Multiagent Planning Supported by Plan Diversity Metrics and Landmark Actions

Jan Tožička[1], Jan Jakubův[1], Karel Durkota[1], Antonín Komenda[2] and Michal Pěchouček[1]
[1]*Agent Technology Center, Department of Computer Science, Czech Technical University, Prague, Czech Republic*
[2]*Technion - Israel Institute of Technology, Haifa, Israel*

Keywords: Multiagent Planning, Diverse Planning, Planning with Landmarks.

Abstract: Problems of domain-independent multiagent planning for cooperative agents in deterministic environments can be tackled by a well-known initiator–participants scheme from classical multiagent negotiation protocols. In this work, we use the approach to describe a multiagent extension of the Generate-And-Test principle distributively searching for a coordinated multiagent plan. The generate part uses a novel plan quality estimation technique based on metrics borrowed from the field of diverse planning. The test part builds upon planning with landmarks by compilation to classical planning. Finally, the proposed multiagent planning approach was experimentally analyzed on one newly designed domain and one classical benchmark domain. The results show what combination of plan quality estimation and diversity metrics provide the best planning efficiency.

1 INTRODUCTION

Multiagent planning is a specific form of distributed planning and problem solving, which was summarized by (Durfee, 1999). Multiagent planning research and literature focused mostly on the *coordination* part of the problem while the *synthesis* part dealing with a particular ordering of actions was studied in the area of classical planning.

The coordination was, for instance, studied in well-known General Partial Global Planning by (Decker and Lesser, 1992) or with additional domain-specific information as TALPlanner by (Doherty and Kvarnström, 2001). The first fusion of the coordination and synthesis parts for domain-independent multiagent planning with deterministic actions was proposed by (Brafman and Domshlak, 2008). The approach was based on the classical planning formalism STRIPS (Fikes and Nilsson, 1971) extended to multiagent settings denoted as MA-STRIPS. (Brafman and Domshlak, 2008) also proposed a solution for the coordination part of the problem by translation to a Distributed Constraint Satisfaction Problem (DCSP). Since the paper was focused primarily on theoretical analysis of computational complexity of MA-STRIPS problems, several algorithmic approaches appeared later in other papers, e.g., in (Nissim et al., 2010) or (Torreño et al., 2012).

In this paper, we propose a novel algorithmic approach to multiagent planning for problems described in MA-STRIPS based on the principle of classical multiagent negotiation protocols as Contract Net with one agent acting as an initiator and the rest acting as participants. The approach can be seen as a protocol describing distribution of the *Generate-And-Test Search* which was a base principle also in multiagent planners described by (Nissim et al., 2010) (using DCSP for the coordination part and classical planner for the generation part) and by (Pellier, 2010) (using backtracking search for the coordination part and planning graphs for the generation part).

The contribution of our work is in the way how plan candidates are generated and tested. Our generative process uses estimation of quality of generated plans based on metrics of diverse planning (particularly from (Bhattacharya et al., 2010) and (Srivastava et al., 2007)). In other words, the idea is to generate good-quality plans and avoid low-quality ones. The quality measure is based on the history of answers of the participants who were trying to extend the initial plan. Therefore it can be understood as a learning of the initiator agent to generate plan candidates which can be more likely extended by more participant agents to a final solution.

The testing part utilizes planning with landmarks (similarly as used by (Nissim et al., 2010)). The difference is that we translate a planning problem with landmarks into an ordinary planning problem, which

Tožička J., Jakubův J., Durkota K., Komenda A. and Pěchouček M..
Multiagent Planning Supported by Plan Diversity Metrics and Landmark Actions.
DOI: 10.5220/0004918701780189
In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence* (ICAART-2014), pages 178-189
ISBN: 978-989-758-015-4
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

can be then solved by a classical planner. Usually, landmarks are incorporated into planners as special heuristics as in (Richter and Westphal, 2010). However, our translation enables a straightforward incorporation of externally defined landmarks, which is required by the proposed planning protocol.

Finally, we provide experimental evaluation of the planner on a newly designed planning domain *tools* and *rovers* planning domain from International Planning Competition extended for multiagent planning.

2 PLANNING MODEL

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 environment wherein the agents act is *classical* with *deterministic* actions. The following formal preliminaries compactly restate the MA-STRIPS problem (Brafman and Domshlak, 2008) required for the following sections.

2.1 Planning Problem

An MA-STRIPS planning problem $\mathcal{P}$ is defined as a quadruple $\mathcal{P} = \langle P, \mathcal{A}, I, G \rangle$, where P is a set of propositions or facts, $\mathcal{A}$ is a set of *agents*, I is an initial state and G is a set of goals. We use α and β to range over agents in $\mathcal{A}$.

An *action* an agent can perform is a triple $a = \langle a_{\mathsf{pre}}, a_{\mathsf{add}}, a_{\mathsf{del}} \rangle$ of subsets of P, where a_{pre} is the set of preconditions, a_{add} is the set of add effects, and a_{del} is the set of delete effects. We define functions $\mathsf{pre}(a)$, $\mathsf{add}(a)$, and $\mathsf{del}(a)$ such that for any action a it holds $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$. Moreover let $\mathsf{eff}(a) = \mathsf{add}(a) \cup \mathsf{del}(a)$.

The set $\mathcal{A}$ contains agents. We identify an *agent* with its capabilities, that is, an agent $\alpha = \{a_1, \ldots, a_n\}$ is characterized by a finite repertoire of actions it can preform in the environment. A *state* $s = \{p_1, \ldots, p_m\} \subseteq P$ is a finite set of facts and we say that p_i holds in s. When no confusion can arise, we use $\mathcal{A}$ also to denote the set of all actions of $\mathcal{P}$, that is, when we write $a \in \mathcal{A}$ then $\mathcal{A}$ is to be considered as a shortcut for $\bigcup \mathcal{A}$.

Example 1. *We shall demonstrate definitions of this section on a simple logistic problem involving three locations Prague, Brno, Ostrava, and a Crown to be delivered from Prague to Ostrava. A Plane can travel from Prague to Brno and back. Similarly, a Truck provides connection between Brno and Ostrava.*

The set of facts P contains (1) facts to describe positions of Plane and Truck like Plane-at-Prague and Truck-at-Ostrava, and (2) facts to describe position of the Crown like Crown-in-Brno and Crown-in-Truck. The initial state and the goal are given as follows.

$I = \{\mathit{Plane\text{-}at\text{-}Prague}, \mathit{Truck\text{-}at\text{-}Brno}, \mathit{Crown\text{-}in\text{-}Prague}\}$
$G = \{\mathit{Crown\text{-}in\text{-}Ostrava}\}$

Agents can execute actions to:

1. *load and unload the Plane or the Truck like* $\mathit{load}_{\mathit{Plane@Prague}}$ *and* $\mathit{unload}_{\mathit{Truck@Ostrava}}$*. The action* $\mathit{load}_{\mathit{Plane@Prague}}$ *has preconditions Plane-at-Prague and Crown-in-Prague, one add effect Crown-in-Plane and it deletes Crown-in-Prague. Other actions are defined similarly.*
2. *fly the Plane and drive the Truck between allowed destinations like* $\mathit{fly}_{\mathit{Brno}\to\mathit{Prague}}$ *and* $\mathit{drive}_{\mathit{Brno}\to\mathit{Ostrava}}$*. For example,* $\mathit{drive}_{\mathit{Brno}\to\mathit{Ostrava}}$ *has precondition Truck-at-Brno and it adds Truck-at-Ostrava while removing Truck-at-Brno.*

Agent Plane is defined as being capable of executing following actions.

$$\mathit{Plane} = \{\ \mathit{fly}_{\mathit{Prague}\to\mathit{Brno}}, \mathit{fly}_{\mathit{Brno}\to\mathit{Prague}}, \mathit{load}_{\mathit{Plane@Prague}}, \mathit{load}_{\mathit{Plane@Brno}}, \mathit{unload}_{\mathit{Plane@Prague}}, \mathit{unload}_{\mathit{Plane@Brno}}\ \}$$

Agent Truck is defined similarly. Agent set $\mathcal{A}$ *is then simply* $\{\mathit{Plane}, \mathit{Truck}\}$.

2.2 Problem Projections

MA-STRIPS problems distinguish between *public* and *internal* facts and actions. Let $\mathsf{facts}(a) = \mathsf{pre}(a) \cup \mathsf{add}(a) \cup \mathsf{del}(a)$ and similarly $\mathsf{facts}(\alpha) = \bigcup_{a \in \alpha} \mathsf{facts}(a)$. An α-internal and public subset of all facts P, denoted $P^{\alpha\text{-int}}$ and P^{pub} respectively, are subsets of P such that the following hold.

$$\begin{aligned} P^{\text{pub}} &\supseteq \bigcup_{\alpha \neq \beta} (\mathsf{facts}(\alpha) \cap \mathsf{facts}(\beta)) \\ P^{\alpha\text{-int}} &= \mathsf{facts}(\alpha) \setminus P^{\text{pub}} \\ P^{\alpha} &= P^{\alpha\text{-int}} \cup P^{\text{pub}} \end{aligned}$$

The set P^{pub} contains all the facts that are used in actions of at least two different agents. The set can possibly contain also other facts, that is, some facts mentioned in actions of one agent only. This definition of public facts differs from other definitions in literature (Brafman and Domshlak, 2008) where P^{pub} is defined using equality instead of superset ($\supseteq$), i.e., our definition gives partial freedom what is treated as public. Our definition allows us to experiment with extensions of the set of public facts. For the purpose of this paper, however, the definition with equality can be considered without any effect on our results. We suppose that P^{pub} is an arbitrary but fixed set which

satisfies the above condition. Set $P^{\alpha\text{-int}}$ of α-internal facts contains facts mentioned only in the actions of agent α, but possibly not all of them. The set P^{α} contains facts *relevant* to agent α.

Example 2. *In our running example the set* facts(*Plane*) *contains Plane-at-Prague, Plane-at-Brno, Crown-in-Prague, Crown-in-Plane, and Crown-in-Brno. The only fact shared by the two agents is Crown-in-Brno but later on we will require also* $G \subseteq P^{\text{pub}}$ *so we have the following.*

$$\begin{aligned} P^{\text{pub}} &= \{\mathit{Crown\text{-}in\text{-}Brno}, \mathit{Crown\text{-}in\text{-}Ostrava}\} \\ P^{\mathit{Plane}\text{-int}} &= \{\mathit{Plane\text{-}at\text{-}Prague}, \mathit{Plane\text{-}at\text{-}Brno} \\ &\qquad \mathit{Crown\text{-}in\text{-}Prague}, \mathit{Crown\text{-}in\text{-}Plane}\} \\ P^{\mathit{Plane}} &= P^{\text{pub}} \cup P^{\mathit{Plane}\text{-int}} \end{aligned}$$

The set P^{Truck} *is defined appropriately.*

The *projection* a^{α} of action a to agent α is an action defined as follows.

$$a^{\alpha} = \langle \mathsf{pre}(a) \cap P^{\alpha}, \mathsf{add}(a) \cap P^{\alpha}, \mathsf{del}(a) \cap P^{\alpha} \rangle$$

Example 3. *In our example we can compute the below action projections. To save space we write* $(\mathit{fly}_{\mathit{Prague}\to\mathit{Brno}})^{\mathit{Plane}}$ *as* $\mathit{fly}^{\mathit{Plane}}_{\mathit{Prague}\to\mathit{Brno}}$ *and so on.*

$$\begin{aligned} &\mathit{fly}^{\mathit{Plane}}_{\mathit{Prague}\to\mathit{Brno}} = \mathit{fly}_{\mathit{Prague}\to\mathit{Brno}} \\ &\mathit{fly}^{\mathit{Truck}}_{\mathit{Prague}\to\mathit{Brno}} = \langle \emptyset, \emptyset, \emptyset \rangle \\ &\mathit{load}^{\mathit{Plane}}_{\mathit{Truck@Brno}} = \langle \{\mathit{Crown\text{-}in\text{-}Brno}\}, \emptyset, \{\mathit{Crown\text{-}in\text{-}Brno}\} \rangle \\ &\mathit{unload}^{\mathit{Plane}}_{\mathit{Truck@Ostrava}} = \langle \emptyset, \{\mathit{Crown\text{-}in\text{-}Ostrava}\}, \emptyset \rangle \end{aligned}$$

The set α^{pub} of *public actions* of agent α is defined as $\alpha^{\text{pub}} = \{a \mid a \in \alpha, \mathsf{eff}(a) \cap P^{\text{pub}} \neq \emptyset\}$, and the set α^{int} of *internal actions* of agent α as $\alpha^{\text{int}} = \alpha \setminus \alpha^{\text{pub}}$. The set $\mathcal{A}^{\text{pub}}$ of *all public actions* of problem $\mathcal{P}$ is defined as $\mathcal{A}^{\text{pub}} = \bigcup_{\alpha\in\mathcal{A}} \alpha^{\text{pub}}$, and the set $\mathcal{A}^{\alpha}$ of all actions *relevant* to agent α is $\mathcal{A}^{\alpha} = \alpha^{\text{int}} \cup \{a^{\alpha} | a \in \mathcal{A}^{\text{pub}}\}$. Note that $a^{\alpha} = a$ for any $a \in \alpha$. Hence in the definition of $\mathcal{A}^{\alpha}$ we do not need to project internal actions, and the only actions which are effected by α-projection are public actions of agents other than α.

Example 4. *In our example we have the following public and relevant actions.*

$$\begin{aligned} \mathit{Plane}^{\text{pub}} &= \{\ \mathit{load}_{\mathit{Plane@Brno}}, \mathit{unload}_{\mathit{Plane@Brno}}\ \} \\ \mathcal{A}^{\mathit{Plane}} &= \{\ \mathit{fly}_{\mathit{Prague}\to\mathit{Brno}}, \mathit{fly}_{\mathit{Brno}\to\mathit{Prague}}, \\ &\qquad \mathit{load}_{\mathit{Plane@Prague}}, \mathit{unload}_{\mathit{Plane@Prague}}, \\ &\qquad \mathit{load}^{\mathit{Plane}}_{\mathit{Plane@Brno}}, \mathit{unload}^{\mathit{Plane}}_{\mathit{Plane@Brno}}, \\ &\qquad \mathit{load}^{\mathit{Plane}}_{\mathit{Truck@Brno}}, \mathit{unload}^{\mathit{Plane}}_{\mathit{Truck@Brno}}, \\ &\qquad \mathit{load}^{\mathit{Plane}}_{\mathit{Truck@Ostrava}}, \mathit{unload}^{\mathit{Plane}}_{\mathit{Truck@Ostrava}}\ \} \end{aligned}$$

Note that $\mathcal{A}^{\mathit{Plane}}$ *has ten actions while* $\mathcal{A}^{\mathit{Truck}}$ *has only eight because* $\mathit{load}_{\mathit{Plane@Prague}}$ *and* $\mathit{unload}_{\mathit{Plane@Prague}}$ *are private for Plane.*

In a MA-STRIPS problem $\mathcal{P}$, all the agents operate on a shared global state. The projection $\mathcal{P}^{\alpha}$ of a problem $\mathcal{P}$ to agent α is a classical STRIPS problem where an agent has an internal copy of the global state. Previously defined relevant actions $\mathcal{A}^{\alpha}$ contain (1) internal actions of agent α, (2) public actions of α, and (3) projections of public actions of other agents which emulate effects of external actions on the internal state. Projection $\mathcal{P}^{\alpha}$ of $\mathcal{P}$ is defined as follows.

$$\mathcal{P}^{\alpha} = \langle P^{\alpha}, \mathcal{A}^{\alpha}, I \cap P^{\alpha}, G \rangle$$

Example 5. *In our example we have*

$$\begin{aligned} I \cap P^{\mathit{Plane}} &= \{\mathit{Plane\text{-}at\text{-}Prague}, \mathit{Crown\text{-}in\text{-}Prague}\} \\ \mathcal{P}^{\mathit{Plane}} &= \langle P^{\mathit{Plane}}, \mathcal{A}^{\mathit{Plane}}, I \cap P^{\mathit{Plane}}, G \rangle \end{aligned}$$

Projection $\mathcal{P}^{\mathit{Truck}}$ *is defined similarly.*

In the rest of this paper we consider only problems where all the facts of the goal state G are public, that is, $G \subseteq P^{\text{pub}}$ which is common in literature (Nissim and Brafman, 2012). This assures that any agent is able to find its local solution fulfilling the goal if it is satisfiable. Then it is up to the agent negotiation to extend this local solution to a valid plan. Moreover we suppose that two different agents do not execute the same action, that is, we suppose that the sets α_i are pairwise disjoint (Brafman and Domshlak, 2008).

2.3 Plans and Solutions

A *plan* π is a sequence of actions $\langle a_1, \ldots, a_k \rangle$. A plan π defines an order in which the actions are executed by their unique owner agents. It is supposed that independent actions can be executed in parallel. A plan π is called a *solution* of $\mathcal{P}$ when it contains actions from $\mathcal{A}$ and a sequential execution of the actions from π by their respective owners transforms the initial state I to a state which is a subset of G. Let $\mathsf{sol}(\mathcal{P})$ denote the set of all solutions of MA-STRIPS problem $\mathcal{P}$. Similarly, let $\mathsf{sol}(\mathcal{P}^{\alpha})$ denote the set of all solutions of the classical STRIPS problem $\mathcal{P}^{\alpha}$.

Example 6. *Let us consider the following plans.*

$$\begin{aligned} \pi_0 &= \langle \mathit{load}_{\mathit{Plane@Prague}}, \mathit{fly}_{\mathit{Prague}\to\mathit{Brno}}, \mathit{unload}_{\mathit{Plane@Brno}}, \\ &\qquad \mathit{load}_{\mathit{Truck@Brno}}, \mathit{drive}_{\mathit{Brno}\to\mathit{Ostrava}}, \mathit{unload}_{\mathit{Truck@Ostrava}} \rangle \\ \pi_1 &= \langle \mathit{unload}^{\mathit{Plane}}_{\mathit{Truck@Ostrava}} \rangle \\ \pi_2 &= \langle \mathit{unload}^{\mathit{Truck}}_{\mathit{Plane@Brno}}, \mathit{load}^{\mathit{Truck}}_{\mathit{Truck@Brno}}, \\ &\qquad \mathit{drive}_{\mathit{Brno}\to\mathit{Ostrava}}, \mathit{unload}^{\mathit{Truck}}_{\mathit{Truck@Ostrava}} \rangle \end{aligned}$$

It is easy to check that π_0 *is a solution of our example* MA-STRIPS *problem* $\mathcal{P}$. *Plan* π_1 *is a solution of projection* $\mathcal{P}^{\mathit{Plane}}$ *because projection* $\mathit{unload}^{\mathit{Plane}}_{\mathit{Truck@Ostrava}}$ *of Truck's public action simply produces the goal state out of the blue. Finally, clearly* $\pi_2 \in \mathsf{sol}(\mathcal{P}^{\mathit{Truck}})$.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

A *public plan* of problem $\mathcal{P}$ is a plan that contains only actions from $\mathcal{A}^{\text{pub}}$, that is, contains only public actions of $\mathcal{P}$. A public plan can be seen as a solution outline that captures execution order of public actions while ignoring agents internal actions. For a solution π of $\mathcal{P}$ we construct the *public projection* π^{pub} by removing internal actions, that is, by restricting π to $\mathcal{A}^{\text{pub}}$. Hence π^{pub} is a public plan of $\mathcal{P}$. For a solution π of $\mathcal{P}^\alpha$ the *public projection* π^{pub} is constructed similarly by removing internal actions and additionally by translating projection images back to their projection origins. That is to say, that π^{pub} is composed of public actions from $\mathcal{A}^{\text{pub}}$ rather than from their projections which are present in π. Thus π^{pub} is again a public plan of $\mathcal{P}$.

Example 7. *In our example we know that* $\pi_0 \in \mathsf{sol}(\mathcal{P})$ *and* $\pi_1 \in \mathsf{sol}(\mathcal{P}^{Plane})$ *and* $\pi_2 \in \mathsf{sol}(\mathcal{P}^{Truck})$. *Thus we can construct the following public plans.*

$$\pi_0^{\text{pub}} = \langle \mathit{unload}_{Plane@Brno}, \mathit{load}_{Truck@Brno}, \mathit{unload}_{Truck@Ostrava} \rangle$$
$$\pi_1^{\text{pub}} = \langle \mathit{unload}_{Truck@Ostrava} \rangle$$
$$\pi_2^{\text{pub}} = \langle \mathit{unload}_{Plane@Brno}, \mathit{load}_{Truck@Brno}, \mathit{unload}_{Truck@Ostrava} \rangle$$

Note that $\pi_0^{\text{pub}} = \pi_2^{\text{pub}}$.

2.4 Public Plan Extensibility

We want to construct a solution of $\mathcal{P}$ from solutions of agent projections $\mathcal{P}^\alpha$. But not all projection solutions can be easily composed to a solution of $\mathcal{P}$. The concept of *public plan extensibility* helps us to select projection solutions which are conductive to our purpose. In this section we use σ to range over public plans to improve readability.

Definition 1. *Let* σ *be a public plan of* $\mathcal{P}$. *We say that* σ *is* internally extensible *if there is* $\pi \in \mathsf{sol}(\mathcal{P})$ *such that* $\pi^{\text{pub}} = \sigma$. *Similarly, we say that* σ *is* internally α-extensible *if there is* $\pi \in \mathsf{sol}(\mathcal{P}^\alpha)$ *such that* $\pi^{\text{pub}} = \sigma$.

Example 8. *In our example it is clear that* π_0^{pub} *is internally extensible because it was constructed from the solution of* $\mathcal{P}$. *From the same reason we see that* π_1^{pub} *is internally Plane-extensible and* π_2^{pub} *is internally Truck-extensible. It is easy to see that* π_2^{pub} *is also internally Plane-extensible. However,* π_1^{pub} *is not internally Truck-extensible because Truck needs to execute other public actions prior to* $\mathit{unload}_{Truck@Ostrava}$.

The following lemma states that a solution of problem $\mathcal{P}$ can be constructed from a public plan σ which is internally α-extensible for all the involved agents. The constructive proof suggests an algorithm to construct a solution.

Lemma 1. *Let public plan* σ *of* $\mathcal{P}$ *be given. Public plan* σ *is internally extensible if and only if* σ *is internally* α*-extensible for every agent* α *that owns some action from* σ.

Proof. Case ($\Rightarrow$) is trivial. When σ is internally extensible then there is $\pi \in \mathsf{sol}(\mathcal{P})$ such that $\pi^{\text{pub}} = \sigma$. We can construct projection π_α of π to agent α by removing internal actions of agents other than α, and by applying projection a^α to the remaining actions a. It holds that $\pi_\alpha \in \mathsf{sol}(\mathcal{P}^\alpha)$ and also $\pi_\alpha^{\text{pub}} = \sigma$. Thus σ is internally α-extensible.

To prove case ($\Leftarrow$) let us suppose that $\alpha_1, \ldots, \alpha_n$ are all the agents that owns some action in σ. For every i, σ is internally α_i-extensible and thus there is π_i such that $\pi_i \in \mathsf{sol}(\mathcal{P}^{\alpha_i})$ and $\pi_i^{\text{pub}} = \sigma$. Now we construct a solution π of $\mathcal{P}$ from projection solutions π_i's as follows. We split each π_i by the public actions from σ and we join the corresponding internal parts of different plans together. Then we construct π from σ by adding the joined parts between corresponding public actions in σ. Note that we do not need to do a reverse projection because for action a internal to agent α it holds that $a^\alpha = a$. Clearly $\pi^{\text{pub}} = \sigma$ and it is not hard to prove that $\pi \in \mathsf{sol}(\mathcal{P})$. Hence σ is internally extensible. □

The consequence of the lemma is that to ensure that $\mathcal{P}$ has a solution it is enough to find a solution $\pi \in \mathsf{sol}(\mathcal{P}^\alpha)$ for some agent α such that π^{pub} is internally extensible.

Example 9. *We have seen previously that* π_2^{pub} *is internally Truck-extensible and also internally Plane-extensible. Hence we know that there is some solution of* $\mathcal{P}$ *even without knowing* π_0. *On the other hand, we know that* π_1^{pub} *is not internally Truck-extensible and thus* π_1^{pub} *is not internally extensible.*

Some public plans of $\mathcal{P}$ can be extended to a valid solution of $\mathcal{P}$ but it might require inserting also public actions into σ. The following definition captures this notion which will be used in the following sections.

Definition 2. *Let public plan* σ *of* $\mathcal{P}$ *be given. We say that* σ *is* publicly extensible *if there is public plan* σ' *of* $\mathcal{P}$ *which is internally extensible and* σ *is a subsequence of* σ'.

Example 10. *We have seen that* π_1^{pub} *is not internally extensible, however, it is still publicly extensible because it is a subsequence of* π_0^{pub}.

Similarly we define that σ is *publicly* α*-extensible*. Projection solution $\pi \in \mathsf{sol}(\mathcal{P}^\alpha)$ is called internally extensible (or publicly extensible) when the corresponding public plan π^{pub} is so.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

3 CONFIRMATION SCHEME

In this section we present a multiagent planning algorithm which effectively iterates over all solutions of one selected agent (*initiator*) in order to find such a solution which is internally extensible by all the other agents (*participants*). The confirmation algorithm provides a sound and complete multiagent planning algorithm (see Theorem 2).

Algorithm 1: Multiagent planning algorithm with iterative deepening.

```
input  : multiagent planning problem P
output : a solution π of P when solution exists
Function MultiPlanIterative(P) is
    l_max ← 1
    loop
        π ← MultiPlan(P, l_max)
        if π ≠ ∅ then
            return π
        end
        l_max ← l_max + 1
    end
end
```

We suppose that we have a separate agent capable of running planning algorithms for each agent mentioned in a given problem $\mathcal{P}$. Procedure `MultiplanIterative` from Algorithm 1 is the main entry point of our algorithms, both in this and the following sections. This procedure is initially executed by one of the agents called *initiator*. It takes a problem $\mathcal{P}$ as the only argument and it iteratively calls procedure $\texttt{MultiPlan}(\mathcal{P}, l_{\mathsf{max}})$ to find a solution of $\mathcal{P}$ of length l_{max}, increasing l_{max} by one on a failure. In this way we ensure completeness of our algorithm because we enumerate the infinite set of all plans in a way that does not miss any solution. To simplify the presentation, we restrict our research only to those problems $\mathcal{P}$ which actually have a solution, that is, $\mathsf{sol}(\mathcal{P}) \neq \emptyset$.

Algorithm 2 presents implementation of `MultiPlan` in the confirmation algorithm. We suppose that $\texttt{SinglePlan}(\mathcal{P}, \mathcal{F}, l_{\mathsf{max}})$ implements a sound and complete classical planner which returns a solution of (an initiator projection of) $\mathcal{P}$ of length l_{max} which is not in $\mathcal{F}$. Moreover we suppose that `SinglePlan` always terminates and that it returns $\emptyset$ when there is no solution.

Initially, we set $\mathcal{F}$ to $\emptyset$. Then we invoke `SinglePlan` to obtain a solution of $\mathcal{P}$ denoted as π. Afterwards, we ask the participant agents whether or not the public plan π^{pub} is internally α-extensible. How participant agents fulfill this task is described in Section 5.1 When answers from all of the agents are affirmative then π is returned as a result. Otherwise π is added to the set of forbidden plans $\mathcal{F}$ and `SinglePlan` is called to compute a different solution.

Algorithm 2: $\texttt{MultiPlan}(\mathcal{P}, l_{\mathsf{max}})$ in the confirmation scheme. Function $\texttt{SinglePlan}(\mathcal{P}, \mathcal{F}, l_{\mathsf{max}})$ returns a plan of length l_{max} solving problem $\mathcal{P}$ omitting forbidden plans from $\mathcal{F}$ or $\emptyset$ if there is no such plan. Method $\texttt{AskAllAgents}(\pi^{\text{pub}})$ ask all agents α mentioned in the plan whether they consider the public plan π^{pub} to be internally α-extensible and returns `OK` if all agents reply `YES`.

```
input  : problem P and a maximum plan length l_max
output : a solution π of P when solution exists
Function MultiPlan(P, l_max) is
    F ← ∅
    loop
        π ← SinglePlan(P, F, l_max)
        if π = ∅ then
            return ∅
        end
        reply ← AskAllAgents(π^pub)
        if reply = OK then
            return π
        end
        F ← F ∪ {π}
    end
end
```

The following states that the (public projection of the) plan returned by the confirmation algorithm is internally extensible to a solution of $\mathcal{P}$ (*soundness*), and that the algorithm finds internally extensible solution when there is one (*completeness*). It is easy to construct a solution of $\mathcal{P}$ given an internally extensible plan.

Theorem 2. *Let procedure* `SinglePlan` *in* `MultiPlan` *(Alg. 2) be sound and complete. Then algorithm* `MultiplanIterative` *(Alg. 1) with confirmation procedure* `MultiPlan` *is sound and complete.*

Proof. To prove soundness, let us suppose that π is the result of `MultiPlanIterative`. Public plan π^{pub} was confirmed by each agent α to be internally α-extensible. Thus, by Lemma 1, it is internally extensible and following the lemma proof we can reconstruct the whole solution of $\mathcal{P}$.

Let us prove completeness. During each loop iteration in `MultiPlan` one plan is added to $\mathcal{F}$. There are only finitely many plans of length l_{max} and thus algorithm `MultiPlan` always terminates because `SinglePlan` is sound and complete. When $\mathcal{P}$ is solvable, then some internally extensible solution π has to be eventually returned by `SinglePlan` at some point because `SinglePlan` is complete. This solution is then the result of `MultiPlan` (and hence

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

`MultiPlanIterative`) because, as a solution of $\mathcal{P}$, it has to be confirmed by all the participants. □

4 GENERATING PLANS USING DIVERSE PLANNING

In the previous section we have supposed that function `SinglePlan`$(\mathcal{P}, \mathcal{F}, l_{\max})$ selects an arbitrary solution of $\mathcal{P}$ of length $l_{\max}$ which is distinct from all the previous solutions stored in $\mathcal{F}$. In this section we present an improved version of `SinglePlan` which selects a solution based on evaluation of *qualities* of previously found solutions.

Section 4.1 defines the notion of plan metrics which are used to describe how much two plans differ. Based on these metrics we define in Section 4.2 a notion of the *relative quality* of a plan based on evaluation of previously considered solutions which were, however, rejected by at least one of the participant agents. Finally, Section 4.3 describes improved version of function `SinglePlan`.

4.1 Plan Metrics

While planning looks for a single solution of a problem, the goal of diverse planning is to find several *different* solutions. There are two main approaches to define how much two plans differ. Firstly, the difference of two plans can be defined by their membership to the same homotopy class (Bhattacharya et al., 2010). Another approach defines a distance between plans. The distance can be defined either on *(i)* actions and their relations, or on *(ii)* states that the execution of a plan goes through, or on *(iii)* causal links between actions and goals (Srivastava et al., 2007). In this paper, we use two metrics of the first type, that is, distance metrics defined on actions and their mutual positions in the plan.

4.1.1 Different Actions Metric

The *Different Actions Metric* counts the ratio of actions which are contained only in one of the plans. It is defined as follows. Let $\pi_0 \setminus \pi_1$ denote the plan π_0 with all the actions from π_1 removed.

$$\delta^{\mathcal{A}}(\pi_A, \pi_B) = \frac{|\pi_A \setminus \pi_B| + |\pi_B \setminus \pi_A|}{|\pi_A| + |\pi_B|}$$

This metric considers neither the ordering of actions nor the fact that some of the actions can be in a plan multiple times. Nevertheless, it is very simple for evaluation.

4.1.2 Levenshtein Distance Metric

The *Levenshtein Distance Metric* (Levenshtein, 1966) is a general distance metric defined on two sequences. Let $\mathsf{trim}(\pi)$ be the plan π with the last action removed. Moreover let $\mathsf{diff}(\pi_A, \pi_B)$ be 1 if the last actions of π_A and π_B differ and 0 otherwise. Then the Levenshtein metric $\delta^{\mathcal{L}}(\pi_A, \pi_B)$ is defined as follows.

$$\begin{aligned}
\delta^{\mathcal{L}}(\pi, \emptyset) &= |\pi| \\
\delta^{\mathcal{L}}(\emptyset, \pi) &= |\pi| \\
\delta^{\mathcal{L}}(\pi_A, \pi_B) &= \min \begin{cases} \delta^{\mathcal{L}}(\mathsf{trim}(\pi_A), \pi_B) + 1 \\ \delta^{\mathcal{L}}(\pi_A, \mathsf{trim}(\pi_B)) + 1 \\ \delta^{\mathcal{L}}(\mathsf{trim}(\pi_A), \mathsf{trim}(\pi_B)) + \\ \qquad + \mathsf{diff}(\pi_A, \pi_B) \end{cases}
\end{aligned}$$

This metric describes how many changes using *elementary operations* have to be performed to convert one plan into another. The elementary operations are *add* an action into the plan, *remove* an action from the plan, and *replace* one action in the plan by another action.

4.2 Plan Quality Estimation

In Algorithm 2, the initiator agent generates its local solution π and asks participant agents to check whether π^{pub} can be extended to a solution of their local problems. Each participant either accepts or rejects π^{pub}. Based on their replies, we can define the quality $Q(\pi)$ of π as the ratio of the number of participants accepting π^{pub} and the total number of participants.

$$Q(\pi) = \frac{\textit{\# of participants accepting } \pi^{\text{pub}}}{\textit{\# of all participants}}$$

Hence the plan π with $Q(\pi) = 1$ is accepted by all of the participants and the algorithm successfully terminates.

Once we have a plan π' whose quality has already been established, we can define a *relative quality* $\Delta(\pi, \pi')$ of an arbitrary π with respect to π' using a selected metric δ on plans as follows.

$$\Delta(\pi, \pi') = \left| Q(\pi') - \delta(\pi, \pi') \right|$$

The relative quality $\Delta(\pi, \pi')$ is high when either $Q(\pi')$ is high and π is close to π', or when $Q(\pi')$ is low and π is distanced from π'. In other cases the value is close to zero.

Suppose we have a set of plans Π whose qualities have already been established. Then we can compute the relative quality $\Delta(\pi, \Pi)$ of an arbitrary plan π with respect to Π in several ways. In our work we work with the following two *quality estimators*.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

4.2.1 Average Quality Estimator

The *average estimator* $\Delta^{\oslash}(\pi,\Pi)$ is defined as the average of the relative qualities of π with respects to the plans from Π.

$$\Delta^{\oslash}(\pi,\Pi) = \frac{\sum_{\pi' \in \Pi} \Delta(\pi,\pi')}{|\Pi|}$$

4.2.2 Minimal Quality Estimator

The *minimal estimator* $\Delta^{\min}(\pi,\Pi)$ is defined as the minimal relative quality.

$$\Delta^{\min}(\pi,\Pi) = \min_{\pi' \in \Pi} \Delta(\pi,\pi')$$

4.3 Generating Diverse Plans

During the execution of Algorithm 2, the initiator agent remembers the qualities Q of generated but rejected plans, that is, it remembers the qualities of all the plans from $\mathcal{F}$. We suppose that Q is updated with every call to `AskAllAgents`. Additionally, the initiator computes the following statistics about actions.

$Q(a)$ = average quality of plans containing a

$Q(a,a')$ = average quality of plans containing a before a'

The function `SinglePlan` executed repeatedly by the initiator is described in Algorithm 3. It calls `DiversePlan` to generate a fixed number (n) of local solutions. Function `DiversePlan` works as follows. Firstly it generates a solution candidate using roulette wheel selection (Bäck, 1996) based on average action qualities $Q(a)$. These actions are then presorted using statistics about action ordering $Q(a,a')$. Note that two actions are swapped only if the difference of the statistics is larger then some threshold Δ^{Q} (0.1 in our experiments). This ordering step allows algorithm to find the correct solution faster, but the price for that is lost of completeness of `SinglePlan` procedure.

Once a solution candidate is generated, the initiator α tests whether this sequence of actions is publicly α-extensible, that is, whether it is its local solution. If so, the solution is added to a set of diverse plans. This process is repeated until the required number of local solutions is found. In our implementation, this process is further extended and occasionally, instead of a roulette selection, those action which have not been used often are chosen. In this way the algorithm gathers further information about unused actions. Finally, function `SinglePlan` selects the diverse plan with the maximum relative quality.

Algorithm 3: `SinglePlan`$(\mathcal{P},\mathcal{F})$ uses `DiversePlan`$(\mathcal{P},n,l_{\max})$ to generate n different solutions to the problem $\mathcal{P}$ and then selects the best one using metric $\Delta(\pi,\mathcal{F})$. The generation of different plans is based on the roulette wheel selection by the quality evaluation received by other agents.

```
input  : classical STRIPS problem P, the set F of
         forbidden plans, and a maximum plan length
         l_max
output : a solution π of P when solution exists
Function SinglePlan(P, F, l_max) is
    /* n is a constant                          */
    Π^div ← DiversePlan(P, F, n, l_max)
    π ← argmax_{π∈Π^div}(Δ(π, F))
    return π
end

input  : problem P and n number of solutions
output : a set of diverse solutions
Function DiversePlan(P, F, n, l_max) is
    Π ← ∅
    while |Π| < n do
        A ← GetRandomActions(P)
        π' ← OrderActions(A)
        π ← CreatePublicExtension(P, π')
        if π ≠ ∅ & π ∉ F & |π| ≤ l_max then
            Π ← Π ∪ {π}
        end
    end
    return Π
end

Function GetRandomActions(P, l_max) is
    n ← RandomInt(1...min(l_max, |A|))
    A ← ∅
    while |A| < n do
        A ← A ∪ {a : roulette selection by Q(a)}
    end
    return A
end

Function OrderActions(A) is
    π ← A
    for i = 2..|π| do
        if Q(π_i, π_{i-1}) − Q(π_{i-1}, π_i) > Δ^Q then
            SwapActions(π_{i-1}, π_i)
        end
    end
    return π
end
```

5 FROM THEORY TO PRACTICE

We have implemented the algorithms described in the previous sections taking advantage of several existing techniques and systems. An overall scheme of the ar-

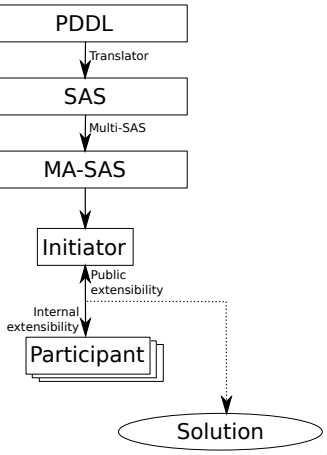

Figure 1: Architecture of the planner.

chitecture of our planner is sketched at Figure 1. An input problem $\mathcal{P}$ described in PDDL is translated into SAS using *Translator* script which is a part of Fast Downward[1] system. Our *Multi-SAS* script then splits SAS representation of the problem $\mathcal{P}$ into agents' projections $\mathcal{P}^\alpha$ using user provided selection of public facts P^{pub}. Initiator then computes public extension of actions to create a solution to its own projection of the problem. Participants are then requested to check whether they consider it to be internally α-extensible.

Next part of this section demonstrates how the public and internal extension can be easily verified using any standard STRIPS planner.

5.1 Computing Plan Extensions

In our algorithms, agents are asked whether a provided sequence of actions can be extended into a solution by adding other actions into the sequence. Technically, this is similar to the planning problem with landmarks (Brafman and Domshlak, 2008). In this section we describe our algorithm to solve this problem. Based on this solution we describe how an initiator agent computes public extensions of a given sequence and how participant agents check whether a sequence of public actions is internally extensible.

Suppose we are given a classical STRIPS planning problem $\mathcal{P} = \langle P, A, I, G\rangle$ together with a sequence $\sigma = \langle a_1, \ldots, a_n\rangle$ of actions build from the facts P. The planning problem with landmarks is the task to find a solution π of the problem $\langle P, A \cup \{a_1, \ldots, a_n\}, I, G\rangle$ such that σ is a subsequence of π, that is, that all the actions from σ are used in π in the proposed order. Note that an action a_i might or might be not in A.

[1] http://www.fast-downward.org/

Definition 3. *A* planning problem with landmarks *is a pair* $\langle \mathcal{P}, \sigma\rangle$ *where* $\mathcal{P} = \langle P, A, I, G\rangle$ *is a classical* STRIPS *problem and* $\sigma = \langle a_1, \ldots, a_n\rangle$ *is a sequence of actions build from the facts of* $\mathcal{P}$.

A solution π *of* $\langle \mathcal{P}, \sigma\rangle$ *is a solution of the classical* STRIPS *problem* $\langle P, A \cup \{a_1, \ldots, a_n\}, I, G\rangle$ *such that* σ *is a subsequence of* π.

We solve a planning problem with landmarks by translating $\langle \mathcal{P}, \sigma\rangle$ into a classical STRIPS problem $\mathcal{P}^\sigma$ such that the solutions of $\mathcal{P}^\sigma$ are in a direct correspondence to the solutions of the original problem with landmarks. Firstly we take a set P_{marks} of $n+1$ facts distinct from P denoted as follows.

$$P_{\text{marks}} = \{mark_0, \ldots, mark_n\}$$

The meaning of fact $mark_i$ is that the landmark actions $a_1, \ldots, a_i$ has already been used in the correct order and that the action a_{i+1} can be used now. We will ensure that only one fact from P_{marks} can hold in any reachable state. We will add $mark_0$ to an initial state and we will require $mark_n$ to be in the goal.

Definition 4. *Let* $\mathcal{P} = \langle P, A, I, G\rangle$ *and* $\sigma = \langle a_1, \ldots, a_n\rangle$ *and* $P_{marks} = \{mark_0, \ldots, mark_n\}$ *such that* P *and* P_{marks} *are distinct be given. For every action* a_i *let us define action* b_i *as follows.*

$$b_i = \langle\ \mathsf{pre}(a_i) \cup \{mark_{i-1}\},\ \mathsf{add}(a_i) \cup \{mark_i\},\ \mathsf{del}(a_i) \cup \{mark_{i-1}\}\ \rangle$$

The translation of the planning problem with landmarks $\langle \mathcal{P}, \sigma\rangle$ *into a classical* STRIPS *problem* $\mathcal{P}^\sigma$ *is defined as follows.*

$$\mathcal{P}^\sigma = \langle\ P \cup P_{marks},\quad A \cup \{b_1, \ldots, b_n\},\ I \cup \{mark_0\},\ G \cup \{mark_n\}\rangle$$

Basically we take action a_i and we add $mark_{i-1}$ to its preconditions and remove $mark_{i-1}$ when a_i is used. Moreover a use of a_i enables us to use the next action a_{i+1} from the list σ by adding $mark_i$ to the effects. It is easy to show the following property.

Lemma 3. *Let* $\langle \mathcal{P}, \sigma\rangle$ *be a planning problem with landmarks. When* π *is a solution of* $\mathcal{P}^\sigma$ *then* π *with* b_i*'s changed back to* a_i*'s is a solution of* $\langle \mathcal{P}, \sigma\rangle$. *Moreover when there is a solution of* $\langle \mathcal{P}, \sigma\rangle$ *then there is a solution of* $\mathcal{P}^\sigma$.

Recall that every agent α is equipped with its local projection $\mathcal{P}^\alpha$ of problem $\mathcal{P}$, that is, a classical STRIPS problem defined as follows.

$$\mathcal{P}^\alpha = \langle P^\alpha, \mathcal{A}^\alpha, I \cap P^\alpha, G\rangle$$

The set $\mathcal{A}^\alpha$ of local actions consists of α-internal actions α^{int} and projections of public actions.

aa $\qquad \mathcal{A}^\alpha = \alpha^{\text{int}} \cup \{a^\alpha | a \in \mathcal{A}^{\text{pub}}\}$

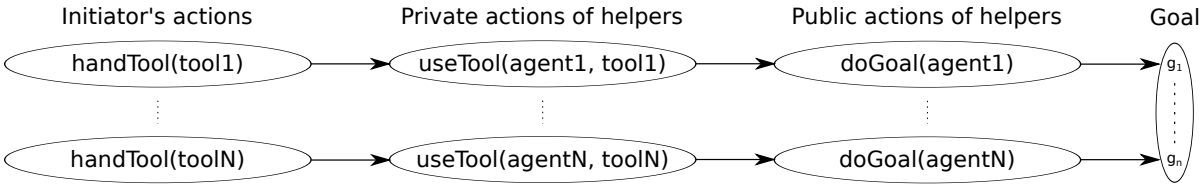

Figure 2: A scheme of the Tool problem.

In Algorithm 3, the initiator agent is asked using function `CreatePublicExtension`$(\mathcal{P},\pi')$ to find a solution of its local projection $\mathcal{P}^\alpha$ that has a given action sequence π' as a subsequence, that is, its public extension. The initiator can simply solve the planning problem with landmarks $\langle \mathcal{P}^\alpha, \pi' \rangle$ as shown in the following Theorem 4. Note that in this case the landmarks from π' are also in the set $\mathcal{A}^\alpha$ of actions of $\mathcal{P}^\alpha$.

Theorem 4. *Plan π is publicly α-extensible to a solution of $\mathcal{P}^\alpha$ if and only if the planning problem with landmarks $\langle \mathcal{P}^\alpha, \pi \rangle$ is solvable. And moreover, the solution of planning problem with landmarks serves as a proof of the extensibility, and vice versa.*

Proof. It is quite straightforward to translate each plan π' proving public extensibility of plan π to a solution of the planning problem with landmarks $\langle \mathcal{P}^\alpha, \pi \rangle$, and vice versa. □

In Algorithm 2, the participant agents are asked from the call to `AskAllAgents`(π^{pub}) to establish whether π^{pub} is internally α-extensible to a solution of $\mathcal{P}^\alpha$. The participant can simply check the solvability of the planning problem with landmarks $\langle a_1^\alpha, \ldots, a_n^\alpha \rangle$ as shown in the following Theorem 5. Note that in this case the landmarks are not in α^{int}.

Theorem 5. *Plan $\pi = \langle a_1, \ldots, a_n \rangle$ is internally α-extensible to a solution of $\mathcal{P}^\alpha$ if and only if the planning problem $\langle P^\alpha, \alpha^{\text{int}}, I \cap P^\alpha, G \rangle$ with landmarks $\langle a_1, \ldots, a_n \rangle$ is solvable. And moreover, the solution of planning problem with landmarks serves as a proof of the extensibility, and vice versa.*

6 EXPERIMENTS

For our experiments, we have designed the *Tool Problem* that allows us to observe a smooth transition in the complexity of the problem.

We focused our experiments on the following criteria: (1) comparison of different estimators and (2) an average number of iterations required to find a solution.

6.1 Tool Problem

In the *Tool Problem*, the goal is that each of N agents performs its public *doGoal* action as it is shown in Figure 2. However, this action must be preceded by its internal *useTool* action first. Only the initiator agent can provide tools with the *handTool* action. Formally, there are N tools *tool1*, ..., *toolN*, and $N+1$ agents (the initiator and N participants). In the initial state, none of the participants has its tool and the initiator has all of them. However, the initiator does not know that the participants need them. One of possible solutions is as follows.

1. *handTool(initiator, tool1)*
⋮
N. *handTool(initiator, toolN)*
N+1. *useTool(participant1, tool1)*
⋮
2N. *useTool(participantN, toolN)*
2N+1. *doGoal(participant1, tool1)*
⋮
3N. *doGoal(participantN, toolN)*

Other permutations of the plan also form a valid solution.

6.2 Results

Let us present our results for the *Tool Problem* with 2, 4, 6, 8, 10, and 12 tools. Graphs in figures 3, 4, and 5 show the results of running our experiments 50 times.

Estimator Average Errors. Firstly, we compare both estimators presented in this paper: Average Estimator (titled AVG in the graphs) and Minimal Estimator (MIN). Each estimator is tested with two different distance metrics: Different Action Metric (DIFF) and Levenshtein Distance Metric (LEV). Figure 3 demonstrates the progress of the estimators errors for the Tool Problem with 10 tools. Errors are computed from the average of 50 runs. As shown in the graph, Average Estimator with Different Action Metric converts quickly to very low error and thus it seems to be the best choice for this problem.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

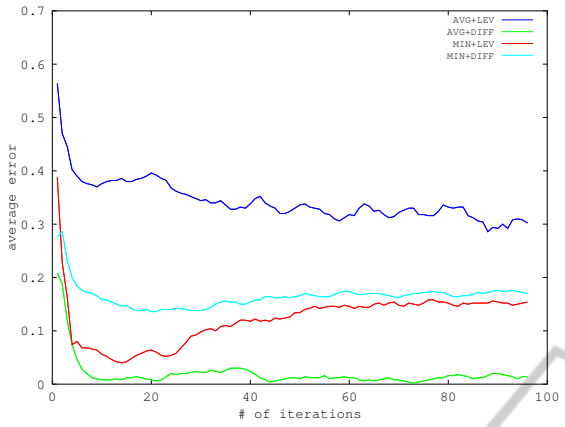

Figure 3: Progress of an average error of plan qualities computed by different estimators for the *Tool Problem* with 10 tools.

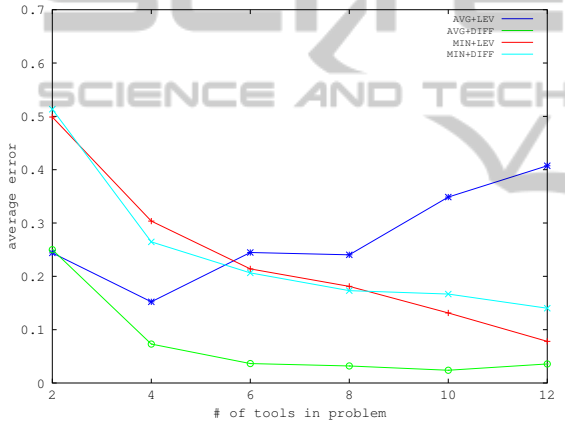

Figure 4: Average errors of plan qualities computed from the first 80 iterations for the *Tool Problem* with a variable number of tools.

Figure 4 shows an average error for each estimator during first 80 iterations for different sizes of the Tool Problem. We can see that the Average Estimator with Different Action Metric again shows the lowest errors for all the cases, and furthermore, that its error decreases with increasing problem complexity.

Results for Tool Problem. Table 1 shows how many Tool Problems of different sizes has been solved during 50 runs using different plan generation techniques. We can see that most of the approaches perform better than a random generation of plans[2].

[2]We have implemented a simple implementation of `SinglePlan` by translating a planning problem into a SAT problem instance and by calling an external SAT solver to solve it. It is easy to instruct a SAT solver to compute a solution different from previously found solutions.

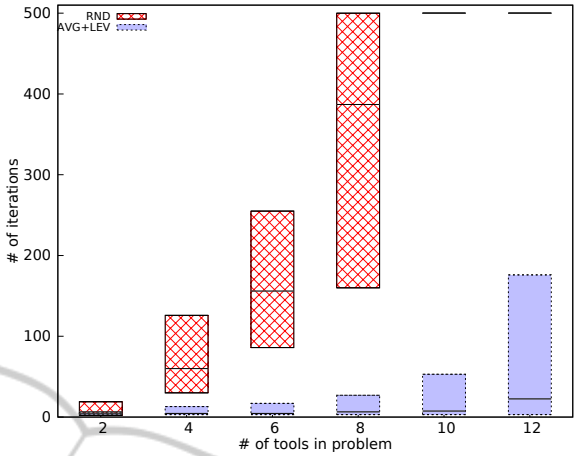

Figure 5: The number of Generate-And-Test iterations needed to solve different sizes of the Tool Problem using random generation of plans (RND) and generation driven by the Average Estimator with the Levenshtein Distance Metric (AVG+LEV). Graph shows median (line in the rectangle) and 25 % and 75 % quantile (lower and upper bound of the rectangle) of the results.

AVG+LEV again shows the best performance. Figure 5 shows more detailed distribution for its results in comparison to the baseline random generation. This graph shows a significant improvement over the baseline solution and that more complex cases of Tool Problem can be solved using this technique.

Results for Rover Problem. Classical planners are compared at the International Planning Competition with a well defined set of problems called *IPC problems*. Unfortunately, most of these problems are by their nature a single-agent problems and there is no standard way to convert them into a multiagent setting. Nevertheless, some of the problems are by their nature multiagent and fulfills all the requirements we have specified above in this article. One of the problems is called *rovers* and its goal is to plan actions for multiple robotic rovers on Mars that need to collect samples and transmit their data back to Earth via a shared base.

Table 2 shows that we were able to solve some problem instances very quickly when the first plan generated by the initiator (*rover0*) was internally α-extensible by all the other agents and thus formed a solution of the problem. When the first generated plan was not a solution of the problem then the search for a solution usually timeouted because it requires a planner to find out that a problem has no solution. This constitutes a challenge for the state-of-the-art planners which usually performs best on problems which actually have a solution. When there is no solution

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 1: Percentage of successfully solved instances of the Tools Problem for different number of tools. Comparison of a reference random plans generator (RND) and different combinations of estimators and plan distance metrics.

	2	3	4	5	6	7	8	9	10	11	12
RND	100%	100%	100%	100%	98%	80%	54%	40%	16%	6%	10%
MIN+DIF	100%	100%	100%	98%	60%	36%	22%	6%	16%	2%	0%
MIN+LEV	100%	100%	100%	94%	96%	100%	90%	96%	68%	100%	76%
AVG+DIF	100%	100%	100%	100%	94%	88%	84%	72%	68%	70%	78%
AVG+LEV	100%	100%	100%	100%	100%	100%	100%	100%	96%	98%	82%

Table 2: Number of iterations needed to successfully solve Rovers problems from the IPC collection of planning problems. Problems marked by ∞ were not solved because the problem was too large for the test of public extensibility and FD did not finish in a reasonable time. Two experiments did not finish because of an error in FD planner during the test of internal extensibility (marked as E-P). The value 0 means that a solution was found immediately and successfully confirmed by all the participants without any need for negotiation.

# rovers	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
iteration	2	0	0	0	0	∞	∞	0	∞	∞	∞	∞	E-P	∞	∞	0	0	E-P

then the planners usually get stuck in an exhaustive search of the whole plan space. Nevertheless, our planner was able to solve quickly few of harder instances of the problem, even faster than other multiagent planners (Nissim and Brafman, 2012).

7 FINAL REMARKS

We have proposed a novel approach to planning for MA-STRIPS problems based on the Generate-And-Test principle and initiator–participant protocol scheme. We have experimentally compared various combinations of plan quality estimators and plan distance metrics improving efficiency of the plan generating approach. Additionally, we have validated a principle of planning with landmarks by compilation to classical planning problem used as the testing part of the planner. The results show that the principle is viable and the best combination of estimator and metric for the designed domain is averaging with action difference metric.

In future work, we plan to test the planner in more planning domains, as it is from the beginning designed as domain-independent and reinforce the plan generation process by elements of backtracking search. Additionally, the approach hinges on efficient solving of plan-(non)existence problems with landmarks (the plan extensibility problem), therefore we will analyze how to improve on that as well.

ACKNOWLEDGEMENTS

This research was supported by the Czech Science Foundation (grant no. 13-22125S) and in part by a Technion fellowship.

REFERENCES

Bäck, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK.

Bhattacharya, S., Kumar, V., and Likhachev, M. (2010). Search-based path planning with homotopy class constraints. In Felner, A. and Sturtevant, N. R., editors, *SOCS*. AAAI Press.

Brafman, R. and Domshlak, C. (2008). From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proceedings of ICAPS'08*, volume 8, pages 28–35.

Decker, K. and Lesser, V. (1992). Generalizing the Partial Global Planning Algorithm. *International Journal on Intelligent Cooperative Information Systems*, 1(2):319–346.

Doherty, P. and Kvarnström, J. (2001). Talplanner: A temporal logic-based planner. *AI Magazine*, 22(3):95–102.

Durfee, E. H. (1999). Distributed problem solving and planning. In Weiß, G., editor, *A Modern Approach to Distributed Artificial Intelligence*, chapter 3. The MIT Press, San Francisco, CA.

Fikes, R. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 608–620.

Levenshtein, V. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707.

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

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

Pellier, D. (2010). Distributed planning through graph merging. In Filipe, J., Fred, A. L. N., and Sharp, B., editors, *ICAART (2)*, pages 128–134. INSTICC Press.

Richter, S. and Westphal, M. (2010). The lama planner: guiding cost-based anytime planning with landmarks. *J. Artif. Int. Res.*, 39(1):127–177.

Srivastava, B., Nguyen, T. A., Gerevini, A., Kambhampati, S., Do, M. B., and Serina, I. (2007). Domain independent approaches for finding diverse plans. In Veloso, M. M., editor, *IJCAI*, pages 2016–2022.

Torreño, A., Onaindia, E., and Sapena, O. (2012). An approach to multi-agent planning with incomplete information. In *ECAI*, pages 762–767.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS