_{h} is a number of messages for the heuristic estimate (each of size of a partial relaxed plan). As h^{FF}_{α} do not build distributed ARPGs, c_r and c_h are always zero. The domains are COOP. PATHFINDING (CP), LOGISTICS (LOG), LINEAR LOGISTICS (LLG), ROVERS (ROV) and SATELLITES (SAT). Runs denoted as – did not finish in the limits.

tion of FF heuristic as in Algorithm 2 with additional information on reachability of projected actions.

The implementation we used is a preliminary prototype which is not competitive with the current single-agent state-of-the-art planners, but nevertheless it gives insights in comparison of the heuristics.

The h^{MAFF} is based on the theory presented in the previous section. For each state each of the agents computes the heuristic estimate, i.e., a complete set of ARPGs is built. In order to manage the distribution and asynchronism, the ARPG building algorithm slightly differs from Algorithm 1 so that ARPGs for several different states can be built simultaneously. The heuristic estimate is then extracted using Algorithm 2.

In Lazy h^{MAFF}, the ARPGs are built lazily, i.e., an agent builds an ARPG for its current search state using its actions and projections similarly as in h^{FF}_{α}, then the Relaxed Plan (RP) is extracted and only when some projected action is added to the RP, request is sent to the owner of the original action. The owner then builds an ARPG from the given state to preconditions of the original actions, extracts a RP using the same procedure and sends back the computed RP. The returned RP is then merged with the original one. This effectively forms a distributed recursion algorithm. Such algorithm significantly lowers communication load and enables the agents to search in parallel, especially in loosely coupled problems. In addition to this, reachability analysis using Algorithm 1 can be done before starting the search to improve the estimate of applicability of the projected actions.

The results in Table 1 show, that h^{FF}_{α} is fast and can effectively solve smaller problem instances, but it is not much informed, as illustrated by the number of visited states. This becomes critical in larger problems. On the other hand, h^{MAFF} is better informed, but it has to build all ARPGs for each state which is estimated. This is extremely communication intensive as shown by the number of exchanged ARPG messages (c_{r}). Also the possibilities of parallel computation are reduced by the fact that all agents have to build the ARPGs for each estimated state.

The best performance is given by the Lazy h^{MAFF}. This implementation keeps the heuristic estimate quality of h^{MAFF}, but since it does not build ARPGs during the search, but only local RPGs enriched by the projections of other agents' actions, the RPGs can be built lazily only for those states where any interaction between the agents is needed and only those agents involved build the RPGs. The ARPGs are computed only in an initial reachability analysis, which can be omitted, but which significantly improves the results.

Final Remarks

Our formal treatment and design of algorithms for computing distributed Relaxed Planning Graph for multiagent planning can be seen as a first step towards efficient MA planners based on satisficing state-space search techniques utilizing relaxation heuristics. Furthermore, we have experimentally shown that appropriate implementation of a decentralized estimation of a global relaxation heuristic can radically improve computational and communication efficiency of the planning process as a whole.

Acknowledgments This work was supported by the *U.S. Air Force EOARD* grant no. FA8655-12-1-2096.

References

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *Proceedings of ICAPS'08*, 28–35. AAAI.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of ICAPS'09*. AAAI.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *Proceedings of ICAPS'07*, 176–183. AAAI.

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

Komenda, A.; Novák, P.; and Pěchouček, M. 2013. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications*. DOI: 10.1016/j.jnca.2012.12.011.

Mattern, F. 1987. Algorithms for distributed termination detection. *Distributed computing* 2(3):161–175.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In van der Hoek, W.; Padgham, L.; Conitzer, V.; and Winikoff, M., eds., *Proceedings of AAMAS'12*, 1265–1266. IFAAMAS.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of AAMAS'10*, 1323–1330.

Pellier, D. 2010. Distributed planning through graph merging. In Filipe, J.; Fred, A. L. N.; and Sharp, B., eds., *Proceedings of ICAART'10*, volume 2, 128–134. IFAAMAS.

Torreño, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In Raedt, L. D.; Bessière, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P. J. F., eds., *Proceedings of ECAI'12*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 762–767. IOS Press.