Multiagent Planning by Iterative Negotiation over Distributed Planning Graphs

Jan Tožička[1], **Jan Jakubův**[1], **Karel Durkota**[1], **Antonín Komenda**[2]
[1] Agent Technology Center, CTU in Prague
Prague, Czech Republic
[2] Technion - Israel Institute of Technolog
Haifa, Israel

Abstract

Multiagent planning for cooperative agents in deterministic environments intertwines *synthesis* and *coordination* of the local plans of involved agents. Both of these processes require an underlying structure to describe synchronization of the plans. A *distributed planning graph* can act as such a structure, benefiting by its compact representation and efficient building. In this paper, we propose a general negotiation scheme for multiagent planning based on planning graphs. The scheme is designed as independent on a particular local plan synthesis approach.

To demonstrate the proposed principle, we have implemented the negotiation scheme as an algorithm with a concrete technique for the local plan synthesis based on compilation of the local planning problems to SAT problems. Results of the negotiation further shape the SAT problems so that agents coordinate their plans and avoid possible conflicts in an iterative manner. The paper is concluded by showing a set of experiments which demonstrate a trade-off between planning efficiency (by means of time and communication) and increasing amount of public information in the planning problem.

Introduction

Intelligent agents embodied in an environment have to be able not only to selfishly push the world towards their own goals but also to cooperate on common goals with their neighbors and solve mutual conflicts if their plans interfere. *Multiagent planning*, an umbrella term for such behavior, deals with challenges both on (i) the synthesis of actions into individual agents' plans and on (ii) the coordination of the agents' plans in a shared environment. A generally usable approach to multiagent planning has to be able to deal with a wide range of application domains without any fixed domain-specific knowledge.

In such cases of domain-independent planning, the input to the planning process does not contain only the initial and goal conditions on the environment, but also description of the problem domain. Such approaches allow the agents to prepare plans flexibly according to their knowledge of the environment mechanics. From the practical perspective, domain-independent techniques can be reused in various circumstances, where the agents are required to plan.

In one of the most cited works on multiagent planning (Durfee 1999), Durfee describes basics of possible coordination schemes for planning agents. From the taxonomy presented there, our approach falls into *a distributed planning of distributed plans* which do not assume either the planning process or the resulting plan to be centralized. There is a large amount of work dealing with another facets of the coordination part of the problem, e.g. GPGP (Decker and Lesser 1992), or TALPlanner (Doherty and Kvarnström 2001). In 2008, Brafman and Domshlak proposed multiagent planning in (Brafman and Domshlak 2008) which targeted deterministic environments and was based on an extension of the classical planning model STRIPS (Fikes and Nilsson 1971). The results of the paper showed that deterministic domain-independent multiagent planning is not exponentially dependent on the number of agents in the computational sense.

The approach we propose in this paper can be seen as a merge and extension of two previous approaches. The first one is from Zhang et al. presented in (Zhang, Nguyen, and Kowalczyk 2007). The idea behind it is based on distribution of a well-known structure—*a planning graph*—and compilation of the planning problem into a DisCSP problem. The other approach authored by Pellier in (Pellier 2010) is also based on distributed planning graphs, however for the local plan extraction each agent uses a centralized CSP solver and the coordination of their plans is done by a backtracking approach resembling prioritized planning.

Our contribution in this work is threefold. Firstly, we have generalized the predetermined coordination part (done by Zhang at al. as DisCSP and Pellier as prioritized planning) by a decentralized approach based on novel negotiation scheme. This negotiation scheme extends our scheme published in (Tožička et al. 2014) by handling new types of other agent responses. Secondly, we propose a way how to supersede the Pellier's CSP-based extraction of local plans by compilation to SAT problems. And finally, we have designed and implemented an extension of the planning graph structure by state-of-the-art planning modeling approach SAS+ (Huang, Chen, and Zhang 2012). We also experimentally show a trade-off between planning efficiency (by means of computation time and communication) and increasing amount of public information in the planning problem.

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.

This model, together with proofs of lemmas and theorems, has been already published in (Tožička et al. 2014). Nevertheless, we consider it necessary to repeat basic definitions and lemmas to make paper standalone.

Planning Problem

An MA-STRIPS planning problem Π is defined as a quadruple $\Pi = \langle P, \mathcal{A}, I, G\rangle$, where P is a set of propositions, $\mathcal{A}$ is a set of *agents* $\alpha_1, \ldots, \alpha_{|\mathcal{A}|}$, I is an initial state and G is a set of goals.

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 $\mathsf{eff}(a) = \mathsf{add}(a) \cup \mathsf{del}(a)$.

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. Let A_Π denote the set of all actions in a problem Π, that is, $A_\Pi = \bigcup_{\alpha\in\mathcal{A}} \alpha$. A *state* $s = \{p_1, \ldots, p_m\} \subseteq P$ is a finite set of facts and we say that p_i's hold in s.

Public and Internal Actions

MA-STRIPS problems distinguish between the *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.

$$P^{\text{pub}} \supseteq \bigcup_{\alpha\in\mathcal{A}} \left(\mathsf{facts}(\alpha) \cap \bigcup_{\beta\in\mathcal{A}\setminus\{\alpha\}} \mathsf{facts}(\beta)\right)$$
$$P^{\alpha\text{-int}} = \mathsf{facts}(\alpha) \setminus P^{\text{pub}}$$

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 ($\supseteq$). Our definition, however, allows us to experiment with extensions of the set of public facts and this is discussed below in experiment section. 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 of facts *relevant* for a single agent α is defined as $P^\alpha = P^{\alpha\text{-int}} \cup P^{\text{pub}}$. The *projection* a^S of an action a to a set of facts S is a restriction of a containing only facts from S, that is, $a^S = \langle \mathsf{pre}(a) \cap S, \mathsf{add}(a) \cap S, \mathsf{del}(a) \cap S\rangle$. The projection a^α of action a to agent α is defined as $a^{(P^\alpha)}$ and the public projection a^{pub} of action a is defined as $a^{(P^{\text{pub}})}$.

The set α^{pub} of *public actions* of agent α is defined as $\alpha^{\text{pub}} = \{a \mid a \in \alpha, \mathsf{eff}(a^{\text{pub}}) \neq \emptyset\}$ and the set α^{int} of *internal actions* of agent α as $\alpha^{\text{int}} = \alpha \setminus \alpha^{\text{pub}}$. The set A_Π^{pub} of *all public actions* of problem Π is defined as $A_\Pi^{\text{pub}} = \bigcup_{\alpha\in\mathcal{A}} \alpha^{\text{pub}}$. A_Π^α the set of all actions known by agent α is then $A_\Pi^\alpha = \alpha^{\text{int}} \cup \{a^\alpha | a \in A_\Pi^{\text{pub}}\}$.

In the rest of this paper we consider only problems Π where all the propositions from the goal state G are public, that is, $G \subseteq P^{\text{pub}}$ which is common in literature (Nissim and Brafman 2012)[1]. Moreover we suppose that two different agents do not execute the same action, that is, we suppose that the sets α_i's are pairwise disjoint (Brafman and Domshlak 2008).

Plans, Solutions, and Projections

The projection Π^α of a problem Π to agent α is a classical STRIPS problem defined as follows.

$$\Pi^\alpha = \langle P^\alpha, A_\Pi^\alpha, I \cap P^\alpha, G\rangle$$

Given an MA-STRIPS problem Π, a *plan* $\pi = \langle a_1, \ldots, a_k\rangle$ is a sequence of actions from A_Π. A plan π defines an order in which actions are to be 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 Π if a sequential execution of the actions from π by their respective owners transforms the initial state I to a subset of the goal G. Let $\mathsf{sol}(\Pi)$ denote the set of all solutions of problem Π. We use $\pi[i]$ to denote a_i, that is, the action from the plan at position i. Moreover $\pi[i \ldots j]$ where $i \leq j$ denotes the plan subsequence $\langle a_i, \ldots, a_j\rangle$.

The *projection* π^{S} of a plan $\pi = \langle a_1, \ldots, a_k\rangle$ is computed from π by projecting each action a_i to S and by subsequent removal of empty projections. Formally we define

$$\pi \overset{S}{\longmapsto} \pi^{\text{S}} \overset{\text{def}}{\iff} \pi^{\text{S}} = \langle a_1^{\text{S}}, \ldots, a_k^{\text{S}}\rangle \,|^{A_\Pi^{\text{S}}},$$

where the restriction $|^{A_\Pi^{\text{S}}}$ creates a subsequence containing only actions of A_Π^{S}. Note that different plans can have the same projection. The *public plan projection* π^{pub} is defined as $\pi^{(A_\Pi^{\text{pub}})}$ and the plan projection π^α to agent α is defined as $\pi^{(P^\alpha)}$. A plan is called *public* w.r.t. Π if $a_i \in A_\Pi^{\text{pub}}$ for all i.

Extensible Plans

The following defines *(α)-internally extensible* plans which are plans that can be transformed to a solution by inserting only internal actions into it.

Definition 1. *Let Π be* MA-STRIPS *problem and let π be a plan public w.r.t. Π. We say that the public plan π is* α-internally extensible *if*

$$\exists \pi' \in \mathsf{sol}(\Pi^\alpha) : \pi' \overset{\text{pub}}{\longmapsto} \pi$$

[1] This condition can be weakened, but we stick to it as it simplifies this paper.

We say that the public plan π is internally extensible *if*

$$\exists \pi' \in \mathsf{sol}(\Pi) : \pi' \overset{\text{pub}}{\longmapsto} \pi$$

The following lemma states that a solution of problem Π can be obtained composing partial α-internally extensible plans of all the involved agents.

Lemma 1. *Let Π be* MA-STRIPS *problem and let π be a plan public w.r.t. Π. A plan π is* α-internally extensible *for every agent α that owns some action from π if and only if π is* internally extensible*.*

Similarly to the Definition 1 the following defines a *publicly extensible* plan which can be transformed to a solution by inserting both public and internal actions into it.

Definition 2. *Let Π be given. We say that a plan π is* publicly extensible *if there exists a plan π' which is* internally extensible *and such that π is a subsequence of π'.*

Plan Domains

The following defines a plan *domain* $\mathcal{D}$ which is a key structure used in our algorithms. A domain $\mathcal{D}$ is a set of plans with operations defined as follows.

$$\begin{array}{lcl} \mathcal{D} \ominus \pi & = & \mathcal{D} \setminus \{\pi\} \\ \mathcal{D} \oplus \langle a, t\rangle & = & \{\pi \in \mathcal{D} | \pi[t] = a\} \\ \mathcal{D} \ominus \langle a, t\rangle & = & \mathcal{D} \setminus (\mathcal{D} \oplus \langle a, t\rangle) \end{array}$$

Moreover let $\mathcal{D}^{l_{\mathsf{max}}}$ denote the set of all plans of length l_{max}.

In our algorithms presented in the following sections we suppose that we have a classical planner which computes a solution of a given classical planning problem which is in a given domain $\mathcal{D}$, that is, that we have an effective procedure that selects a solution from a given domain. We work with plan domains defined as sets of plans to abstract from a concrete implementation and to simplify presentation of the algorithms in the following sections. A plan domain $\mathcal{D}$ can be seen as an abstract data structure which supports the above three operations and whose semantics is defined using aforementioned sets of plans. Our effective implementation of plan domains uses planning graphs and a SAT solver. We encode the search for a plan in a planning graph as a SAT instance and operations on domain $\mathcal{D}$ then add additional conditions to the SAT instance so that the search is restricted to $\mathcal{D}$. The implementation is further described below.

Operation $\mathcal{D} \ominus \pi$ simply removes π from the domain. Operation $\mathcal{D} \oplus \langle a, t\rangle$ restricts the domain so that it contains only those plans which contain action a at position t. Finally, operation $\mathcal{D} \ominus \langle a, t\rangle$ does exactly the opposite, that is, it restricts the domain so that it contains only those plans which do not contain action a at position t.

Confirmation Scheme

In this section we present a multiagent planning algorithm which effectively iterates over all plans in order to find internally extensible solution. This *confirmation* algorithm can also be seen as a skeleton which is further elaborated in next section. 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 Π
output: a solution π of Π when solution exists
Function `MultiPlanIterative`(Π) **is**
  $l_{\mathsf{max}} \leftarrow 1$
  loop
    $\pi \leftarrow$ `MultiPlan`(Π, l_{max})
    if $\pi \neq \emptyset$ **then**
      return π
    end
    $l_{\mathsf{max}} \leftarrow l_{\mathsf{max}} + 1$
  end
end

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 Π as the only argument and it iteratively calls procedure `MultiPlan`(Π, l_{max}) to find a solution of Π 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 Π which actually have a solution.

Algorithm 2: `MultiPlan`(Π, l_{max}) in the confirmation scheme. Method `SinglePlan`($\Pi, \mathcal{D}$) returns a plan from domain $\mathcal{D}$ solving problem Π or $\emptyset$ if there is no such plan. Constructor `PlanDomain` constructs a plan domain with a given semantics. Method `AskAllAgents`(π^{pub}) asks all agents mentioned in the plan whether they consider the *public projection* of this plan to be *internally extensible* and returns `OK` if all agents reply `YES`.

input: problem Π and a maximum plan length l_{max}
output: a solution π of Π when solution exists
Function `MultiPlan`(Π, l_{max}) **is**
  $\mathcal{D} \leftarrow$ **new** `PlanDomain`($\{\pi : |\pi| = l_{\mathsf{max}}\}$)
  loop
    $\pi \leftarrow$ `SinglePlan`($\Pi, \mathcal{D}$)
    if $\pi = \emptyset$ **then**
      return $\emptyset$
    end
    $reply \leftarrow$ `AskAllAgents`(π^{pub})
    if $reply =$ `OK` **then**
      return π
    end
    $\mathcal{D} \leftarrow \mathcal{D} \ominus \pi$
  end
end

Algorithm 2 presents implementation of `MultiPlan` in the confirmation algorithm. Operator `PlanDomain` con-

structs a planning domain with semantics described by its argument. We suppose that $\texttt{SinglePlan}(\Pi, \mathcal{D})$ implements a sound and complete classical planner which returns a solution of Π^α within a given domain $\mathcal{D}$ where α correspond to the agent executing the task. Moreover we suppose that `SinglePlan` always terminates and that it returns $\emptyset$ when there is no solution. Our effective implementation of `SinglePlan` is described in local plan extraction section.

Initially, we create a domain that contains all the plans of length l_{max}. Then we invoke `SinglePlan` to obtain a solution of Π^α denoted as π. Afterwards, we ask all involved agents whether or not the public projection π^{pub} is internally extensible. To answer this question, each agent invokes `SinglePlan` for a problem considering only actions from π^{pub} together with its internal actions while using a plan domain to describe possible partial solutions. When the answers from all of the agents are affirmative then π is returned as a result. Otherwise π is excluded from domain $\mathcal{D}$ and `SinglePlan` is called to compute a different solution.

The following states that the plan returned by the confirmation algorithm is internally extensible to a solution of Π (*soundness*), and that the algorithm finds internally extensible solution when there is one (*completeness*). It is easy to construct a solution of Π given an internally extensible plan.

Theorem 2. *Algorithm* `MultiplanIterative` *(Alg. 1) with confirmation procedure* `MultiPlan` *(Alg. 2) is sound and complete.*

Iterative Negotiation Scheme

A drawback of the confirmation scheme from the previous section is that it requires an initiator agent to find a plan which is internally extensible to a problem solution. It means that the other agents, called *participants*, can insert only their internal actions into the plan and this can be too limiting. Our iterative negotiation algorithm from this section tries to overcome this drawback by attempting to correct a publicly extensible plan to a solution. Hence we distinguish the following cases depending on a result π returned by `SinglePlan`.

`CASE-I` **– π is an *internally extensible* plan** – all participants can extend the plan adding internal actions only.

`CASE-II` **– π is a *publicly extensible* plan** – all participants can extend the plan but some can require another public action to be performed prior to their action.

`CASE-III` **– Otherwise** – negotiation fails, the initiator restricts the domain $\mathcal{D}$ and replans.

The confirmation algorithm handles only situations described in `CASE-I`. Plans of `CASE-II` are handled as `CASE-III`, that is, the search for an internally extensible plan continues in a restricted domain. The following subsections describe improved handling of `CASE-I` and `CASE-II` in the iterative negotiation algorithm.

Handling of Internally Extensible Plans

Handling of `CASE-I` in the iterative negotiation algorithm is presented in Algorithm 3. One of the agents, called *initiator*, starts the negotiation. Other agents are called *partici-*

Algorithm 3: $\texttt{MultiPlan}(\Pi, l_{\mathsf{max}})$ in iterative negotiation scheme: Systematic search through all possible plans with backtracking (confirming actions from the beginning of the plan). Procedure $\texttt{AskAgent}(\alpha, \pi)$ queries agent α how it rates the plan π. It returns `CASE-I` if it is prefix of some *internally extensible* plan, otherwise returns `CASE-II` if it is prefix of some *publicly extensible* plan, otherwise it returns `CASE-III`.

input: problem Π and a maximum plan length l_{max}
output: a solution π of Π when it exists, $\emptyset$ otherwise
Function $\texttt{MultiPlan}(\Pi, l_{\mathsf{max}})$ **is**
  $\mathcal{D}^{l_{\mathsf{max}}} \leftarrow$ **new** $\texttt{PlanDomain}(\{\pi : |\pi| = l_{\mathsf{max}}\})$
  $\pi \leftarrow \texttt{SinglePlan}(\Pi, \mathcal{D}^{l_{\mathsf{max}}})$
  $\pi^{\checkmark} \leftarrow \texttt{CorrectPlan}(\pi, 1, \Pi, \mathcal{D}^{l_{\mathsf{max}}})$
  return $\pi^{\checkmark}$
end

input: plan π, index of first action that can be changed l, problem Π and the domain $\mathcal{D}$
output: a solution π of Π when it exists, $\emptyset$ otherwise
Function $\texttt{CorrectPlan}(\pi, l, \Pi, \mathcal{D})$ **is**
  repeat
    $\alpha \leftarrow \texttt{OwnerOf}(\pi[l])$
    reply $\leftarrow \texttt{AskAgent}(\alpha, (\pi[1 \dots l])^{\mathrm{pub}})$
    switch *reply* **do**
      case `CASE-I`
        if $|\pi| = l$ **then**
          return π
        end
        $\pi^{\checkmark} \leftarrow$ $\texttt{CorrectPlan}(\pi, l+1, \Pi, \mathcal{D} \oplus \langle \pi[l], l \rangle)$
        if $\pi^{\checkmark} \neq \emptyset$ **then**
          return $\pi^{\checkmark}$
        end
      end
      case `CASE-II`
        `/* Do nothing for now,`
           `will be handled by`
           `Algorithm 4.  */`
      end
      otherwise (`CASE-III`)
        `/* Do nothing. Subplan`
           $\pi[1 \dots l]$ `is not prefix of`
           `any solution.  */`
      end
    endsw
    $\mathcal{D} \leftarrow \mathcal{D} \ominus \langle \pi[l], l \rangle$
    $\pi \leftarrow \texttt{SinglePlan}(\Pi, \mathcal{D})$
  until $\pi = \emptyset$
  return $\emptyset$
end

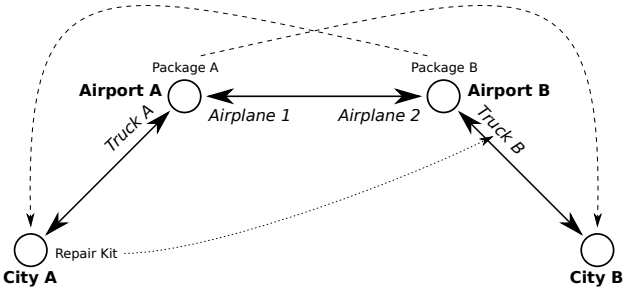

Figure 1: Logistics problem. Two packages need to be transported as shown by dashed arrows. Each of four vehicles can transport objects only between a pair of neighboring locations. In one analyzed variant, the `TruckB` is broken and it thus requires be fixed using the `RepairKit` before it can move anywhere.

pants. The initiator selects its local solution from the initial domain $\mathcal{D}^{l_{\max}}$ a tries to correct it to an internally extensible solution of a given problem. Procedure `CorrectPlan` iterates over the actions from π a tries to confirm the actions one by one, by asking the action owner to confirm action position. If the answer of `AskAgent` is `CASE-I` then it continues to query the next action. Otherwise the initiator remembers that this action can not be performed at specified position (under assumption that previously confirmed actions precede) and tries to find a different plan where this action is not required, while the previous actions in the plan remain the same. The following example illustrates Algorithm 3.

Example 1. *Let us demonstrate our algorithms on a simple logistics problem illustrated by Figure 1. They are two packages located at two airports. The goal is to transport them to distant cities. In order to achieve this goal, it is necessary to transport each package to the second airport by a plane and then to load it onto the truck and move it to the target city.*

Let's suppose that all the facts describing the location of the packages are public and all other facts are internal. Let's also suppose that the agent `AirplaneA` *decides to solve this task and starts the planning process. In the first run of planning process, it creates a plan that seems to solve the problem. An example of such a plan follows:*

```
1. load(AirplaneB, PackageB, AirportB)
2. unload(AirplaneB, PackageB, AirportA)
3. load(AirplaneA, PackageA, AirportA)
4. fly(AirplaneA, AirportA, AirportB)
5. unload(AirplaneA, PackageA, AirportB)
6. load(TruckA, PackageB, AirportA)
7. load(TruckB, PackageA, AirportB)
8. unload(TruckA, PackageB, CityA)
9. unload(TruckB, PackageA, CityB)
```

Now, the `AirplaneA` *verifies that all the agents, that are required to perform some action in the plan, can really perform the requested action. Agent* `AirplaneA` *first asks* `AirplaneB` *to load* `PackageB` *at* `AirportB` *at time 1. This is directly possible and thus the* `AirplaneB` *replies with* `CASE-I`. `AirplaneA` *then queries* `AirplaneB` *with first two actions.* `AirplaneB` *cannot perform the* `unload` *action immediately after the* `load` *action, nevertheless it is required to insert only one internal action* `fly` *to create a valid plan that can be prefix of some solution. Therefore,* `AirplaneB` *replies with* `CASE-I` *again.*

Similarly, the negotiation continues action by action to the end of the plan and then agent `AirplaneA` *can confirm that the plan is* internally extensible.

Handling of Publicly Extensible Plans

It is a more complex problem to detect whether a plan is *publicly extensible* and then to convert it into a *internally extensible* plan. In this case, the initiator α creates a plan solving Π^{α} which misses some public actions required by some participant to allow him to cooperate on this plan. When the participant is queried with a plan containing such an action, it replies with `CASE-II` as demonstrated by Example 2. Then initiator asks him for a list of missing required public action. These actions are inserted into the original plan by the initiator and they have to be confirmed by owner agents. They can contain actions that cannot be performed – then the initiator asks the participant for some alternative plan that would allow him to perform required action. Additionally, the actions returned by the participant can also contain actions owned by another participant. These actions of another participant need again to be verified.

In the `CASE-II`, the plan extension is searched in depth-first manner and thus it can yield in infinite cycle in some cases. Therefore, there has to be some limitation to stop the deepening. It can be limited by the number of plan extensions '$\circ reqActs \circ$' or by the maximal length of π' (e.g. $|\pi'| \leq 2 \cdot l_{\max}$).

Algorithm 4: If the participant marks the plan as *publicly extensible* (`CASE-II`), the initiator asks him for missing public actions using method `askRequiredActions`(α). These actions are then inserted into the current plan π.

case `CASE-II`
  while $(reqActs \leftarrow$ `askRequiredActions`$(\alpha)) \neq \emptyset$ **do**
    $\pi' \leftarrow \pi[1 \ldots (l-1)] \circ reqActs \circ \pi[l \ldots]$
    $\mathcal{D}' \leftarrow \{\pi_0 \in$
    $\mathcal{D} |\ \pi_0[1 \ldots (l-1)] \circ reqActs \circ \pi_0[l \ldots]\}$
    $\pi^{\checkmark} \leftarrow$ `CorrectPlan`$(\pi', l, \Pi, \mathcal{D}')$
    if $\pi^{\checkmark} \neq \emptyset$ **then**
      return $\pi^{\checkmark}$
    end
  end
end

Example 2. *Let's extend the previous example by* `TruckA`*'s internal state that it has broken engine and thus it cannot perform action* `unload` *requiring internal action* `move` *unless it is fixed. In order to perform the* `move`

action it has to `fixEngine` *using the* `RepairKit`*. The* `RepairKit` *is placed at* `CityB` *and thus it has to be transported by* `TruckB` *and a plane to the* `AirportA` *(location where* `TruckA` *is placed). There is no reason why the initiator would plan to move the* `RepairKit`[2]

When broken truck `TruckA` *is asked to to fulfill action* `unload` *at time 8, it creates plan containing following actions (apart from the action specified by the request):*

```
 8. load(TruckB, RepairKit, CityB)
 9. unload(TruckB, RepairKit, AirportB)
10. load(AirplaneA, RepairKit, AirportB)
11. unload(AirplaneA, RepairKit, AirportA)
12. fixEngine(TruckA, RepairKit, AirportA)
13. move(TruckA, CityA)
14. unload(TruckA, PackageB, CityA)
```

First four actions are public actions that need to be performed by other agents before the `TruckA` *can fix its engine and move to destination where it will unload the package. Therefore, the reply is* `CASE-II` *with a subplan containing these four public actions. Initiator will insert this subplan into his plan and continues the negotiation.*

Obviously, this approach can change *publicly extensible* plan into an *internally extensible* plan only if it is possible to insert required public actions immediately before the action which requires them. In some domains, this does not have to be true, and the *publicly extensibly* plan can require some public action to be inserted before some other already planned action. Nevertheless, this problem does not reduce the completeness of proposed algorithm, because the required action will be planned later by the initiator once it tries all other possible plans having that part of the plan fixed (this situation is handled similarly by Algorithm 3).

From theory to practice

We have implemented the algorithms described in the previous sections taking advantage of several existing techniques and systems. A overall scheme of the architecture of our planner is sketched at Figure 2. An input problem Π described in PDDL is translated into SAS using *Translator* script which is a part of Fast Downward[3] system. Our *Multi-SAS* script then splits SAS representation of the problem Π into agents' projections Π^α using user provided selection of public facts P^{pub}. Each agent can then compute its local *planning graph* up to level l_{max} where l_{max} is always increased by one on failure. Each planning graph represents a set of plans including all solutions if any exists. We then encode the search for a local solution in a planning graph into a SAT instance and we use MiniSat[4] solver to find a solution to the problem Π^α.

[2]Actions moving with the `RepairKit` are part of the domain $\mathcal{D}^{l_{\mathsf{max}}}$ (for some l_{max}) but let's suppose that we have a planner that prefers NOOPs to moving some object which is not required by the goal. Nevertheless, initiator uses complete method and thus it will come to the solution soon or later.

[3]`http://www.fast-downward.org/`

[4]`http://minisat.se/`

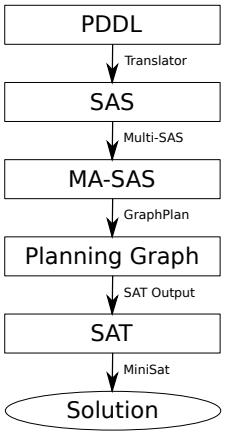

Figure 2: Architecture of the planner.

It is crucial to use suitable representation of plan domain $\mathcal{D}$ to allow effective implementation of all operators and function used in our algorithms under reasonable memory requirements. Direct listing of all the plans is obviously unfeasible. We have chosen to use *planning graphs* represented as SAT problems. In following sections we define multiagent planning graphs and describe (i) how to encode the search for a solution in a planning graph into a SAT instance and (ii) how the SAT instance is altered so that it encodes the search for a solution in a restricted domain. We conclude with a discussion of possible algorithm improvements.

Agent Planning Graphs

Agent planning graphs (APGs) stem from the classical planning graphs. Building distributed planning graphs was previously studied with focus on distribution of the Graphplan algorithm (Pellier 2010). Multiagent planning graphs were also studied recently in its relaxed form (Torreño, Onaindia, and Sapena 2012)[removed for review].

An APG is a directed, labeled and layered graph $\mathcal{G}^\alpha = (P \cup A, E)$ of one particular agent α for a local agent's planning task. As in a classical planning graph, the nodes of the graph represent propositions P and actions A. The arcs E represent linkup of propositions and actions.

In more detail, an *i*-th proposition layer and action layer will be denoted as P_i and A_i respectively. The layers alternate, so that $(P_0, A_0, P_1, A_1, \ldots, A_{n-1}, P_n)$ and all layers $P_i \subseteq P$ and all layers $A_i \subseteq A$. The first proposition layer P_0 contains nodes labeled by propositions of the agent's projection of the initial state: $P_0 = I \cap P^\alpha$.

Each action layer contains action nodes for all applicable actions of the agent α in a state represented by the previous fact layer and external projections of other agents' public actions reachable in the same layer $A_i = \{a | a \in A^\alpha_\Pi, \mathsf{pre}(a) \subseteq P_i\}$. In all successive fact layers, the nodes copy the previous fact layer according to the frame axiom and transforms the facts by actions in the previous action layer: $P_i = P_{i-1} \cup \{p | p \in \mathsf{add}(a), a \in A_{i-1}\}$.

Additionally, there is defined relation between pairs of ac-

tions and pairs of facts called *mutexes*. It is constructed during the creation of the planning graph using specific rules described in (Ghallab, Nau, and Traverso 2004). Whenever two actions or facts are in mutex, it means that they cannot be achieved simultaneously.

In our implementation the APG is built until all the facts of the goal are supported and there is no mutex between them. Then we add layers as described by Algorithm 1.

Local Plan Extraction

Initial domain $\mathcal{D}^{l_{\max}}$ for a *Planning graph* containing $l_{\max}$ layers in SAT representation contains a variable for each:

- action at each PG layer (including the *noop* actions) indicating whether the action is activated (i.e. part of the plan)
- fact at each PG layer indicating whether this atom is supported by some active action

Then it contains a formula for each:

- mutex, assuring that two mutexed actions must not be activated at the same time
- precondition of each action to ensure that it will be true if the action is activated
- fact, because it has to be supported by some active action to be true
- fact of initial state, to set it to true
- fact of goal state to require it to be true in a solution

Operations on $\mathcal{D}$ are then defined as follows:

$\mathcal{D} \oplus \langle a, l \rangle$ – a variable representing action a at layer l is required to be *true* (new formula added to the SAT representation)

$\mathcal{D} \ominus \langle a, l \rangle$ – a variable representing action a at layer l is required to be *false* (new formula added to the SAT representation)

$\texttt{SinglePlan}(\Pi, \mathcal{D})$ – a SAT solver is used to find a solution to the problem Π

Moreover, it is necessary to define how participant agent should handle a query from the initiator $\texttt{AskAgent}(\alpha, \pi)$. As described in caption of Algorithm 3, this query asks participant to assess the category the plan π – whether it is a prefix of *internally*, or *publicly extensible* plan. The participant first tries whether the provided plan is *internally extensible*, while it knows that the $\pi[1...(|\pi| - 1)]$ is *internally extensible*. Participant also stores information about internal actions that had to be inserted into this plan in order to mark it as *internally extensible*. It first match this subplan together with this information to its own planning graph $\mathcal{G}^{\alpha}$. Then, in first iteration, it tries whether the action $\pi[|\pi|]$ can directly follow previous actions by forcing this action to be used at appropriate layer. If it is not possible it tries this action at another layer allowing only internal action to be inserted in created gap. This continues until some limit ($l_{\max}$). If this process did not succeed then we know that this plan is not *internally extensible*. Similar procedure is used to detect *publicly extensible* plans.

Improvements

The described algorithm and its implementation can be improved in several ways. We use planning graph to create initial set of plans $\mathcal{D}^{l_{\max}}$. It can contain several actions at one layer, that are independent and can be executed in parallel. The initiator can query these actions in parallel. If all agents reply `CASE-I` then the initiator can continue with next action. If some agent replies `CASE-III`, the whole layer of action is forbidden in domain $\mathcal{D}$ and new plan has to be generated. If some agents reply `CASE-II`, their required action can be added into the plan in any order. It is also possible to only add actions required by one agent and continue with algorithm, because once it reaches the queried action again other agents will probably reply `CASE-III` again and then other required actions will be injected into the plan.

A participant replying with `CASE-III` can also take the initiative and continue the negotiation instead of the original initiator. This can be easily implemented in the case when only one agent is queried at a time. In the case of parallel queries, it is necessary to handle commitments of agents to different initiators to not promise excluding actions.

Experiments

For our experiments, we have used the *Tool Problem* (Tožička et al. 2014) that allows us to smoothly change an amount of public and internal actions between two extreme cases: (i) there is no internal action and (ii) there are as many internal actions as possible (implied by the equal sign in the definition of P^{pub}). Case (i) allows the initiator to construct a correct plan immediately without any communication, while case (ii) might require some negotiation. An advantage of case (ii) is, however, that initiator's plan is not so complex and contains less actions.

We have focused on the following three criteria: (1) number of SAT solver invocations by an initiator and participants, (2) the complexity of communication between the initiator and the participants (number of positive and negative responses), and (3) time complexity. We have measured these criteria as a function of the *publicness* of a problem, where the publicness is a number between 0 and 100 defined as $\frac{\#\text{ public actions}}{\#\text{ all actions}} * 100$. All our experiments have been carried out on CPU Intel Core-Duo 1.4GHz.

Tool Problem

In the *Tool Problem*, the goal requires that each of N agents performs its `doGoal` action. Nevertheless, in order to perform this action, agent need to `useTool` first. Then, there is an agent that provides the tools (`handTool`). This agent will be in the role of initiator. Initial state is that none of the agents has its tool and initiator has all of them. Goal state is that all tools have been used by actions `doGoal`.

In this problem, minimal publicness is 66.6% – all participants' actions `doGoal` have to be public because facts contained in goal are public and these actions have them as their effects; Initiator's actions `handTool` also have to be public because some of their effects are required by some other agents' actions; and actions `useTool` are internal. The maximal possible publicness is 100% when all actions

are public. In our experiments we continuously change the visibility of `useTool` actions and thus the publicness of the whole problem.

Results

Let us present results for the Tool Problem where $N = 5$. Each publicness settings was run 20 times. Another experiments for $N = 3$ and 4 yield similar results. All results are presented as a function of publicness of the problem (X-axis).

SAT Solver Invocations Figure 3 shows that with the increasing publicness the number of initiator's and participants' SAT Solver invocations decrease. The reason for it is that the more actions are available to the initiator the easier it is to find a local plan that all participants mark as *internally extensible*. In an extreme case, when all actions are public, the initiator performs only one SAT solver invocations and finds an *internally extensible* plan immediately.

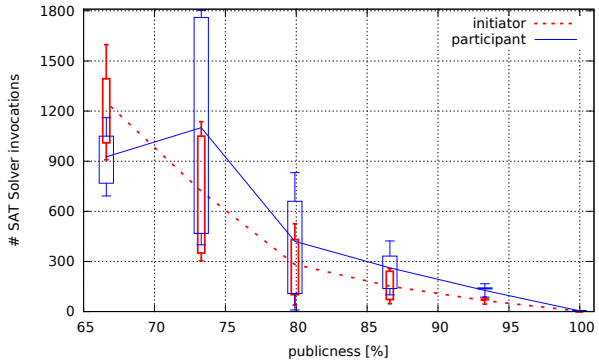

Figure 3: Average initiator's and participants' SAT Solver invocations depending on the problem publicness.

Number of `CASE-I` and `CASE-III` Replies The number of participants' `CASE-I` and `CASE-III` replies depending on the problem publicness is shown in Figure 4. It can be seen that in case of the minimal publicness (66%), the initiator creates many plans that participants reply with `CASE-III`. Note that the number of `CASE-I` replies first increases and then decreases with growing publicness. That is caused by the increase of relative number of `CASE-I` replies because a plan for one agent (whose actions are public) is correct, while the whole number of generated plans does not decrease significantly.

Time Complexity An average time of finding the solution depending on the amount of publicness is shown in Figure 5. Note that in the case of publicness of 66%, less time is required than in the case of publicness of 73%, although there are more SAT solver invocations in total. That is because the SAT problem is more constrained and it often does not have any solution, which can be often proved very easily by the SAT solver.

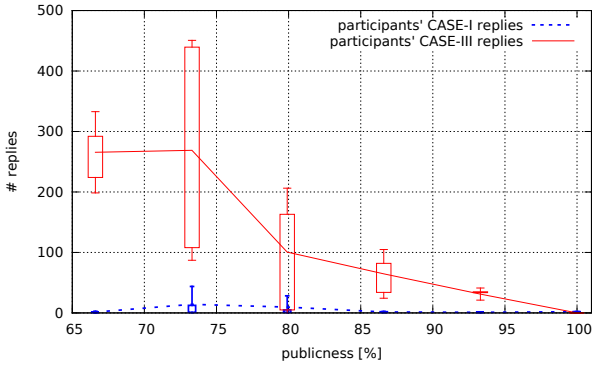

Figure 4: Average `CASE-I` and `CASE-III` replies depending on the problem publicness.

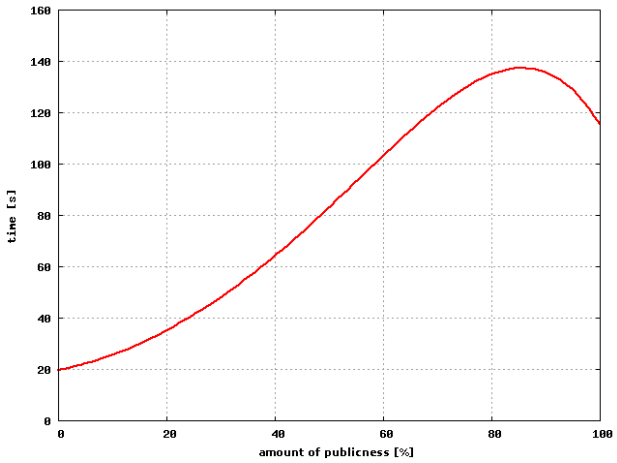

Figure 5: Amount of time required to solve the problem depending on the problem publicness.

Final remarks

The planning technique we proposed is carried out by a group of cooperative planning agents. Initially, each agent awaits a relevant part of a deterministic planning domain and a problem. After receiving the inputs, the agents firstly prepare their own local planning graphs based on the parts of the problem they have. Secondly, the agents use their individual SAT solvers to extract a local solution from their planning graphs. And finally, they negotiate actions from other agents to help them and not to interfere with the other agents by additional constraining of the SAT solving processes, forming a negotiation loop. If all the agents find a local plan and the plans provide all the requested goals without any conflicts, the planning process ends. Otherwise, the negotiation process continues until a solution is found.

Acknowledgements

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

References

Brafman, R., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of ICAPS'08*, volume 8, 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., ed., *A Modern Approach to Distributed Artificial Intelligence*. San Francisco, CA: The MIT Press. chapter 3.

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*, 608–620.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Huang, R.; Chen, Y.; and Zhang, W. 2012. Sas+ planning as satisfiability. *J. Artif. Intell. Res. (JAIR)* 43:293–328.

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

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

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

Tožička, J.; Jakubův, J.; Durkota, K.; Komenda, A.; and Pěchouček, M. 2014. Multiagent planning supported by plan diversity metrics and landmark actions. In *International Conference on Agents and Artificial Intelligence (ICAART)*.

Zhang, J. F.; Nguyen, X. T.; and Kowalczyk, R. 2007. Graph-based multiagent replanning algorithm. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, 122:1–122:8. New York, NY, USA: ACM.