Relaxation Heuristics for Multiagent Planning

Michal Štolba[1] and **Antonín Komenda**[2]
stolba@agents.fel.cvut.cz, akomenda@tx.technion.ac.il
[1]Department of Computer Science and Engineering,
Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic
[2]Faculty of Industrial Engineering and Management,
Technion—Israel Institute of Technology, Haifa, Israel

Abstract

Similarly to classical planning, in MA-STRIPS multiagent planning, heuristics significantly improve efficiency of search-based planners. Heuristics based on solving a relaxation of the original planning problem are intensively studied and well understood. In particular, frequently used is the delete relaxation, where all delete effects of actions are omitted. In this paper, we present a unified view on distribution of delete relaxation heuristics for multiagent planning.

Until recently, the most common approach to adaptation of heuristics for multiagent planning was to compute the heuristic estimate using only a projection of the problem for a single agent. In this paper, we place such approach in the context of techniques which allow sharing more information among the agents and thus improve the heuristic estimates. We thoroughly experimentally evaluate properties of our distribution of *additive*, *max* and *Fast-Forward* relaxation heuristics in a planner based on distributed Best-First Search. The best performing distributed relaxation heuristics favorably compares to a state-of-the-art MA-STRIPS planner in terms of benchmark problem coverage. Finally, we analyze impact of limited agent interactions by means of recursion depth of the heuristic estimates.

Introduction

Planning in a shared deterministic environment for and by a team of cooperative agents with common goals is a natural extension of classical planning. To model such planning problems, (Brafman and Domshlak 2008) proposed a multiagent extension of the classical STRIPS formalization, MA-STRIPS. The model presumes a set of cooperative agents defined by their capabilities in form of a set of actions partitioned from the original planning problem. In general, not all the agents need to (or even can not) consider the complete planning problem, therefore only subsets of facts and actions are marked as *public* and known to the whole team.

In recent years, several multiagent planning techniques solving MA-STRIPS problems were proposed. One focusing on optimality, scalability and efficiency was proposed by (Nissim and Brafman 2012). It adopted one of the currently most successful approaches in classical planning—Best-First Search with highly informed automatically derived heuristics. The heuristics used were LM-cut (Helmert and Domshlak 2009) with pathmax equation and merge-and-shrink (Helmert, Haslum, and Hoffmann 2007), both admissible as is necessary for optimal planning. The heuristics used only local information of the respective agents, effectively working only with their own facts and actions and public actions of other agents. (Crosby, Rovatsos, and Petrick 2013) proposed a centralized planning approach based on multiagent decomposition and local heuristic estimates. The approach used delete relaxation, particularly the Fast-Forward (FF) (Hoffmann and Nebel 2001) heuristics and focused on satisfiability. Another approach proposed by (Borrajo 2013) can in principle end up as planning with a global heuristic estimations, however requires private information of other agents[1]. On the contrary, the approach by (Torreño, Onaindia, and Sapena 2013) preserves private knowledge and proposes distributed heuristic estimate, however it is not based on relaxation of the original problem, but on Domain Transition Graphs (Helmert 2006). 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"*. A recent work by (Štolba and Komenda 2013) targeted this challenge for distributed Fast-Forward heuristic with focus on obtaining the same estimates as by a centralized solution, rather than searching for a general solution for wide variety of relaxation heuristics.

In this work, we focus on distribution of the general principle of delete relaxation heuristics in MA-STRIPS planning with state-of-the-art implementation approaches. We evaluate properties of such distributed heuristics both from computational and communication perspectives and analyze the quality of the heuristics based on estimation depth in a sense of participating agents, i.e., *agent coupling relaxation*.

Multiagent Planning

A MA-STRIPS (Brafman and Domshlak 2008) 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 cooperative *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 hold in s. An *action*

[1]The proposed solution used obfuscation of the private information, which can be prohibited in cases where even the "structure" of the information has not to be revealed.

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 from $\mathcal{L}$.

An *agent* $\alpha = \{a_1, \ldots, a_n\}$ is characterized precisely by its capabilities, a finite repertoire of actions it can preform in the environment. MA-STRIPS problems distinguish between the *public* and *internal* (or *private*) 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 subset of all facts $\mathcal{L}$ of agent α will be denoted as $\mathcal{L}^{\alpha-\mathrm{int}}$, where $\mathcal{L}^{\alpha-\mathrm{int}} = \mathsf{atoms}(\alpha) \setminus \bigcup_{\beta \in \mathcal{A} \setminus \alpha} \mathsf{atoms}(\beta)$ and a subset of all public facts as $\mathcal{L}^{\mathrm{pub}} = \mathcal{L} \setminus \bigcup_{\alpha \in \mathcal{A}} \mathcal{L}^{\alpha-\mathrm{int}}$. All facts relevant for one particular agent α are denoted as $\mathcal{L}^{\alpha} = \mathcal{L}^{\alpha-\mathrm{int}} \cup \mathcal{L}^{\mathrm{pub}}$ 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 $\alpha^{\mathrm{pub}} = \{a \mid a \in \alpha, \mathsf{atoms}(a) \cap \mathcal{L}^{\mathrm{pub}} \neq \emptyset\}$ and *internal actions* as $\alpha^{\mathrm{int}} = \alpha \setminus \alpha^{\mathrm{pub}}$. 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 propositions of other agents, formally $\mathsf{atoms}(a^{\alpha}) = \mathsf{atoms}(a) \cap \mathcal{L}^{\alpha}$. Finally, α^{proj} will denote the set of all public actions of other agents $\mathcal{A} \setminus \alpha$ projected for agent α.

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.

Multiagent Best-First Search

The planning algorithm we assume for the further analysis of the distributed heuristics is based on a multiagent Best-First Search (MA-BFS) derived from the work in (Nissim and Brafman 2012) with deferred-evaluation of (lazy) heuristics. It is outlined in Algorithm 1. The MA-BFS algorithm is based on the standard textbook BFS. Firstly, the $Open$ list is initialized with the initial state s_i and the $Closed$ list is empty (line 1). In an infinite loop (lines 2–15), the $Open$ list is polled to obtain state s (line 4). If s was not processed yet, it is marked as closed (lines 5, 6), next, if s is a goal state, the search is terminated and the plan is reconstructed (lines 7–9), otherwise, the heuristic estimate of the state is computed (line 10).

The MA-BFS differs from the textbook BFS in several aspects. Because the computation of an heuristic estimate may invoke communication among multiple agents, the heuristic estimators are asynchronous. The estimator is called with a callback in its parameter and the main loop immediately continues (line 10). When the heuristic estimation is finished, the callback (lines 16–23) is invoked and it performs the standard procedure of setting the heuristic value (line 17) and expanding the state using applicable actions (line 22). In addition to that, if the state was obtained by expanding a public action, the state is sent to all other agents.

Finally at the end of each loop (lines 13 and 14) the messages related to the heuristic estimation (see next section) and the messages containing states sent by other agents (sent at line 20) are processed. The received states are added to the $Open$ list. Note, that in MA-STRIPS a global state $s \subseteq \mathcal{L}$ is seen by sending agent α as a projection s^{α} and by receiving agent β as a projection s^{β}.

Algorithm 1 Multiagent Best-First Search

Input: Initial state s_i, goal $S_G \subseteq \mathcal{L}$, set of agent's actions α, heuristic estimator $\mathcal{H}$.

```
1:  Open ← {s_i}, Closed ← ∅
2:  while true do
3:    if Open ≠ ∅ then
4:      s ← poll(Open)
5:      if s ∉ Closed then
6:        Closed ← Closed ∪ {s}
7:        if s unifies with S_G then
8:          reconstruct the plan
9:        end if
10:       H(s, heuristicComputedCallback(s, h))
11:     end if
12:   end if
13:   process heuristic messages
14:   process search messages
15: end while
```

```
16: heuristicComputedCallback(s, h):
17: set the heuristic estimate of s to h
18: if h ≠ ∞ then
19:   if s obtained by a s.t. a ∈ α^pub then
20:     send state s
21:   end if
22:   Open ← Open ∪ expand(s, α)
23: end if
```

Distribution of Relaxation Heuristics

One general approach to compute heuristic estimates is to compute a solution of a *relaxed planning problem*. Such problems have some constraints removed (relaxed) in order to make it easier to solve them. One of well-known and thoroughly studied relaxations is obtained by removing the delete effects of all actions. Our motivation in this paper is to extend this concept to multiagent planning, therefore we will focus on classical delete relaxation heuristics: (i) inadmissible h_{add}, (ii) admissible h_{max} both (Bonet and Geffner 1999) and (iii) inadmissible h_{FF}, which was published in (Hoffmann and Nebel 2001). In the following subsections, we will present efficient multiagent distributions of those three heuristics.

Distribution of h_{add} and h_{max}

Both additive and max heuristics follow a very similar principle and are typically formalized as a set of recursive equations, such the following for h_{add}:

$$h_{add}(P,s) = \sum_{p \in P} h_{add}(p,s) \quad (1)$$

$$h_{add}(p,s) = \begin{cases} 0 & if\ p \in s \\ h_{add}(argmin_{a \in O(p)} [h_{add}(a,s)], s) & \text{otherwise} \end{cases} \quad (2)$$

$$h_{add}(a,s) = cost(a) + h_{add}(\mathsf{pre}(a), s), \quad (3)$$

where P is a set of propositions (i.e., goal or action preconditions), s is a state, a is an action and $O(p)$ is a set of actions which achieve p, formally $O(p) = \{a \in \alpha | p \in \mathsf{add}(a)\}$. The equations for h_{max} are the same except for Equation 1 where is a max function instead of sum, therefore everything we state about h_{add} applies similarly to h_{max}.

In the multiagent setting, some of the actions in the $argmin$ clause in Equation 2, where we are choosing the minimal cost action among actions achieving the proposition p, may be projections of other agent's public actions ($a \in \alpha \cup \alpha^{proj}$). Let α be the agent currently computing the heuristic estimate of the state s and β be some other agent. Let for some proposition p exist an action $a \in \beta$ s.t. $a^\alpha \in O(p)$. In such a case, there are two options how to handle the situation.

One option is to ignore the fact that the action is a projection and continue as if it was an ordinary action. This way, we may leave out some preconditions of the action (private to the owning agent), but we still get lower or equal estimate of the action cost (by including the private preconditions we can only increase the cost). We will denote the approach as a *projected heuristic*. Projected heuristics require no communication at all.

The other option is to always compute the true estimate. In order to do so, the agent α sends request $r = \langle a^\alpha, s \rangle$ to agent β to obtain true estimate of the cost of the action a^α. Upon receiving the request, agent β calls $h_{add}(\mathsf{pre}(a), s)$ and returns the result in a reply. It is obvious that in order to compute the heuristic estimate, agent β may need to send similar requests to other agents, or even back to agent α. This way, we end up with a distributed recursive function, which returns exactly the same results as a centralized h_{add} on a global problem $\Pi_G = \langle \mathcal{L}, \bigcup_{\alpha \in \mathcal{A}} \alpha, s_0, S_g \rangle$, since for every projection a^α of action $a \in \beta$, the true cost of a is obtained from the agent β.

A middle ground between the presented two extremes is to limit the recursion depth δ. If the *maximum recursion depth* δ_{max} is reached, all projected actions are evaluated without sending any further requests. This limit introduces another relaxation of the original problem where the interaction between agents is limited—the *agent coupling relaxation*. Such heuristic estimation is always lower or equal than would be the heuristic estimation in the global problem Π_G using a centralized heuristic estimator, because ignoring preconditions of an action in its projection can never increase the cost of the action . By limiting the recursion depth to $\delta_{max} = 0$, we return back to the *projected heuristic*, where all interactions between agents are relaxed away.

Relaxed exploration. Although the definition of h_{add} by a set of recursive equations is intuitively clear and provides good theoretical background, in practice, the recursive functions are typically not used. Recursive calls are limited by the call stack. Converting such recursion, where the recursive call is within a complex function $argmin$, into iteration is possible, but rather cumbersome. Instead, the idea of *relaxed exploration* is typically utilized.

The *relaxed exploration* is in fact a reachability analysis of the relaxed planning problem, which can be conveniently seen as building a *relaxed planing graph* (RPG). A relaxed planning graph is a layered (alternating fact and ac-

Algorithm 2 Distributed Relaxed Exploration

Input: Boolean flag r (true when first called), global exploration queue $\mathcal{Q}$

relaxedExploration(r)**:**

```
1: while Q ≠ ∅ do
2:   p ← poll(Q)
3:   if p ∈ goal and achieved(p') : ∀p' ∈goal then
4:     return
5:   end if
6:   O_p ← {a ∈ α ∪ α^proj | p ∈ pre(a)}
7:   for all a ∈ O_p do
8:     increment cost(p) by cost(a)
9:     if achieved(p') : ∀p' ∈ pre(a) then
10:      enqueueProposition(a, eff(a), r)
11:    end if
12:  end for
13: end while
```

Input: Action a, proposition p, Boolean flag r

enqueueProposition(a, p, r)**:**

```
14: if cost(p) = ⊥ or cost(p) > cost(a) then
15:   cost(p) ← cost(a)
16:   if r and a ∈ α^proj then
17:     send request message M_req = ⟨s, a, 0⟩
          to owner(a),
          process the reply by
            receiveReplyEnqueueCallback(p,_)
18:   else
19:     Q ← Q ∪ {p}
20:   end if
21: end if
```

Input: Heuristic estimate h, proposition p (set from enqueueProposition)

receiveReplyEnqueueCallback(p, h)**:**

```
22: if cost(p) > h then
23:   cost(p) ← h
24:   Q ← Q ∪ {p}
25: end if
26: relaxedExploration(false)
27: if no unresolved requests then
28:   return compute the total cost
29: end if
```

Algorithm 3 Request Processing

Input: Request message $M_{req} = \langle s, a, \delta \rangle$, where s is state, a action, δ recursion depth, β the sender

processRequest($M_{req} = \langle s, a, \delta \rangle$**,** β**):**

1: $\mathcal{Q} \leftarrow \{s\}$
2: relaxedExploration(false)
3: $h \leftarrow$compute the total cost
4: $P \leftarrow$mark public actions
5: send reply message $M_{re} = \langle h, P, \delta \rangle$ to β

Algorithm 4 Reply Processing

Input: Reply message $M_{re} = \langle h, P, \delta \rangle$, where h is heuristic estimate, P set of actions, δ recursion depth

processReply($M_{re} = \langle h, P, \delta \rangle$**):**

1: **if** $\delta < \delta_{max}$ **then**
2: $h_{sum} \leftarrow h$
3: **for all** $a \in P$ **do**
4: send request message $M_{req} = \langle s, a, \delta + 1 \rangle$ to $owner(a)$, process the reply by receiveReplyCallback(h)
5: **end for**
6: **end if**
7: receiveReplyEnqueueCallback(_,h)

Input: Heuristic estimate h

receiveReplyCallback(h**):**

8: $h_{sum} \leftarrow h_{sum} + h$
9: **if** all replies received **then**
10: receiveReplyEnqueueCallback(_,h_{sum})
11: **end if**

tion layers) directed graph. In its first layer it contains all facts which hold in the initial state, the next layer contains all actions of which preconditions are satisfied in the previous layer (and noop actions), the next layer contains all (add) effects of the actions from previous layer and so forth. In practice, a RPG is not built explicitly, but the exploration is achieved via effective representation we will refer to as an *exploration queue* (based on the Fast-Downward planning system (Helmert 2006)).

The *exploration queue* considers only *unary actions*—actions which have a single proposition as an add effect (any relaxed problem can be converted so it contains only unary actions). The *exploration queue* is supported by a data structure representing the precondition-of and achieved-by relations. The queue is initialized with the propositions which are true in the state s. Until the queue is empty, a proposition p is polled, it is checked whether p is a goal proposition and if so, whether all goals are satisfied. If not, for each action that depends on p ($p \in \mathsf{pre}(a)$ where $a \in \alpha \cup \alpha^{proj}$), the action cost is incremented by the cost of proposition p (that is either added for h_{add}, or maxed for h_{max}) and if there are no more unsatisfied preconditions of the action, the action is applied. The process is detailed in Algorithm 2, lines 1–13. Thanks to the sole use of *unary operators*, the application of an action a can be interpreted as adding the (only one) proposition $p = \mathsf{eff}(a)$ to the *exploration queue*, thus the procedure is named enqueueProposition (line 10).

The effectiveness of this approach lays in fact that during the relaxed exploration, cost estimates of facts and actions can be conveniently computed and once all goal facts are reached, the heuristic can be computed by simple sum or max of costs of all goal facts respectively.

Distributed relaxed exploration. An algorithm capable of building RPGs in a distributed manner was presented in (Štolba and Komenda 2013). The major drawback of the used approach was the necessity to build the RPG for each state by all agents at once, thus preventing the search to run independently in parallel. It was shown, that the resulting heuristic estimate is equal to the centralized estimate. In this paper, we will not place the requirement of obtaining the same value as in the centralized variant, which will allow us to build a much more efficient algorithm. The algorithm is based on building the *exploration queue* and requesting other agents when projections of their actions are encountered. Moreover the presented algorithm allows for precise control of the recursion depth and thus enables us to trade-off the estimation precision with the computation and communication complexity.

The basic process of building the *exploration queue* $\mathcal{Q}$ is similar to the centralized version as described in the previous section. The main principle of the distributed process is that whenever a projection of some other agent's action should be applied (and its effect added to the queue), a request is send to the owner of the action to obtain its true cost. The effect of the action is added to the queue only after the reply is received. Note, that when computing the reply, the agent may need to send requests as well, thus ending up with a distributed recursion. In order to effectively handle the recursion it is flattened so that all requests are sent by the initiator agent and the replies are augmented with the parameters of the next recursive call.

The exploration part of the algorithm is shown in Algorithm 2, whereas Algorithms 3 and 4 details the inter-agent communication. The entry point of the algorithm is the **relaxedExploration** procedure. First, it is invoked with the r parameter set to true, indicating, that whenever a projected action is encountered, a request is sent to its owner.

The main difference between the centralized and distributed approaches lays in the **enqueueProposition** procedure. If the cost of the action improves the current cost of the proposition, the cost of the proposition is set equal to the cost of the action, as usual, but if the the action $a \in \alpha^{proj}$ is a projection and sending of requests is enabled, i.e. $r = \mathsf{true}$, a request message $M_{req} = \langle s, a, \delta \rangle$, where s is the current state, a is the action and initial recursion depth $\delta = 0$, is sent to the owner of the action a, i.e. agent β. Otherwise, the proposition is added to the *exploration queue*.

Processing the messages. When the request message is received by the agent β (see Algorithm 3, **processRequest**), the *relaxed exploration* is run with the goal being the preconditions of the requested action a and without sending any requests, i.e., $r = \mathsf{false}$. After finishing the exploration, public actions which have contributed to the resulting heuristic estimate are determined (line 4). In principle, the procedure is similar to extracting a relaxed plan in the FF heuristic. A reply $M_{re} = \langle h, P, \delta \rangle$ is sent, where h is the computed heuristic value, P is the set of the contributing public actions and δ is the current recursion depth.

Receiving the reply from agent β is managed by procedure **processReply** in Algorithm 4. If the recursion depth has already reached the limit $\delta > \delta_{max}$, the original receiveReplyEnqueueCallback(p, h) from Algorithm 3 for action a is called, the cost estimate of proposition p is finalized and p is added to the *exploration queue*. Since the messaging process is asynchronous, the original *relaxed exploration* has already terminated, therefore it is started again (line 26), with the original data structures and with the newly evaluated proposition added to the queue. When the exploration is finished and there are no pending requests, the final heuristic estimate is computed depending on the actual heuristic (sum or max) and is returned via a callback to the search, so that the evaluated state can be expanded.

Otherwise, if $\delta \leq \delta_{max}$, agent α iterates through all actions $a' \in P$ and sends requests to their respective owners. The heuristic estimate received in each reply is added to the shared h_{sum}. When all replies are received (the replies undergo the same procedure, if there are any other public actions involved) and all costs are added together, again the receiveReplyEnqueueCallback(p, h) from Algorithm 2 is called with $h = h_{sum}$.

The **processReply** procedure stands for the distributed recursion, but the deeper recursive call is not called by the agent β, but the parameters of the recursion (the set of actions P which should be resolved next) are sent back to the initiator agent. This is rather an optimization to avoid having multiple heuristic evaluation contexts needed to handle multiple interwoven request/reply traces. Each context would need to have separate instance of the *exploration queue* data structure, which would present major inefficiency. Instead, the initiator agent is responsible for tracking the recursion and the replying agent only processes one reply at a time, locally, without sending any requests. Therefore, each agent needs to have only two instances of the *exploration queue*, one used to compute their own heuristic estimates (and possibly send requests and await replies), and one used to compute the local estimates for the replies.

Distribution of h_{FF}

The Fast-Forward h_{FF} heuristic is not directly based on estimation of the cost of actions in the relaxed problem, but on actually finding a plan solving the relaxed problem (a relaxed plan or RP). The heuristic is not typically described using recursive equations, but the implementation based on *relaxed exploration* can be easily reused. The difference is, that the evaluation does not end when the exploration is finished (all goal propositions have been reached), but continues with the relaxed plan extraction. The extraction of RP starts with the goal propositions and traverses the data structure towards the initial state, while marking the relaxed plan.

Since the algorithm is implementation-wise very similar to the h_{add} heuristic, one of the possible approaches to distribution of h_{FF} is to perform the distributed *relaxed exploration* exactly as in h_{add} and simply add RP extraction routine at the end of the heuristic evaluation (as part of the total cost computation). Another approach was introduced in (Štolba and Komenda 2013) as lazy multiagent FF heuristic h_{lazyFF}, which we have adopted and compared with the previously described approach and both additive and max heuristics.

Our version of the lazy FF algorithm starts by building a local *exploration queue*. When all goal propositions are reached, a relaxed plan π' is extracted. For all actions $a \in \pi'$, which are projections $a \in \alpha^{proj}$, request message $M_{req} = \langle s, a, \delta \rangle$ is sent to the owning agent $\beta = owner(a)$ of the action a. When agent β receives the request, he constructs a local relaxed plan from state s (by local *relaxed exploration* and local RP extraction without sending any requests), satisfying the preconditions (both public and private) of the action a. Then, agent β sends a reply $M_{re} = \langle h, P, \delta \rangle$, where h is the length of the relaxed plan and P is a set of projected actions contained in the plan. When the reply is received by agent α, the algorithm iterates through all actions $a' \in P$ and sends requests to their respective owners. Each of the requests undergo the same procedure as the original request, adding the returned heuristic estimates to the resulting h_{sum}. When all requests are processed, h_{sum} is added to the length of the local relaxed plan of agent α and returned via callback as the heuristic estimate of state s.

The recursion depth of the heuristic estimate can be limited in a similar manner as in the add/max heuristics. Whenever a request should be sent and the maximum recursion limit δ_{max} has been reached, the request is not sent and the possible relaxed sub-plan is ignored.

Experiments

To analyze properties of the proposed distributed heuristics and their implementations, we have prepared a set of experiments covering various efficiency aspects of the heuristics.

All experiments were performed on FX-8150 8-core processor at 3.6GHz, each run limited to 8GB of RAM and 10 minutes. Each measurement is a mean from 5 runs. We have used the translation to SAS+ formalism and preprocessing from the Fast-Downward planning system (Helmert 2006) and new implementation of the search and heuristic estimators.

Initial comparison

The first batch of experiments focused on two classical planning metrics used in comparison of heuristic efficiency: planning time t and number of explored states e. Those metrics were supplied by a multiagent metric of communicated bytes b among the agents during the planning process. Used planning problems stem from IPC domains modified for multiagent planning as presented, e.g., in (Nissim and Braf-

prob. ($\|\mathcal{A}\|$)	δ_{max}	h_{FF} t[s]	h_{FF} e[k-]	h_{FF} b[MB]	h_{add} t[s]	h_{add} e[k-]	h_{add} b[MB]	h_{max} t[s]	h_{max} e[k-]	h_{max} b[MB]	h_{lazyFF} t[s]	h_{lazyFF} e[k-]	h_{lazyFF} b[MB]
Rov8 (4)	1	1.2	0.4	0.5	1.2	0.5	0.7	**1.1**	0.4	0.5	–	–	–
Rov8 (4)	∞	**1.1**	0.5	0.7	1.2	0.6	0.8	**1.1**	0.5	0.6	–	–	–
Rov12 (4)	0	70.9	4617.3	35.7	40.7	3939.5	31.2	–	–	–	57	4583.5	35.6
Rov12 (4)	1	1.2	0.8	0.7	1.3	0.3	0.2	**1.1**	0.3	0.3	–	–	–
Rov12 (4)	∞	**1.1**	0.4	0.4	1.2	0.7	0.6	1.2	0.3	0.3	–	–	–
Rov14 (4)	1	21	112.1	157.4	18.7	101.9	136.6	21.9	127.2	170.6	–	–	–
Rov14 (4)	∞	23	124.7	175.1	19.2	98.5	138.7	**16.1**	90.6	127.6	–	–	–
Sat9 (5)	1	3.2	6.2	14.7	3.2	7.8	17.7	3.1	6.9	15.6	3.1	9.8	10.3
Sat9 (5)	∞	3.7	10.6	25	3.5	7.4	17.4	3.3	6.2	14.6	**3**	7.4	8.4
Sat10 (5)	1	3.7	4.1	9.6	**3.4**	3.2	7.1	3.4	2.4	5.3	4.1	13.5	6.6
Sat10 (5)	∞	3.7	4.3	9.9	4	8.2	19	3.3	2.3	5.3	3.9	7.8	6
Sat* (14)	1	69.2	9.3	36.9	68	8.7	33.4	69	9	34.8	**60.6**	9	17.5
Sat* (14)	∞	69	9.3	36.7	68.5	9.3	37	68.5	9	35.7	61.3	9.7	18.4
Sat* (16)	1	133.8	11.1	57.5	136.6	11.7	59.4	133.6	10.7	54.1	126.4	12.8	31
Sat* (16)	∞	132.7	11.1	57.8	137.3	12.4	64.6	133.5	11.1	57.8	**124.5**	12.3	28.8
Log* (6)	0	**0.7**	6.8	0	0.7	5.7	0	0.7	7	0	**0.7**	7.2	0
Log* (6)	1	1.2	0.5	0.3	1.6	1.3	0.6	1.2	0.7	0.3	1.6	5.4	0.6
Log* (6)	∞	1.1	0.4	0.2	1.3	0.8	0.5	1.2	0.5	0.3	1.4	0.6	0.8
CP* (6)	0	1.8	136.1	2.2	1.6	99.5	1.7	2.6	351.5	5.5	1.8	137.7	2.2
CP* (6)	1	1.7	127.5	2.1	7.3	72.1	51.6	10	100.2	71.8	5.6	43.8	66.1
CP* (6)	∞	1.7	122.5	2	**1.4**	76.2	1.3	2.6	352.6	5.4	44.3	91.8	973.5
CP* (7)	0	2.2	183.2	2.4	2.2	223.1	3.2	6.3	1252.5	15.7	2.3	205	2.7
CP* (7)	1	**1.9**	162.1	2.2	18.3	248.1	150.8	35.5	451.4	274.5	50.4	371.2	738.6
CP* (7)	∞	2	188.9	2.5	2.1	225.2	3.2	6.3	1255.6	15.5	160.9	249.5	249.5
Sok* (2)	0	1.6	8.5	0.5	1.5	7.7	0.5	1.7	11.7	0.7	1.6	8.8	0.6
Sok* (2)	1	1.5	7.6	0.5	17.1	22	66.8	3.9	4.3	12.1	4.3	12.9	24.1
Sok* (2)	∞	1.5	8.3	0.5	**1.4**	7.8	0.5	1.6	11.7	0.7	–	–	–

Table 1: Comparison of all presented heuristics for selected problems and metrics. The planning time metrics t is in seconds, the explored states e in thousands of states and the communicated information b in megabytes. Abbreviations: Rov = rovers, Sat = satellites, Log = logistics, CP = cooperative path-finding, Sok = sokoban.

man 2012). The problems with * in their names were either based on IPC domains, but simplified, or other state-of-the-art multiagent benchmarks, e.g., from (Komenda, Novák, and Pěchouček 2013). The recursion depth was limited to three values $\delta_{max} = \{0, 1, \infty\}$ as other settings of δ_{max} showed similar results. Missing rows were not successfully planned with any of the tested heuristics.

The results are summarized in Table 1. No single heuristic and δ_{max} dominates the other ones. In Rovers, the most successful seems to be h_{max}. In Satellites, h_{lazyFF} performs well, but in other domains it does not solve some problems at all. The Logistics is dominated by h_{FF} and in Cooperative Path-Finding and Sokoban, the best are h_{FF} and h_{add}.

Problem coverage

In this experiment, we have evaluated the coverage of all the described heuristics (h_{add},h_{max},h_{FF} and h_{lazyFF}) with the maximum recursion depth δ_{max} set to 0, 1, 2, 4 and ∞. The coverage has been evaluated over two sets of benchmarks. First set consists of 40 specifically multiagent problems, which are typically not that combinatorially hard, but contain more agents (taken from (Štolba and Komenda 2013) and (Komenda, Novák, and Pěchouček 2013)). Second set consists of 21 problems converted directly form IPC benchmarks (as in (Nissim and Brafman 2012)), which are typically much combinatorially harder, but with less agents. The results are summarized in Table 2.

The results show clear dominance of h_{FF}, but interestingly the other distribution approach of the Fast-Forward heuristic, h_{lazyFF}, is on the other side of the spectrum. This is most probably because one of the biggest strengths of the FF heuristic, compared to other delete relaxation heuristics used here, is that it does not suffer from *over-counting* (one action is included in the estimate several times) thanks to the explicit relaxed plan extraction. In the h_{lazyFF}, we partially lose this advantage, because when sending reply, only the length of the plan is sent. Therefore, single action can be included several times in multiple replies from single agent, or even multiple agents.

Another observation is that the setting of $\delta_{max} = 0$ is dominated by other values. This may be due to the choice of the domains, the effect of various δ_{max} settings is thoroughly analyzed in the next set of experiments. Also, various settings of δ_{max} for $\delta_{max} > 0$ affect the coverage only marginally.

We can also state, that in the terms of coverage, the use of distributed heuristics compares favorably to the state of the art, as in (Nissim and Brafman 2012), the planner using

δ_{max}	0	1	2	4	∞
h_{FF}	35 /7	38/15.4	38 /15	38 /14.4	38 /15
h_{add}	35 /7	38 /14	38 /14	38 /14	38 /14
h_{max}	35 /3.2	38 /14	38 /14	38 /14	38 /14
h_{lazyFF}	35.2 /6.8	38 /8	36.2 /8	36.5 /8	36.8 /8

Table 2: Coverage for various heuristics and recursions depth δ_{max}. The results are in the form of multiagent domains / IPC domains.

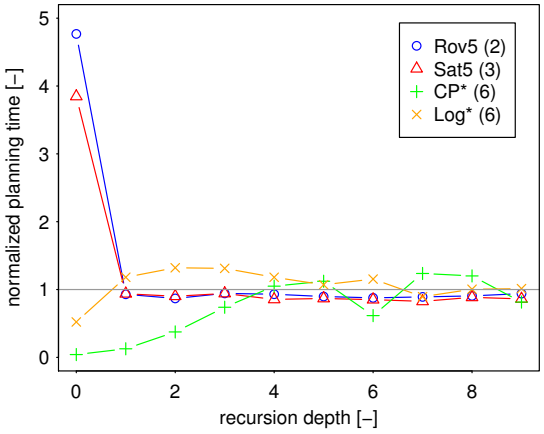

Figure 1: Planning time normalized to result for of $\delta_{max} = \infty$ for h_{lazyFF} heuristics.

projected heuristics LM-cut and Merge&Shrink has a coverage of 11 resp. 12 problems, which was exceeded by all distributed heuristic with $\delta_{max} > 0$, except for h_{lazyFF}. But it is important to emphasize, that the comparison is not completely fair, as the approach used in (Nissim and Brafman 2012) is optimal and although we use the admissible h_{max} heuristic, our approach is not optimal. To do so requires special handling of the search termination, as detailed in (Nissim and Brafman 2012).

Effect of the recursion depth

In the following set of experiments, we have evaluated the effect of changing the *maximal recursion depth* δ_{max} on the speed and communication requirements of the planning process. The data set was measured on four selected domains with varied couplings (rovers, satellites, cooperative path-finding and logistics), each represented by a single problem. The *maximal recursion depth* ranged from 0 (a *projected heuristic*) to 9, for comparison, the results were normalized against the result of run with $\delta_{max} = \infty$.

By coupling, we understand the concept formalized in (Brafman and Domshlak 2008), which can be rephrased as "the more interactions must take place among the agents in order to solve the problem, the more coupled the problem is"—at one extreme there are problems, where all actions interact with other agents (containing only public actions) meaning *full coupling*. In problems of the other extreme, the agents can solve their individual problems without any interaction. Because of our decision to treat all goals as public, we cannot achieve *full decoupling*—at least goal-achieving actions are public and thus causing some level of coupling. The experimental domains were chosen such that rovers and satellites are loosely coupled. In satellites, only the assumption that all goal-achieving actions are public introduces some coupling, in rovers, there are also interacting preconditions among the goal-achieving actions. Logistics is moderately coupled (private movement of agents and public handling of packages) and cooperative path-finding is fully coupled.

The experimental results for the h_{lazyFF} heuristic are plotted in Figures 1 and 2. In the fully coupled cooperative path-finding, the results are best for $\delta_{max} = 0$ and are converging to the results for $\delta_{max} = \infty$ as δ_{max} grows. This is because in a fully coupled problem, all actions are public and in cooperative path-finding all their preconditions and effects are also public (which does not have to be always the case). Therefore each agent has complete information about the problem in form of the action projections ($a^{\alpha} = a$ for all actions and agents) and the *projected heuristic* gives perfect estimate (the same as would global heuristic give). For $\delta_{max} > 0$, requests are sent for every projected action, causing more communication and computation without bringing any improvement to the heuristic estimate.

Completely different picture give the results for the loosely coupled problems. The results are significantly worse for $\delta_{max} = 0$, from $\delta_{max} = 1$ they are practically equal to $\delta_{max} = \infty$. The solution of those problems typically consist of long private parts finished by a singe public action (the goal achieving action). When estimated by a *projected heuristic*, the private parts of other agents get ignored and the estimates are thus much less informative. Even the fact, that when a state is expanded by a public action, it is sent with the original agent's heuristic estimate, does not help, because estimation of states expanded further from such state ignore the information again. But even $\delta_{max} = 1$ is enough to resolve this issue.

Lastly, in the logistics problem, the $\delta_{max} = 0$ estimates are rather good (but not as good as in the cooperative path-finding) and with growing δ_{max}, the results converge towards $\delta_{max} = \infty$, but for $0 < \delta_{max} < \infty$ the results are slightly worse. This may suggest, that as the coupling is moderate, it is best either to fully exploit the coupled part of the problem and use *projected heuristics*, or to rely on the decoupled part of the problem and employ the full recursion approach, depending on the exact balance.

The results for communication are in Figure 2. The left chart compares the total bytes communicated and shows the same tendencies as the planning time in Figure 1. In fact, limiting the interactions may lead to increased communication. The right table shows the data for heuristic requests, there we see the expected result for $\delta_{max} = 0$, where no requests are sent, otherwise the tendencies are surprisingly similar. This indicates, that the communication complexity is dominated by the search communication complexity (the longer the search takes, the more messages are passed).

Presented results suggest, that for tightly coupled problems, sharing of the information is not only less important,

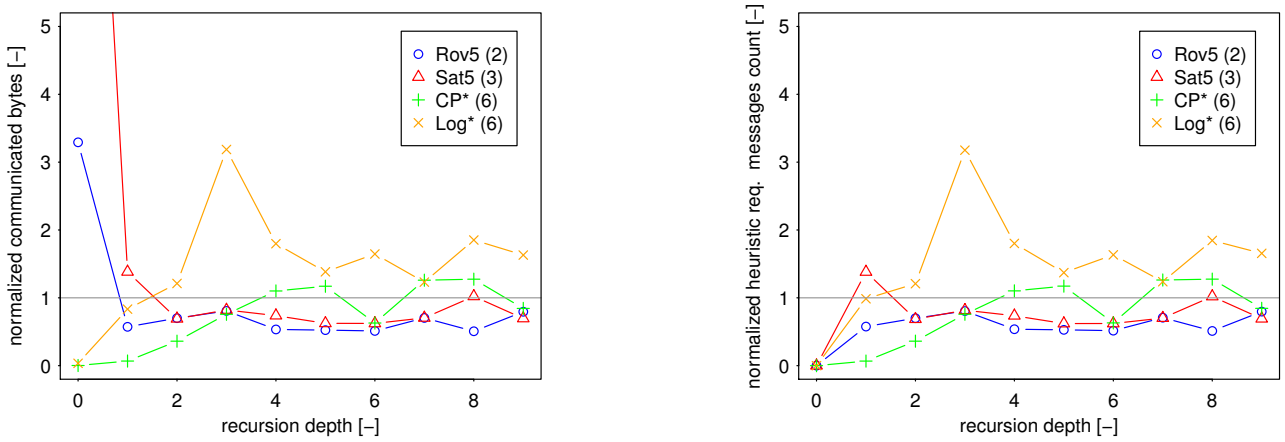

Figure 2: Communicated bytes and heuristic message requests normalized to $\delta_{max} = \infty$ for h_{lazyFF} heuristics.

because the agents have most of the information in their problem projections, but may even lower the effectiveness by redundant communication. For loosely coupled problems, the communication is vital, even if the communication is very limited. For moderately coupled problems, both extremes are equally good. In general, it is hard to determine, which approach will yield the best results, but it is sensible to choose from either no communication $\delta_{max} = 0$, full communication $\delta_{max} = \infty$, or even communication limited to very low recursion depth limits, i.e., $\delta_{max} = 1$. If we can expect some properties of the problems at hand, we can suggest preferred approach much easier—if we are *not* expecting loosely coupled problems, $\delta_{max} = 0$ is the best choice, for *no* tightly coupled problems $\delta_{max} = \infty$ and for *no* moderately coupled problems, $\delta_{max} = 1$ seems to be the best choices.

The results in the presented figures are for h_{lazyFF} mainly because they are the most illustrative, other heuristics follow the same patterns as described here.

Final Remarks

We have proposed an efficient distribution approach for three classical delete-relaxation heuristics h_{add}, h_{max} and h_{FF}. The heuristics were experimentally compared in a planner utilizing a mutliagent Best-First Search on various multiagent planning problems stemming from classical IPC domains. The comparison comprised metrics of planning time and communication and expanded states, coverage comparison and comparison of the effect of changing maximum recursion depth.

The results did not show any single heuristic to be dominating others, but brought to light interesting and rather unintuitive conclusion. For tightly coupled problems, it is more efficient to limit the information sharing, whereas loosely coupled problems strongly benefit form the distributed heuristic estimation.

Acknowledgments This research was supported by the Czech Science Foundation (13-22125S), A. Komenda was supported in part by *USAF EOARD* (FA8655-12-1-2096), in part by a Technion fellowship.

References

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In Biundo, S., and Fox, M., eds., *ECP*, volume 1809 of *Lecture Notes in Computer Science*, 360–372. Springer.

Borrajo, D. 2013. Plan sharing for multi-agent planning. In *Proc. of DMAP Workshop of ICAPS'13*, 57–65.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of ICAPS'08*, 28–35.

Crosby, M.; Rovatsos, M.; and Petrick, R. 2013. Automated agent decomposition for classical planning. In *Proceedings of ICAPS'13*.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of ICAPS'09*.

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

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. 14:253–302.

Komenda, A.; Novák, P.; and Pěchouček, M. 2013. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications*.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In *Proceedings of AAMAS'12*, 1265–1266.

Štolba, M., and Komenda, A. 2013. Fast-forward heuristic for multiagent planning. In *Proc. of DMAP Workshop of ICAPS'13*, 75–83.

Torreño, A.; Onaindia, E.; and Sapena, O. 2013. Fmap: a heuristic approach to cooperative multi-agent planning. In *Proc. of DMAP Workshop of ICAPS'13*, 84–92.