Recursive Reductions of Internal Dependencies in Multiagent Planning

Jan Tožička, Jan Jakubův and Antonín Komenda
Agent Technology Center, Department of Computer Science, Czech Technical University in Prague, Karlovo namesti 13, 121 35, Prague, Czech Republic

Keywords: Automated Planning, Multiagent Systems, Problem Reduction.

Abstract: Problems of cooperative multiagent planning in deterministic environments can be efficiently solved both by distributed search or coordination of local plans. In the current coordination approaches, behavior of other agents is modeled as public external projections of their actions. The agent does not require any additional information from the other agents, that is the planning process ignores any dependencies of the projected actions possibly caused by sequences of other agents' private actions.
In this work, we formally define several types of internal dependencies of multiagent planning problems and provide an algorithmic approach how to extract the internally dependent actions during multiagent planning. We show how to take an advantage of the computed dependencies by means of reducing the multiagent planning problems. We experimentally show strong reduction of majority of standard multiagent benchmarks and nearly doubling of solved problems in comparison to a variant of a planner without the reductions. The efficiency of the method is demonstrated by winning in a recent competition of distributed multiagent planners.

1 INTRODUCTION

Cooperative intelligent agents acting in a shared environment have to coordinate their steps in order to achieve their goals. A well-established model for multiagent planning in deterministic environments was described by (Brafman and Domshlak, 2008) as MA-STRIPS, which is a minimal extension of classical planning model STRIPS (Fikes and Nilsson, 1971). MA-STRIPS provides problem partitioning in form of separated sets of actions of particular agents, and notion of local private information the agents are not willing to share. By definition, private actions and facts about the environment do not affect other agents and cannot be affected by other agents. Shared facts and actions which can influence more than one agent are denoted as public.

In multiagent planning modeled as MA-STRIPS, agents can either plan only with their own actions and facts and inform the other agents about public achieved facts, as for instance in the MAD-A* planner (Nissim and Brafman, 2012). Or, agents can also use other agents' public actions provided that the actions are stripped of the private facts in preconditions and effects. Thus agents plan actions, in a sense, for other agents and then coordinate the plans (Tožička et al., 2014b).

Only a complete stripping of all private information from public actions was used in literature so far. Such approach can, however, lead to tangible loss of information on causal dependencies of the actions described by the private actions. A seeming remedy is to borrow techniques from classical planning on problem reduction (e.g., in (Haslum, 2007; Chen and Yao, 2009; Coles and Coles, 2010)). As our motivation is to "pack" sequences of public and private actions, the most suitable are recursive macro actions as proposed by (Jonsson, 2009; Bäckström et al., 2012). A macro action can represent a sound sequence of actions and, provided that it allows for recursive reductions, it can be used repeatedly with possibly radical downsizing of the reduced planning problem.

In a motivation logistic problem, when an agent transports a package from one city to another and wants to keep its current load internal, it is not practical to publish two actions: *load(package, fromCity)* and *unload(package, toCity)*. Instead it should publish action *transport(package, fromCity, toCity)*, which is capturing the hidden (private) relation between this pair of actions while it is not disclosing it in an explicit way.

We propose to keep the pair of actions and to add new public predicate that says that some action requires another action to precede it (because it better fits the proposed coordination algorithm). In the sim-

Tožička, J., Jakubův, J. and Komenda, A.
Recursive Reductions of Internal Dependencies in Multiagent Planning.
DOI: 10.5220/0005754901810191
In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016) - Volume 2*, pages 181-191
ISBN: 978-989-758-172-4

plest case, the new public fact would directly correspond to the internal fact *isLoaded(package)*, but in realistic cases, it could also capture more complex dependencies, for example, the transshipment between different vehicles belonging to the transport agent.

In this paper, we build on our previous work (Tožička et al., 2015a) where internal dependencies of public actions where studied with a restriction that every action consumes all of its preconditions. This restriction no longer applies here because it can be very limiting in practice. Furthermore, we demonstrate the effect of our approach on a benchmark set from the CoDMAP competition held on International Conference on Automated Planning and Scheduling (ICAPS'15). Our planner, employing the theory from this paper, won the distributed track of CoDMAP 2015.

2 MULTIAGENT PLANNING

This section provides condensed formal prerequisites of multiagent planning based on the MA-STRIPS formalism (Brafman and Domshlak, 2008). Refer to (Tožička et al., 2014a) for more details.

An MA-STRIPS *planning problem* Π is a quadruple $\Pi = \langle P, \{\alpha_i\}_{i=1}^{n}, I, G\rangle$, where P is a set of facts, α_i is the set of actions of i-th agent[1] $I \subseteq P$ is an initial state, and $G \subseteq P$ is a set of goal facts. We define selector functions $\mathsf{facts}(\Pi)$, $\mathsf{agents}(\Pi)$, $\mathsf{init}(\Pi)$, and $\mathsf{goal}(\Pi)$ such that $\Pi = \langle \mathsf{facts}(\Pi), \mathsf{agents}(\Pi), \mathsf{init}(\Pi), \mathsf{goal}(\Pi)\rangle$. An *action* $a \in \alpha$, the agent α can perform, is a triple of subsets of P called *preconditions*, *add effects*, *delete effects*. Selector functions $\mathsf{pre}(a)$, $\mathsf{add}(a)$, and $\mathsf{del}(a)$ are defined so that $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)\rangle$. Note that an agent is identified with the actions the agent can perform in an environment.

In MA-STRIPS, out of computational or privacy concerns, each fact is classified either as *public* or as *internal*. A fact is *public* when it is mentioned by actions of at least two different agents. A fact is *internal for agent* α when it is not public but mentioned by some action of α. A fact is *relevant for* α when it is either public or internal for α. MA-STRIPS further extends this classification of facts to actions as follows. An action is *public* when it has a public (add- or delete-) effect, otherwise it is *internal*. An action from Π is *relevant for* α when it is either public or owned by (contained in) α.

[1]Whereas, in STRIPS, the second parameter is a set of actions, in MA-STRIPS, the second parameter is actually a set of sets of actions.

We use $\mathsf{int\text{-}facts}(\alpha)$ and $\mathsf{pub\text{-}facts}(\alpha)$ to denote in turn the sets internal facts and the set of public facts of agent α. Moreover, we write $\mathsf{pub\text{-}facts}(\Pi)$ to denote all the public facts of problem Π. We write $\mathsf{pub\text{-}actions}(\alpha)$ to denote the set of public actions of agent α. Finally, we use $\mathsf{pub\text{-}actions}(\Pi)$ to denote all the public actions of all the agents in problem Π.

In multiagent planning with external actions, a *local planning problem* is constructed for every agent α. Each local planning problem for α is a classical STRIPS problem where α has its own internal copy of the global state and where each agent is equipped with information about public actions of other agents called *external actions*. These local planning problems allow us to divide an MA-STRIPS problem into several STRIPS problems which can be solved separately by a classical planner.

The *projection* $F \triangleright \alpha$ of a set of facts F to agent α is the restriction of F to the facts relevant for α, representing F as seen by α. The *public projection* $a \triangleright \star$ of action a is obtained by restricting the facts in a to public facts. Public projection is extended to sets of actions element-wise.

A *local planning problem* $\Pi \triangleright \alpha$ of agent α, also called *projection of* Π to α, is a classical STRIPS problem containing all the actions of agent α together with external actions, that is, public projections of other agents' public actions. The local problem of α is defined only using the facts relevant for α. Formally,

$$\Pi \triangleright \alpha = \langle P \triangleright \alpha, \alpha \cup \mathsf{exts}(\alpha), I \triangleright \alpha, G\rangle$$

where the set of external actions $\mathsf{exts}(\alpha)$ is defined as follows.

$$\mathsf{exts}(\alpha) = \bigcup_{\beta \neq \alpha} (\mathsf{pub\text{-}actions}(\beta) \triangleright \star)$$

In the above, β ranges over all the agents of Π. The set $\mathsf{exts}(\alpha)$ can be equivalently described as $\mathsf{exts}(\alpha) = (\mathsf{pub\text{-}actions}(\Pi) \setminus \alpha) \triangleright \star$. To simplify the presentation, we consider only problems with public goals and hence there is no need to restrict goal G.

3 PLANNING WITH EXTERNAL ACTIONS

The previous section allows us to divide an MA-STRIPS problem into several classical STRIPS local planning which can be solved separately by a classical planner. Recall that the local planning problem of agent α contains all the actions of α together with α's external actions, that is, with projections of public actions of other agents. This section describes conditions which allow us to compute a solution of the

Algorithm 1: Distributed MA planning algorithm.

```
1  Function MaPlanDistributed($\Pi \triangleright \alpha$) is
2      $\Phi_\alpha \leftarrow \emptyset$;
3      loop
4          generate new $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$;
5          $\Phi_\alpha \leftarrow \Phi_\alpha \cup \{\pi_\alpha \triangleright \star\}$;
6          exchange public plans $\Phi_\alpha$ with other
           agents;
7          $\Phi \leftarrow \bigcap_{\beta \in \mathsf{agents}(\Pi)} \Phi_\beta$;
8          if $\Phi \neq \emptyset$ then
9              return $\Phi$;
10         end
11     end
12 end
```

original MA-STRIPS problem from solutions of local problems.

A *plan* π is a sequence of actions. A *solution* of Π is a plan π whose execution transforms the initial state to a subset of the goals. A *local solution* of agent α is a solution of $\Pi \triangleright \alpha$. Let $\mathsf{sols}(\Pi)$ denote the set of all the solutions of MA-STRIPS or STRIPS problem Π. A *public plan* σ is a sequence of public actions. The *public projection* $\pi \triangleright \star$ *of plan* π is the restriction of π to public actions.

A public plan σ is *extensible* when there is $\pi \in \mathsf{sols}(\Pi)$ such that $\pi \triangleright \star = \sigma$. Similarly, σ is α-*extensible* when there is $\pi \in \mathsf{sols}(\Pi \triangleright \alpha)$ such that $\pi \triangleright \star = \sigma$. Extensible public plans give us an order of public actions which is acceptable for all the agents. Thus extensible public plans are very close to solutions of Π and it is relatively easy to construct a solution of Π once we have an extensible public plan. Hence our algorithms will aim at finding extensible public plans.

The following theorem (Tožička et al., 2014a) establishes the relationship between extensible and α-extensible plans. Its direct consequence is that to find a solution of Π it is enough to find a local solution $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ which is β-extensible for every agent β.

Theorem 1. *Public plan σ of Π is extensible if and only if σ is α-extensible for every agent α.*

The theorem above suggests the distributed multiagent planning algorithm described in Algorithm 1. Every agent executes the loop from Algorithm 1, possibly on different machine. Every agent keeps generating new solutions of its local problem and stores solution projections in set Φ_α. These sets are exchanged among all the agents so that every agent can compute their intersection Φ. Once the intersection Φ is non-empty, the algorithm terminates yielding Φ as the result. Theorem 1 ensures that every public plan in the resulting Φ is extensible. Consult (Tožička et al., 2014a) for more details on the algorithm.

4 INTERNAL DEPENDENCIES OF ACTIONS

One of the benefits of planning with external actions is that every agent can plan separately its local problem which involves planning of actions for other agents (external actions). Other agents can then only verify whether a plan generated by another agent is α-extensible for them. A con of this approach is that agents have only a limited knowledge about external actions because internal facts are removed by projection. Thus it can happen that an agent plans external actions inappropriately in a way that the resulting public plan is not α-extensible for some agent α.

In the rest of this paper we try to overcome the limitation of partial information about external actions. The idea is to equip agents with additional information about external actions without revealing internal facts. The rest of this section describes *dependency graphs* which are used in the following sections as a formal ground for our analysis of public and external actions.

4.1 Dependency Graphs

Local planning problem $\Pi \triangleright \alpha$ of agent α contains information about external actions provided by the set $\mathsf{exts}(\alpha)$. The idea is to equip agent α with more information described by a suitable structure. A dependency graphs is a structure we use to encapsulate information about public actions which an agent shares with other agents.

Dependency graphs are known from literature (Jonsson and Bäckström, 1998; Chrpa, 2010). In our context, a *dependency graph* Δ is a bipartite directed graph defined as follows.

Definition 1. *A* dependency graph Δ *is a bipartite directed graph whose nodes are actions and facts. We write* $\mathsf{actions}(\Delta)$ *and* $\mathsf{facts}(\Delta)$ *to denote action and fact nodes respectively. Given the nodes, graph Δ contains the following three kinds of edges.*

$$
\begin{array}{lr}
(a \rightarrow f) \in \Delta \textit{ iff } f \in \mathsf{add}(a) & (a\ \textit{produces}\ f) \\
(f \rightarrow a) \in \Delta \textit{ iff } f \in \mathsf{pre}(a) \setminus \mathsf{del}(a) & (a\ \textit{requires}\ f) \\
(f \dashrightarrow a) \in \Delta \textit{ iff } f \in \mathsf{pre}(a) \cap \mathsf{del}(a) & (a\ \textit{consumes}\ f)
\end{array}
$$

Additionally, a fact can be marked as initial in Δ. The set of states marked as initial is denoted $\mathsf{init}(\Delta)$.

Hence edges of a dependency graph Δ are uniquely determined by the set of nodes. Note that action nodes are themselves actions, that is, triples of fact sets. These action nodes can contain additional facts other than fact nodes $\mathsf{facts}(\Delta)$. We use dependency graphs to represent internal dependencies of public actions. Dependencies determined by public facts are known to other agents and thus we do not need them in the graph as fact nodes. From now on we suppose that $\mathsf{facts}(\Delta)$ contains no public facts as fact nodes. Action nodes, however, can contain public facts in their public actions.

Definition 2. *Let an* MA-STRIPS *problem* Π *be given. The* minimal dependency graph $\mathsf{MD}(\alpha)$ *of agent* $\alpha \in \mathsf{agents}(\Pi)$ *is the dependency graph uniquely determined by the following set of nodes.*

$$\begin{aligned} \mathsf{actions}(\mathsf{MD}(\alpha)) &= \mathsf{pub\text{-}actions}(\alpha) \\ \mathsf{facts}(\mathsf{MD}(\alpha)) &= \emptyset \\ \mathsf{init}(\mathsf{MD}(\alpha)) &= \emptyset \end{aligned}$$

Hence $\mathsf{MD}(\alpha)$ has no edges as there are no fact nodes. Thus the graph contains only separated public action nodes. Furthermore, the set $\mathsf{exts}(\alpha)$ of external actions of agent α can be trivially expressed as follows.

$$\mathsf{exts}(\alpha) = \bigcup_{\beta \neq \alpha} (\mathsf{actions}(\mathsf{MD}(\beta)) \triangleright \star)$$

Thus we see that dependency graphs can carry the same information as provided by $\mathsf{exts}(\alpha)$.

Definition 3. *The* full dependency graph $\mathsf{FD}(\alpha)$ *of agent* α *contains all the actions of* α *and all the internal facts of* α.

$$\begin{aligned} \mathsf{actions}(\mathsf{FD}(\alpha)) &= \alpha \\ \mathsf{facts}(\mathsf{FD}(\alpha)) &= \mathsf{int\text{-}facts}(\alpha) \\ \mathsf{init}(\mathsf{FD}(\alpha)) &= \mathsf{init}(\Pi) \cap \mathsf{int\text{-}facts}(\alpha) \end{aligned}$$

Hence $\mathsf{FD}(\alpha)$ contains all the information known by α. By publishing $\mathsf{FD}(\alpha)$, an agent reveals all his internal dependencies which might be a potential privacy risk. On the other hand, other agents are by $\mathsf{FD}(\alpha)$ provided the most precise information about dependencies of public actions of α. Every plan of another agent, computed with $\mathsf{FD}(\alpha)$ in mind, is automatically α-extensible. Thus we see that dependency graphs can carry dependencies information with a varied precision.

4.2 Dependency Graph Collections

A dependency graph represents information about public actions of one agent. Every agent needs to know information from all the other agents. We use *dependency graph collections* to represent all the required information. A *dependency graph collection* $\mathcal{D}$ of an MA-STRIPS problem Π is a set of dependency graphs which contains exactly one dependency graph for every agent of Π. We write $\mathcal{D}(\alpha)$ to denote the graph of α. We write $\mathsf{actions}(\mathcal{D})$, $\mathsf{facts}(\mathcal{D})$, and $\mathsf{init}(\mathcal{D})$ to denote in turn all the action, fact, and initial fact nodes from all the graphs in $\mathcal{D}$.

Definition 4. *Given problem* Π, *we can define the* minimal collection $\mathsf{MD}(\Pi)$ *and the* full collection $\mathsf{FD}(\Pi)$ *as follows.*

$$\begin{aligned} \mathsf{MD}(\Pi) &= \{\mathsf{MD}(\alpha) : \alpha \in \mathsf{agents}(\Pi)\} \\ \mathsf{FD}(\Pi) &= \{\mathsf{FD}(\alpha) : \alpha \in \mathsf{agents}(\Pi)\} \end{aligned}$$

Later we shall show some interesting properties of the minimal and full collections.

4.3 Local Problems and Dependency Collections

In order to define local problems informed by $\mathcal{D}$, we need to define facts and action projections which preserve information from $\mathcal{D}$. We use symbol $\triangleright_{\mathcal{D}}$ to denote *projections accordingly to* $\mathcal{D}$. Recall that the public projection $a \triangleright \star$ of action a is the restriction of the facts of a to $\mathsf{pub\text{-}facts}(\Pi)$. The *public projection* $a \triangleright_{\mathcal{D}} \star$ *of action* a *accordingly to* $\mathcal{D}$ is the restriction of the facts of a to $\mathsf{pub\text{-}facts}(\Pi) \cup \mathsf{facts}(\mathcal{D})$. Public projection is extended to sets of actions elementwise. Furthermore, *external actions of* α *according to* $\mathcal{D}$, denoted $\overline{\mathsf{exts}}_{\mathcal{D}}(\alpha)$, contain public projections (according to $\mathcal{D}$) of actions of other agents. In other words, $\overline{\mathsf{exts}}_{\mathcal{D}}(\alpha)$ carries all the information published by other agents for agent α. It is computed as follows.

$$\overline{\mathsf{exts}}_{\mathcal{D}}(\alpha) = \bigcup_{\beta \neq \alpha} (\mathsf{actions}(\mathcal{D}(\beta)) \triangleright_{\mathcal{D}} \star)$$

This equation captures distributed computation of $\overline{\mathsf{exts}}_{\mathcal{D}}(\alpha)$ where every agent β separately computes published actions, applies public projection, and sends the result to α.

In order to define a local planning problem of agent α which would take information from $\mathcal{D}$ into consideration, we need to extract from $\mathcal{D}$ facts and initial facts of other agents. Below we define sets $\overline{\mathsf{facts}}_{\mathcal{D}}(\alpha)$ and $\overline{\mathsf{init}}_{\mathcal{D}}(\alpha)$ which contain those facts and initial facts published by other agents, that is, all the facts from $\mathcal{D}$ except of the facts of α.

$$\begin{aligned} \overline{\mathsf{facts}}_{\mathcal{D}}(\alpha) &= \mathsf{facts}(\mathcal{D}) \setminus \mathsf{facts}(\mathcal{D}(\alpha)) \\ \overline{\mathsf{init}}_{\mathcal{D}}(\alpha) &= \mathsf{init}(\mathcal{D}) \setminus \mathsf{init}(\mathcal{D}(\alpha)) \end{aligned}$$

Now we are ready to define *local planning problems according to* $\mathcal{D}$ which extends local planning problems by the information contained in $\mathcal{D}$.

Definition 5. *Let* Π *be* MA-STRIPS *problem. The* local problem $\Pi \triangleright_{\mathcal{D}} \alpha$ of agent $\alpha \in \mathsf{agents}(\Pi)$ accordingly to $\mathcal{D}$ *is the classical* STRIPS *problem* $\Pi \triangleright_{\mathcal{D}} \alpha = \langle P_0, A_0, I_0, G_0 \rangle$ *where*

(1) $P_0 = \mathsf{facts}(\Pi \triangleright \alpha) \cup \overline{\mathsf{facts}}_{\mathcal{D}}(\alpha)$,

(2) $A_0 = \alpha \cup \overline{\mathsf{exts}}_{\mathcal{D}}(\alpha)$,

(3) $I_0 = \mathsf{init}(\Pi \triangleright \alpha) \cup \overline{\mathsf{init}}_{\mathcal{D}}(\alpha)$, *and*

(4) $G_0 = \mathsf{goal}(\Pi)$.

We can see that a local problem $\Pi \triangleright_{\mathcal{D}} \alpha$ according to $\mathcal{D}$ extends the local problem $\Pi \triangleright \alpha$ by the facts and actions published by $\mathcal{D}$.

Let us consider two boundary cases of dependency collections $\mathsf{MD}(\Pi)$ and $\mathsf{FD}(\Pi)$. Given an MA-STRIPS problem Π, we can construct local problems using the minimal dependency collection $\mathsf{MD}(\Pi)$. It is easy to see that $\Pi \triangleright_{\mathsf{MD}(\Pi)} \alpha = \Pi \triangleright \alpha$ for every agent α. With the full dependency collection $\mathsf{FD}(\Pi)$ we obtain equal projections, that is, $\Pi \triangleright_{\mathsf{FD}(\Pi)} \alpha = \Pi \triangleright_{\mathsf{FD}(\Pi)} \beta$ for all agents α and β. Moreover, local solutions equal MA-STRIPS solutions, that is, $\mathsf{sols}(\Pi \triangleright_{\mathsf{FD}(\Pi)} \alpha) = \mathsf{sols}(\Pi)$ for every α.

4.4 Publicly Equivalent Problems

We have seen that dependency collections can provide information about internal dependencies with a varied precision. Given two different collections, two different local problems can be constructed for every agent. However, when the two local problems of the same agent equal on public solutions, we can say that they are equivalent because their public solutions are equally extensible.

In order to define equivalent collections, we first define public equivalence on problems. Two planning problems Π_0 and Π_1 are *publicly equivalent*, denoted $\Pi_0 \simeq \Pi_1$, when they have equal public solutions. Formally as follows.

$$\Pi_0 \simeq \Pi_1 \quad \Leftrightarrow \quad \mathsf{sols}(\Pi_0) \triangleright \star = \mathsf{sols}(\Pi_1) \triangleright \star$$

Public equivalence can be extended to dependency graph collections as follows. Two collections $\mathcal{D}_0$ and $\mathcal{D}_1$ of the same MA-STRIPS problem Π are *equivalent*, written $\mathcal{D}_0 \simeq \mathcal{D}_1$, when for any agent α, it holds that the local problems $\Pi \triangleright_{\mathcal{D}_0} \alpha$ and $\Pi \triangleright_{\mathcal{D}_1} \alpha$ are publicly equivalent. Formally as follows.

$$\mathcal{D}_0 \simeq \mathcal{D}_1 \quad \Leftrightarrow \quad (\Pi \triangleright_{\mathcal{D}_0} \alpha) \simeq (\Pi \triangleright_{\mathcal{D}_1} \alpha) \ \ (\text{for all } \alpha)$$

Example 1. *Given an* MA-STRIPS *problem* Π, *with the full dependency collection* $\mathsf{FD}(\Pi)$ *we can see that* $\Pi \simeq \Pi \triangleright_{\mathsf{FD}(\Pi)} \alpha$ *holds for any agent. Hence to find a public solution of* Π *it is enough to solve the local problem (accordingly to* $\mathsf{FD}(\Pi)$*) of an arbitrary agent. The same holds for any dependency collection* $\mathcal{D}$ *such that* $\mathcal{D} \simeq \mathsf{FD}(\Pi)$. *Note that* $\mathcal{D}$ *can be much smaller and provide less private information than the full dependency collection.*

The above definitions allow us to recognize problems without any internal dependencies which we can define as follow.

Definition 6. *An* MA-STRIPS *problem* Π *is* internally independent *when* $\mathsf{MD}(\Pi) \simeq \mathsf{FD}(\Pi)$.

In order to solve an internally independent problem, it is enough to solve the local problem $\Pi \triangleright \alpha$ of an arbitrary agent. Any local public solution is extensible which makes internally independent problems easier to solve because there is no need for interaction and negotiation among the agents. Later we shall show how to algorithmically recognize internally independent problems. The following formally captures the above properties.

Lemma 2. *Let* Π *be an internally independent* MA-STRIPS *problem. Then* $(\Pi \triangleright \alpha) \simeq \Pi$.

Proof. $(\Pi \triangleright \alpha) \simeq (\Pi \triangleright_{\mathsf{MD}(\Pi)} \alpha) \simeq (\Pi \triangleright_{\mathsf{FD}(\Pi)} \alpha) \simeq \Pi$ □

5 SIMPLE ACTION DEPENDENCIES

Let us consider dependency collections without internal actions, that is, collections $\mathcal{D}$ where $\mathsf{actions}(\mathcal{D})$ contains no internal actions. When $\mathcal{D}$ is published, then no agent publishes actions additional to $\mathsf{exts}(\alpha)$ which is desirable out of privacy concerns. Furthermore, the plan search space of $\Pi \triangleright_{\mathcal{D}} \alpha$ is not increased when compared to $\Pi \triangleright \alpha$. Even more, every additionally published fact in $\mathcal{D}$ providing a valid dependency prunes the search space. Action dependencies captured by collections without internal actions can be expressed by requirements on the order of actions in a plan. This further abstracts the published information providing privacy protection. Thus it seems reasonable to publish dependency collections without internal actions.

5.1 Simply Dependent Problems

The following defines simply dependent MA-STRIPS problems, where internal dependencies of public actions can be expressed by a dependency collection free of internal actions.

Definition 7. *An* MA-STRIPS *problem* Π *is* simply dependent *when there exists* $\mathcal{D}$ *such that* $\mathsf{actions}(\mathcal{D})$ *contains no internal actions and* $\mathcal{D} \simeq \mathsf{FD}(\Pi)$.

Suppose we have a simply dependent MA-STRIPS problem and a dependency collection $\mathcal{D}$ which proves the fact. In order to solve Π, once again, it is enough to solve only one local problem $\Pi \triangleright_{\mathcal{D}} \alpha$ (of an arbitrary agent α).

Lemma 3. *Let Π be a simply dependent MA-STRIPS problem. Let $\mathcal{D}$ be a dependency collection which proves that Π is simply dependent. Then $(\Pi \triangleright_{\mathcal{D}} \alpha) \simeq \Pi$ holds for any agent $\alpha \in \mathsf{agents}(\Pi)$.*

Proof. $(\Pi \triangleright_{\mathcal{D}} \alpha) \simeq (\Pi \triangleright_{\mathsf{FD}(\Pi)} \alpha) \simeq \Pi$ □

The above method requires all the agents to publish the information from $\mathcal{D}$. However, the information does not need to be published to all the agents as it is enough to select one *trusted* agent and send the information only to him. Hence it is enough for all the agents to agree on a single trusted agent.

5.2 Dependency Graph Reductions

Recognizing simply dependent MA-STRIPS problems might be difficult in general. That is why we define an approximative method which can provably recognize some simply dependent problems. We define a set of reduction operations on dependency graphs and we prove that the operations preserve relation $\simeq$. Then we apply the reductions repeatedly starting with $\mathsf{FD}(\Delta)$ obtaining a dependency graph which can not be reduced any further. This is done by every agent. When the resulting graphs contain no internal actions, then we know that the problem is simply dependent. Additionally, when the resulting graphs contain no internal facts, then we know that the problem is independent.

Our previous work (Tožička et al., 2015a) was restricted to problems where $\mathsf{pre}(a) = \mathsf{del}(a)$ holds for every action a. This impractical limitation is removed here. We still restrict our attention to problems where $\mathsf{del}(a) \subseteq \mathsf{pre}(a)$ holds for every action a. This is not considered limiting because a problem not meeting this requirement can be easily transformed to a permissible equivalent problem.

Finally, to abstract from the set of initial facts of a dependency graph Δ, we introduce to the graph a special *initial action* $\langle \emptyset, \mathsf{init}(\Delta), \emptyset \rangle$. We suppose that every dependency graph has exactly one initial action and hence we do not need to remember the set of initial facts. The initial action is handled as *public* even when it has no public effect. Both definitions of dependency graphs are trivially equivalent but the one with an initial action simplifies the presentation of reduction operations.

We proceed by informal descriptions of dependency graph reductions. The formal definition is given below. The operations are depicted in Figure 1.

(R1) Remove Simple Action Dependency. If some internal action has only one delete effect and one add effect and there is no other action depending on f_1 we can merge both facts into one and remove that action.

(R2) Remove Simple Fact Dependency. If some fact is the only effect of some action and there is only one action that consumes this effect without any side effects, we can remove this fact and merge both actions.

(R3) Remove Small Action Cycle. In many domains, there are reversible internal actions that allow transitions between two (or more) states. All these states can be merged into a single state and the actions changing them can be omitted.

(R4) Merge Equivalent Nodes. If two nodes (facts or actions) equal on incoming and outgoing edges, then we can merge these two nodes. Mostly this is not directly in the domain but this structure might appear when we simplify a dependency graph using the other reductions.

(R5) Remove Invariants. After several reduction steps, it can happen that all the delete effects on some fact are removed and the fact is always fulfilled from the initial state. This happens, for example, in *Logistics*, where the location of a vehicle is internal knowledge and can be freely changed as described by reduction (R3). Once these cycles are removed, only one fact remains. The remaining fact represents that the vehicle is *somewhere*, which is always true. This fact can be freely removed from the dependency graph.

In order to formally define the above reductions we first define operator $[F]_{f_1 \to f_2}$ which renames fact f_1 to f_2 in the set of facts $F \subseteq P$.

$$[F]_{f_1 \to f_2} = \begin{cases} F & \text{if } f_1 \notin F \\ (F \setminus \{f_1\}) \cup \{f_2\} & \text{otherwise} \end{cases}$$

Similarly, we define operator $[F]_{\text{-}f} = F \setminus \{f\}$ which removes fact f from the set of facts F. These operators are extended to actions (applying the operator to preconditions, add, and delete effects) and to action sets (element-wise). The operators can be further extended to dependency graphs, where $[\Delta]_{\text{-}f}$ is the dependency graph determined by $[\mathsf{actions}(\Delta)]_{\text{-}f}$ and $[\mathsf{facts}(\Delta)]_{\text{-}f}$. Finally, for two actions a_1 and a_2 we define the merged action $a_1 \oplus a_2$ as the action obtained by unifying separately preconditions, add, and delete effects of both the actions.

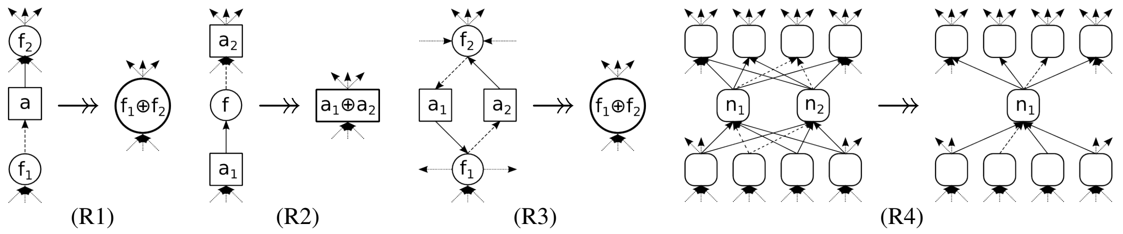

Figure 1: Graphical illustration of reduction operations (R1)–(R4). Circles represent fact nodes and rectangles represent action nodes. Rounded boxes in (R4) represent any node (either fact or action node).

The following formally defines *reduction relation* $\Delta_0 \rightarrow \Delta_1$ which holds when Δ_0 can be transformed to Δ_1 using one of the reduction operations.

Definition 8. *The* reduction relation $\Delta_0 \rightarrow \Delta_1$ *on dependency graphs is defined by the following four rules.*

(R1) *Rule (R1) is applicable to Δ_0 when*

(1) Δ_0 contains edges $(f_1 \dashrightarrow a \rightarrow f_2)$,

(2) a is internal, and

(3) there are no other edges from/to a, and

(4) there are no other edges from f_1.

Then $\Delta_0 \rightarrow \Delta_1$ where Δ_1 is defined as $\Delta_1 = [\Delta_0]_{f_1 \rightarrow f_2}$. The initial action is preserved.

(R2) *Rule (R2) is applicable to Δ_0 when*

(1) Δ_0 contains edges $(a_1 \rightarrow f \dashrightarrow a_2)$,

(2) there are no other edges from/to f, and

(3) there are no other edges from a_1, and

(4) a_2 has no other delete effects, and

(5) a_2 is internal action.

Then $\Delta_0 \rightarrow \Delta_1$ where Δ_1 is given by the following.

$$\begin{aligned} \mathsf{actions}(\Delta_1) &= \{[a_1 \oplus a_2]_{-f}\} \cup (\mathsf{actions}(\Delta_0) \setminus \{a_1, a_2\}) \\ \mathsf{facts}(\Delta_1) &= [\mathsf{facts}(\Delta_0)]_{-f} \end{aligned}$$

If a_1 is the initial action of Δ_0 then the new merged action becomes the initial action of Δ_1. Otherwise, the initial action is preserved.

(R3) *Rule (R3) is applicable to Δ_0 when*

(1) Δ_0 contains edges $(f_1 \dashrightarrow a_1 \rightarrow f_2)$, and

(2) Δ_0 contains edges $(f_2 \dashrightarrow a_2 \rightarrow f_1)$, and

(3) a_1 and a_2 are both internal, and

(4) there are no other edges from/to a_1 or a_2.

Then $\Delta_0 \rightarrow \Delta_1$ where Δ_1 is given by the following.

$$\begin{aligned} \mathsf{actions}(\Delta_1) &= [\mathsf{actions}(\Delta_0) \setminus \{a_1, a_2\}]_{f_2 \rightarrow f_1} \\ \mathsf{facts}(\Delta_1) &= [\mathsf{facts}(\Delta_0)]_{f_2 \rightarrow f_1} \end{aligned}$$

The initial action is preserved as it is public.

(R4) *Rule (R4) is applicable to Δ_0 when Δ_0 contains two nodes n_1 and n_2 (either action or fact nodes) such that*

(1) nodes n_1 and n_2 have equal sets of incoming and outgoing edges, and

(2) n_1 and n_2 are not public actions.

Then $\Delta_0 \rightarrow \Delta_1$ where, in the case n_1 and n_2 are actions, Δ_1 is given by the following.

$$\begin{aligned} \mathsf{actions}(\Delta_1) &= \{n_1 \oplus n_2\} \cup (\mathsf{actions}(\Delta_0) \setminus \{n_1, n_2\}) \\ \mathsf{facts}(\Delta_1) &= \mathsf{facts}(\Delta_0) \end{aligned}$$

When n_1 or n_2 is the initial action of Δ_0 then the new merged action becomes the initial action of Δ_1. Otherwise, the initial action is preserved.
In the case n_1 and n_2 are facts, $\Delta_1 = [\Delta_0]_{n_2 \rightarrow n_1}$.

(R5) *Let a_{init} be the initial action of Δ_0. Rule (R5) is applicable to Δ_0 when there exists fact f such that*

(1) Δ_0 contains edge $(a_{init} \rightarrow f)$, and

(2) Δ_0 contains no edge $(f \dashrightarrow a)$ for any a.

Then $\Delta_0 \rightarrow \Delta_1$ where Δ_1 is defined as $\Delta_1 = [\Delta_0]_{-f}$. The initial action of Δ_1 is $[a_{init}]_{-f}$.

The following defines *reduction equivalence relation* $\Delta_0 \sim \Delta_1$ as a reflexive, symmetric, and transitive closure of $\rightarrow$. In other words, Δ_0 and Δ_1 are *reduction equivalent* when one can be transformed to another using the reduction operations. Dependency collections $\mathcal{D}_0$ and $\mathcal{D}_1$ are *reduction equivalent* when graphs of corresponding agents are reduction equivalent.

Definition 9. *Dependency graphs* reduction equivalence relation*, denoted $\Delta_0 \sim \Delta_1$, is the least reflexive, symmetric, and transitive closure generated by the relation $\rightarrow$.*

Given MA-STRIPS *problem Π, dependency collections $\mathcal{D}_0$ and $\mathcal{D}_1$ of Π are* reduction equivalent*, written $\mathcal{D}_0 \sim \mathcal{D}_1$, when $\mathcal{D}_0(\alpha) \sim \mathcal{D}_1(\alpha)$ for any agent* $\alpha \in \mathsf{agents}(\Pi)$.

The following theorem formally states that reduction operations preserves public equivalence.

Theorem 4. *Let Π be an* MA-STRIPS *problem and let $\mathsf{pre}(a) \subseteq \mathsf{del}(a)$ hold for any internal action. Let $\mathcal{D}_0$ and $\mathcal{D}_1$ be dependency collections of problem Π. Then $\mathcal{D}_0 \sim \mathcal{D}_1$ implies $\mathcal{D}_0 \simeq \mathcal{D}_1$.*

Proof sketch. It can be shown that none of the reduction operations changes the set of public plans $\mathsf{sols}(\mathcal{D}_0(\alpha)) \triangleright \star$ of any agent $\alpha \in \mathsf{agents}(\Pi)$. Therefore repetitive application of reductions assures that $\mathcal{D}_0 \simeq \mathcal{D}_1$.

Algorithm 2: Compute the dependency graph to be published by agent α.

```
Function ComputeSharedDG($\alpha$) is
    $\Delta_0 \leftarrow \mathsf{FD}(\alpha)$;
    loop
        if $\exists \Delta_1 : \Delta_0 \rightarrow \Delta_1$ then
            $\Delta_0 \leftarrow \Delta_1$;
        else
            break;
        end
    end
    if $\Delta_0$ contains only public actions then
        return $\Delta_0$;
    else
        return $\mathsf{MD}(\alpha)$;
    end
end
```

To avoid possible action confusion caused by value renaming, we suppose that actions are assigned unique ids which are preserved by the reduction, and that plans are sequences of these ids. □

The consequences of the theorem are discussed in the following section.

5.3 Recognizing Simply Dependent Problems

Let us have an MA-STRIPS problem Π where $\mathsf{pre}(a) \subseteq \mathsf{del}(a)$ holds for every internal action a. Suppose that every agent α can reduce its full dependency collection $\mathsf{FD}(\alpha)$ to a state where it contains no internal action. Then there is $\mathcal{D}$ such that $\mathcal{D} \sim \mathsf{FD}(\Pi)$ and hence $\mathcal{D} \simeq \mathsf{FD}(\Pi)$ by Theorem 4. Hence Π is simply dependent and its public solution can be found without agent interaction, provided all the agents allow to publish $\mathcal{D}$. Important idea here is that publicly equivalent dependency graphs do not need to reveal the same amount of sensitive information. Moreover when $\mathcal{D} \sim \mathsf{MD}(\alpha)$ then Π is independent and can be solved without any interaction and without revealing other than public information. This gives us an algorithmic approach to recognize some independent and simply dependent problems.

6 PLANNING WITH DEPENDENCY GRAPHS

This section describes how agents use dependency graphs in order to solve MA-STRIPS problem Π (Algorithm 3). At first, every agent computes the dependency graph it is willing to share using function `ComputeSharedDG` described by Algorithm 2. Every agent α starts with the full dependency graph $\mathsf{FD}(\alpha)$ and tries to apply reduction operations repeatedly as long as it is possible. When the resulting reduced dependency graph Δ_0 contains only public actions, then the agent publishes Δ_0. Otherwise, the agent publishes only the minimal dependency graph $\mathsf{MD}(\alpha)$. Algorithm 2 clearly terminates for every input because every reduction decreases the number of nodes in the dependency graph. Hence the algorithm loop (lines 3–9 in Algorithm 2) can not be iterated more than n times when n is the count of nodes in $\mathsf{FD}(\alpha)$. Moreover, every reduction operation can be performed in a time polynomial to the size of the problem, and thus the whole algorithm is polynomial-time.

Algorithm 3: Distributed planning with dependency graphs.

```
Function DgPlanDistributed($\alpha$) is
    $\Delta \leftarrow$ ComputeSharedDG($\alpha$);
    send $\Delta$ to other agents;
    construct $\mathcal{D}$ from other agent's graphs;
    compute local problem $\Pi \triangleright_{\mathcal{D}} \alpha$;
    return MaPlanDistributed($\Pi \triangleright_{\mathcal{D}} \alpha$);
end
```

Once the shared dependency graph Δ is computed, Algorithm 3 continues by sending Δ to other agents. Then shared dependency graphs of other agents are received. This allows every agent to complete the dependency collection $\mathcal{D}$, and to construct the local problem $\Pi \triangleright_{\mathcal{D}} \alpha$. The rest of the planning procedure is the same as in the case of Algorithm 1.

The algorithm can be further simplified when all the agents succeeds in reducing $\mathsf{FD}(\alpha)$ to an equivalent dependency graph without internal actions, that is, when Π is provably simply dependent. Then it is enough to select one agent to compute public solution of Π. When at least one agent α fails to share dependency collection equivalent to the full dependency collection $\mathsf{FD}(\alpha)$ then iterated negotiation is required. When some agent α (but not all the agents) succeeds in reducing $\mathsf{FD}(\alpha)$ then every plan created by any other agent will be automatically α-extensible.

7 EXPERIMENTS

For experimental evaluation we use benchmark problems from the CoDMAP'15 competition[2]. The

[2]See http://agents.fel.cvut.cz/codmap

Table 1: Results of the analysis of internal dependencies of public actions in benchmark domains.

Domain	Facts	Public facts	Merge facts	Fact disclosure	Actions	Public actions	Success
Blocksworld	787	733	53	100 %	1368	1368	100 %
Depots	1203	1139	56	85 %	2007	2007	100 %
Driverlog	1532	1419	16	25 %	7682	7426	100 %
Elevators	509	343	43	29 %	2060	1767	70 %
Logitics	240	154	56	63 %	342	298	100 %
Rovers	2113	1251	31	3 %	3662	1555	13 %
Satellite	846	578	0	0 %	8839	914	1 %
Taxi	177	173	0	0 %	107	107	100 %
Woodworing	1448	1425	7	27 %	4126	4126	100 %
Zenotravel	1349	1204	0	0 %	13516	2364	0 %

benchmark set contains 12 domains with 20 problems per domain. Each agent has its own domain and problem files containing description of known facts and actions. Additionally, some facts/predicates are specified as private and thus should not be communicated to other agents. The privacy classification roughly corresponds to MA-STRIPS.

We firstly present analysis of internal dependencies and their reductions in Section 7.1. In Section 7.2, we present results independently evaluated by organizers of the CoDMAP'15 competition.

7.1 Domain Analysis

In this section we present analysis of internal dependencies of public action in the benchmark problems.

We have evaluated internal dependencies of public actions within benchmark problems by constructing full dependency graph for every agent in every benchmark problem. We have applied Algorithm 2 to reduce full dependency graphs to an irreducible publicly equivalent dependency graph. The results of the analysis are presented in Table 1. The table columns have the following meaning. Column **(Facts)** represents an average number of all facts in a domain problem. Column **(Public facts)** represents an average number of public facts in a domain problem. Column **(Merge facts** represents an average size of $\mathsf{facts}(\Delta)$ in the resulting irreducible dependency graph. Column **(Fact disclosure)** represents the percentage of published merge facts with respect to all the internal facts. Column **(Actions)** represents an average number of all actions in a domain problem. Column **(Public actions)** represents an average number of public actions in a domain problem. Column **(Success)** represents the percentage of agents capable of reducing their full dependency graph to to a publicly equivalent graph without internal actions.

We can see that five of the benchmark domains, namely *Blocksworld*, *Depots*, *Driverlog*, *Logistics*, *Taxi*, *Wireless*, and *Woodworking* were found simply dependent. All the problems in these domains can be solved by solving a local problem of a single agent. On the contrary, in most problems of domains *Rovers*, *Satellite*, and *Zenotravel*, none of the agents were able to reduce its full dependency graph so that it contains no internal actions. Hence the agents in these domain publish only the minimal dependency graphs and hence the analysis does not help in solving them. Finally, in *Elevators* domain, some of the agents succeeded in reducing their full dependency graphs and thus the analysis can partially help to solve them.

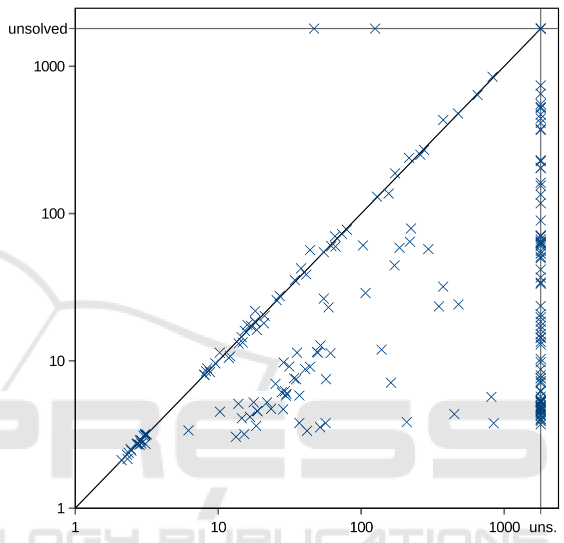

Figure 2: Comparison of planning times (in seconds) of PSM-VR algorithm without (X axis) and with (Y axis) internal problem reduction.

7.2 Experimental Results

To evaluate the impact of dependency analysis, we use our PSM-based planners (Tožička et al., 2014b) submitted to the CoDMAP'15 competition. Namely, we use planner PSM-VR (Tožička et al., 2015b) and its extension with dependency analysis PSM-VRD.

Figure 2 evaluates the impact of the dependency analysis on CoDMAP benchmark problems. For each

Table 2: Results at CoDMAP competition (`http://agents.fel.cvut.cz/codmap/results/`). Top table shows overall coverage of solved problem instances. Middle table shows IPC score over the plan quality Q (a sum of Q^*/Q over all problems, where Q^* is the cost of an optimal plan, or of the best plan found by any of the planners for the given problem during the competition). Bottom table shows IPC Agile score over the planning time T (a sum of $1/(1+\log_{10}(T/T^*))$ over all problems, where T^* is the runtime of the fastest planner for the given problem during the competition). [‡]This is optimal planner. [†]This post-submission domain was not supported by the planner parser. These results are presented with the consent of CoDMAP organizers.

Domain	**#**	**MAPlan LM-Cut**[‡]	**MAPlan MA-LM-Cut**[‡]	**MH-FMAP**	**MAPlan FF+DTG**	**PSM-VR**	**PSM-VRD**
Blocksworld	20	2	1	0	14	12	**20**
Depots	20	5	2	2	10	1	**16**
Driverlog	20	15	9	18	18	16	**20**
Elevators	20	2	0	**9**	**9**	2	5
Logitics	20	4	5	4	**16**	0	**16**
Openstacks	20	1	1	8	**18**	14	**18**
Rovers	20	2	4	18	**19**	13	13
Satellite	20	13	4	4	14	7	**17**
Taxi	20	19	14	**20**	19	9	**20**
Wireless	20	3	2	0	**4**	0[†]	0[†]
Woodworing	20	3	4	8	14	9	**19**
Zenotravel	20	6	6	16	**19**	16	16
Total Coverage	240	75	52	107	174	99	**180**

Domain	**#**	**MAPlan LM-Cut**[‡]	**MAPlan MA-LM-Cut**[‡]	**MH-FMAP**	**MAPlan FF+DTG**	**PSM-VR**	**PSM-VRD**
Blocksworld	20	2	1	0	7	11	**17**
Depots	20	5	2	2	6	1	**15**
Driverlog	20	15	9	**17**	12	14	16
Elevators	20	2	0	**8**	6	1	4
Logitics	20	4	5	4	13	0	**15**
Openstacks	20	1	1	8	**18**	9	12
Rovers	20	2	4	**18**	16	5	5
Satellite	20	**13**	4	4	10	6	**13**
Taxi	20	**19**	14	17	15	6	16
Wireless	20	3	2	0	**4**	0[†]	0[†]
Woodworing	20	3	4	7	13	8	**17**
Zenotravel	20	6	6	**15**	**15**	10	10
IPC Score	240	75	52	100	135	72	**140**

Domain	**#**	**MAPlan LM-Cut**[‡]	**MAPlan MA-LM-Cut**[‡]	**MH-FMAP**	**MAPlan FF+DTG**	**PSM-VR**	**PSM-VRD**
Blocksworld	20	1	0	0	**14**	5	**14**
Depots	20	3	1	1	9	0	**14**
Driverlog	20	10	4	11	**17**	7	14
Elevators	20	1	0	4	**8**	1	4
Logitics	20	3	2	2	13	0	**14**
Openstacks	20	0	0	3	**18**	7	8
Rovers	20	1	1	7	**19**	6	6
Satellite	20	10	2	1	**13**	3	12
Taxi	20	14	7	10	**19**	3	15
Wireless	20	**3**	2	0	2	0[†]	0 [†]
Woodworing	20	2	3	4	9	5	**18**
Zenotravel	20	5	4	10	**18**	8	8
IPC Agile Score	240	52	27	52	**159**	45	127

problem, a point is drawn at the position corresponding to the runtime without dependency analysis (x-coordinate) and the runtime with dependency analysis (y-coordinate). Hence the points below the diagonal constitute improvements. Results show that the dependency analysis decreases overall planning time of PSM algorithm. We can see that in few cases the time increases which is caused by the time consumed by reduction process. Also, by publishing additional facts, the problem size can grow and thus it can become harder to solve.

Table 2 shows official results of the CoDMAP competition. We can see that the dependency analysis significantly improved the performance of PSM-VR planner. Moreover, PSM-VRD achieved the overall best coverage in 8 out of 12 domains. As expected, the highest coverage directly corresponds to the success of dependency analysis. The table also shows results of two additional criteria comparing the quality (IPC Score) of solutions and the time (IPC Agile Score) need to find the solution. In both criteria PSM-VRD performed very well even though it was outperformed by MAPlan-FF+DTG planner in the IPC Agile Score.

8 CONCLUSIONS

We have formally and semantically defined internally independent and simply dependent MA-STRIPS problems and proposed a set of reduction rules utilizing the underlying dependency graph. To identify internally independent and simply dependent problems, we have proposed technique which can build a full dependency graph and try to reduce it to an irreducible publicly equivalent dependency graph. This provides an algorithmic procedure for recognizing provably internally independent and simply dependent problems. We have shown that provably independent and simply dependent problems can be solved easily without agent interaction. The proposed reduction rules were defined over structural information of the dependency graph and provided possibly recursive removal of superfluous facts and actions by analysis of simple dependency, cycles, equivalency, and state invariants.

We experimentally showed that reduction of the standard multiagent planning benchmarks using the dependencies provides overall 71% downsizing and nearly doubled the number of solved problems in comparison to the same algorithm used without the reductions. Finally, in comparison with the latest distributed multiagent planners the proposed approach outperformed all and won the distributed track of the recent multiagent planning competition CoDMAP.

ACKNOWLEDGEMENTS

This research was supported by the Czech Science Foundation (no. 13-22125S and 15-20433Y) and by the Czech Ministry of Education (no. SGS13/211/OHK3/3T/13). Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the program "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

REFERENCES

Bäckström, C., Jonsson, A., and Jonsson, P. (2012). Macros, reactive plans and compact representations. In *ECAI 2012*, pages 85–90.

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

Chen, Y. and Yao, G. (2009). Completeness and optimality preserving reduction for planning. *Proceedings of 21st IJCAI*, pages 1659–1664.

Chrpa, L. (2010). Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Eng. Review*, 25(3):281–297.

Coles, A. and Coles, A. (2010). Completeness-preserving pruning for optimal planning. In *Proceedings of 19th ECAI*, pages 965–966.

Fikes, R. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. In *IJCAI'71*, pages 608–620.

Haslum, P. (2007). Reducing Accidental Complexity in Planning Problems. In *Proceedings of 20th IJCAI*, pages 1898–1903.

Jonsson, A. (2009). The role of macros in tractable planning. *Journal of Artificial Intelligence Research*, 36:471–511.

Jonsson, P. and Bäckström, C. (1998). Tractable plan existence does not imply tractable plan generation. *Annals of Mathematics and Artificial Intelligence*, 22:281–296.

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

Tožička, J., Jakubův, J., Durkota, K., Komenda, A., and Pěchouček, M. (2014a). Multiagent Planning Supported by Plan Diversity Metrics and Landmark Actions. In *Proceedings ICAART'14*.

Tožička, J., Jakubův, J., and Komenda, A. (2014b). Generating multi-agent plans by distributed intersection of finite state machines. In *ECAI2014*, pages 1111–1112.

Tožička, J., Jakubův, J., and Komenda, A. (2015a). On internally dependent public actions in multiagent planning. In *Proceedings of DMAP Workshop of ICAPS'15*.

Tožička, J., Jakubův, J., and Komenda, A. (2015b). PSM-based Planners Description for CoDMAP 2015 Competition. In *CoDMAP-15*.