Research Workshop of the
Israel Science Foundation

Proceedings of the 3rd Workshop on

Distributed and Multi-Agent Planning (DMAP-15)

Edited By:

Antonín Komenda, Michal Štolba, Dániel L. Kovacs and Michal Pěchouček

Jerusalem, Israel 7/6/2015

Organizing Committee

Antonín Komenda
Czech Technical University in Prague, Czech Republic

Michal Štolba
Czech Technical University in Prague, Czech Republic

Dániel Laszlo Kovacs
Budapest University of Technology and Economics, Hungary

Michal Pěchouček
Czech Technical University in Prague, Czech Republic

Program committee

Daniel Borrajo, Universidad Carlos III de Madrid, Spain
Ronen Brafman, Ben-Gurion University of the Negev, Israel
Raz Nissim, Ben-Gurion University of the Negev, Israel
Eva Onaindia, Universidad Politecnica de Valencia, Spain
Michael Rovatsos, The University of Edinburgh, UK
Alessandro Saetti, Università degli Studi di Brescia, Italy
Scott Sanner, National ICT Australia (NICTA), Australia
Matthijs Spaan, Delft University of Technology, Netherlands
Roni Stern, Harvard University, USA
Alejandro Torreño, Universidad Politecnica de Valencia, Spain
Mathijs de Weerdt, Delft University of Technology, Netherlands
Gerhard Wickler, The University of Edinburgh, UK
Shlomo Zilberstein, University of Massachusetts, Amherst, USA

Foreword

This volume compiles the scientific papers accepted at DMAP'15, the 3rd Distributed and Multi-Agent Planning workshop, held at ICAPS 2015 in Jerusalem, Israel, on June 7th, 2015. This event follows up the successful first and second edition of DMAP, held at ICAPS 2013 and 2014, and continues the tradition of the "Multiagent Planning and Scheduling" workshop series held at ICAPS 2005 and 2008, and the joint AAMAS-ICAPS session on multi-agent planning in 2010.

Distributed and Multi-Agent Planning is a broad field of growing interest within the planning and scheduling community. However, its subfields remain dispersed and uncoordinated. Most works in this field are generally published in major AI conferences, such as IJCAI, and AAAI, AAMAS or ICAPS. Nevertheless, most of these approaches are based or built upon planning and scheduling technologies developed by the ICAPS community.

The aim of the workshop is to bring together the single- and multi-agent planning community, offering researchers a forum to introduce and discuss their works in the different subfields of this area and thus narrow the gap between centralized and distributed planning. Similarily as the last year, the authors were given the opportunity to introduce their works also in the poster session of the main ICAPS conference.

This year's papers cover many of topics of multi-agent planning, such as privacy-preservation, probabilistic and non-deterministic multiagent planning, game theory, adversarial plan recognition, epistemic multiagent-planning and heuristics, among others. Therefore, the workshop offers a wide view of the state-of-the-art in multi-agent planning, fostering works that have the potential to push this field forward.

Altogether 11 submissions were received from 7 different countries, 10 full papers and 1 short position paper, matching the numbers of the first edition of the workshop. These papers were reviewed by a Program Committee composed of 13 members and coordinated by the 4 PC chairs. The accepted papers will be presented orally at the workshop and 6 will be also presented in the poster session of the main ICAPS conference.

Newly, the workshop is accompanied by a preliminary multiagent planning competition, CoDMAP (Competition of Distributed and Multiagent Planners). The competition is meant to be a preliminary version of possible future MAP track at the International Planning Competition (IPC). The competition focuses on comparing domain-independent, offline planners for cooperative agents. The MAP problems the planners have to solve during the competition are discrete-time, deterministic and fully observable, except for the introduction of privacy, as described in MA-STRIPS. To cope with the multiple agents and privacy, the base language of determinstic IPC, PDDL3.1, was extended with the required features to serve as a base language of CoDMAP. For maximal openness, CoDMAP consists of two tracks: centralized highly compatible "transitional" track for various multiagent planners and "experimental" proper track for distributed multi-agent planners. Altogether 8 planners were submitted to the competition and 3 of them were apt to compete in both tracks, the rest compete only in the centralized track.

We thank the members of the Program Committee for their dedicated effort at reviewing and ensuring the quality of the works presented at DMAP'15. We also thank the chairs of the ICAPS main conference for their continuous support throughout the organization of this workshop. Finally, we would like to thank our authors and the multi-agent planning community for submitting their work to DMAP'15. Your research, collaboration and active participation are critical to thesuccess of this workshop and for the progress of the field of multi-agent planning.

– Antonín Komenda, Michal Štolba, Dániel L. Kovacs, Michal Pěchouček
DMAP-2015 Chairs

Table of Contents

A Privacy Preserving Algorithm for Multi-Agent Planning and Search

Ronen I. Brafman
Ben-Gurion University of the Negev
Be'er Sheva, Israel
brafman@cs.bgu.ac.il

Abstract

To engage diverse agents in cooperative behavior, it is important, even necessary, to provide algorithms that do not reveal information that is private or proprietary. A number of recent planning algorithms enable agents to plan together for shared goals without disclosing information about their private state and actions. But these algorithms lack clear and formal privacy guarantees: the fact that these algorithms do not require agents to *explicitly* reveal private information, does not imply that such information cannot be deduced. The main contribution of this paper is an enhanced version of the distributed forward-search planning framework of Nissim and Brafman that reveals less information than the original algorithm, and the first, to our knowledge, discussion and formal proof of privacy guarantees for distributed planning and search algorithms.

Introduction

As our world becomes better connected and more open ended, production becomes more customized, and autonomous agents are no longer science fiction, a need arises for enabling groups of agents to cooperate in generating a plan for diverse tasks that none of them can perform alone in a cost-effective manner. Indeed, much like ad-hoc networks, one would expect various contexts to naturally lead to the emergence of ad-hoc teams of agents that can benefit from cooperation. Such teams could range from groups of manufacturers teaming up to build a product that none of them can build on their own, to groups of robots sent by different agencies or countries to help in disaster settings. To perform complex tasks, these agents need to combine their diverse skills effectively. Planning algorithms can help achieve this goal.

Most planning algorithms require full information about the set of actions and state variables in the domain. However, often, various aspects of this information are private to an agent, and it is not eager to share them. For example, the manufacturer is eager to let everyone know that it can supply motherboards, but it will not want to disclose the local process used to construct them, or its inventory levels. Similarly, rescue forces of country A may be eager to help citizens of country B suffering from a tsunami, but without having to provide detailed information about the technology behind their autonomous bobcat to country B, or to country C's humanoid evacuation robots. In both cases, agents have public capabilities they are happy to share, and private processes and information that support these capabilities, which they prefer (or possibly require) to be kept private.

With this motivation in mind, a number of algorithms have recently been devised for distributed privacy-preserving planning [Bonisoli *et al.*, 2014; Torreño *et al.*, 2014; Luis and Borrajo, 2014; Nissim and Brafman, 2014]. In these algorithms, agents supply a public interface only, and through a distributed planning process, come up with a plan that achieves the desired goal without being required to share a complete model of their actions and local state with other agents. But there is a major caveat: it is well known from the literature on secure multi-party computation [Yao, 1982; 1986] that the fact that a distributed algorithm does not require an agent to *explicitly* reveal private information does not imply that other agents cannot deduce such private information from other information communicated during the run of the algorithm. Consequently, given that privacy is the raison-d'etre for these algorithms, it is important to strive to improve the level of privacy provided, and to provide formal guarantees of such privacy properties. To date, no such guarantees have been provided.

In this paper we focus on the multi-agent forward search (MAFS) algorithm [Nissim and Brafman, 2014]. Forward search algorithms are the state-of-the-art in centralized classical planning, and their distributed version scales up best among distributed algorithms for classical multi-agent planning. We describe a modified version of MAFS, called SECURE-MAFS in which less information is exposed to other agents, and we provide the first formal treatment of privacy in the context of distributed planning algorithms. SECURE-MAFS is not guaranteed to provide complete privacy always; indeed, we doubt that search based algorithm can provide such guarantees without fundamental new techniques because of the fact that information about intermediate states is being constantly exchanged. Yet, we are able to provide a sufficient condition for privacy, and use it to prove a number of results, placing the discussion of privacy in multi-agent planning on much firmer ground.

The Model

We use a multi-valued variable variant of the MA-STRIPS model [Brafman and Domshlak, 2008]. This framework minimally extends the classical STRIPS language to MA planning for cooperative agents. The main benefits of this model are its simplicity, and the fact that it is easily extended to handle the case of *non-cooperative* agents [Nissim and Brafman, 2013]. There is abundant work on multi-agent planning that uses setting much richer than the classical one (e.g., [Durfee, 2001; Bernstein *et al.*, 2005; Jonsson and Rovatsos, 2011; ter Mors *et al.*, 2010] to name a few). We focus on MA-STRIPS because it is so basic – a sort of "classical planning" for multi-agent system. As such, most planning models that use a factored state space, extend it in various way, and it would be easier to import the ideas and techniques for distribution and privacy preservation from it to such models. Moreover, the techniques we use are not specific to planning, but can be used for other applications in which distributed, privacy preserving search is needed.

Definition 1. *A MA-STRIPS planning task for a set of agents* $\Phi = \{\varphi_i\}_{i=1}^k$ *is given by a 4-tuple* $\Pi = (V, \{A\}_{i=1}^k, I, G)$ *with the following components:*

- V *is a finite set of finite-domain variables,*
- *For* $1 \leq i \leq k$, A_i *is the set of actions that* φ_i *can perform. Each action* $a = \langle pre,eff,c \rangle \in A = \cup A_i$ *is given by its preconditions, effects, and cost, where preconditions and effects are partial assignments to* V.

A *solution* to a planning task is a plan $\pi = (a_1, \ldots, a_k)$ such that $G \subseteq a_k(\ldots(a_1(s)\ldots)$. That is, it is a sequence of actions that transforms the initial state into a state satisfying the goal conditions. A solution is *cost-optimal* if it has minimal total cost over all possible solutions.

As an example, consider the well known Logistics classical planning domain, in which packages should be moved from their initial to their target locations. The agents are vehicles: trucks and airplanes, that can transport these packages. The packages can be loaded onto and unloaded off the vehicles, and each vehicle can move between a certain subset of locations. Variables denoting possible locations are associated with each package and vehicle. Possible actions are *drive, fly*, *load*, and *unload*, each with its suitable parameters (e.g., *drive(truck, origin, destination)* and *load(package, truck, location)*). The actions associated with each particular agent are ones of loading and unloading from the corresponding vehicle, and moving it around its possible locations. MA-STRIPS specifies this association of actions to agents explicitly as part of its problem description.

Privacy

Privacy guarantees in MA planning come in the form of *private variables, values,* and *private actions*. If a value is private to an agent, then (ideally) only it knows about its existence. If an action is private to an agent, (ideally) only it knows about its existence, its form, and its cost. In this paper we shall use the deduced notion of private variables and private actions described below, following [Brafman and Domshlak, 2008]. Alternatively, one can specify privacy requirements as part of the input. In that case, if privacy is defined over variable values, our results, with the exception of completeness (which can be affected by the choice of private variables) carry through.

A *private* variable-value of agent φ is a value required and affected *only* by the actions of φ. A *private* variable is one, all of whose values are private. We refer to all other variables and variable-values as *public*. The *private state* of agent φ_i in state s, denoted by s_{pr-i}, consists of the private variable-value pairs that hold in s. An action is *private* if all its preconditions and effects are private. All other actions are classified as *public*. That is, φ's private actions affect and are affected only by the actions of φ, while its public actions may require or affect the actions of other agents. Note, however, that the public actions of an agent may have private preconditions and effects. Thus, the *public projection* of an action a is an action that contains only the public preconditions and effects of a. For ease of the presentation, we assume that all actions that achieve a goal condition are *public*. Our methods are easily modified to remove this assumption.

In Logistics, all vehicle locations are private (affect and are affected only by their respective agent's *move* actions). Thus, the location of a vehicle is a private variable. *move* actions are also private, as they depend on and affect only the private location values. Package location values can be either private or public, depending on whether they are required/affected by more than one agent. Thus, *at(package)=location* is private if *location* is accessible only to one agent and public otherwise. Since, typically, some package locations are public (e.g., the airport), package location is not a private variable, although some of its values are private. Therefore, *load* (respectively *unload*) actions that require (respectively affect) public package location propositions are public.

We can now talk about the privacy guarantees of an algorithm. We say that an algorithm is *weakly private* if no agent communicates a private value of a variable (in the initial state, the goal, or any other intermediate state) to an agent to whom this value is not private during a run of the algorithm, and if the only description of its own actions it needs to communicate to another agent φ is their public projection.

Private values of one agent do not appear in preconditions or effects of other agents' actions. Thus, they do not influence their applicability. Hence, weak privacy is easily achieved by encrypting the private values of a variable. However, agents may deduce the existence and properties of private variables, values, and action preconditions, from information communicated during the run of the algorithm. For example, *at(package)=truck* is a value private to the truck because it appears only in the preconditions and effects of its load and unload actions. However, suppose a package is in location A at one state, but is no longer there at the following state. Other agents can deduce that the variable *at(package)* has some private value. Or, consider the location of the truck. This is a private variable, and thus, other agents should not be able to distinguish between states that differ only in its value. However, when the truck is in location a, it has children in which packages appear or disappear from a, whereas when it is in location b, packages will appear or disappear from loca-

tion b. From this, other agents can deduce the existence of a private variable that distinguishes between these two states. Moreover, they can also infer the number of its possible values. While it is not necessarily the case that agents will be able to make these deductions, the lack of formal guarantees to the contrary is worrisome.

In the area of *secure multi-party computation* [Yao, 1982; 1986], a subfield of Cryptography, stronger privacy guarantees are sought from multi-party algorithms. The goal of methods for secure multi-party computation is to enable multiple agents to compute a function over their inputs, while keeping these inputs private. More specifically, agents $\varphi_1, \ldots, \varphi_n$, having *private* data $x_1, \ldots, x_n$, would like to jointly compute some function $f(x_1, \ldots, x_n)$, without revealing any information about their private information, other than what can be reasonably deduced from the value of $f(x_1, \ldots, x_n)$. Thus, we will say that a multi-agent planning algorithm is *strongly private* if no agent can deduce information beyond the information that can be deduced from its own actions' description, the public projection of other agents' actions, and the public projection of the solution plan, about the existence of a value or a variable that is private to another agent, or about the model of an action. More specifically, we will say that a variable or, a specific value of a variable is *strongly private* if the other agents cannot deduce its existence from the information available to them.

Strong privacy is not simply a property of an algorithm. It requires that we consider issues such as: the nature of the other agents, their computational power, and the nature of the communication channel. For example, the other agents may be curious but honest, that is, they follow the algorithm faithfully, but will try to learn as much as they can from the other agents. Or, they may be malicious, i.e., they will deviate from the algorithm if this allows them to gain private information. How much information agents can learn will depend on the information they are exposed to within the algorithm, whether they share information with other agents, what their computational power is, and even the nature of the communication channel – e.g., whether it is synchronous or asynchronous, and whether it is lossy.

Existing algorithms for classical multi-agent planning guarantee only weak privacy, which, as observed above, is a somewhat superficial notion. The first discussion of potentially stronger privacy guarantees appeared in [Nissim and Brafman, 2014] in the context of the MAFS algorithm. But that discussion came short of providing formal results and tools. Indeed, in MAFS agents can infer information about an agent's private state from the public part of its descendent states, as illustrated above for the truck's location variable. The goal of this paper is to provide an enhanced version of MAFS, called SECURE-MAFS, that reveals less information and comes with clearer privacy guarantees and tools.

The MAFS Algorithm

The multi-agent forward search algorithm (MAFS) [Nissim and Brafman, 2014] performs a distributed version of forward-search planning. In fact, it is a schema for distributing search algorithms while preserving some privacy. The distribution is along the "natural" lines. An agent φ_i will expand a state using its own actions only. If the public projection of an action of φ_j is applicable in this new state, φ_i will send the state to φ_j who will expand it. This idea can be applied to any search algorithm, although a bit of care needs to be taken to identify solutions, especially if an optimal one is sought (see [Nissim and Brafman, 2014] for the details).

Algorithms 1-3 depict the MAFS algorithm for agent φ_i. In MAFS, each agent maintains a separate search space with its own *open* and *closed* lists. It expands the state with the minimal f value in its open list, which is initialized with the agent's projected view of the initial state. When an agent expands state s, *it uses its own operators only*. This means that two agents (that have different operators) expanding the same state, will generate *different* successor states.

Since no agent expands all relevant search nodes, messages must be sent between agents, informing one agent of open search nodes relevant to it expanded by another agent. Agent φ_i characterizes state s as relevant to agent φ_j if φ_j has a public operator whose public preconditions (the preconditions φ_i is aware of) hold in s, and the creating action of s is public. In that case, Agent φ_i will send s to Agent φ_j.

Algorithm 1 MAFS for agent φ_i

```
1: Insert I into open list
2: while TRUE do
3:    for all messages m in message queue do
4:       process-message(m)
5:    s ← extract-min(open list)
6:    expand(s)
```

Algorithm 2 process-message($m = \langle s, g_{\varphi_j}(s), h_{\varphi_j}(s) \rangle$)

```
1: if s is not in open or closed list or g_{φi}(s) > g_{φj}(s) then
2:    add s to open list and calculate h_{φi}(s)
3:    g_{φi}(s) ← g_{φj}(s)
4:    h_{φi}(s) ← max(h_{φi}(s), h_{φj}(s))
```

Algorithm 3 expand(s)

```
1: move s to closed list
2: if s is a goal state then
3:    broadcast s to all agents
4:    if s has been broadcasted by all agents then
5:       return s as the solution
6: if the last action leading to s was public then
7:    for all agents φj ∈ Φ with a public action whose public preconditions are satisfied in s do
8:       send s to φj
9: apply φi's successor operators to s
10: for all successors s' do
11:    update g_{φi}(s') and calculate h_{φi}(s')
12:    if s' is not in closed list or f_{φi}(s') is now smaller than it was when s' was moved to closed list then
13:       move s' to open list
```

Messages sent between agents contain the full state s, the cost of the best plan from the initial state to s found so far, and the sending agent's heuristic estimate of s. The values of private variables in s are encrypted so that only the relevant agent can decipher it. By definition, if q is private to an agent,

other agents do not have operators that affect its value, and so they do not need to know its value. They can simply copy the encrypted value to the next state.

When φ receives a state via a message, it checks whether this state exists in its open or closed lists. If it does not appear in these lists, it is inserted into the open list. If a copy with higher g value exists, its g value is updated, and if it is in the closed list, it is reopened. Otherwise, it is discarded. Once an agent expands a *solution* state s, it sends s to all agents and awaits their confirmation, which is sent whenever they expand, and then broadcast state s (Line 3 of Algorithm 3). For more details, see [Nissim and Brafman, 2014].

Secure MAFS

MAFS does not require agents to know private information of other agents, nor is such information provided to them. Although agents see the private state of other agents, it is encrypted, and they cannot and need not manipulate it. Thus, it is weakly private. However, agents are exposed to more information than the projected public actions of other agents and the solution plan because they receive many intermediate states during the search process. From this information, it may be possible, in principle, to deduce private information. For example, from the complete search tree, even with encrypted private states, an agent may be able to deduce that the state of an agent is identical in two different search states based on the public parts of the sub-tree rooted at them. Thus, it could deduce the number of private variables of other agents. And maybe, potentially, construct a model of the transitions possible between them.

In reality, agents are not exposed to the entire search space, as only a small portion of it is explored typically, some states are not sent to them, and it may not be easy to deduce that one state is a parent of another state. However, this observation is neither a proof nor a guarantee that MAFS maintains privacy. We now develop SECURE-MAFS, a variant of MAFS for which we can offer better guarantees.

The basic idea is as follows: the effect of φ_i's action on the components of the state that are not private to φ_j is the same, regardless of φ_j's private state. Thus, if φ_j knows the effect of φ_i's action on a state s_1, it knows its effect on a similar state s_2 that differs only in its private state. In fact, the same holds true for a sequence of actions executed by agents other than φ_j. Hence, based on the work done on s_1 by other agents, φ_j can simulate the effect of the work that will be done on s_2 without actually sending it. The resulting algorithm provides better privacy, and can also lead to fewer messages.

Given a state s, recall that s_{pr-i} is the private state of φ_i in s. We write s_{-i} to denote the part of state s that is not private to agent φ_i. Thus, s can be partitioned to s_{pr-i} and s_{-i}. We write s_i to denote the part of the state that consists of variables private to i and public variables. That is, the information about s that is readily accessible to φ_i. We say that s' is an i-child of state s (and s is an i-parent of s') if s' is obtained from s by the application of a sequence $a_1, \ldots, a_l$ of actions of agents other than φ_i, and the last action in this sequence, a_l, is public. Intuitively, this means that s' is a state that would be sent by some agent to φ_i, and φ_i was not involved in the generation of any state between s and s'.

Instead of a regular closed list, each agent φ_i maintains a set of three-tuples. The first part is the *key* which contains a non-private part of a state that was sent (i.e., something of the form s_{-i}). The second part is a list of private parts of states consistent with the key, that were sent or were supposed to be sent. The third part is the non-private part of their i-children. That is, if state s was sent, there must be a three-tuple with key s_{-i} and second part containing s_{pr-i}. The second part may contain additional private states, s'_{pr-i}, for every state s' sent by the agent that satisfies: $s'_{-i} = s_{-i}$. The third part may be initially empty, but will grow to contain s''_{-i} for every state s'' such that s'' is an i-child of s or of any s' whose private part appears in the second list. That is, the third part is a list of i-children of states whose non-private part is identical to s that were sent or would have been sent in MAFS to φ_i.

The modified *expand*, called *secure-expand*, differs in two elements. Instead of sending the state only to agents that have an applicable action (line 7 in *expand*), we send it to all agents. And we replace "send" (line 8 in *expand*), with "virtual-send." The virtual-send (algorithm 6) ensures that no two states with the same non-private component will be sent by an agent. Suppose that φ_i wants to send state s to other relevant agents. First, it checks whether in the past it sent a state s' such that $s_{-i} = s'_{-i}$. If not, it sends s, as before, but adds a new three-tuple to the list of sent items of the form $\langle s_{-i}, \{s_{pr-i}\}, \{\}\rangle$. If, however, a state s' such that $s_{-i} = s'_{-i}$ was sent earlier, then it simply adds s_{pr-i} to the second part of the three-tuple whose key is s'_{-i}, and it *does not* send s. Instead, it "virtually" sends it, emulating the answer that it would get. That is, for every i-child s'' of s' in this three-tuple, it sends itself a message $\hat{s}$ such that $\hat{s}_{-i} = s''_{-i}$ and $\hat{s}_{pr-i} = s'_{-i}$. That is, it knows that if in the past it sent a message s' and received an i-child s'' then if it sends s now, it will receive something that is identical to s'', except that its private part will be the same as in s.

Algorithm 4 secure-process-message($m = \langle s, g_{\varphi_j}(s), h_{\varphi_j}(s)\rangle$)

1: Let $S = \{s\}$
2: **if** s_{-i} has an i-parent **then**
3: Let s'_{-i} be the i-parent of s_{-i}
4: **for all** element s''_{pr-i} in the 2nd component of the tuple with key s'_{-i} **do**
5: Replace s_{pr-i} with s''_{pr-i} in s
6: $S = S \cup \{s\}$
7: **else**
8: Replace s_{pr-i} with I_{pr-i}
9: $S = S \cup \{s\}$
10: **for all** $s \in S$ **do**
11: **if** s is not in open or closed list **or** $g_{\varphi_i}(s) > g_{\varphi_j}(s)$ **then**
12: add s to open list **and** calculate $h_{\varphi_i}(s)$
13: $g_{\varphi_i}(s) \leftarrow g_{\varphi_j}(s)$
14: $h_{\varphi_i}(s) \leftarrow max(h_{\varphi_i}(s), h_{\varphi_j}(s))$

Secure-process-message modifies *process-message* to be consistent with the above ideas. First, when a state s'' is re-

Algorithm 5 secure-expand(s)

```
1: move s to closed list
2: if s is a goal state then
3:    broadcast s to all agents
4:    if s has been broadcasted by all agents then
5:       return s as the solution
6: if the last action leading to s was public then
7:    virtual-send s to all agents
8: apply φ_i's successor operators to s
9: for all successors s' do
10:   update g_{φ_i}(s') and calculate h_{φ_i}(s')
11:   if s' is not in closed list or f_{φ_i}(s') is now smaller than
      it was when s' was moved to closed list then
12:      move s' to open list
```

ceived by φ_i, we identify the state s such that s'' is an i-child of s. (Technically, this is done by maintaining a state id in the private part of states sent). We then record this fact in the three-tuple that corresponds to s_{-i}. The fact that s'' is an i-child of s implies that for every s' such that $s'_{-i} = s_{-i}$ there exists an i-child $\hat{s}$ such that $\hat{s}_{-i} = s''_{-i}$ and $\hat{s}_{pr-i} = s'_{pr-i}$. Thus, we treat $\hat{s}$ as if it, too, was just received by φ_i and add it to φ_i's open list. The modified code for the two procedures appears in Algorithms 4 & 5.

Algorithm 6 virtual-send(s,φ_j)

```
1: if a three-tuple with key s_{-i} exists then
2:    for all i-children s'_{-i} in this three-tuple do
3:       Let s̄ = s'_{-i} · s_{pr-i}. That is, augment s'_{-i} with the
         local state of φ_i in s.
4:       Add s̄ to your open list
5: else
6:    Replace s_{pr-i} with a unique id for tracking i-children
7:    Send s to φ_j
8:    Create new three-tuple: {s_{-i}, {s_{pr-i}}, {}}.
```

The net effect of these changes is that no two states with the same non-private part are ever sent, yet the entire reachable search-space can be explored. This may have computational advantages because fewer messages are sent, but here we focus on the impact on privacy. The other agents will always see a single private state associated with every non-private part of the global state.

Soundness and Completeness

Lemma 1. *If s is inserted into the open list of an agent φ_i then s is reachable.*

Proof. First, we note that MAFS and SECURE-MAFS generate only action sequences in which a private action of an agent is followed by another action of that agent. It was shown in [Nissim and Brafman, 2014] that for any goal involving public variables only, it is sufficient to consider such sequences only.

The proof is by induction on the step in which the state s was inserted into the open list. We assume an interleaving semantics, as not true concurrency is required by the algorithm. The base case is I, which is the first state inserted into any open list, which is reachable. For the inductive step, suppose s was inserted into φ_i's open list. There are three cases:

1. s was inserted by an expansion of some state s' via action a (line 12 of *secure-expand*). Since s' was inserted into the open list of φ_i earlier, it is reachable, and so is s.

2. s was inserted in line 12 of *secure-process-message*. This means that prior to the insertion of s, a message containing some state s' was received by φ_i from φ_j. If s' was sent, it was taken out of the open list of φ_j earlier, and hence it is reachable. We know that s is obtained from s' as follows: we find the i-parent of s', s''. s'' was sent earlier by i, and hence it is reachable. There is some state s''' that appeared in the open list of φ_i prior to the insertion of s, such that $s'''_{-i} = s''_{-i}$ and s is defined so that $s_{-i} = s'_{-i}$ and $s_{pr-i} = s'''_{pr-i}$. We claim that s is reachable.

To see this, notice that all states between s'' and s' must have been obtained by expansion. Otherwise, some component of s'' and s', other than s''_{pr-i} must be different, in which case $s'''_{-i} \neq s''_{-i}$. Because $s'''_{-i} = s''_{-i}$, the same sequence of actions is applicable at s''', and must lead to a state whose non-private component must be the same as s'_{-i}. Its private component must be the same as s'''_{pr-i} because that private state is not affected by these actions and none of these actions are in A_i (by definition of i-parent).

3. s was inserted in line 4 of *virtual send.* This is similar to the above case. s' is taken out of φ_i's open list (and thus, reachable). We see that some other state s'', such that $s''_{-i} = s'_{-i}$ has already been sent by φ_i, and is thus reachable. It has an i-child s'''. Following the same line of reasoning, s, which is obtained by applying to s' the same actions that led from s'' to s''' is also reachable.

□

To simplify the remaining proofs we assume, without loss of generality, that there are public actions only. This can be achieved by replacing every set A_i with all legal sequences of private actions followed by a single public action, all from A_i. There is no loss of generality because no information is exchanged following private actions, and a pubic goal is reachable iff it is reachable via a sequence of actions in which a private action by an agent is always followed by another action (private or public) by the same agent.

Lemma 2. *If s is reachable via some sequence of actions then s will be generated by some agent.*

Proof. The proof is by induction on the length of the action sequence $a_1, \ldots, a_k$ leading to s.

For $k = 0$, $s = I$ which is inserted initially to all open lists. Next, consider a sequence of length $k+1$ reaching s. Assume $a_{k+1} \in A_i$. Denote the state reached following the first k actions by s'. By the inductive hypothesis, s' was generated by some agent φ_j. If $i = j$ then that agent will also generate s. Otherwise, we know that φ_j must virtual-send s' to all agents eventually (as it inserts it into its open list and, unless a goal is found earlier, will eventually expand it and send it) and in particular, to φ_i. If it actually sends s', then we know φ_i will insert it into its open list and eventually expand it, generating s. If it does not send it to φ_i, we know that φ_j sent some other state s'' to φ_i, where $s'_{-j} = s''_{-j}$. a_{k+1} is applicable at

s'' because of this. Thus, at some point φ_i will apply it and send $a_{k+1}(s'')$ to φ_j. The latter would recognize that it is an i-child of s'', and would generate a state s''' such that $s'''_{-j} = (a_{k+1}(s''))_{-j}$ and $s'''_{pr-j} = (a_{k+1}(s''))_{pr-j}$. However, since $a_{k+1} \notin A_j$, $s''' = a_{k+1}(s') = s$, as required. □

We note that the soundness and completeness of SECURE-MAFS also implies that the secure variant of MAD-A* remains optimal. SECURE-MAFS can also lead to the transmission of fewer messages compared to MAFS. In MAFS' *expand,* a state is sent to all agents that have an applicable public action at that state. From the completeness proof above, it can be seen that SECURE-MAFS is still complete if the state is sent only to agents that have an applicable action in that state *and* the agent that sent the parent state. Thus, SECURE-MAFS will send at most one more message per state than MAFS, whereas if there are multiple paths that differ only on the private effects (e.g., driving in different routes, or using different vehicles, or tools) its caching mechanism will refrain from sending all but one state to *all* agents with applicable actions.

Privacy Guarantees

We now show that SECURE-MAFS provides strong privacy guarantees under certain conditions. While a search-based approach in which intermediate search nodes are shared among agents is unlikely to be strongly private always, we can show that SECURE-MAFS maintains strong privacy of some aspects of the problem.

The forward search tree G^{Π} associated with a planning task Π is a well known concept. It contains the initial state, and for every state in it, it contains, as its children, all states obtained by applying all applicable actions to it. Leaf nodes are goal states that satisfy the optimization criteria (e.g., in satisficing planning, all goal states; in optimal planning, goal states with optimal f value) and dead-ends. In principle, the states in the tree can be generated in any topological order, but the use of a particular search algorithm restricts the order further. We define the *$-i$-projected tree*, G^{Π}_{-i}, to be the same tree, except that each state s is replaced by s_{-i}.

At present we do not have techniques that can take into account differences in cost and heuristic value and their impact on the order of generation of different sequences of actions by a particular search algorithm (but see the discussion). Thus, as before, we focus on the simpler case where all actions are public, have unit cost, and the heuristic value for all states does not depend on the private variables. While there is no loss of generality in assuming all actions are public, assuming that actions have identical cost and the heuristic value is not affected by the private part of the state is a restriction, especially if we assume that private actions are compiled away (in which case their cost is ignored).

Theorem 1. *Let Π and Π' be two planning problems. If $G^{\Pi}_{-i} = G^{\Pi'}_{-i}$ and A_i contains unit cost public actions only, and the heuristic value of a node depends only on its non-private component, then agents other than φ_i cannot distinguish between an execution of* SECURE-MAFS *on Π and Π'*

Proof. $G^{\Pi}_{-i} = G^{\Pi'}_{-i}$ implies that the search-tree is identical, except for, possibly, the various private parts (and multiplicity) of states with identical non-private part. Given our assumption on g and h, the expansion order of this tree depends only on the non-private component of the states, which is identical in both cases. Since messages depend only on the non-private part, this means that in both cases, φ_i will send identical messages. Different messages may be virtually sent, but this will not impact what other agents see. Since other agents see the same messages, they will send the same messages to φ_i. □

Theorem 1 provides a tool for proving strong privacy. As an example, consider the case of independent private variables. Formally, v is an *independent private* variable of φ_i if v is a private variable of φ_i and if v appears in a precondition of a then all the variables that appear in a's description are private, and all its effects are on independent private variables. For example, suppose that the truck doesn't only deliver goods, but the driver sells drugs on the side, having actions of buying and selling them at various places. The action of selling/buying a drug at a location has no effect on public variables, nor on private variables that can impact public variables.

Given a set of variables, the actions in the domain induced by the removal of these variables contain the original actions, except for those in which the removed variables appear as preconditions and/or effects. In our example, if we remove the variable *has-drugs(agent)*, the actions of selling and buying drugs are also removed.

Lemma 3. *Let Π be a planning problem and let R be a set of independent private variables of φ. Agents other than φ_i cannot distinguish between an execution of* SECURE-MAFS *on Π and Π_{-R}, the problem obtained by removing R, assuming R does not influence h.*

Proof. The addition or removal of private independent variables does not influence G^{Π}. It simply leads to additional public actions (when we compile away private actions) that differ only in their effect on the variables in R. Since the value of variables in R does not influence our ability to apply any of the other actions, $G^{\Pi}_{-i} = G^{\Pi_{-R}}_{-i}$. □

We can further illustrate the power of the above techniques by showing that SECURE-MAFS is strongly private for logistics. We say that a location is public if more than one agent can reach it.

Lemma 4. *Under the above assumptions,* SECURE-MAFS *is strongly private for logistics.*

Proof. The public information in logistics is whether or not a package is in a public location. Thus, consider two planning problems Π and Π' that have the same set of packages, the same set of public locations, and, initially, identical locations for packages that are in locations that are not private to φ_i. We claim that $G^{\Pi}_{-i} = G^{\Pi'}_{-i}$. To see this, note that two states that differ only in the location of packages that are in places private to φ_i have the same sub-tree, projected to $-i$ under the assumption that every private location is reachable from every other private location. Thus, from Theorem 1 it follows that agents cannot distinguish between these domains. □

Suppose, further, that logistics is augmented with public actions that are executable in a private location. For example, suppose that the amount of fuel of an agent is public and agents can fuel in private locations. The above result remains true. On the other hand, consider the, admittedly contrived, logistics variant in which certain private locations are reachable from some other private locations only by passing through some public locations, and that trucks must unload their cargo when passing in a public location. In that case, $G^{\Pi}_{-i} \neq G^{\Pi'}_{-i}$, and we would not be able to apply Theorem 1. Thus, we see that our strong privacy proofs are sensitive to the specifics of the domains. Finally, note that in all proofs above, we assume that the agents are honest but curious, and these proofs are correct even if all agents collude, combining their information.

We now briefly, and informally, consider two other domains. The *satellites* domain is strongly private, both for MAFS and SECURE-MAFS because no satellite supplies or destroys preconditions of actions of another satellite, and the only public actions are actions that achieve a sub-goal. The only information one agent can learn about another agent when this property holds is whether or not it is able to achieve a sub-goal. Private sub-goals can be modelled by adding a proposition such as *private-sub-goals-achieved* and an action that achieves it with the set of private sub-goals as its (private) preconditions, and the above remains true. The *rovers* domain is more similar to *logistics*. Agents can block or free a shared resource, the channel, required for communication, and some of the sub-goals are achievable by multiple agents. Because of the shared channel, all communication actions are public. Private information includes the location of the agent, the state of its internal instruments, the existence of such instruments, and what information the agent has acquired. We claim that this information is strongly private in SECURE-MAFS in the following sense. If an agent can move between locations freely, without requiring intermediate public actions (which implies that it can collect information in whatever order it wishes), the projected search trees of two planning problems in which the agents have identical capabilities, that is, they can acquire the same information (pictures, soil samples, etc.), are identical. Thus, external agents cannot differentiate between them.

Discussion and Summary

We presented SECURE-MAFS, a new variant of the state-of-the-art MAFS algorithm with better privacy guarantees and potentially fewer messages. Beyond SECURE-MAFS, our central contribution is the first formal discussion of strong privacy in planning, a sufficient condition for strong privacy in SECURE-MAFS, and an illustration of its use. Consequently, we believe this work can play an important role in placing the discussion and analysis of privacy preserving planning algorithms on much firmer ground.

There is much left for future work. We feel that the notion of strong privacy requires additional development to capture some of our intuitions, and new proof techniques. Specifically, our current central technique focuses on showing the equivalence of two problem domains, but can we give a constructive technique for showing that a certain variable is strongly private? Another issue is the restriction to unit cost actions and heuristics that are not sensitive to private variables. A simple variant of SECURE-MAFS with a form of ϵ-exploration might overcome this problem: with probability ϵ the choice of next node to expand from the open list is random, and with probability $1 - \epsilon$ we expand the first node. In that case, the expansion order is less sensitive to the heuristic value, and it is more difficult to differentiate between search trees for different problems. We note that in this respect, MAFS has a certain advantage: it appears that an agent cannot tell the relation between a sequence of nodes generated by another agents, i.e., whether the nodes are siblings or are ancestors of one another. This property could be useful for proving privacy guarantees for MAFS.

As pointed out by [Bonisoli *et al.*, 2014], one can and should consider planning models with finer grained notions of privacy. Specifically, they allow variables and actions that are private to an arbitrary subset of agents, and specify such requirements as part of the input. It is also possible to introduce a similar *derived* notion of privacy with the aim of introducing as much privacy as possible without sacrificing goal reachability. This can be done as follows: we say that $v = val$ is *private* to φ_i if there exists an action $a \in A_i$ such that $v = val$ is a precondition of a. Note that $v = val$ can be private to a number of agents, but if $v = val$ was private to φ_i according to the previous definition, it will be private to φ_i only, according to the current definition. If $v = val$ was public according to the previous definition, it can now be private to a strict subset of the agents. We use $(v = val)_{prv}$ to denote the set of agents to whom $v = val$ is private. An action is *private* to an agent φ_i, if for all values $v = val$ in its description, $(v = val)_{prv} = \{\varphi_i\}$. Otherwise, we still refer to the action as *public*.

To adapt MAFS to this finer notion of privacy, agents must encrypt the value of variable q using a key that is shared among the agents in q_{prv}. All other elements remain the same. We believe it is clear that, at least as far as variable values are concerned, one cannot require stricter encryption requirements, as an agent must know when its actions are applicable, which requires knowing when the value of their preconditions hold. It is, however, an open question whether a variants of SECURE-MAFS exists for this finer notion of privacy, that supports the requirement that agents will not see two states that differ only on variables that are not private to them.

Acknowledgements: I would like to thank the anonymous IJCAI'15 and DMAP'15 WS reviewers for their useful comments. This work was supported by ISF Grant 933/13 and by the Lynn and William Frankel Center for Computer Science, and the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev.

References

[Bernstein *et al.*, 2005] Daniel S. Bernstein, Eric A. Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized pomdps. In *IJCAI*, pages 1287–1292, 2005.

[Bonisoli *et al.*, 2014] Andrea Bonisoli, Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. A privacy-preserving model for the multi-agent propositional planning problem. In *ICAPS'14 Workshop on Distributed and Multi-Agent Planning*, 2014.

[Brafman and Domshlak, 2008] Ronen I. Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, pages 28–35, 2008.

[Durfee, 2001] Edmund H. Durfee. Distributed problem solving and planning. In *EASSS*, pages 118–149, 2001.

[Jonsson and Rovatsos, 2011] Anders Jonsson and Michael Rovatsos. Scaling up multiagent planning: A best-response approach. In *ICAPS*, 2011.

[Luis and Borrajo, 2014] Nerea Luis and Daniel Borrajo. Plan merging by reuse for multi-agent planning. In *ICAPS'14 Workshop on Distributed and Multi-Agent Planning*, 2014.

[Nissim and Brafman, 2013] Raz Nissim and Ronen I. Brafman. Cost-optimal planning by self-interested agents. In *AAAI*, 2013.

[Nissim and Brafman, 2014] Raz Nissim and Ronen I. Brafman. Distributed heuristic forward search for multi-agent planning. *Journal of AI Research*, 51:292–332, 2014.

[ter Mors *et al.*, 2010] Adriaan ter Mors, Chetan Yadati, Cees Witteveen, and Yingqian Zhang. Coordination by design and the price of autonomy. *Autonomous Agents and Multi-Agent Systems*, 20(3):308–341, 2010.

[Torreño *et al.*, 2014] Alejandro Torreño, Eva Onaindia, and Oscar Sapena. Fmap: Distributed cooperative multi-agent planning. *Applied Intelligence*, 41(2):606–626, 2014.

[Yao, 1982] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

[Yao, 1986] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

Privacy Preserving Pattern Databases

Shlomi Maliah and **Guy Shani** and **Roni Stern**
Information Systems Engineering
Ben Gurion University of the Negev, Beer-Sheva, Israel
shlomima@post.bgu.ac.il and shanigu@cs.bgu.ac.il and roni.stern@gmail.com

Abstract

In a privacy preserving multi-agent planning problem agents have some private information they do not wish to share with each other, but operate to achieve common goals. Based on the Multi-Agent STRIPS (MA-STRIPS) formalism, several search-based algorithms were proposed to solve these type of planning problems. To guide the search, these algorithms require a heuristic function that estimates the cost of reaching a goal. Developing such heuristics is challenging because the computation of the heuristic may violate the agents privacy. In this work we show how pattern databases, a general-purpose class of heuristic that can be very effective in many domains, can be implemented in a privacy preserving manner. As a secondary contribution, we present an improvement to the GPPP, a recent effective suboptimal privacy preserving planner and also demonstrate how privacy preserving heuristics can be used even when the goals are not known to all agents. Experimentally, we show the effectiveness of the proposed heuristic and improved GPPP planner, showing substantial speedups and lower solution cost over previously proposed heuristics on some benchmark domains.

Introduction

Many modern organizations outsource some of their tasks to outside companies. The organization must then work together with the outsourcing companies to achieve its goals, while disclosing as little as possible about its abilities. Consider for example the case of a military organization that has outsourced its food service to an outside company. The food service company must deliver food into logistics centers, from where it is picked by army trucks and distributed to the army bases. The military would not want, however, to disclose the whereabouts of these bases, the number of people in each base, or the number and location of army trucks. The two organizations must then collaborate while maintaining privacy about their actions, and communicating only over public actions, such as the interactions at the logistics centers.

The scenario above is an example of a multi-agent planning problem where the agents agree to cooperate to achieve a joint goal but do not agree to share some of their private information. This type of problems were recently defined in the context of the Multi-Agent STRIPS formalism (Brafman and Domshlak 2008), and several privacy preserving planners were introduced (Maliah, Shani, and Stern 2014; Nissim and Brafman 2014; Torreño, Onaindia, and Sapena 2014; Luis and Borrajo 2014; Bonisoli et al. 2014). Privacy in MA-STRIPS is embodied by partitioning the set of actions each agent can perform into public and private actions. Similarly, the set of facts are partitioned into public and private facts. During planning and execution, the agents only agree to share the state of the public facts and which public actions are performed. Any knowledge about the private facts and actions must not be passed to the other agents. Stronger forms of privacy are possible but are not addressed in this paper.

Some of the most successful privacy preserving MA-STRIPS planners are based on heuristic search techniques (Maliah, Shani, and Stern 2014; Nissim and Brafman 2014). The efficiency of heuristic search planners, in general, greatly depends on having an accurate heuristic function that estimates the cost of reaching a goal from a given state. Several methods have been developed for generating effective domain-independent heuristic functions for single-agent planners. Migrating these single agent heuristic functions to be used in a privacy preserving planner is challenging because part of the evaluated state is private, and thus unavailable to the heuristic function. Ignoring the private part of a state can result in a grossly uninformed heuristic.

Maliah et al. (2014) showed how a privacy preserving landmark heuristic can be computed based on the single-agent landmark heuristic (Hoffmann, Porteous, and Sebastia 2004; Richter, Helmert, and Westphal 2008). In this paper we propose a privacy preserving heuristic function that is based on pattern databases (PDBs) (Culberson and Schaeffer 1998; Edelkamp 2001; Felner, Korf, and Hanan 2004), which are a very popular and successful family of single-agent heuristic functions.

The proposed privacy preserving heuristic is called *Privacy Preserving Pattern Database* (3PDB), and works as follows. In a preprocessing stage a lookup table – the PDB – is populated with costs of plans for transitioning between public facts. Importantly, this PDB is populated in a collaborative and privacy preserving manner. When planning,

a state is evaluated by searching for public facts that can be achieved from that state using private actions, and then considering the distance stored in the PDB for reaching goal facts from these achievable public facts. The resulting inadmissible heuristic is privacy preserving and outperforms the previous landmark-based heuristic substantially in some MA-STRIPS domains. In addition, we also discuss how to compute 3PDB even for cases where the goal is private.

As a secondary contribution, we propose an improvement to the Greedy Privacy Preserving Planner (GPPP) (Maliah, Shani, and Stern 2014). GPPP is a state-of-the-art suboptimal privacy preserving MA-STRIPS planners that operates in two phases. First, a greedy best-first search is performed on a special type of relaxed planning problem, resulting in a high-level plan comprising a sequence of public actions the agents need to perform to achieve their goals. Then, each agent attempts to find a plan in which it will perform the public actions dictated to it by the high-level plan. We propose an improvement on GPPP that considers which public facts each public action achieves, and then analyzes when each of these facts can be achieved. Then, each agent plans to achieve a set of public facts together, instead of performing a single public action. This leverages the strength of current single agent planners and results in runtime speedup and better solution cost.

Both contributions of this work, the 3PDB heuristic and the improved GPPP, are empirically evaluated on 6 domains based on known single-agent IPC domains, adapted to the multi-agent setting. Results show that in some domains the 3PDB heuristic substantially outperformed the previously proposed privacy preserving landmark heuristic, and the combination of 3PDB with the improved GPPP almost always outperforms the landmark based heuristic with the original GPPP.

Background

We review required background on single-agent heuristics and privacy preserving multi-agent planning.

Single-Agent Heuristics

One of the dominant approaches for single-agent planning these days is forward heuristic search. Key to the success of heuristic search planners is a highly effective heuristic function. Heuristic functions can be roughly classified into four families: *landmarks*, *abstractions*, *delete relaxation*, and *critical paths*. Helmert and Domshlak (2009) analyzed the theoretical relations between these families, and many recent planners employ a combination of heuristics from different families. Our approach is motivated by pattern databases heuristics, which are a highly successful form of abstraction heuristics, and also draws from delete relaxation heuristics, used in the well-known FF planner (Hoffmann 2001) and many other state-of-the-art planners (Richter and Westphal 2010). We provide here only a very brief, high level, description of these heuristics.

A *pattern database* (PDB) is a lookup table that stores costs of plans in an *abstract version* of the original planning problem. This abstract version of the problem is based on a homomorphism f mapping every state s in the state space of the original problem to an abstract state $f(s)$ in the abstract state space. The PDB stores the cost of plans between abstract states. To compute a heuristic for reaching state s' from state s, we use the cost stored in the database for reaching $f(s')$ from $f(s)$. Perhaps the most classical example of PDBs is in the classical tile-puzzle problem, where the identity of some tiles is ignored in the abstract state space. Usually, the PDB is constructed before planning for many start and goal states, and then the heuristic can be computed very fast when planning.

Delete relaxation heuristics estimate the cost of reaching a goal from a state s by solving a *relaxed version* of the original problem. The terms "abstraction" and "relaxed", while similar in their semantic meaning, have different formal meaning in the planning literature. Abstractions are homomorphisms between the original state space and an abstract state space. Relaxed planning problems are planning problems in which the delete effects of actions are ignored, i.e., an action only adds facts to the state and never deletes facts. For example, the action of moving a block from one place to another, would result in the block being in two locations in a relaxed planning problem. While solving relaxed planning problems optimally is NP-Hard (Bylander 1994), it is possible to solve them suboptimally in a computationally efficient manner. Most delete relaxation heuristics assign to a state s the estimated cost of the plan from s to a goal in the relaxed planning problem.

Preserving Privacy in MA-STRIPS

The MA-STRIPS model is defined as follows.

Definition 1 (MA-STRIPS) *An MA-STRIPS problem is represented by a tuple $\langle P, \{A_i\}_{i=1}^k, I, G\rangle$ where:*

- *P is the set of possible literals (facts about the world)*
- *$I \subseteq P$ is the set of literals true at the start state*
- *k is the number of agents*
- *A_i is the set of actions that agent i can perform.*
- *$G \subseteq P$ is the set of literals that needs to be true after executing the generated plan (i.e., the goal state)*

Each action in A_i has the standard STRIPS syntax and semantic, that is $a = \langle$ pre(a), add(a), del(a) $\rangle$. The sets A_i are disjoint, that is, no public action can be executed by two different agents.

Each agent has a set of private actions and private facts that are only known to it. In earlier work, the set of private actions and private facts were deduced from the problem definition as follows. A fact is regarded as private if only one agent has actions that has this fact as a precondition or as an effect. All other facts are public. An action is regarded as a public action, if at least one of its preconditions or effects is a public fact. In our experiments, we used this domain-deduced partition to public/private actions/facts, but the proposed heuristic and algorithm can accept any such partition.

Privacy Preserving Planners There are many ways to define the "privacy preserving" property. In this work we aim for a *weak form of privacy preserving planning* (Nissim and Brafman 2014) in which private states and actions are not directly revealed to other agents during planning or execution. Of course this does not mean that agents learn nothing about the abilities of other agents. For example, when agent a publishes, as we do, that it can achieve a fact using n actions, the other agents may deduce some lower bound on the number of private actions of a. In principal, one can measure the information revealed by every message sent to other agents, for example by computing the entropy reduction on the possible states of the agent following the message. That being said, we leave a thorough discussion of privacy measurements to future research, and constrain our algorithms here not to explicitly reveal any private facts that can be achieved, or private actions that can be executed.

Recently, several privacy preserving planners have been proposed for MA-STRIPS. For a complete survey see Nissim and Brafman (2014). Two leading privacy preserving planners, called Multi-Agent Forward Search (MAFS) (Nissim and Brafman 2014) and Greedy Privacy Preserving Planner (GPPP) (Maliah, Shani, and Stern 2014), are based on forward heuristic search. In MAFS, each agent runs a best-first search independently, but when generating a state with a public action, that state is broadcasted to all agents, which can then continue to expand that state by performing their actions. Privacy is preserved by encrypting the private part of the broadcasted state. MAFS is very general in the sense that it can be configured to return optimal solutions and suboptimal solutions (Nissim and Brafman 2014); and is even applicable for self-interested agents (Nissim and Brafman 2013). GPPP is a recent suboptimal privacy preserving planner shown to perform better than MAFS. We describe GPPP in detail later in this paper, as improving it is one of our contributions.

Privacy Preserving Pattern Database

We now describe our novel heuristic, called 3PDB. Next, we describe 3PDB in details, including: 1) how the PDB is created offline, and 2) how the 3PDB heuristic uses this PDB during search.

Computing the PDB

The PDB used for computing the 3PDB heuristic stores for pairs of public facts f_1 and f_2 the estimated cost of a plan that achieves f_2, starting from a state in which f_1 is true. The construction of this PDB is done in an offline phase, listed in Algorithm 1. Initially, the PDB is empty (line 1). Then, each agent tries to find a plan for reaching f_2 from a state with f_1 using only its private and public actions. This phase is denoted in the pseudo code as **SearchForPlan** (line 4), and was implemented in our experiments by running the FF planner.[1] If a plan from f_1 to f_2 was found, its cost is stored in the PDB. For clarity of presentation we assume the same PDB is shared by all agents but such "shared memory" can be implemented in a distributed way using message passing — each agent maintains its own PDB, and broadcasts notifications after every modification to the PDB. Importantly, the shared PDB does not compromise the weak privacy constraint, as the PDB only contains pairs of public facts and the cost of the corresponding plan. Thus, this process does not explicitly reveal any private fact or action.

Algorithm 1: Construct the PDB for the 3PDB heuristic

```
1 PDB ← ∅
2 foreach Agent a_i do
3     foreach f_1 ∈ P_pub do
4         foreach f_2 ∈ P_pub do
5             SearchForPlan(a_i, f_1, f_2, PDB)
6 UpdateTransitiveClosure(PDB)
```

In some cases, cooperation of multiple agents is needed to achieve f_2 from f_1. For example, maybe no one agent can achieve f_2 from f_1, but one agent can achieve f_3 from f_1 and another agent can achieve f_2 from f_3. Thus, together they can achieve f_2 from f_1. To reflect these kind of collaborative plans, the PDB construction process iteratively tries to improve the existing entries in the PDB, and add new entries to the PDB. This is done by computing the *transitive closure* of the PDB entries, denoted in the pseudo code as **UpdateTransitiveClosure** (line 6). For example, if there is an entry for $\langle f_1, f_2 \rangle$ with cost 10, and an entry for $\langle f_2, f_3 \rangle$ with cost 5 then computing the transitive closure will add an entry for $\langle f_1, f_3 \rangle$ with cost 10+5=15, unless a lower cost entry already exist in the PDB. Computing the transitive closure can be done either by exhaustively iterating through all pairs of facts until no improvements are achieved (similar to dynamic programming), or by considering each PDB entry as an action transitioning between facts, and searching the lowest cost transition between any two pairs using an optimal search algorithm. We implemented the latter option, which is more efficient.

One may consider filling all entries in the PDB by running a privacy preserving multi-agent solver for every pair of public facts. In a preliminary study we observed that computing a PDB in this way was very time consuming, and orders of magnitude slower than the method we described above.

Computing the 3PDB Heuristic

After the PDB is computed, it is used for every state to compute the 3PDB heuristic as follows. The pseudo code describing the 3PDB heuristic computation for a given state s is presented in Algorithm 2. Each agent a_i computes all the public facts it can achieve from the start state in relaxed planning problem, i.e., where delete effects are ignored (line 2), and records the cost of achieving each of these public facts. The set of resulting public facts is denoted by $Pubs_i$ and the cost of achieving a public fact $f \in Pubs_i$ in this relaxed

[1] We also experimented with using the A* (Hart, Nilsson, and Raphael 1968) algorithm, to find optimal plans for the PDB. Experimentally, we did not observe substantial difference between the performance of the two algorithm for solving these simple single fact problem.

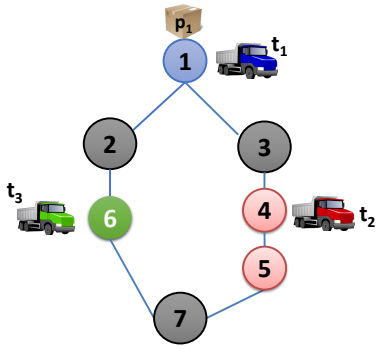


Figure 1: Illustration of the scenario

planning problem is denoted by $cost(f)$. Then, each agent computes the cost of reaching g from s, for every goal g that is not true in s. This is done by summing $cost(f)$ and the stored entry in the PDB for reaching g from f (denoted $PDB(f, g)$), for every fact $f \in Pubs_i$. The minimum over these costs is denoted by $cost(g, i)$, and represents an estimation of the cost it will take agent a_i to achieve g (line 4).

For every goal g that is not true in s, every agent a_i computes and publishes $cost(g, i)$ to all agents. Note that if a_i cannot achieve g or achieve a fact f that has a PDB entry from f to g, then a_i publishes ∞. The resulting heuristic for state s sums for every g the minimum $cost(g, i)$ over all agents (line 7). This approach, of adding the estimated cost of reaching each goal independently, is also used in the h^{add} heuristics (Bonet and Geffner 2001).

As an example of how 3PDB works, consider the simple logistics scenario in Figure 1. Each agent controls a single truck, grayed nodes (2, 3, and 7) are known to all agents, and the colored nodes (1, 4, 5, 6) are private locations, known only to a single agent. Location 1 is private for agent t_1, location 6 for agent t_3, and locations 4 and 5 for agent t_2. Thus, every fact related to these specific locations would be private. The goal is to get package p_1 to location 7.

Table 1 shows the PDB precomputed for this scenario. For example, the PDB entry for moving package p_1 from location 2 to location 3 has a cost of 4. Indeed, the shortest plan would be to load p_1, move to location 1, then location 3, and then unloading p_1 at location 2. Note that there is no

Algorithm 2: Compute the 3PDB heuristics

Input: s, the state for which to compute the heuristic
1 **foreach** *Agent* a_i **do**
2 $Pubs_i \leftarrow$ all public facts a_i can achieve
3 **foreach** *g, a fact from the goal not satisfied in* s **do**
4 $cost(g, i) \leftarrow \min_{f \in Pubs_i}(PDB(f, g)$+cost($f$))
5 $h \leftarrow 0$
6 **foreach** *g, a fact from the goal not satisfied in* s **do**
7 $h \leftarrow h + \min_{a_i}(cost(g, i))$
8 **return** h

Fact 1	Fact 2	Cost
package p_1 at 3	package p_1 at 2	4
package p_1 at 7	package p_1 at 2	4
package p_1 at 2	package p_1 at 3	4
package p_1 at 7	package p_1 at 3	5
package p_1 at 2	package p_1 at 7	4
package p_1 at 3	package p_1 at 7	5

Table 1: The generated PDB for Figure 1.

entry for getting from p_1 at 1 to the goal because p_1 at 1 is a private fact. The 3PDB computes all public facts achievable from the current state. In our case, only agent t_1 can achieve public facts, which are: p_1 at location 2 and p_1 at location 3. Achieving each of these public facts costs 3 (load p_1, move, unload p_1). Thus, the 3PDB heuristic will return $h = 3 + 4$, corresponding to the cost of achieving p_1 at location 2 (=3) plus the PDB entry for getting p_1 from location 2 to the goal (=4).

The scenario in Figure 1 also emphasizes a case where our 3PDB heuristic is more informed than a landmark-based heuristic. Consider the two states generated by expanding the state where p_1 is at location 1. These are the states where p_1 is at location 2, denoted s_2, and the state where p_1 is at location 3, denoted s_3. Our 3PDB heuristic will return $h(s_2) = 4$ and $h(s_3) = 5$, thus expanding first s_2 and eventually reaching the goal with a low cost plan. By contrast, landmark-based heuristic, such as the previously proposed privacy preserving landmark heuristic (PPLM) (Maliah, Shani, and Stern 2014), would give s_2 and s_3 the same heuristic value, because the disjunctive landmark p_1 at location 2 or location 3 is satisfied in both states.

Handling Private Goals

Previous research on privacy preserving planing often assumes that all goals are public facts, and does not explicitly address the case where some of the facts in the goal are private (Nissim and Brafman 2014). Such a cases poses a serious challenge to heuristic search – how can an agent estimate the cost of a plan for a goal that it is not aware of? This is also a challenge for our 3PDB heuristic, as private goal facts will not have a corresponding entry in the PDB.

We have implemented the following method to overcome such cases. Every agent that knows about a private goal preforms a backwards search from that goal in order to find all the public facts that, if achieved, would enable reaching the private goal. Then, that agent publishes these public facts to all agents as additional goals. It is possible that there are several sets of such public facts, where achieving all the facts in any one of these sets would enable reaching the goal. In such cases, the disjunction of all these sets of facts is added as a goal.

The way these "artificial" goals are handled is exactly the same as all other goal facts, except for the following differences.

- When the 3PDB heuristic computes the cost of achieving an "artificial" goal, the agent that published this goal also

adds to the computation the cost of reaching the real, private, goal after achieving that goal.

- In case of a disjunction "artificial" goal, the heuristic value would be the minimum over all the sets of "artificial" goals.

This technique for handling private goals is based on classical landmark detection algorithms (Hoffmann, Porteous, and Sebastia 2004). A deeper study of different ways to handle private goals is beyond the scope of this paper.

Improved GPPP

We developed an improvement to GPPP that results in a reduction of runtime and solution cost. To describe this improvement, we first explain how GPPP works.

The GPPP Algorithm

The GPPP algorithm can be explained as having two phases.

- **High-level planning.** This is a collaborative search effort, aiming to find a sequence of public actions the agents should perform to achieve the goal.

Algorithm 3: GPPP

```
1  Plan(start)
       Input: start, the start state
2      OPEN ← {start}
3      while OPEN is not empty and goal not found do
4          s ← choose best s from OPEN
5          foreach Agent a_i do
6              children ← GetChildren(s,a_i)
7              foreach child ∈ children do
8                  if child is a goal node then
9                      P_public ← the plan for this goal
10                     P_full ← Ground(start,P_public)
                       /* If public plan Ok   */
11                     if P_full is not null then
12                         return P_full
13                 else
14                     Compute h(child) and insert child
                       to OPEN
15 Ground(start,P_public)
       Input: start, the start state
       Input: P_public, a sequence of public actions
16     P_full ← ⟨⟩
17     current ← start
18     foreach action a ∈ P_public do
19         P_a ← Plan(current,a)
20         if P_a was found then
21             current ← execute P_a on current
22             Append P_a to P_full
23         else
24             return Null
25     return plan_full
```

- **Grounding.** This is a set of single-agent planning problems, where each agent generates a private plan to perform the public actions dictated by the plan generated in the high-level planning phase.

A key technique used in GPPP (introduced earlier in MAFS) is to encrypt the private facts of the state. For a given state, each agent maintains a *private state index* (PSI). The PSI of state s and agent a_i is an index that a_i can map to the set of its private facts that are true in s. Importantly, only agent a_i is able to map its PSIs to the private facts they represent. A state s can be fully represented by a set of public facts and a set of PSIs, one per agent. This representation of a state can be safely broadcasted and shared by all agents without loss of privacy. Below, we assume that states are always represented in such a way.

Algorithm 3 lists the pseudo-code for GPPP. GPPP can be implemented in a distributed manner, but we describe it here as a centralized algorithm for ease of exposition. Maliah et al. (2014) further discusses a distributed implementation for GPPP. The high-level planning phase in GPPP is a best-first search, maintaining an open list (denoted OPEN) of nodes that are considered for expansion. Initially, the initial state is inserted into OPEN (line 2). Then, in every iteration the best node according to the heuristic function (3PDB or any other heuristic) is popped out of OPEN (line 4). Each agent then checks which public actions it can perform next. This is done approximately by considering a relaxed planning problem in which private actions do not have delete effect, and performing all applicable private actions until we find all private facts that can be achieved without performing public actions. We call this set of facts the *achievable private facts*. Assuming that all achievable private facts are true, each agent applies all public actions that can be performed at this state (line 6) and the generated states are added to OPEN (line 14).[2]

If one of the generated states is a goal state, then the corresponding plan is extracted (line 9) and the grounding phase begins (line 10). Note that the plan passed to the grounding component consists of only public actions. We call this plan the *public plan*, denoted by P_{public}. Grounding a public plan is done by solving a sequence of single-agent planning problems, one for each action in the public plan. Let $P_{public}[i]$ be the i^{th} action in the public plan, $a_{P(i)}$ be the agents that needs to perform it, and $P_{public}[1]$ be the first action. The single-agent planning problem that corresponds to the $P_{public}[1]$ is for agent $a_{P(1)}$ to find a private plan from the start state that enables it to perform the action $P_{public}[1]$. Agent $a_{P(1)}$ then applies this plan and action $P_{public}[1]$ to the start state. The resulting state is the initial state for the subsequent planning problem, which is for agent $a_{P(2)}$ to find a plan that will enable it to perform action $P_{public}[2]$. This continues until either the goal state is reached (after all the actions in the public plan were performed), or until one of these single-agent planning problems fails to find a solution. In the former case, the grounding succeeded and the

[2]For a generated state s', the facts that are mutually exclusive with the preconditions or effects of the action used to generate s' are removed from the set of achievable private facts of s'.

agents have a full plan to achieve the goal (line 12). In the latter case, the grounding fails and the high-level planning phase continues to search for a different public plan. Note that the plans generated during grounding are not shared between the agents. All that is shared is the current state, and thus, privacy is preserved.[3]

Improved Grounding

Grounding, as outlined in Algorithm 3, is done sequentially, one action at a time. This can be inefficient, both in terms of runtime and in terms of solution cost. Consider for example, the logistic example shown in Figure 2c, where the goal is to put package p_1 at location 4, and package p_2 at location 6. Locations 4 and 6 are known to both agents. Facts related to locations 1,2, and 3 are private facts of agent t_1, and facts related to location 5 are private facts of agent t_2. The public plan generated for this problem consists of the following public actions:

- $A1$: Agent t_1 unloading p_1 at location 4
- $A2$: Agent t_1 unloading p_2 at location 4
- $A3$: Agent t_2 loading p_2 at location 4
- $A4$: Agent t_2 unloading p_2 at location 6

Grounding these public actions one at time would result in truck t_1 driving to location 2, loading p_1, driving to location 4 to unload it, and then driving back to location 1 to pick up p_2. Table 2a lists the full grounding from public plan to concrete plans. Clearly, it would be more efficient for t_1 to drive to pick up both packages and then drive to location 4 to unload the both packages. A better grounding could have been found if agent t_1 was tasked to achieve the effects of $A1$ and $A2$ (which are that p_1 and p_2 are at location 4) in a single planning problem. Table 2b lists this improved grounding, showing that the resulting plan has a lower cost (15 instead of 18) compared to the plan generated by the regular grounding. Moreover, this solution may be found faster, as fewer planning problem are solved in order to ground the problem (3 instead of 4).

Our improved GPPP tries to capture the intuition behind the example given in Figure 2c. The first step in our improved grounding is to consider *grounding public effects of actions instead of actions*. This means that the task of a grounding agent is not to find a plan that reaches a state where a public action can be performed, but to find a plan that reaches a state where the required public effect is achieved.

Definition 2 (Public Effect) *The public effect of an action A is a tuple $\langle add_P(A), del_P(A)\rangle$, where $add_P(A)$ is the list of public facts in A's add list, and $del_P(A)$, is the list of public facts in A's delete list. Achieving a public effect of an action A means reaching a state in which the facts in $add_P(A)$ are true and the facts in $del_P(A)$ are not.*

[3]For clarity, the pseudo code of Algorithm 3 shows that each agent returns its grounding plan, but in fact only the current state is shared.

Grounding a public effect corresponds to finding a plan in which that public effect is achieved. Grounding public effects instead of actions offer greater flexibility, as there may be several plans that achieve the same public effect.

The second step in our improved grounding is to try identify which action effects can be grounded together. Grounding a set of public effects E corresponds to reaching a state in which all the facts in the add lists of all the public effects in E are true, and all the facts in the delete lists of all the public effects are false.

The challenge is how to know which public effects can be grounded toghether, and how to order these groups of public effects. One may consider applying algorithms for plan parallelisation (Bäckström 1998) to identify which public actions can be performed in parallel, and ground together their public effects. Plan parallelisation algorithms, however, cannot be directly applied in here, as these algorithms require that preconditions and effects of all actions in the plan are known, while in privacy preserving planning, some of these preconditions and effects may be private.

We propose a privacy preserving method to estimate which public effects can be grounded together. The approach we take is that we only allow grounding public effects together if it does not prevent any public effects of the public plan from being achieved. More formally, let $E_P[i]$ be the public effects of that action. We allow a public effect $E_P[i]$ to be grounded with a public effect $E_P[j]$ if:

1. $E_P[i]$ can be achieved when $E_P[j]$ is grounded.
2. Achieving $E_P[i]$ with $E_P[j]$ do not prevent achieving any of the public effects in the public plan.
3. Public effects between j and i do not negate the effects of $E_P[i]$

The first condition checks if $E_P[i]$ can be achieved after the first j-1 public effects were achieved. The second condition verifies that by grounding $E_P[i]$ earlier than planned, i.e., before achieving the public effects $E_P[j], E_P[j+1], ..., E_P[i-1]$, do not prevent these intermediate public effects from being achievable. The third condition checks that the effects of $E_P[i]$ would not be deleted due to grounding $E_P[i]$ earlier.

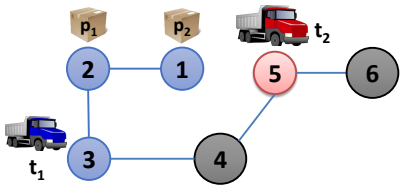
To fully implement these conditions, we would need to know what effects enable or prevent achieving other effects. Due to privacy constraints, we do not know this. As an approximation, we consider for every action A in the public plan. the impact of performing it on the set of public effects that can be achieved. Let $Pos[i]$ be the list of public effects that were not achievable before performing the i^{th} action in the public plan, and became so after performing it, and let $Neg[i]$ be the list of public effects that were achievable before performing the i^{th} action but not so after performing it. We compute $Pos[i]$ and $Neg[i]$ by storing for every action in the public plan the list of alternative public actions that could have been performed instead of it. Identifying this set of alternative actions can be done easily during the high-level planning phase, as these are simple alternative branches in the search tree. Let $Alt[i]$ be this set of public effects of the alternative actions of the i^{th} action in the public plan. Us-

Public plan	Grounding	Cost
Unload p_1 at 4	Move 3 $\rightarrow$ 2	1
	Load p_1	1
	Move 2$\rightarrow$ 4	2
	Unload p_1	1
Unload p_2 at 4	Move 4 $\rightarrow$ 1	3
	Load p_2	1
	Move 1 $\rightarrow$ 4	3
	Unload p_2	1
Load p_2 at 4	Move 5 $\rightarrow$ 4	1
	Load p_2	1
Unload p_2 at 6	Move 4 $\rightarrow$ 6	2
	Unload p_2	1
Total:		**18**

(a) Regular grounding

Public plan	Grounding	Cost
Unload p_1 at 4 and Unload p_2 at 4	Move 3 $\rightarrow$ 1	2
	Load p_2	1
	Move 1 $\rightarrow$ 2	1
	Load p_1	1
	Move 2 $\rightarrow$ 4	2
	Unload p_1	1
	Unload p_2	1
Load p_2 at 4	Move 5 $\rightarrow$ 4	1
	Load p_2	1
Unload p_2 at 6	Move 4 $\rightarrow$ 6	2
	Unload p_2	1
Total:		**14**

(b) Improved grounding



(c) Illustration of the scenario

Figure 2: A scenario demonstrating the problem of sequential grounding.

ing $Alt[i]$, we can compute $Pos[i]$ and $Neg[i]$ as follows: $Pos[i] = Alt[i{+}1] \setminus Alt[i]$ and $Neg[i] = Alt[i] \setminus Alt[i{+}1]$.

Finally, we can describe how we chose which public effects could be grounded together. We allowed $E_P[i]$ to be grounded with $E_P[j]$, for $j < i$, if the following conditions hold:

1. $E_P[i] \in Alt[j]$
   (achieving $E_P[i]$ is possible after $E_P[j\text{-}1]$ is achieved)
2. $\forall k$ s.t. $j \leq k < i$: $E_P[k] \notin Neg[i]$
   (achieving $E_P[i]$ will not prevent achieving $E_P[k]$)
3. $E_P[i] \notin Neg[j]$
   (achieving $E_P[j]$ will not prevent achieving $E_P[i]$)
4. $\forall k$ s.t. $j \leq k < i$: $E_P[i]$ is not mutex with E_P[k]
   (achieving $E_P[k]$ do not destroy $E_P[i]$)

Our improved grounding considers each of the public effects in increasing order (starting from $E_P[0]$), trying to group every public effect $E_P[i]$ as early as possible in the sequence of public effects, where the above conditions hold. Whenever a public effect $E_P[i]$ was successfully grouped with $E_P[j]$, we update $E_P[j]$ to contain both public effects; and add $Pos[i]$ and remove $Neg[i]$ from $Alt[j]$.

The benefits of our improved grounding is two-fold. First, since several action effects can be grounded together, the grouding phase involves solving fewer planning problems to solve, potentially saving runtime. Also, the quality of the resulting plan is better, since agents can optimize their plan to achieve a set of facts together, instead of considering these facts one at a time. Our improved grounding, however, is not always helpful. First, the public effects that were grouped together may not be groundable together. If this occurs, we revert back to the sequential grounding of the original GPPP. Second, the grouping mechanism also consumes runtime. Third, the planning problems solved in the improved grounding are potentially harder, as they require to achieve more than in the grounding of the original GPPP. The experimental results provided next validate the benefit of our improved grounding in most domains.

Experimental Results

Next, we demonstrate the gains of using the 3PDB heuristic and the improved grounding for GPPP. Since 3PDB is an inadmissible heuristic, it is not suitable for optimal planners. Thus, we evaluated 3PDB when used in GPPP, which was shown to outperform MAFS as a suboptimal solver (Maliah, Shani, and Stern 2014). We compared GPPP using our novel 3PDB heuristic against the GPPP with the Privacy Preserving Landmark heuristic (PPLM), which was the best-performing configuration of GPPP in prior work.

Domains

Multi-agent planning is perhaps most interesting when agents have complementing abilities, and when tasks that can be performed by several agents require different costs from different agents. Such domains make the high-level decision making process of assigning tasks to agents more challenging. Most existing benchmarks do not exhibit this behavior. For example, in the original logistics problem planes can fly between every pair of airports, and trucks can drive between every pair of cities. This results in minor differences between the various paths of a package to its destination. In the satellite domain, on the other hand, all agents are identical, and can perform all actions with the same cost. Again, in such domains there is little reason to prefer one agent to another for performing a certain task.

We hence created new benchmark domains, MABlocksWorld and MALogistics, that force the decision process to be smarter about task allocation. In MABlocksWorld there is a set of stacks over which blocks can be piled. Each agent possess an arm that can reach only a part of the stacks. Several stacks are shared and provide the public collaboration locations. In MALogistics there is a graph of cities connected by roads, divided into areas. Each agent is responsible for one area, and neighboring areas share a city. In both cases, an object (package or block) can be moved through various paths to reach its goal location. As various agents control areas of different size and structure, it may be better to prefer one agent over the other to pass the object to its goal.

Table 2 shows the number of instances solved under 5

Domain	PPLM	3PDB
MABlocksWorld	9	**14**
MALogistics	12	**14**
Satellite	12	12
Elevators	16	16
Logistics	12	12
Zenotravel	11	11

Table 2: Instances solved under 5 minutes

minutes. In 2 out of our 6 domains, GPPP with the 3PDB heuristic was able to solve more instances than with the PPLM heursitic. The improved grounding in GPPP did not lead to more instances being solved, but as shown below, led to improved runtime.

GPPP	Original		Improved	
Heuristic	PPLM	3PDB	PPLM	3PDB
MABlocksWorld	5.18	**0.38**	5.18	**0.38**
MALogistics	1.91	**1.73**	1.27	**1.05**
Satellite	**1.53**	3.51	**1.01**	2.56
Elevators	0.68	**0.64**	0.47	**0.41**
Logistics	1.05	**1.00**	0.57	**0.47**
Zenotravel	**0.76**	0.78	**0.29**	**0.29**

Table 3: Avg. runtime (in sec.) for instances solved by all algorithms

Table 3 shows the average runtime over the instances solved by all algorithms. The results under the "Original" column are for GPPP, and the results under the "Improved" column are for the GPPP modification we proposed above, grounding several public effects together. In each domain, we marked in bold the best performing heuristic (PPLM/3PDB) in each type of GPPP (Original/Improved), and marked in an underline the best performing configuration (PPLM/3PDB and Original/Improved).

First, we can see that the improved GPPP almost always results in a runtime reduction over the original GPPP. In the Zenotravel, for example, the improved GPPP was more than two times faster than the original GPPP. Second, we observe that in most cases, the 3PDB, presented in this paper, is superior to the PPLM. In the MABlocksWorld domain, the advantage was most substantial, where 3PDB was on average more than an order of magnitude faster than the previously proposed PPLM heuristic. In the other domains, however, the improvement was more modest. Moreover, in a single domains – Satellite – the PPLM heuristic was better than our 3PDB heuristic. Thus, we cannot conclude that any of these heuristics strictly dominates the other. This is also observed in single-agent planning – different heuristics perform best on different domains. This has lead to several techniques for combining different heuristics, such as dovetailing (Valenzano et al. 2010) and alternating (Röger and Helmert 2010). We leave the exploration of combining several heuristics in privacy preserving MA-STRIPS to future work.

Table 4 presents the average plan cost over all instances solved by all algorithms. Bold and underline were used to denote best heuristic and best configuration, as explained for Table 3. We see that in most cases both the 3PDB heuristic and the improved GPPP resulted in lower cost compared to PPLM and the original GPPP, respectively. Moreover, we note that the advantage of 3PDB over PPLM, in terms of solution cost, is more substantial in the MABlocksWorld and MALogistics domains. This is expected, since in these domains, agents have several ways to achieve the goal, involving different agents which have different capabilities and resulting in plans with different costs. As discussed above in the context of Figure 1, such cases are example cases in which 3PDB is superior over PPLM.

GPPP	Original		Improved	
Domain	PPLM	3PDB	PPLM	3PDB
MABlocksWorld	17.56	**13.11**	17.56	**13.11**
MALogistics	222.50	**183.33**	203.50	**184.50**
Satellite	**35.67**	41.58	**32.17**	**32.17**
Elevators	**29.56**	**29.56**	27.94	**27.88**
Logistics	59.08	**55.08**	**43.92**	**43.92**
Zenotravel	24.18	**22.45**	**18.55**	**18.55**

Table 4: Avg. plan cost for instances solved by all algorithms

Thus, in conclusion, we observe that on most domains, 3PDB is able to solve more instances faster than PPLM and produces plans of similar or lower cost. In addition, the improved GPPP contributes as well to reduce runtime and lower solution cost.

Conclusion and Future Work

We proposed a novel privacy preserving heuristic called Privacy Preserving Pattern Database (3PDB). 3PDB is inspired by two classical single-agent heuristics: pattern databases and delete relaxation. In a preprocessing stage, 3PDB populates a pattern database with the costs of plans between pairs of public facts. When planning, this PDB is used to estimate the cost of achieving the goal. We showed how 3PDB can be used even if the goal is private. Then, we proposed an improvement to GPPP, a current state-of-the-art suboptimal privacy preserving MA-STRIPS planner. In this improved GPPP, agents can plan to achieve several public facts together instead of planning to achieve one action at a time.We evaluated the 3PDB and the improved GPPP experimentally, revealing that both contributions lead to better or similar runtime and lower solution cost, in most domains.

Developing privacy preserving heuristics has only recently begun to be explored by the planning community, and there are many questions to address. One such question raised earlier in this paper is how to combine several privacy preserving heuristics. Another, perhaps broader, research question is how to address a more flexible privacy constraint, where compromising some privacy is allowed but incurs a cost. This would raise interesting challenges related to balancing loss of privacy and improved heuristic accuracy.

References

Bäckström, C. 1998. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research (JAIR)* 9(1):99–137.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1):5–33.

Bonisoli, A.; Gerevini, A. E.; Saetti, A.; and Serina, I. 2014. A privacy-preserving model for the multi-agent propositional planning problem. In *ICAPS workshop on Distributed and Multi-Agent Planning (DMAP)*.

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

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1):165–204.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S. 2001. Planning with pattern databases. In *the European Conference on Planning (ECP)*, 13–34.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research (JAIR)* 22:279–318.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research (JAIR)* 22:215–278.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI magazine* 22(3):57.

Luis, N., and Borrajo, D. 2014. Plan merging by reuse for multi-agent planning. In *ICAPS workshop on Distributed and Multi-Agent Planning (DMAP)*.

Maliah, S.; Shani, G.; and Stern, R. 2014. Privacy preserving landmark detection. In *the European Conference on Artificial Intelligence (ECAI)*, 597–602.

Nissim, R., and Brafman, R. I. 2013. Cost-optimal planning by self-interested agents. In *AAAI*.

Nissim, R., and Brafman, R. I. 2014. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research (JAIR)* 51:293–332.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39(1):127–177.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, volume 8, 975–982.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *ICAPS*, 246–249.

Torreño, A.; Onaindia, E.; and Sapena, Ó. 2014. Fmap: Distributed cooperative multi-agent planning. *Applied Intelligence* 1–21.

Valenzano, R. A.; Sturtevant, N. R.; Schaeffer, J.; Buro, K.; and Kishimoto, A. 2010. Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In *ICAPS*, 177–184.

On Internally Dependent Public Actions in Multiagent Planning

Jan Tožička, Jan Jakubův, Antonín Komenda
Agent Technology Center, Czech Technical University, Prague, Czech Republic,
{jan.tozicka, jan.jakubuv, antonin.komenda}@agents.fel.cvut.cz

Abstract

Agents planning under STRIPS-related model using separation of facts and actions to private and public can model behavior of other agents as public external projections of their actions. In the most simplistic case, 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 internally 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 computed dependencies in multiagent planning. Additionally, we analyze the standard benchmarks used for mutliagent planning and present overview of various sub-types of internal dependencies of public actions in particular planning domains.

Introduction

In multiagent planning modeled as MA-STRIPS (Brafman and Domshlak 2008), 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 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, Jakubův, and Komenda 2014).

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.

In this article, we propose to keep the pair of actions and add new public predicate that says that some action requires another action to precede it (because it better fits proposed coordination algorithm). In the simplest 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 the transport agent.

Multiagent Planning

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

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, $I \subseteq P$ 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* an agent 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 $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)\rangle$.

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.

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 to several STRIPS problems which can be solved separately by a classical planner.

The *projection* $F \triangleright \alpha$ of 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 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.

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 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 sections describe conditions which allow us to compute a solution of the 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. 2014) 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*.*

Algorithm 1: Distributed MA planning algorithm.

```
1  Function MaPlanDistributed(Π ▷ α) is
2      Φ_α ← ∅;
3      loop
4          generate new π_α ∈ sols(Π ▷ α);
5          Φ_α ← Φ_α ∪ {π_α ▷ ⋆};
6          exchange public plans Φ_β with other agents;
7          Φ ← ⋂_{β ∈ agents(Π)} Φ_β;
8          if Φ ≠ ∅ then
9              return Φ;
10         end
11     end
12 end
```

The theorem above suggests a distributed multiagent planning algorithm described in Algorithm 1. Every agent executes the loop from Algorithm 1, possibly on a 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. 2014) for more details on the algorithm.

Internal Dependencies of Public 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.

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 directed graph whose nodes are actions and facts. Dependency graphs contain three kind of edges,

namely precondition, add, and delete edges. Given the set of nodes, the dependency graph contains edge $(f \to a)$ whenever $f \in \mathsf{pre}(a)$. Furthermore, there is the edge $(a \to_+ f)$ iff $f \in \mathsf{add}(a)$, and there is the edge $(a \to_- f)$ iff $f \in \mathsf{del}(a)$. A dependency graph Δ is thus uniquely determined by the set of nodes. We write $\mathsf{actions}(\Delta)$ and $\mathsf{facts}(\Delta)$ to denote in turn the set of action and fact nodes of Δ. Furthermore, we suppose some fact nodes can be mark as initial and we use $\mathsf{init}(\Delta)$ to denote the set of initial nodes of Δ.

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 corresponding actions.

Definition 1. *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 2. *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.

Dependency Graph Collections

A dependency graph represents information about public actions of one agent. Every agent need 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 3. *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.

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 element-wise. Furthermore, *external actions of* α *accordingly to* $\mathcal{D}$, denoted $\mathsf{exts}_{\mathcal{D}}(\alpha)$, contain public projections (accordingly to $\mathcal{D}$) of actions of other agents. In other words, $\mathsf{exts}_{\mathcal{D}}(\alpha)$ carries all the information published by other agents for agent α. It is computed as follows.

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

This equation captures distributed computation of $\mathsf{exts}_{\mathcal{D}}(\alpha)$ 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 $\mathsf{facts}_{\mathcal{D}}(\alpha)$ and $\mathsf{init}_{\mathcal{D}}(\alpha)$ 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.

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

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

Definition 4. *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) $P_0 = \mathsf{facts}(\Pi \triangleright \alpha) \cup \mathsf{facts}_{\mathcal{D}}(\alpha)$,
(2) $A_0 = \alpha \cup \mathsf{exts}_{\mathcal{D}}(\alpha)$,
(3) $I_0 = \mathsf{init}(\Pi \triangleright \alpha) \cup \mathsf{init}_{\mathcal{D}}(\alpha)$, *and*
(4) $G_0 = \mathsf{goal}(\Pi)$.

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

Example 1. *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* $\Pi \rhd_{\mathsf{MD}(\Pi)} \alpha = \Pi \rhd \alpha$ *for every agent* α*. With the full dependency collection* $\mathsf{FD}(\Pi)$ *we obtain equal projections, that is,* $\Pi \rhd_{\mathsf{FD}(\Pi)} \alpha = \Pi \rhd_{\mathsf{FD}(\Pi)} \beta$ *for all agents* α *and* β*. Moreover, local solutions equal* MA-STRIPS *solutions, that is,* $\mathsf{sols}(\Pi \rhd_{\mathsf{FD}(\Pi)} \alpha) = \mathsf{sols}(\Pi)$ *for every* α*.*

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.

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

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 the for any agent α, it holds that the local problems $\Pi \rhd_{\mathcal{D}_0} \alpha$ and $\Pi \rhd_{\mathcal{D}_1} \alpha$ are publicly equivalent. Formally as follows.

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

Example 2. *Given an* MA-STRIPS *problem* Π*, with the full dependency collection* $\mathsf{FD}(\Pi)$ *we can see that* $\Pi \simeq \Pi \rhd_{\mathsf{FD}(\Pi)} \alpha$ *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 5. *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 $\Pi \rhd \alpha$ 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* $(\Pi \rhd \alpha) \simeq \Pi$.

Proof. $(\Pi \rhd \alpha) \simeq (\Pi \rhd_{\mathsf{MD}(\Pi)} \alpha) \simeq (\Pi \rhd_{\mathsf{FD}(\Pi)} \alpha) \simeq \Pi$ □

Simple Public Action Dependencies

Let us consider dependency collections without internal actions, that is, collections $\mathcal{D}$ where $\mathsf{actions}(\mathcal{D})$ contains only public actions. Hence no agent publishes actions additional to $\mathsf{exts}(\alpha)$ which is desirable out of privacy concerns. Furthermore, the plan search space of $\Pi \rhd_{\mathcal{D}} \alpha$ is not increased when compared to $\Pi \rhd \alpha$. Even more, every additionally published fact in $\mathcal{D}$ providing valid dependency prunes the search space. Action dependencies of internal-actions-free dependency collections can be expressed by a 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.

Simply Dependent Problems

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

Definition 6. *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 $\Pi \rhd_{\mathcal{D}} \alpha$ (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* $(\Pi \rhd_{\mathcal{D}} \alpha) \simeq \Pi$ *holds for any agent* $\alpha \in \mathsf{agents}(\Pi)$.

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

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.

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.

Furthermore we restrict our attention to MA-STRIPS problems where all the internal actions are *precondition consuming*, that is, where $\mathsf{pre}(a) = \mathsf{del}(a)$ holds for every internal action. Additionally, for public actions we require $\mathsf{del}(a) \subseteq \mathsf{pre}(a)$. With this requirement, we can simplify

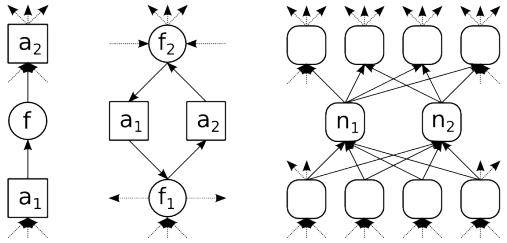


Figure 1: Application conditions for reduction operators. *(R1)* simple dependency, *(R2)* small action cycle, and *(R3)* two equivalent nodes.

the syntax of dependency graphs because precondition and delete edges are always paired. That is, Δ contains the edge $(f \to a)$ iff Δ contains the edge $(a \to_{\text{-}} f)$. Hence delete edges are implicitly determined by precondition edges and thus we can remove them from the graph. This allows us to remove the index + from add edges. Hence dependency graphs in this simplified syntax contain only edges of the form $(f \to a)$ and $(a \to f)$ where all the edges outgoing an action node are implicitly add edges.

Finally, to abstract from the set of initial facts of a dependency graph Δ, we introduce to the graph the 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* although it has no public precondition or effect. Both formal definitions 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. Formal definition is given below. The operations are depicted in Figure 1.

(R1) Remove Simple Dependencies. If some fact is the only effect of some action and there is only one action that uses this effect, we can remove this fact and merge both actions.

(R2) Remove Small Action Cycles. 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.

(R3) 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.

(R4) 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 (R2). 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.

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

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 (elementwise). 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.

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

Definition 7. *The* reduction relation $\Delta_0 \to \Delta_1$ *on dependency graphs is defined by the following four rules.*

(R1) *Rule (R1) is applicable to Δ_0 when*
(1) Δ_0 contains edges $(a_1 \to f \to a_2)$,
(2) there are no other edges from/to f, and
(3) there are no other edges from a_1, and
(4) a_1 and a_2 are not both public actions.
Then $\Delta_0 \to \Delta_1$ where Δ_1 is given by the following.

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

When a_1 or a_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.

(R2) *Rule (R2) is applicable to Δ_0 when*
(1) Δ_0 contains a cycle $(f_1 \to a_1 \to f_2 \to a_2 \to f_1)$,
(2) a_1 and a_2 are both internal, and
(3) there are no other edges from/to a_1 or a_2.
Then $\Delta_0 \to \Delta_1$ where Δ_1 is given by the following.

$$\mathsf{actions}(\Delta_1) = [\mathsf{actions}(\Delta_0) \setminus \{a_1, a_2\}]_{f_2 \to f_1}$$
$$\mathsf{facts}(\Delta_1) = [\mathsf{facts}(\Delta_0)]_{f_2 \to f_1}$$

The initial action is preserved as it is public.

(R3) *Rule (R3) is applicable to Δ_0 when Δ_0 contains two nodes n_1 and n_2 (either action or fact nodes) such that*
(1) $(n_1 \to n) \in \Delta_0 \Leftrightarrow (n_2 \to n) \in \Delta_0$ for any node n,
(2) $(n \to n_1) \in \Delta_0 \Leftrightarrow (n \to n_2) \in \Delta_0$ for any node n,
(3) and, n_1 and n_2 are not both public actions.
Then $\Delta_0 \to \Delta_1$ where, in the case n_1 and n_2 are actions, Δ_1 is given by the following.

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

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, $\Delta_1 = [\Delta_0]_{n_2 \to n_1}$.

(R4) *Rule (R4) is applicable to Δ_0 when Δ_0 contains fact f with no outgoing edges. Then $\Delta_0 \to \Delta_1$ where Δ_1 is defined as $\Delta_1 = [\Delta_0]_{-f}$. The initial action is preserved.*

The following defines *reduction equivalence relation* $\Delta_0 \sim \Delta_1$ as a reflexive, symmetric, and transitive closure of $\to$. 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 8. *Dependency graphs* reduction equivalence relation*, denoted $\Delta_0 \sim \Delta_1$, is the least reflexive, symmetric, and transitive closure generated by the relation $\to$.*

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 $\mathsf{pre}(a) = \mathsf{del}(a)$ 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 $\mathcal{D}_0 \simeq \mathcal{D}_1$.*

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

Recognizing Simply Dependent Problems

Let us have an MA-STRIPS problem Π where $\mathsf{pre}(a) = \mathsf{del}(a)$ 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 public actions. 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 does 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.

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 $\mathsf{FD}(\alpha)$.

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

```
1  Function ComputeSharedDG(α) is
2      Δ0 ← FD(α);
3      loop
4          if ∃Δ1 : Δ0 → Δ1 then
5              Δ0 ← Δ1;
6          else
7              break;
8          end
9      end
10     if Δ0 contains only public actions then
11         return Δ0;
12     else
13         return MD(α);
14     end
15 end
```

Algorithm 3: Distributed planning with dependency graphs.

```
1 Function DgPlanDistributed(α) is
2     Δ ← ComputeSharedDG(α);
3     send Δ to other agents;
4     construct D from other agent's graphs;
5     compute local problem Π ▷_D α accordingly to D;
6     return MaPlanDistributed(Π ▷_D α);
7 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 its 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.

Domain Analysis

In this section we present analysis of internal dependencies of public action in 10 benchmark domains containing 144 problems in total. We use IPC problem converted to multiagent setting as presented in (Torreño, Onaindia, and Sapena 2014) and we use MA-STRIPS model to setup public/internal classification of facts and actions.

We have evaluated internal dependencies of public actions within benchmark problems by constructing full dependency graph for every agent in every benchmark prob-

Domain	#	Facts	Public facts	Merge facts	Fact disclosure	Actions	Public actions	Success
Blocksworld	34	641	594	42	89 %	1100	1100	100 %
Depots	20	1199	1047	97	61 %	1594	1541	100 %
Driverlog	20	1536	1421	16	25 %	7682	7439	100 %
Elevators	30	256	112	13	9 %	617	404	38 %
Logitics	20	454	238	106	47 %	679	477	100 %
Openstacks	30	197	177	19	100 %	949	949	100 %
Rovers	20	597	325	0	0 %	837	338	0 %
Satellite	20	587	387	0	0 %	3063	625	0 %
Woodworing	30	380	301	0	0 %	1144	1137	75 %
Zenotravel	20	802	690	0	0 %	7803	1566	0 %

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

lem. 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 **(#)** represents the number of problems in the domain. 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*, and *Openstacks*, 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 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* and *Woodworing* domains, some of the agents succeeded in reducing their full dependency graphs and thus the analysis can partially help to solve them. An experimental evaluation of the impact of dependency analysis is left for future research.

Conclusions

We have semantically defined internally independent and simply dependent MA-STRIPS problems. To identify internally independent and simply dependent problems, we 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. Furthermore, we have presented an multiagent planning algorithm which can be used with only partially dependent problems. Finally, we have analyzed the commonly used domains for comparison of MA-STRIPS-compatible multiagent planners and showed high numbers of simply dependent problems within them.

Acknowledgements

This research was supported by the Czech Science Foundation (grant no. 13-22125S) and by the Ministry of Education of the Czech Republic within the SGS project no. SGS13/211/OHK3/3T/13.

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.

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

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 the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*, 1265–1266.

Torreño, A.; Onaindia, E.; and Sapena, . 2014. Fmap: Distributed cooperative multi-agent planning. *Applied Intelligence* 41(2):606–626.

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

Tožička, J.; Jakubův, J.; and Komenda, A. 2014. Generating multi-agent plans by distributed intersection of Finite State Machines. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, 1111–1112.

Improved Planning for Infinite-Horizon Interactive POMDPs Using Probabilistic Inference

Xia Qu and Prashant Doshi
THINC Lab, Department of Computer Science
University of Georgia, Athens, GA 30602
pdoshi@cs.uga.edu

Abstract

This paper provides the first formalization of self-interested multiagent planning using expectation-maximization (EM). Our formalization in the context of *infinite-horizon* and finitely-nested interactive POMDPs (I-POMDP) is distinct from EM formulations for POMDPs and other multiagent planning frameworks. Specific to I-POMDPs, we exploit the graphical model structure and present a new approach based on block-coordinate descent for further speed up. Forward filtering-backward sampling – a combination of exact filtering with sampling – is utilized to exploit problem structure.

Introduction

Generalization of bounded policy iteration (BPI) to finitely-nested interactive partially observable Markov decision processes (I-POMDP) (Sonu and Doshi 2014) is currently the leading method for *infinite-horizon* self-interested multiagent planning, and obtaining finite-state controllers as solutions. However, interactive BPI is prone to converge to local optima, which severely limits the quality of its solutions despite the limited ability to escape from these local optima.

Attias [2003] posed planning using MDP as a likelihood maximization problem where the "data" is the initial state and the final goal state or the maximum total reward. Toussaint et al. [2006] extended this to infer finite-state automata for infinite-horizon POMDPs. Experiments reveal good quality controllers of small sizes although run time is a concern. Given the limitations of BPI and the compelling potential of this approach in bringing advances in inferencing to bear on planning, we generalize it to infinite-horizon and finitely-nested I-POMDPs. Our generalization here allows its use toward planning for an individual agent in non-cooperation where we may not assume common knowledge of an initial belief or common rewards, due to which others' beliefs, capabilities and preferences are modeled.

Analogously to POMDPs, we formulate a mixture of finite-horizon DBNs. However, these differ by including models of other agents in a special **model node**. *Our approach, labeled as I-EM, improves on the straightforward extension of Toussaint et al.'s EM to I-POMDPs by utilizing various types of structure.* Instead of ascribing as many level 0 finite-state controllers as candidate models and improving each using its own EM, we use the underlying *graphical structure* of the model node and its update to formulate a single EM that directly provides the expected distribution of others actions across *all* models. This rests on a new insight, which considerably simplifies and speeds EM at level 1.

We present a general approach based on block-coordinate descent (Fessler and Hero 1994; Tseng and Mangasarian 2001) for speeding up the *non-asymptotic rate of convergence* of the iterative EM. The problem is decomposed into optimization subproblems in which the objective function is optimized with respect to a small subset (block) of variables, while holding other variables fixed. We discuss the unique challenges and present the *first* effective application of this iterative scheme to multiagent planning.

Finally, sampling offers a way to exploit the embedded problem structure such as information in distributions. The exact forward-backward E-step is replaced with *forward filtering-backward sampling (FFBS)* that generates trajectories weighted with rewards, which are used to update the parameters of the controller. While sampling has been integrated in EM previously (Wu, Zilberstein, and Jennings 2013), FFBS specifically mitigates error accumulation over long horizons.

Background: Interactive POMDP

We briefly review finitely-nested I-POMDPs and outline previous EM-based planning in the context of POMDPs.

Interactive POMDP

A finitely-nested I-POMDP (Gmytrasiewicz and Doshi 2005) for an agent i with strategy level, l, interacting with agent j is: I-POMDP$_{i,l} = \langle IS_{i,l}, A, T_i, \Omega_i, O_i, R_i, OC_i \rangle$

- $IS_{i,l}$ denotes the set of *interactive states* defined as, $IS_{i,l} = S \times M_{j,l-1}$, where $M_{j,l-1} = \{\Theta_{j,l-1} \cup SM_j\}$, for $l \geq 1$, and $IS_{i,0} = S$, where S is the set of physical states. $\Theta_{j,l-1}$ is the set of computable, intentional models ascribed to agent j: $\theta_{j,l-1} = \langle b_{j,l-1}, \hat{\theta}_j \rangle$. Here $b_{j,l-1}$ is agent j's level $l-1$ belief, $b_{j,l-1} \in \triangle(IS_{j,l-1})$ where $\Delta(\cdot)$ is the space of distributions, and $\hat{\theta}_j = \langle A, T_j, \Omega_j, O_j, R_j, OC_j \rangle$, is j's *frame*. At *level l=0*, $b_{j,0} \in \triangle(S)$ and a intentional model reduces to a POMDP.

SM_j is the set of subintentional models of j, an example is a finite state automaton.

- $A = A_i \times A_j$ is the set of joint actions of all agents.
- Other parameters – transition function, T_i, observations, Ω_i, observation function, O_i, and preference function, R_i – have their usual semantics analogously to POMDPs.
- Optimality criterion, OC_i, here is the discounted infinite horizon sum.

An agent's belief over its interactive states is a sufficient statistic fully summarizing the agent's observation history. Given the associated belief update, solution to an I-POMDP is a *policy*. Using the Bellman equation, each belief state in an I-POMDP has a value which is the maximum payoff the agent can expect starting from that belief and over the future.

Gmytrasiewicz and Doshi (2005) provide additional details on I-POMDPs and how they compare with other frameworks.

Expectation-Maximization for POMDPs

To infer a FSC, Toussaint et al. (2006; 2006) formulate a series of DBNs unrolled over increasing time steps, $T = 0 \ldots$, each of which emits a single reward, $r^T = 1$. To infer the controller, its nodes and parameters are included in the DBN. The likelihood maximization problem becomes that of finding π_i, which maximizes the likelihood: $\arg\max_{\pi_i} \sum_{T=0}^{\infty} Pr(r^T = 1|T; \pi_i)\ Pr(T)$, where $Pr(T)$=$\gamma^T(1-\gamma)$.

As the realized trajectory of states, observations, nodes and actions are hidden at planning time, the likelihood maximization is solved in the schema of expectation-maximization (EM) (Dempster, Laird, and Rubin 1977). The E-step takes an expectation of the hidden sequences in the DBN:

$$Q(\pi_i'|\pi_i) = \sum_{T=0}^{\infty} \sum_{z_i^{0:T}} Pr(r^T = 1, z_i^{0:T}, T; \pi_i) \log Pr(r^T = 1, z_i^{0:T}, T; \pi_i') \quad (1)$$

where sequence, $z_i^{0:T} = \{s^t, o_i^t, n_i^t, a_i^t\}_0^T$. The full joint, $Pr(r_T = 1, z_i^{0:T}, T; \pi_i)$, may be computed using a simultaneous forward and backward pass. The M-step derives the distributions, $\mathcal{V}_i$, $\mathcal{T}_i$, and $\mathcal{L}_i$, of the controller, π_i', which maximize the expectation, $Q(\pi_i'|\pi_i)$.

Instead of revising the parameters of the previous iteration, a *greedy* variant of the M-step selects parameter values for the controller, $\mathcal{V}_i(\cdot)$, $\mathcal{T}_i(\cdot|n_i, a_i, o_i)$, and $\mathcal{L}_i(\cdot|n_i)$, for each n_i, a_i, and o_i, that maximize the function, which may be seen as the expectation divided by the previous parameters. This is equivalent to greedily performing the exact M-step an infinite number of times without updating the expectation in between. To avoid quick local maxima, we may soften the maximization by assigning small probabilities to the other parameter values as well.

Planning in I-POMDP as Inference

We generalize planning as inference to the framework of I-POMDP$_{i,l}$. For presentation simplicity, we focus on a two-agent setting; the other agent could have differing frames, which need not be the same as that of agent i. Our approach generalizes to multiple other agents as mentioned below.

Problem Formulation, Mixture Models

Attias (2003) casts planning under uncertainty in the context of POMDPs as an action sequence likelihood maximization problem. Solution of I-POMDP$_{i,l}$ is a policy, which goes beyond a simple sequence of actions. We may represent the policy of agent i for the infinite-horizon case as a stochastic *finite state controller (FSC)*, defined as: $\pi_i = \langle \mathcal{N}_i, \mathcal{T}_i, \mathcal{L}_i, \mathcal{V}_i \rangle$ where $\mathcal{N}_i$ is the set of nodes in the controller. $\mathcal{T}_i : \mathcal{N}_i \times A_i \times \Omega_i \times \mathcal{N}_i \to [0, 1]$ represents the node transition function; $\mathcal{L}_i : \mathcal{N}_i \times A_i \to [0, 1]$ denotes agent i's action distribution at each node; and an initial distribution over the nodes is denoted by, $\mathcal{V}_i : \mathcal{N}_i \to [0, 1]$. For convenience, we group $\mathcal{V}_i$, $\mathcal{T}_i$ and $\mathcal{L}_i$ in $\hat{f}_i$.

Define a controller at level l for agent i as, $\pi_{i,l} = \langle\ \mathcal{N}_{i,l}, \hat{f}_{i,l}\ \rangle$, where $\mathcal{N}_{i,l}$ is the set of nodes in the controller and $\hat{f}_{i,l}$ groups remaining parameters of the controller as mentioned before. Analogously to POMDPs (Toussaint and Storkey 2006), we formulate planning in multiagent settings formalized by I-POMDPs as a likelihood maximization problem:

$$\pi_{i,l}^* = \arg\max_{\Pi_{i,l}} (1-\gamma) \sum_{T=0}^{\infty} \gamma^T\ Pr(r_i^T = 1|T; \pi_{i,l}) \quad (2)$$

where $\Pi_{i,l}$ are all level-l FSCs of agent i, r_i^T is a binary random variable whose value is 0 or 1 emitted after T time steps with probability proportional to the reward, $R_i(s, a_i, a_j)$.

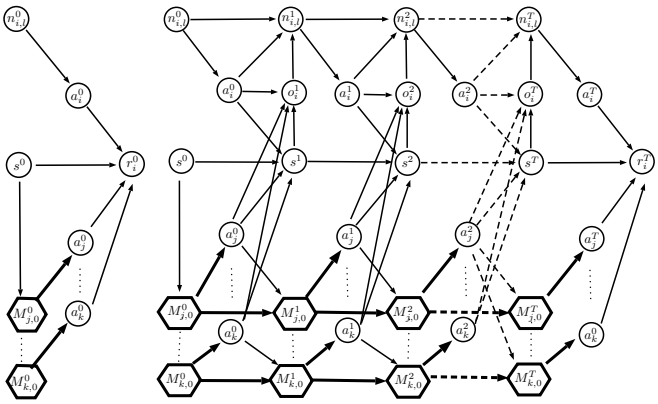


Figure 1: DBN for I-POMDP$_{i,1}$ with i's level-1 policy represented as a standard FSC, with "node state" denoted by $n_{i,l}$. These DBNs differ from those for POMDPs by containing special *model nodes* (hexagons) whose values are the candidate models of other agents.

The planning problem is modeled as a mixture of DBNs of increasing time from T=0 onwards (Fig. 1). The transition and observation functions of I-POMDP$_{i,l}$ parameterize the chance nodes s and o_i, respectively, along with $Pr(r_i^T|a_i^T, a_j^T, s^T) \propto \frac{R_i(s^T, a_i^T, a_j^T) - R_{max}}{R_{max} - R_{min}}$. Here, R_{max} and R_{min} are the maximum and minimum reward values in R_i.

The networks include nodes, $n_{i,l}$, of agent i's level-l FSC. Therefore, functions in $\hat{f}_{i,l}$ parameterize the network

as well, which are to be inferred. *Additionally, the network includes the hexagonal* model nodes *– one for each other agent – that contain the candidate level 0 models of the agent.* Each model node provides the expected distribution over another agent's actions. Without loss of generality, no edges exist between model nodes in the same time step. Correlations between agents could be included as state variables in the models.

Agent j's model nodes and the edges (in bold) between them, and between the model and chance action nodes *represent* a DBN of length T as shown in Fig. 2. Values of the chance node, $m^0_{j,0}$, are the candidate models of agent j. Agent i's initial belief over the state and models of j becomes the parameters of s^0 and $m^0_{j,0}$. The likelihood maximization at level 0 seeks to obtain the distribution, $Pr(a_j|m^0_{j,0})$, for each candidate model in node, $m^0_{j,0}$, using EM on the DBN.

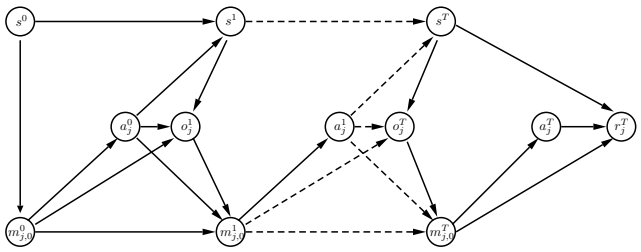


Figure 2: Hexagonal model nodes and edges in bold for one other agent j in Fig. 1 decompose into this DBN. Values of the node, $m^t_{j,0}$, are the candidate models. CPT of chance node, a^t_j, denoted by $\phi_{j,0}(m^t_{j,0}, a^t_j)$ is inferred using likelihood maximization.

Proposition 1 (Correctness)**.** *The likelihood maximization problem as defined in Eq. 2 with the mixture models as given in Fig. 1 is equivalent to the problem of solving the original I-POMDP$_{i,l}$ of discounted infinite horizon whose solution assumes the form of a finite state controller.*

Proofs of all propositions in this paper are in an online appendix (App 2015). Given the unique mixture models above, the challenge is to generalize the EM-based iterative maximization for POMDPs to the framework of I-POMDPs.

Single EM for Level 0 Models

The straightforward approach is to infer a likely FSC for each level 0 model. However, this approach does not scale to many models. Proposition 2 below shows that the *dynamic* $Pr(a^t_j|s^t)$ is sufficient information about predicted behavior of the other agent from its candidate models at time t, in order to obtain the most likely policy of agent i. This is markedly different from using behavioral equivalence (Zeng and Doshi 2012) that clusters models with identical solutions. The latter continues to require the full solution for *each* model.

Proposition 2 (Sufficiency)**.** *Distributions, $Pr(a^t_j|s^t)$ across actions, $a^t_j \in A_j$, for each state s^t is sufficient predictive information about other agent j to obtain the most likely policy of i.*

In the context of Proposition 2, we seek to infer $Pr(a^t_j|m^t_{j,0})$ for each (updated) model of j at all time steps, which is denoted as $\phi_{j,0}$. Other terms in the computation of $Pr(a^t_j|s^t)$ are known parameters of the level 0 DBN.

The likelihood maximization for the level 0 DBN is: $\phi^*_{j,0} = \arg\max_{\phi_{j,0}} (1-\gamma) \sum_{T=0}^{\infty} \sum_{m_{j,0} \in M^T_{j,0}} \gamma^T Pr(r^T_j = 1|T, m_{j,0}; \phi_{j,0})$

As the trajectory consisting of states, models, actions and observations of the other agent is hidden at planning time, we may solve the above likelihood maximization using EM.

E-step Let $z^{0:T}_j = \{s^t, m^t_{j,0}, a^t_j, o^t_j\}^T_0$ where the observation at $t = 0$ is null, be the hidden trajectory. The log likelihood is obtained as an expectation of these hidden trajectories:

$$Q(\phi'_{j,0}|\phi_{j,0}) = \sum_{T=0}^{\infty} \sum_{z^{0:T}_j} Pr(r^T_j = 1, z^{0:T}_j, T; \phi_{j,0}) \times log\ Pr(r^T_j = 1, z^{0:T}_j, T; \phi'_{j,0}) \quad (3)$$

The full joint in Eq. 3 expands as:

$$\begin{aligned} &Pr(r^T_j = 1, z^{0:T}_j, T; \phi_{j,0}) = Pr(r^T_j = 1, z^{0:T}_j|T; \phi_{j,0}) Pr(T) \\ &= (1-\gamma)\gamma^T R_{m_j}(s^T, a^T_j)\ b^0_{i,1}(s)\ b^0_{i,1}(m^0_{j,0}|s^0) \\ &\times \prod_{t=1}^{T} \phi_{j,0}(m^{t-1}_{j,0}, a^{t-1}_j)\ T_{m_j}(s^{t-1}, a^{t-1}_j, s^t) \\ &\times O_{m_j}(s^t, a^{t-1}_j, o^t_j)\ Pr(m^t_{j,0}|a^{t-1}_j, o^t_j, m^{t-1}_{j,0}) \end{aligned} \quad (4)$$

Here, $Pr(m^t_{j,0}|a^{t-1}_j, o^t_j, m^{t-1}_{j,0})$ is the Kronecker-delta function that is 1 when j's belief in $m^{t-1}_{j,0}$ updated using a^{t-1}_j and o^t_j equals the belief in $m^t_{j,0}$; otherwise 0.

The "data" in the level 0 DBN consists of the initial belief over the state and models, $b^0_{i,1}$, and the observed reward at T. Analogously to EM for POMDPs, this motivates forward filtering-backward smoothing on a network with joint state, (s^t,$m^t_{j,0}$), for computing the log likelihood. The transition function for the forward and backward steps is:

$$\begin{aligned} &Pr(s^t, m^t_{j,0}|s^{t-1}, m^{t-1}_{j,0}) = \sum\nolimits_{a^{t-1}_j, o^t_j} \phi_{j,0}(m^{t-1}_{j,0}, a^{t-1}_j) \\ &\times T_{m_j}(s^{t-1}, a^{t-1}_j, s^t)\ Pr(m^t_{j,0}|m^{t-1}_{j,0}, a^{t-1}_j, o^t_j) O_{m_j}(s^t, a^{t-1}_j, o^t_j) \end{aligned} \quad (5)$$

where m_j in the subscripts is j's model at $t-1$. Forward filtering gives the probability of the next state as follows:

$$\alpha^t(s^t, m^t_{j,0}) = \sum_{s^{t-1}, m^{t-1}_{j,0}} Pr(s^t, m^t_{j,0}|s^{t-1}, m^{t-1}_{j,0})\ \alpha^{t-1}(s^{t-1}, m^{t-1}_{j,0})$$

where $\alpha^0(s^0, m^0_{j,0})$ is the initial belief of agent i.

The smoothing by which we obtain the joint probability of the state and model at $t-1$ from the distribution at t is:

$$\beta^h(s^{t-1}, m^{t-1}_{j,0}) = \sum_{s^t, m^t_{j,0}} Pr(s^t, m^t_{j,0}|s^{t-1}, m^{t-1}_{j,0})\ \beta^{h-1}(s^t, m^t_{j,0})$$

where h denotes the number of time steps until T (horizon), and $\beta^0(s^T, m^T_{j,0}) = E_{a^T_j|m^T_{j,0}}[Pr(r^T_j = 1|s^T, m^T_{j,0})]$.

Messages α^t and β^h give the probability of a state at some time slice in the DBN. As we consider a mixture of BNs, we seek the probabilities for all states in the mixture model. Subsequently, we may compute the forward and backward

messages at all states for the entire mixture model in one sweep.

$$\widehat{\alpha}(s, m_{j,0}) = \sum_{t=0}^{\infty} Pr(T=t)\ \alpha^t(s, m_{j,0}) \quad (6)$$

$$\widehat{\beta}(s, m_{j,0}) = \sum_{h=0}^{\infty} Pr(T=h)\ \beta^h(s, m_{j,0}) \quad (7)$$

Model growth As the other agent performs its actions and makes observations, the space of j's models grows exponentially: starting from a finite set of $|M^0_{j,0}|$ models, we obtain $\mathcal{O}(|M^0_{j,0}|(|A_j||\Omega_j|)^t)$ models at time t. This greatly increases the number of trajectories in $Z_j^{0:T}$. We limit the growth in the model space by sampling models at the next time step from the distribution, $\alpha^t(s^t, m^t_{j,0})$, as we perform each step of forward filtering. It limits the growth by exploiting the structure present in $\phi_{j,0}$ and O_j, which guide how the models grow.

M-step We obtain the updated $\phi'_{j,0}$ from the full log likelihood in Eq. 3 by separating the terms as given below:

$$Q(\phi'_{j,0}|\phi_{j,0}) = \langle \text{terms independent of } \phi'_{j,0} \rangle + \sum_{T=0}^{\infty} \sum_{z_j^{0:T}} Pr(r_i^T = 1, z_j^{0:T}, T; \phi'_{j,0})\ \sum_{t=0}^{T} \phi'_{j,0}(a_j^t | m_{j,0}^t)$$

and maximizing it w.r.t. $\phi'_{j,0}$:

$$\phi'_{j,0}(a_j^t, m_{j,0}^t) \propto \phi_{j,0}(a_j^t, m_j^t) \sum_{s^t} R_{m_j}(s^t, a_j^t)\ \widehat{\alpha}(s^t, m_{j,0}^t) + \sum_{s^t, s^{t+1}, m_{j,0}^{t+1}, o_j^{t+1}} \frac{\gamma}{1-\gamma} \widehat{\beta}(s^{t+1}, m_{j,0}^{t+1})\ \widehat{\alpha}(s^t, m_{j,0}^t) \times T_{m_j}(s^t, a_j^t, s^{t+1})\ Pr(m_{j,0}^{t+1} | m_{j,0}^t, a_j^t, o_j^{t+1})\ O_{m_j}(s^{t+1}, a_j^t, o_j^{t+1})$$

EM for Level l I-POMDP

At strategy levels, $l \geq 1$, Eq. 2 defines the likelihood maximization problem, which is iteratively solved using EM. We show the E- and M-steps next beginning with $l = 1$.

E-step In a multiagent setting, the hidden variables additionally include what the other agent may observe and how it acts over time. However, a key insight is that Proposition 2 allows us to limit attention to the conditional distribution over other agents' actions given the state. Therefore, let $z_i^{0:T} = \{s^t, o_i^t, n_{i,l}^t, a_i^t, a_j^t, \ldots, a_k^t\}_0^T$, where the observation at $t = 0$ is null, and other agents are labeled j to k; this group is denoted $-i$. The full log likelihood involves an expectation over the hidden variables:

$$Q(\pi'_{i,l}|\pi_{i,l}) = \sum_{T=0}^{\infty} \sum_{z_i^{0:T}} Pr(r_i^T = 1, z_i^{0:T}, T; \pi_{i,l}) \times \log\ Pr(r_i^T = 1, z_i^{0:T}, T; \pi'_{i,l}) \quad (8)$$

Due to the subjective perspective in I-POMDPs, Q computes the likelihood of agent i's FSC only (and not of joint FSCs).

In the above expected log likelihood, $Pr(r_i^T = 1, z_i^{0:T}, T; \pi_{i,l})$ may be written as:

$$Pr(r_i^T = 1, z_i^{0:T}, T; \pi_{i,l}) = Pr(r_i^T = 1, z_i^{0:T} | T; \pi_{i,l})\ Pr(T) = (1-\gamma)\gamma^T Pr(r_i^T = 1 | a_i^T, a_{-i}^T, s^T) \mathcal{V}_i(n_{i,l}^0) b_{i,l}^0(s^0) \mathcal{L}_i(n_{i,l}^0, a_i^0) \times \prod_{-i} Pr(a_{-i}^0 | s^0) \prod_{t=1}^{T} \mathcal{T}_i(n_{i,l}^{t-1}, a_i^{t-1}, o_i^t, n_{i,l}^t)\ T_i(s^{t-1}, a_i^{t-1}, a_{-i}^{t-1}, s^t)\ O_i(s^t, a_i^{t-1}, a_{-i}^{t-1}, o_i^t)\ \mathcal{L}_i(n_{i,l}^t, a_i^t) \prod_{-i} Pr(a_{-i}^t | s^t) \quad (9)$$

In addition to parameters of I-POMDP$_{i,l}$, which are given, parameters of agent i's controller and those relating to other agents' predicted actions, $\phi_{-i,0}$, are present in Eq. 9. If the latter are available, Eq. 9 allows us to find parameters of i's level l controller that maximize the likelihood. Notice that in consequence of Proposition 2, Eq. 9 precludes j's observation and node transition functions.

In the T-step DBN of Fig. 1, observed evidence includes the reward, r_i^T, at the end and the initial belief. We seek the likely distributions, $\mathcal{V}_i$, $\mathcal{T}_i$, and $\mathcal{L}_i$, across time slices. We may again realize the full joint in the expectation using a forward-backward algorithm on a hidden Markov model whose state is $(s^t, n_{i,l}^t)$. The transition function of this model is,

$$Pr(s^t, n_{i,l}^t | s^{t-1}, n_{i,l}^{t-1}) = \sum_{a_i^{t-1}, a_{-i}^{t-1}, o_i^t} \mathcal{L}_i(n_{i,l}^{t-1}, a_i^{t-1}) \times \prod_{-i} Pr(a_{-i}^{t-1} | s^{t-1})\ \mathcal{T}_i(n_{i,l}^{t-1}, a_i^{t-1}, o_i^t, n_{i,l}^t) \times T_i(s^{t-1}, a_i^{t-1}, a_{-i}^{t-1}, s^t)\ O_i(s^t, a_i^{t-1}, a_{-i}^{t-1}, o_i^t) \quad (10)$$

The forward message, $\alpha^t = Pr(s^t, n_{i,l}^t)$, represents the probability of being at some state of the DBN at time t:

$$\alpha^t(s^t, n_{i,l}^t) = \sum_{s^{t-1}, n_{i,l}^{t-1}} Pr(s^t, n_{i,l}^t | s^{t-1}, n_{i,l}^{t-1})\ \alpha^{t-1}(s^{t-1}, n_{i,l}^{t-1}) \quad (11)$$

where, $\alpha^0(s^0, n_{i,l}^0) = \mathcal{V}_i(n_{i,l}^0) b_{i,l}^0(s^0)$.

The backward message gives the probability of observing the reward in the final T^{th} time step given some state of the Markov model, $\beta^t(s^t, n_{i,l}^t) = Pr(r_i^T = 1 | s^t, n_{i,l}^t)$:

$$\beta^h(s^t, n_{i,l}^t) = \sum_{s^{t+1}, n_{i,l}^{t+1}} Pr(s^{t+1}, n_{i,l}^{t+1} | s^t, n_{i,l}^t)\ \beta^{h-1}(s^{t+1}, n_{i,l}^{t+1}) \quad (12)$$

where, $\beta^0(s^T, n_{i,l}^T) = \sum_{a_i^T, a_{-i}^T} Pr(r_i^T = 1 | s^T, a_i^T, a_{-i}^T)\ \mathcal{L}_i(n_{i,l}^T, a_i^T) \prod_{-i} Pr(a_{-i}^T | s^T)$, and $1 \leq h \leq T$ is the horizon. Here, $Pr(r_i^T = 1 | s^T, a_i^T, a_{-i}^T) \propto R_i(s^T, a_i^T, a_{-i}^T)$.

A side effect of $Pr(a_{-i}^t | s^t)$ being dependent on t is that we can no longer conveniently define $\widehat{\alpha}$ and $\widehat{\beta}$ for use in M-step at level 1. Instead, the computations are folded in the M-step.

M-step We update the parameters, $\mathcal{L}_i$, $\mathcal{T}_i$ and $\mathcal{V}_i$, of $\pi_{i,l}$ to obtain $\pi'_{i,l}$ based on the expectation in the E-step. Specifically, take log of the likelihood in Eq. 9 with $\pi_{i,l}$ substituted with $\pi'_{i,l}$ and focus on terms involving the parameters of $\pi'_{i,l}$:

$$\log Pr(r^T = 1, z_i^{0:T}, T; \pi'_{i,l}) = \langle \text{terms indep. of } \pi'_{i,l} \rangle + \sum_{t=0}^{T} \log \mathcal{L}'_i(n_{i,l}^t, a_i^t) + \sum_{t=0}^{T-1} \log \mathcal{T}'_i(n_{i,l}^t, a_i^t, o_i^{t+1}, n_{i,l}^{t+1}) + \log \mathcal{V}'_i(n_{i,l})$$

Update of action probabilities – $\mathcal{L}_i$:

In order to update, $\mathcal{L}_i$, we partially differentiate the Q-function of Eq. 8 with respect to $\mathcal{L}'_i$. To facilitate differentiation, we focus on the terms involving $\mathcal{L}_i$, as shown below.

$$Q(\pi'_{i,l}|\pi_{i,l}) = \langle \text{terms indep. of } \mathcal{L}'_i \rangle + \sum_{T=0}^{\infty} \Pr(T) \times \sum_{t=0}^{T} \sum_{z_i^{0:T}} \Pr(r_i^T = 1, z_i^{0:t}|T; \pi_{i,l}) \log \mathcal{L}'_i(n_{i,l}^t, a_i^t)$$

$\mathcal{L}'_i$ on maximizing the above equation is:

$$\mathcal{L}'_i(n_{i,l}^t, a_i^t) \propto \mathcal{L}_i(n_{i,l}^t, a_i^t) \sum_{T=0}^{\infty} \prod_{-i} \sum_{s^T, a_{-i}^T} \frac{\gamma^T}{1-\gamma} \times Pr(r_i^T = 1|s^T, a_i^T, a_{-i}^T)\ Pr(a_{-i}^T|s^T)\ \alpha(s^T, n_{i,l}^T)$$

Update of node transition probabilities – $\mathcal{T}_i$:

In order to facilitate maximizing the Q-function with respect to $\mathcal{T}'_i$, we rewrite it to focus on terms involving $\mathcal{T}'_i$.

$$Q(\pi'_{i,l}|\pi_{i,l}) = \langle \text{terms indep. of } \mathcal{T}'_i \rangle + \sum_{T=0}^{\infty} \Pr(T) \sum_{t=0}^{T} \sum_{z_i^{0:T}} \Pr(r_i^T = 1, z_i^{0:t}|T; \pi_{i,l}) \log \mathcal{T}'_i(n_{i,l}^t, a_i^t, o_i^{t+1}, n_{i,l}^{t+1})$$

On maximizing the above equation for $\mathcal{T}'_i$ we get:

$$\mathcal{T}'_i(n_{i,l}^t, a_i^t, o_i^{t+1}, n_{i,l}^{t+1}) \propto \mathcal{T}_i(n_{i,l}^t, a_i^t, o_i^{t+1}, n_{i,l}^{t+1})\ \mathcal{L}_i(n_{i,l}^t, a_i^t) \times \prod_{-i} \sum_{s^t, s^{t+1}, a_{-i}^t} \beta^0(s^{t+1}, n_{i,l}^{t+1})\ T_i(s^t, a_i^t, a_{-i}^t, s^{t+1}) \times O_i(s^{t+1}, a_i^t, a_{-i}^t, o_i^{t+1}) \sum_{T=0}^{\infty} Pr(T) \alpha^T(s^T, n_{i,l}^T) Pr(a_{-i}^T|s^T)$$

Node distribution $\mathcal{V}_i$ for $\pi'_{i,l}$, is updated analogously to $\mathcal{L}_i$ and $\mathcal{T}_i$.

Greedy M-step Analogously to the greedy maximization in EM for POMDPs (Toussaint, Charlin, and Poupart 2008), we obtain a greedy variant of the M-step. It may be viewed as greedily performing an infinite number of maximizations while keeping the expectation function fixed.

Because a FSC is inferred at level 1, at strategy levels, $l = 2$ and greater, lower-level candidate models are FSCs. EM at these higher levels proceeds by replacing the state of the DBN, $(s^t, n_{i,l}^t)$, with $(s^t, n_{i,l}^t, n_{j,l-1}^t, \ldots, n_{k,l-1}^t)$. The joint probability of Eq. 9 and the transition function of the DBN, Eq. 10, expand to accommodate this larger state.

Interleaved iterations for anytime behavior The presence of models at different levels for the agents leads to novel challenges and candidate approaches. An obvious way of extending EM to the context of finitely-nested I-POMDPs would be to improve $\phi_{j,0}$ at level 0 until convergence before moving to the higher level. However, the higher-level controller may not be improved for some time until the lower level converges. Consequently, Sonu and Doshi (2014) introduced a more sophisticated approach in the context of BPI that interleaves improvements of the other agent's controllers (or action distributions in this paper) with improvements of the subject agent's controller. This facilitates *anytime behavior* with the challenge that the interactive state space may change dynamically at every iteration; we we adapt it to EM as well.

Block-Coordinate Descent for Speed Up

Block-coordinate descent (BCD) (Fessler and Hero 1994; Saha and Tewari 2013; Tseng and Mangasarian 2001) is an iterative scheme to gain faster non-asymptotic rate of convergence in the context of large-scale N-dimensional optimization problems. In this scheme, within each iteration, a set of variables referred to as coordinates are chosen and the objective function, Q, is optimized with respect to one of the coordinate blocks while the other coordinates are held *fixed*. BCD may speed up the non-asymptotic rate of convergence of EM for both I-POMDPs and POMDPs. *The specific challenge here is to determine which of the many variables should be grouped into blocks and how.*

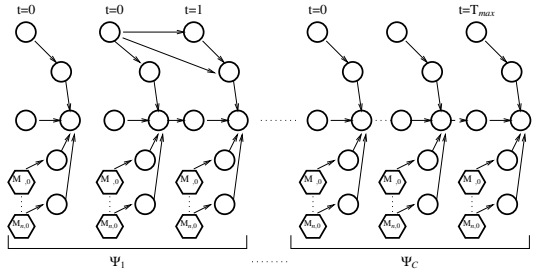


Figure 3: A schematic illustrating the mixture of DBNs grouped into blocks. Block Ψ_1, for e.g., contains 2 DBNs of length 1 and 2.

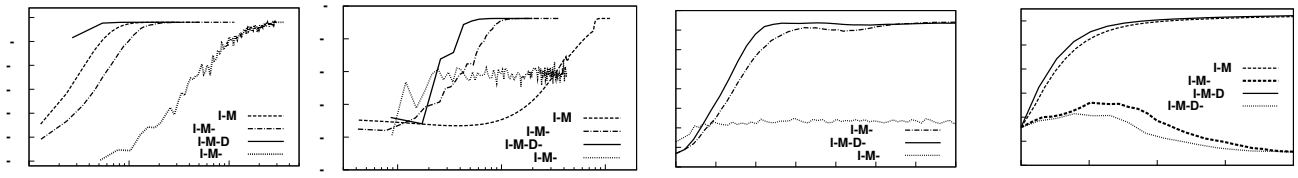
We empirically show in Section that grouping the number of time slices, t, and horizon, h, in Eqs. 11 and 12, respectively, at each level, into coordinate blocks of equal size is beneficial. *In other words, we decompose the mixture model into blocks containing equal numbers of Bayesian networks.* Formally, let Ψ_1^t be a subset of $\{T = 1, T = 2, \ldots, T = T_{max}\}$. Then, the set of blocks is, $B_t = \{\Psi_1^t, \Psi_2^t, \Psi_3^t, \ldots\}$. In practice, because both t and h are finite (say, T_{max}), the cardinality of B_t is bounded by some $C \geq 1$. Analogously, we define the set of blocks of h, denoted by B_h.

In the M-step now, we compute α^t for the time steps in a single coordinate block, Ψ_c^t, only, while using the values of α^t from the *previous* iteration for the complementary coordinate blocks, $\tilde{\Psi}_c^t$. Analogously, we compute β^h for the horizons in Ψ_c^h only, while using β values from the previous iteration for the remaining horizons. We cyclically choose a block, Ψ_c^t, at iterations, $i = c + qC$ where $q \in \{0, 1, 2, \ldots\}$.

Forward Filtering - Backward Sampling

Another approach for exploiting embedded structure in the transition and observation functions is to replace the exact forward-backward message computations with exact forward filtering and backward sampling (FFBS) (Carter and Kohn 1996) to obtain a sampled reverse trajectory consisting of $\langle s^T, n_{i,l}^T, a_i^T \rangle$, $\langle n_{i,l}^{T-1}, a_i^{T-1}, o_i^T, n_{i,l}^T \rangle$, and so on from T to 0. Here, $Pr(r_i^T = 1|s^T, a_i^T, a_{-i}^T)$ is the likelihood weight of this trajectory sample. Parameters of the updated FSC, $\pi'_{i,l}$, are obtained by summing and normalizing the weights.

Each trajectory is obtained by first sampling $\hat{T} \sim Pr(T)$, which becomes the length of i's DBN for this sample. Forward message, $\alpha^t(s^t, n_{i,l}^t)$, $t = 0 \ldots \hat{T}$ is computed exactly



(a) 5-agent tiger: all EM methods (b) 2-agent ML: all EM methods (c) 3-agent UAV: all EM methods (d) 5-agent policing: all EM methods

Figure 4: FSCs improve with time for I-POMDP$_{i,1}$ in the (a) 5-agent tiger, (b) 2-agent money laundering,(c) 3-agent UAV, and (d) 5-agent policing contexts. Observe that BCD causes substantially larger improvements in the initial iterations until we are close to convergence. We explored various block configurations eventually settling on 3 equal-sized blocks for both the tiger and money laundering problems, 5 blocks for UAV and 2 for policing protest. All experiments were run on Linux with Intel Xeon 2.6GHz CPUs and 32GB RAM.

(Eq. 11) followed by the backward message, $\beta^h(s^t, n^t_{i,l})$, $h = 0 \ldots \hat{T}$ and $t = \hat{T} - h$. *Computing β^h differs from Eq. 12 by utilizing the forward message*:

$$\beta^h(s^t, n^t_{i,l}|s^{t+1}, n^{t+1}_{i,l}) = \sum\nolimits_{a^t_i, a^t_{-i}, o^{t+1}_i} \alpha^t(s^t, n^t_{i,l})\ \mathcal{L}_i(n^t_{i,l}, a^t_i)$$
$$\prod\nolimits_{-i} Pr(a^t_{-i}|s^t)\ T_i(s^t, a^t_i, a^t_{-i}, s^{t+1}) \mathcal{T}_i(n^t_{i,l}, a^t_i, o^{t+1}_i, n^{t+1}_{i,l})$$
$$O_i(s^{t+1}, a^t_i, a^t_{-i}, o^{t+1}_i) \quad (13)$$

where $\beta^0(s^T, n^T_{i,l}, r^T_i) = \sum_{a^t_i, a^t_{-i}} \alpha^T(s^T, n^T_{i,l}) \prod_{-i} Pr(a^T_{-i}|s^T)\ \mathcal{L}(n^T_{i,l}, a^T_i)\ Pr(r^T_i|s^T, a^T_i, a^T_{-i})$. Subsequently, we may easily sample $\langle s^T, n^T_{i,l}, r^T_i \rangle$ followed by sampling $s^{T-1}_i, n^{T-1}_{i,l}$ from Eq. 13.

We sample $a^{T-1}_i, o^T_i \sim Pr(a^t_i, o^{t+1}_i|s^t, n^t_{i,l}, s^{t+1}, n^{t+1}_{i,l})$, where:

$$Pr(a^t_i, o^{t+1}_i|s^t, n^t_{i,l}, s^{t+1}, n^{t+1}_{i,l}) \propto \prod\nolimits_{-i} Pr(a^t_{-i}|s^t)\ \mathcal{L}_i(n^t_{i,l}, a^t_i)$$
$$\mathcal{T}_i(n^t_{i,l}, a^t_i, o^{t+1}_i, n^{t+1}_{i,l})\ T_i(s^t, a^t_i, a^t_{-i}, s^{t+1})\ O_i(s^{t+1}, a^t_i, a^t_{-j}, o^{t+1}_i)$$

Computational Complexity

Our EM at level 1 is significantly quicker compared to ascribing FSCs to other agents. In the latter, nodes of others' controllers must be included alongside s and $n_{i,l}$.

Proposition 3 (E-step speed up)**.** *Each E-step at level 1 using the forward-backward pass as shown previously results in a net speed up of $\mathcal{O}((|M||\mathcal{N}_{-i,0}|)^{2K}|\Omega_{-i}|^K)$ over the formulation that ascribes $|M|$ FSCs each to K other agents with each having $|\mathcal{N}_{-i,0}|$ nodes.*

Analogously, updating the parameter, $\mathcal{L}_i$ and $\mathcal{T}_i$ in the M-step exhibits a speedup of $\mathcal{O}((|M||\mathcal{N}_{-i,0}|)^{2K}|\Omega_{-i}|^K)$, while $\mathcal{V}_i$ leads to $\mathcal{O}((|M||\mathcal{N}_{-i,0}|)^K)$. This improvement is exponential in the number of other agents. On the other hand, the E-step at level 0 exhibits complexity that is typically greater compared to the total complexity of the E-steps for $|M|$ FSCs.

Proposition 4 (E-step ratio at level 0)**.** *E-steps when $|M|$ FSCs are inferred for K agents exhibit a ratio of complexity, $\mathcal{O}(\frac{|\mathcal{N}_{-i,0}|^2}{|M|})$, compared to the E-step for obtaining $\phi_{-i,0}$.*

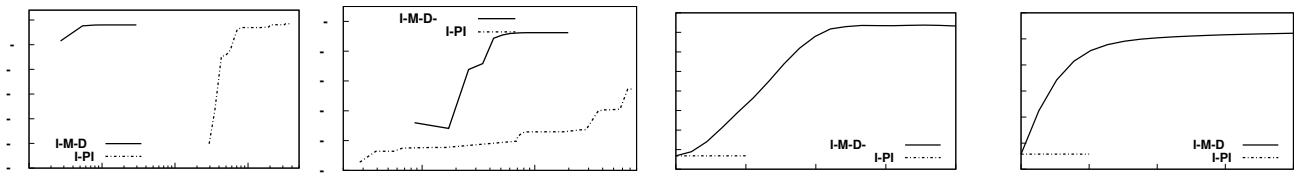
The ratio in Proposition 4 is less than 1 when smaller-sized controllers are sought and there are several models.

Experiments

Five variants of EM are evaluated as appropriate: the exact EM inference-based planning (labeled as I-EM); replacing the exact M-step with its greedy variant (I-EM-Greedy); iterating EM based on coordinate blocks (I-EM-BCD) and coupled with a greedy M-step (I-EM-BCD-Greedy); and lastly, using forward filtering-backward sampling (I-EM-FFBS).

We use 4 problem domains: the noncooperative *multiagent tiger problem* (Doshi and Gmytrasiewicz 2009) ($|S|$= 2, $|A_i|$= $|A_j|$= 3, $|O_i|$= $|O_j|$= 6 for level $l \geq 1$, $|O_j|$= 3 at level 0, and $\gamma = 0.9$) *with a total of 5 agents* and 50 models for each other agent. A larger noncooperative *2-agent money laundering (ML) problem* (Ng et al. 2010) forms the second domain. This is a game between law enforcement (blue team) and money launderers (red team) who aim to move their assets from a 'dirty' pot to a 'clean' one through a series of financial transactions while evading capture by the blue team. It exhibits 99 physical states for the subject agent (blue team), 9 actions for blue and 4 for the red team, 11 observations for subject and 4 for the other, with about 100 models for red team. We also evaluate a *3-agent UAV reconnaissance problem* involving a UAV tasked with intercepting two fugitives in a 3x3 grid before they both reach the safe house (Zeng and Doshi 2012). It has 162 states for the UAV, 5 actions, 4 observations for each agent, and 200,400 models for the two fugitives. Finally, a new *policing protest problem* is used in which police must maintain order in 3 designated protest sites populated by 4 groups of protesters who may be peaceful or disruptive. It exhibits 27 states, 9 policing and 4 protesting actions, 8 observations, and 600 models per protesting group. The latter two domains are historically the largest test problems for I-POMDPs. In contrast to previous works (Kumar and Zilberstein 2010; Toussaint and Storkey 2006), we report time elapsed instead of number of iterations because each iteration could consume excessive time and BCD introduces several quick iterations.

Comparative performance of all methods In Fig. 4, we compare the variants on all problems. Each method starts with a random seed, and the converged value is significantly better than a random FSC for all methods and problems. Increasing the sizes of FSCs gives better values in general but also increases time; using FSCs of sizes 5, 3, 9 and 5, for



(a) 5-agent tiger: I-EM-BCD, I-BPI (b) 2-agent ML: I-EM-BCD-Greedy, I-BPI (c) 3-agent UAV: I-EM-BCD-Greedy, I-BPI (c) 5-agent policing: I-EM-BCD, I-BPI

Figure 5: I-EM-BCD converges significantly quicker than I-BPI to similar-valued FSCs for (a) multiagent tiger, (b) money laundering, (c) UAV reconnaissance, and (d) policing problems. All were run on Linux with Intel Xeon 2.6GHz CPUs and 32GB RAM.

the 4 domains respectively demonstrated a good balance. I-EM and the Greedy and BCD variants clearly exhibit an anytime property on the tiger, UAV and policing problems. The noncooperative ML shows delayed increases because we show the value of agent i's controller and initial improvements in the other agent's controller may maintain or decrease the value of i's controller. This is not surprising due to competition in the problem. Nevertheless, after a small delay the values improve steadily which is desirable. Importantly, absent interleaving there is a much longer period with no increase until all lower levels converge.

I-EM-BCD consistently improves on I-EM and is often the fastest: the corresponding value improves by large steps initially (fast non-asymptotic rate of convergence). In the context of ML and UAV, I-EM-BCD-Greedy shows substantive improvements leading to controllers with much improved values compared to other approaches. Despite a low sample size of about 1,000 for the problems, I-EM-FFBS obtains FSCs whose values improve in general for tiger and ML, though slowly and not always to the level of others. This is because the EM gets caught in a worse local optima due to sampling approximation – this strongly impacts the UAV problem; more samples did not escape these optima. However, forward filtering only (as used in Wu et al. [2013]) required a much larger sample size to reach these levels. FFBS did not improve the controller in the fourth domain.

Characterization of local optima While an exact solution for the smaller tiger problem with 5 agents (or the larger problems) could not be obtained for comparison, I-EM climbs to the optimal value of 8.51 for the downscaled 2-agent version (not shown in Fig. 4). In comparison, BPI does not get past the local optima of -10 using an identical-sized controller – corresponding controller predominantly contains listening actions – relying on adding nodes to eventually reach optimum. While we are unaware of any general technique to escape local convergence in EM, I-EM can reach the global optimum given an appropriate seed. This may not be a coincidence: the I-POMDP value function space exhibits a single fixed point – the global optimum – which in the context of Proposition 1 makes the likelihood function, $Q(\pi'_{i,l}|\pi_{i,l})$, unimodal (if $\pi_{i,l}$ is appropriately sized as we do not have a principled way of adding nodes). If $Q(\pi'_{i,l}|\pi_{i,l})$ is continuously differentiable for the domain on hand, Corollary 1 in Wu [1983] indicates that $\pi_{i,l}$ will converge to the unique maximizer.

Improvement on I-BPI We compare the quickest of the I-EM variants with previous best algorithm, I-BPI (Figs. 5), allowing the latter to escape local optima as well by adding nodes. Observe that FSCs improved using I-EM-BCD converges to values similar to those of I-BPI almost *two orders of magnitude faster*. Beginning with 5 nodes, I-BPI adds 4 more nodes to obtain the same level of value as EM for the tiger problem. For money laundering, I-EM-BCD-Greedy converges to controllers whose value is at least 1.5 times better than I-BPI's given the same amount of allocated time and less nodes. I-BPI failed to improve the seed controller and could not escape for the UAV and policing protest problems.

To summarize, this makes I-EM *variants, with emphasis on BCD, the fastest iterative approaches for infinite-horizon I-POMDPs currently.*

Discussion

Recent applications of I-POMDPs in diverse areas testify to its broad significance and motivate better approximations. The EM formulation of Section builds on the EM for POMDP and differs drastically from the E- and M-steps for the cooperative DEC-POMDP (Kumar and Zilberstein 2010). The differences reflect how I-POMDPs build on POMDPs and differ from DEC-POMDPs. These begin with the structure of the DBNs where the DBN for I-POMDP$_{i,1}$ in Fig. 1 adds to the DBN for POMDP hexagonal model nodes that contain candidate models; chance nodes for action; and model update edges for each other agent at each time step. This differs from the DBN for DEC-POMDP, which adds controller nodes for all agents and a joint observation chance node. The I-POMDP DBN contains controller nodes for the subject agent only, and each model node collapses into an efficient distribution on running EM at level 0. These differences in DBNs immediately translate into differences in the expected log likelihoods. This results in E- and M-steps that generalize those of POMDPs and differ significantly from those of DEC-POMDPs.

In domains where the joint reward function may be decomposed into factors encompassing subsets of agents, ND-POMDPs allow the value function to be factorized as well. Kumar et al. [2011] exploit this structure by simply decomposing the whole DBN mixture into a mixture for each factor and iterating over the factors. Interestingly, the M-step may be performed individually for each agent and this approach scales beyond two agents. Pajarinen and Pelto-

nen [2011] improve the approach mainly by limiting to forward message passing only in the E-step. We exploit both graphical and problem structures to speed up and scale in a way that is contextual to I-POMDPs. BCD also decomposes the DBN mixture into equal blocks of horizons. While it has been applied in other areas (Arimoto 1972; Fessler and Kim 2011), these applications do not transfer to planning.

Additionally, problem structure is considered by using FFBS that exploits *information* in the transition and observation distributions of the subject agent. FFBS could be viewed as a tenuous example of Monte Carlo EM, which is a broad category and also includes the forward sampling utilized by Wu et al. [2013] for DEC-POMDPs. However, fundamental differences exist between the two: forward sampling may be run in simulation and does not require the transition and observation functions. Indeed, Wu et al. utilize it in a model free setting. FFBS is model based utilizing exact forward messages in the backward sampling phase. This reduces the accumulation of sampling errors over many time steps in extended DBNs, which otherwise afflicts forward sampling.

The advance in this paper has wider relevance to areas such as game play and ad hoc teams where agents are modeled. However, EM as with BPI is also susceptible to locality leading to controllers whose quality is unpredictable.

Acknowledgments

This research is supported in part by a NSF CAREER grant, IIS-0845036, and a grant from ONR, N000141310870. We thank Akshat Kumar and Feng Wu for valuable feedback that led to improvements in the paper.

References

2015. Online appendix for proofs only (anonymized). https://www.dropbox.com/s/irnb60aioe1o9tz/ijcai15_730_supplementary.pdf?dl=0. Accessed: Feb 12, 2015.

Arimoto, S. 1972. An algorithm for computing the capacity of arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory* 18(1):14–20.

Attias, H. 2003. Planning by probabilistic inference. In *Proc. of the 9th Int. Workshop on Artificial Intelligence and Statistics*.

Carter, C. K., and Kohn, R. 1996. Markov chainmonte carlo in conditionally gaussian state space models. *Biometrika* 83:589–601.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal Of The Royal Statistical Society, Series B* 39(1):1–38.

Doshi, P., and Gmytrasiewicz, P. J. 2009. Monte carlo sampling methods for approximating interactive pomdps. *Journal of Artificial Intelligence Research* 34:297–337.

Fessler, J. A., and Hero, A. O. 1994. Space-alternating generalized expectation-maximization algorithm. *IEEE Transactions on Signal Processing* 42:2664–2677.

Fessler, J. A., and Kim, D. 2011. Axial block coordinate descent (ABCD) algorithm for X-ray CT image reconstruction. In *International Meeting on Fully Three-dimensional Image Reconstruction in Radiology and Nuclear Medicine*, volume 11, 262–265.

Gmytrasiewicz, P. J., and Doshi, P. 2005. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research (JAIR)* 24:49–79.

Kumar, A., and Zilberstein, S. 2010. Anytime planning for decentralized POMDPs using expectation maximization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 294–301.

Kumar, A.; Zilberstein, S.; and Toussaint, M. 2011. Scalable multiagent planning using probabilistic inference. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2140–2146.

Ng, B.; Meyers, C.; Boakye, K.; and Nitao, J. 2010. Towards applying interactive pomdps to real-world adversary modeling. In *Innotative Applications of AI (IAAI)*, 1814–1820.

Pajarinen, J., and Peltonen, J. 2011. Efficient planning for factored infinite-horizon DEC-POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 325–331.

Saha, A., and Tewari, A. 2013. On the nonasymptotic convergence of cyclic coordinate descent methods. *SIAM Journal on Optimization* 23(1):576–601.

Sonu, E., and Doshi, P. 2014. Scalable solutions of interactive POMDPs using generalized and bounded policy iteration. *Journal of Autonomous Agents and Multi-Agent Systems* DOI: 10.1007/s10458–014–9261–5, in press.

Toussaint, M., and Storkey, A. 2006. Probabilistic inference for solving discrete and continuous state markov decision processes. In *International Conference on Machine Learning (ICML)*, 945–952.

Toussaint, M.; Charlin, L.; and Poupart, P. 2008. Hierarchical POMDP controller optimization by likelihood maximization. In *Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI)*, 562–570.

Toussaint, M.; Harmeling, S.; and Storkey, A. 2006. Probabilistic inference for solving (po)mdps. Technical report, Technical Report EDIINF-RR-0934, Univ. of Edinburgh.

Tseng, P., and Mangasarian, C. O. L. 2001. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications* 475–494.

Wu, F.; Zilberstein, S.; and Jennings, N. R. 2013. Monte-carlo expectation maximization for decentralized POMDPs. In *Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, 397–403.

Wu, C. F. J. 1983. On the convergence properties of the em algorithm. *Annals of Statistics* 11(1):95–103.

Zeng, Y., and Doshi, P. 2012. Exploiting model equivalences for solving interactive dynamic influence diagrams. *Journal of Artificial Intelligence Research* 43:211–255.

A Generative Game-Theoretic Framework for Adversarial Plan Recognition

Nicolas Le Guillarme and **Abdel-Illah Mouaddib**
GREYC (UMR 6072)
University of Caen Basse-Normandie
Caen, France
{nicolas.leguillarme,abdel-illah.mouaddib}@unicaen.fr

Xavier Lerouvreur
Signal Processing, Algorithms & Performance
Airbus Defence and Space
Toulouse, France
xavier.lerouvreur@astrium.eads.net

Sylvain Gatepaille
Advanced Information Processing
Airbus Defence and Space
Val-de-Reuil, France
sylvain.gatepaille@cassidian.com

Abstract

Adversarial reasoning is of the first importance for defence and security applications since it allows to (1) better anticipate future threats, and (2) be proactive in deploying effective responses. In this paper, we address the two subtasks of adversarial reasoning, namely adversarial plan recognition and strategy formulation, from a generative, game-theoretic perspective. First, a set of possible future situations is computed using a contextual action model. This projected situation serves as a basis for building a set of Markov games modeling the planning strategies of both the defender and his adversary. Finally, a library of critical plans for the attacker and a library of best responses for the defender are generated automatically by computing a Nash equilibrium in each game. The adversarial plan recognition task therefore consists of inferring a probability distribution over the set of possible plans of the adversary, while the strategy formulation problem reduces to the selection of the most appropriate response. Initial results on a urban warfare scenario suggest that our framework can be useful to model complex strategic interactions inherent to plan recognition in adversarial situations.

Introduction

Defence and security activities, such as military operations or cybersecurity to name a few, are adversarial by nature. In such situations, where a decision-maker ("the defender") is threatened by one or several intelligent opponents with conflicting goals ("the attackers"), a key enabler for the success of operations is the ability for the defender to "read the opponent's mind" (Kott and McEneaney 2006). Adversarial reasoning generally consists of inferring the intentions of an adversary from the available evidence so as to make possible the prediction of his future actions, which then enables the planning of an effective response. Adversarial reasoning can therefore be divided into two main subtasks, namely *adversarial plan recognition*, or the identification of an adversary's plans and goals on the basis of observations of his actions, and *strategy formulation*, where the defender has to formulate his own plan of action taking into account possible counteractions of the adversary.

Compared to other types of plan recognition, namely intended plan recognition, where the observed agent is cooperative, and keyhole plan recognition, where the observed agent is not aware or indifferent to the recognition process (Geib and Goldman 2001), adversarial plan recognition is characterized by the fact that the observer (defender) and the observed agent (attacker) evolve in the same environment and that the observed agent is at least not cooperative, if not actively hostile to the inference of his plans. These characteristics of adversarial plan recognition result in two types of adversarial interactions between the defender and the attacker. First, since both agents evolve in the same environment and pursue conflicting goals, each agent will have to reason about the planning strategy of his opponent to plan his actions, keeping in mind that his adversary will do the same. Secondly, should the observed agent be actively hostile to the recognition process, he will plan his actions accordingly so as to minimize the probability for his plans to be recognized (e.g. using concealment and/or deceptive actions), while the plan recognition process will have to reason about the adversary's planning strategy to predict his action.

Game-theory is a natural and popular formalism to address problems involving interactions between agents ranging from total cooperation to full hostility (Burkov and Chaib-draa 2008). In (Lisý et al. 2012), the problem of adversarial plan recognition is defined as an imperfect information two-player zero-sum game in extensive form between an actor and an observer. From a game-theoretic perspective, the solution is the strategies for both players that optimize the tradeoff between plan utility and plan detectability. The drawbacks of this method are that a plan library must be provided explicitly, the model does not handle simultaneous decisions, and the observer can only perform activity recognition actions. In (Braynov 2006), the interaction between adversarial planning and adversarial plan recognition is modeled as a two-player zero-sum game over an attack graph representing the possible plans of the attacker. This approach addresses both types of strategic interactions between the observer and the observed agent. However, it relies on a plan recognition algorithm as a plugin. Closest to the present paper is (Chen et al. 2007) where a set of optimal courses of action (COAs) for the enemy and friendly forces is generated using a decentralized Markov game. The state space is defined as the set of all the possible COAs for enemy, friendly, and neutral forces. The effectiveness of the approach has been demonstrated through combat simulation, but the authors are not explicit about the way

their method can be used for goal/plan recognition.

In this work, we propose a generative game-theoretic framework to tackle the problem of adversarial plan recognition in a fully observable, multi-agent setting where actions have stochastic effects. The adversarial plan recognition task therefore consists of inferring a probability distribution over the set of possible plans of an agent whose behavior results from a finite two-player zero-sum stochastic game, where the players are respectively the attacker/observed agent and the defender/observer. The problem of strategy formulation then reduces to the selection of the most likely best response. Stochastic games are used to model the strategic interactions between the defender and the attacker planning strategies due to the common environment and their conflicting goals. The use of deception and concealment by the attacker is out of the scope of this paper and is left for future work. Since generative plan recognition techniques require the definition of an action model to encode the possible behaviors of the observed agent, we define a contextual action model based on the COI model of intentional action (Steinberg 2007) which enables the automatic assessment of all opportunities for action available to each player in the current situation.

The rest of this paper is organized as follows. In Section 2, we provide a brief background on generative plan recognition and stochastic game theory. Section 3 contains an extensive description of PRESAGE, our generative game-theoretic framework for adversarial plan recognition. Section 4 applies this to a urban warfare scenario and presents preliminary experimental results. In Section 5 we discuss the current progress of our work and the planned extensions.

Background

Generative Plan Recognition

Most of previous research in plan recognition relies on an a priori, handcrafted plan library specifying all the possible plans of the observed agent (Avrahami-Zilberbrand and Kaminka 2007; Geib and Goldman 2009; Lisỳ et al. 2012). This library is generally assumed to be exhaustive, which is quite impractical in real-world applications. Recently, a new paradigm known as *generative plan recognition* (also called *model-based* plan recognition or plan recognition *by inverse planning*) has been proposed by Baker et al. (Baker, Saxe, and Tenenbaum 2009) and further studied in a series of work by Ramírez and Geffner (Ramırez and Geffner 2009; 2010; 2011). In plan recognition by inverse planning, observed agents are assumed to be rational, i.e. they will plan optimally to achieve their goals. The first advantage of generative methods is that the need for an explicit declaration of the possible agent behaviors as a plan library is replaced by an implicit encoding, using an agent action model and a set of possible goals, thus making generative methods far more flexible and less dependent on the availability of expert knowledge. The second advantage is that the set of optimal plans can be generated automatically using state-of-the-art planners, including classical planners (Ramırez and Geffner 2009; 2010), Markov Decision Processes (MDPs) (Baker, Saxe, and Tenenbaum 2009), and partially-observable MDPs (Ramırez and Geffner 2011).

Stochastic Games

Definition Stochastic games - also called Markov games - have been introduced by Lloyd Shapley (Shapley 1953) as an extension of MDPs to the multi-agent case. In a stochastic game, the players perform joint actions that determine both the new state of the environment, according to transition probabilities, and the reward obtained by each agent. Formally, a stochastic game is defined as a tuple $\langle Ag, S, \mathbf{A}, \{R^i, i = 1, ..., |Ag|\}, T\rangle$ where

- $Ag = \{1, ..., |Ag|\}$ is a finite set of players.
- S is a finite set of states.
- $A_i = \{a^i_1, ..., a^i_{|A_i|}\}$ is a finite set of actions available to player i.
- $\mathbf{A} \equiv \times_{i \in Ag} A_i$ is the set of joint actions
- $R^i : S \times \mathbf{A} \to \mathbb{R}$ is the payoff function of player i.
- $T : S \times \mathbf{A} \times S \to [0, 1]$ is the transition function.

The game is played in discrete time steps. In each time step t, the players choose their actions simultaneously and the joint action $\mathbf{a} = \{a^1, ..., a^{|A_g|}\}$ is obtained. Each agent i receives a reward $R^i(s_t, \mathbf{a})$ depending on the current state of the environment and the joint action, and the players are transferred to the next state s_{t+1} according to the transition function T. A policy $\pi_i : S \to \Delta A_i$ for agent i defines for each state of the game a local, mixed strategy, i.e. a probability distribution over the set of available actions.

Solution concepts Given a joint strategy $\pi = \langle \pi_i, \pi_{-i}\rangle$ where π_{-i} is the joint strategy of all players except player i, the expected utility of π for player i is defined in each $s \in S$ as the expected value of the utility function of normal form games (Burkov and Chaib-draa 2008)

$$u^\pi_i(s) = E_{\mathbf{a} \in \mathbf{A}}[R^i(s, \mathbf{a})] \tag{1}$$

The state utilities $U^\pi_i(s)$ for player i associated with joint policy π can be quantified using the same performance criteria than for MDPs, i.e. the long-term expected value over i's reward. For instance, using the γ-discounted criterion

$$\begin{aligned} U^\pi_i(s) &= E\left[\sum_{t=0}^{\infty} \gamma^t u^\pi_i(s_t) | s_0 = s\right] \\ &= u^\pi_i(s) + \gamma \sum_{\mathbf{a} \in \mathbf{A}} \sum_{s' \in S} T(s, \mathbf{a}, s') \pi^{\mathbf{a}}(s) U^\pi_i(s'), \end{aligned} \tag{2}$$

where $\pi^{\mathbf{a}}(s)$ is the probability of joint action $\mathbf{a}$ in s given joint policy π, s_0 is the initial state of the game and $\gamma \in [0, 1]$ is the discount factor. For other performance criterion, see (Garcia and Rachelson 2008).

The concept that is most commonly used as a solution of non-cooperative stochastic games is the one of Nash equilibrium, i.e. a combination of strategies where each player selects the best response to the other players' strategies, and no player can improve its utility by unilaterally deviating from this strategy. Formally, a joint strategy $\pi^* = \langle \pi^*_i, \pi^*_{-i}\rangle$ is a Nash equilibrium if

$$\forall s \in S,\ \forall i \in Ag,\ \forall \pi_i \in \Pi_i, \qquad U_i^{\langle \pi^*_i, \pi^*_{-i}\rangle}(s) \geq U_i^{\langle \pi_i, \pi^*_{-i}\rangle}(s), \tag{3}$$

where Π_i is the set of policies available to agent i. The existence of at least one Nash equilibrium for 2-player stochastic games has been demonstrated by Shapley (Shapley 1953).

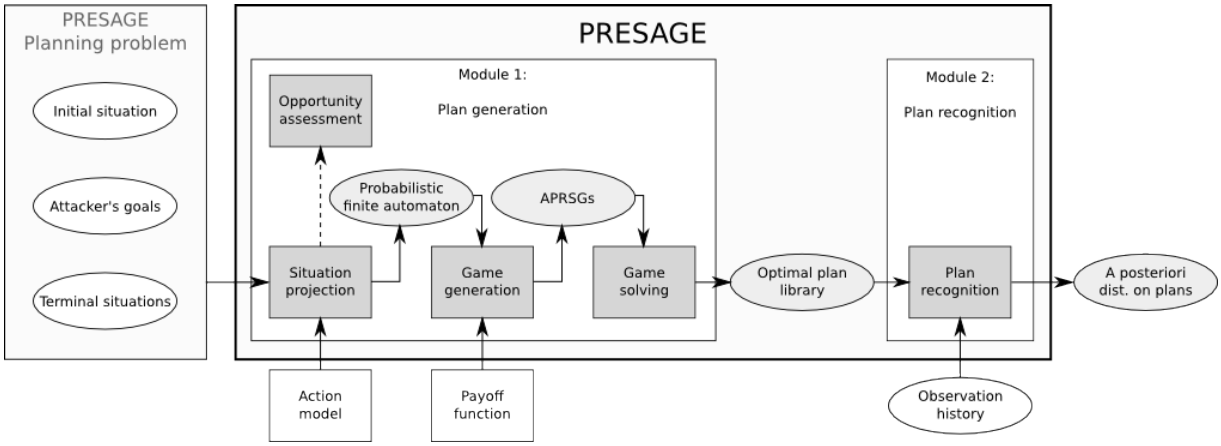


Figure 1: PRESAGE general architecture.

Plan Recognition using Stochastic Games

In this section, we describe PRESAGE (Plan REcognition using StochAstic GamEs), a generative game-theoretic framework for adversarial plan recognition. This framework models the strategic interactions between the decision-making strategies of an attacker (observed agent) and a defender (observer) as Markov games. As any other generative method, our framework requires the definition of an action model. We propose an adaptation of the COI model of action (Steinberg 2007) to the domain of automated planning so that the opportunities (equivalent to MDPs' actions) available to each player can be automatically assessed from the situation and used to generate valid plans. Finally, we design a probabilistic plan recognition method whose aim is to create and maintain a belief state over the set of optimal strategies of the attacker from observations of his behavior.

General Architecture

Figure 1 is a representation of PRESAGE functional architecture. PRESAGE is divided into two independent modules. The first module (Module 1) is dedicated to the automatic generation of the optimal plan library. Starting from the definition of a planning problem P, including the initial situation and the set of possible goals of the attacker, the plan generation module executes a three-step procedure to build the library of possible plans for the attacker:

1. **Situation projection:** first, the set of possible future situations is generated and modeled as a *probabilistic finite automaton* Σ (Stoelinga 2002), whose transitions are labeled by joint opportunities of the attacker and the defender. Opportunities for action available to each actor are assessed in each state of Σ using a generic opportunity assessment engine.
2. **Game generation:** for each possible goal g of the attacker, we build an Adversarial Plan Recognition Stochastic Game (APRSG) Γ_g by considering each state of Σ where g is achieved as a terminal state, and by representing each remaining state as a two-player zero-sum static game using a predefined payoff function.
3. **Game solving:** each APRSG is solved independently of the others using off-the-shelf solving algorithms so as to obtain an optimal plan library Π_P^* containing one optimal strategy for each possible goal of the attacker.

The second module (Module 2) encapsulates a plan recognition engine which, given a plan library and an observation history of both actors' actions, returns an a posteriori distribution over the set of possible plans of the attacker.

Assessing Opportunities for Action

Our action model is an adaptation of the COI model of threat to the planning domain. This model was first proposed in (Steinberg 2005; Little and Rogova 2006) and has been further extended in (Steinberg 2007) to represent any intentional action. It considers the *capability*, *opportunity*, and *intent* of an agent to carry out actions on some targets to be the necessary and sufficient factors to characterize these actions:

Capability the possession of the resources and/or skills required to perform an action.

Opportunity "the right context" to perform an action, i.e. the presence of an operating environment in which potential targets are susceptible to be acted upon.

Intent the plans and goals an agent wants to achieve.

These action components are tightly interrelated. For instance, the intent of an agent can evolve in light of his capabilities and current opportunities, e.g. by changing his target. Inversely, the agent's intent can motivate the acquisition of new capabilities/opportunities, etc. Since our goal is to infer the intent of an agent from observations of his actions as restrained by its capabilities and opportunities, intent is implicit in our model and we postulate that it is constrained by the rationality principle. We also assume that we have full and certain knowledge of the capabilities of each agent. We therefore focus on the opportunity assessment task which consists of determining which action(s) can be performed on which target(s) in the current situation and estimating the expected outcome of these actions. To understand the con-

cept of opportunity assessment, let us consider the example depicted on Figure 2.

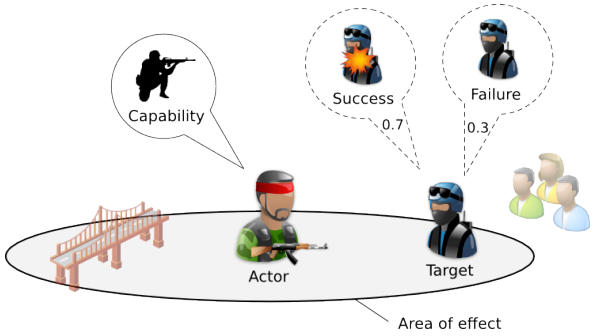


Figure 2: An example of opportunity assessment.

In this situation, the actor owns a resource *assault rifle* which provides him with the capability to perform action *shooting*. However, if the agent wants this action to have an effect on a target t, he must first acquire the opportunity to act upon t. Given a situation s, we will consider that an agent has the opportunity to perform action a on a target t if:

- the agent has the capability to perform action a in s
- the agent has access to target t in s,
- target t is vulnerable to action a in s.

We propose a simple, yet expressive opportunity model that takes into account the three aforementioned conditions and associates probabilities to the possible consequences of an action, namely success or failure. Enhanced action definitions may include more than two outcomes.

Definition 1. *An opportunity o^i for actor i is a tuple $\langle a,t,p\rangle$, where a is an action, t is the target of the action, and p is the probability of success of action a when performed on t.*

Given a situation s, the goal of the opportunity assessment engine is to compute for each actor i, the set $O_i(s)$ of opportunities available to i in s. In our model, each capability (feasible action) is associated with one area of effect (AoE).

Definition 2. *The area of effect associated with a capability a is a tuple $AoE(a) = \langle \mathscr{P}, P_a(success)\rangle$ where $\mathscr{P}$ is a set of preconditions necessary for a target t to be considered accessible (e.g. lying within a spatial area), and $P_a(success)$ is the probability of success of action a.*

Definition 3. *Assuming that actor i has the capability to perform action a, there exists an opportunity o^i for i to perform action a on a target t iff*

1. *t satisfies the accessibility conditions listed in $\mathscr{P}$,*
2. *the probability of success $p > 0$.*

p is determined by both the probability of success of the action $P_a(success)$ and the vulnerability of the target t to action a when t satisfies the accessibility conditions (written $t \in AoE(a)$).

$$\begin{aligned} p \quad = \quad & P(success(a), t \in AoE(a), vulnerable(t,a)) \\ & P_a(success) P(vulnerable(t,a) | success(a), t \in AoE(a)) \end{aligned} \tag{4}$$

The vulnerability of a target to an action may be seen as its ability to thwart some possible effects of this action. For instance, considering the two possible effects of the *shooting* action to be *target killed* (success) and *target missed* (failure), we can assume that a target wearing a bulletproof vest will be less vulnerable to this action, i.e. the probability to obtain effect *target killed* on this particular target will be smaller than with an unprotected target.

Definition 4. *The vulnerability of a target t to an action a is the conditional probability $V_t(a)$, where $V_t(a) = 0$ means that t is not vulnerable to action a and $V_t(a) = 1$ represents the highest level of vulnerability.*

The overall probability of success p is therefore given by

$$p = P_a(success) \times V_t(a) \tag{5}$$

Example 1. *In the situation depicted in Figure 2, the only opportunity for the attacker is to attack the vulnerable ($V_{blueforce}(shooting) = 1.0$) blue force with a probability of success $P_{shooting}(success) = 0.7$; the civilian group cannot be acted upon because it does not lie within the (spatial) AoE, and the attacker does not possess the capability bombbridge required to act upon the bridge. Therefore, we have $O_i(s) = \{\langle shooting, blueforce, 0.7\rangle\}$.*

PRESAGE Planning Problem

A PRESAGE planning problem is defined as a tuple $P = \langle Ag, I, G, \mathscr{T}\rangle$, where Ag is a finite set of actors, I is the initial situation, G is the set of goals of the attacker, and $\mathscr{T}$ is a set of terminal situations. Elements of G are not necessarily elements of $\mathscr{T}$, thus offering the possibility to account for an adversary trying to achieve several goals sequentially.

Situation Projection

The goal of the situation projection engine is, given the definition of a PRESAGE planning problem, to generate a probabilistic finite automaton Σ which aims at formally representing the possible future situations and their dynamics. Σ is defined as a tuple $\langle s_0, S, S^{\mathscr{T}}, \mathbf{O}, T\rangle$. The definitions for the different parameters are compiled in Table 1.

Par.	Description	Expression
S	A finite set of states	$S = \{s_0, ..., s_{|S|}\}$
s_0	The initial state	$s_0 \in S$
$S^{\mathscr{T}}$	A finite set of terminal states	$S^{\mathscr{T}} \subset S$
O_i	A mapping assigning to each $s \in S \setminus S^{\mathscr{T}}$ the set of opportunities for player i in s	$O_i(s) = \{o^i_1, ..., o^i_n\}$ with $o^i_j = (a^i_j, t^i_j, p^i_j)$
O(s)	The set of possible joint opportunities in state $s \in S \setminus S^{\mathscr{T}}$	$\mathbf{O(s)} \equiv \times_{i \in Ag} O_i(s)$
SO	The set of all possible joint opportunity profiles	$SO = \{(s, \mathbf{o}) \mid s \in S \setminus S^{\mathscr{T}}, \mathbf{o} \in \mathbf{O(s)}\}$
T	A transition function	$T : SO \times S \to [0,1]$

Table 1: Definitions for the parameters of Σ

Given a PRESAGE planning problem P and an opportunity assessment engine OA with $\forall s \in S \setminus S^{\mathscr{T}}, i \in Ag$,

$\texttt{OA}(s,i) \rightarrow O_i(s)$, we build Σ using the situation projection algorithm shown in Algorithm 1. The algorithm first checks the terminal conditions (lines 5-6). If the current state s is not terminal, opportunities available to each player in s are assessed using the opportunity assessment engine so as to build the set of possible joint opportunities $\mathbf{O}(\mathbf{s})$ (line 8-10). Line 11 loops over all possible joint opportunities $\mathbf{o}$ in $\mathbf{O}(\mathbf{s})$. Given a state s and a joint opportunity $\mathbf{o}$, the `findSuccessors` function (whose algorithm is not detailed here due to a lack of space) computes the set of successor states S' obtained when executing joint action $\mathbf{o}$ in s, as well as the transition probability $P(s'|s,\mathbf{o})$ for each $s' \in S'$ (line 12). Each successor state s' in S', if not already processed, is added to S and the corresponding transition probability is used to update the transition function T (lines 13-16). The recursive situation projection procedure is applied to each state until no new state can be added to S (line 17).

Algorithm 1: Situation projection algorithm

```
   Data: P = ⟨Ag, I, G, 𝒯⟩, OA
   Result: Σ
1  begin
2      s_0 ← I, S ← {s_0}, S^𝒯 ← {}, T ← {}
3      project (P, s_0, S, S^𝒯, T)
4  procedure project (P, s, S, S^𝒯, T)
5      if s ∈ 𝒯 then
6          S^𝒯 ← S^𝒯 ∪ {s}
7      else
8          foreach i ∈ Ag do
9              O_i(s) ← OA (s, i)
10         O(s) ← ×_{i∈Ag} O_i(s)
11         foreach o ∈ O(s) do
12             S' ← findSuccessors (s, o)
13             foreach s' ∈ S' such as P(s'|s,o) ≠ 0 do
14                 T(s, o, s') ← P(s'|s, o)
15                 if s' ∉ S then
16                     S ← S ∪ {s'}
17                     project (Ag, P, s', S, S^𝒯, T)
```

Stochastic Games Generation and Solving

The projected situation Σ is used as a basis for building one Adversarial Plan Recognition Stochastic Game (APRSG) planner per possible goal of the attacker. An APRSG for goal $g \in G$ is a finite two-player zero-sum stochastic game which is formally defined as a tuple $\Gamma_g = \langle \Sigma_g, \{R_g^i, i = 1,...,|Ag|\}\rangle$ where $\Sigma_g = \langle Ag, S_g, S_g^{\mathscr{T}}, \mathbf{O_g}, T_g\rangle$ and $R_g^i : SO_g \rightarrow \mathbb{R}$ is the payoff function of player i. The only addition to the classical definition of stochastic games (as discussed above) is the set $S_g^{\mathscr{T}} \subset S_g$ of terminal states for goal g.

Players — An APRSG is a two-player game between an attacker (player 1, the observed agent), which tries to maximize his payoff, and a defender (player 2, the observer), which aims at minimizing player 1's payoff. Therefore we have $Ag = \{1,2\}$.

State space and transition function — The game generation procedure is shown in Algorithm 2. First, S_g and $S_g^{\mathscr{T}}$ are initialized as copies of their respective equivalent set in Σ. Then, each state s in S_g corresponding to a situation where goal g is achieved is marked as a terminal state of game Γ_g (lines 3-5). The `pruneDisconnectedStates` function (line 6) acts by removing all the edges originating from terminal states ($\forall s \in S_g^{\mathscr{T}}, \forall \mathbf{o} \in \mathbf{O_g}(\mathbf{s}), \forall s' \in S_g, T(s,\mathbf{o},s') \leftarrow 0$). States in S_g and $S_g^{\mathscr{T}}$, which are no more connected to the initial state s_0 due to the fact that goal states are now considered as terminal, are pruned using depth-first-search algorithm. The state space S_g of game Γ_g is therefore at most as large as the initial state space S. Finally, T_g and $\mathbf{O_g}(\mathbf{s})$ are respectively defined as the restriction of T and $\mathbf{O}(\mathbf{s})$ to the set $S_g \setminus S_g^{\mathscr{T}}$ (lines 7-8). This algorithm is applied for each attacker's goal $g \in G$. A simple example illustrating this game generation procedure is depicted in Figure 3.

Algorithm 2: Game generation algorithm

```
   Data: Σ, g, {R_g^i, i = 1,...,|Ag|}
   Result: Γ_g
1  begin
2      S_g ← copy(S), S_g^𝒯 ← copy(S^𝒯)
3      foreach s ∈ S_g do
4          if g is achieved in s then
5              S_g^𝒯 ← S_g^𝒯 ∪ {s}
6      pruneDisconnectedStates(s_0, S_g, S_g^𝒯, T)
7      T_g ← T_{|S_g \ S_g^𝒯}
8      O_g(s) ← O_{|S_g \ S_g^𝒯}(s)
9      Σ_g ← ⟨Ag, S_g, S_g^𝒯, O_g, T_g⟩
10     Γ_g ← ⟨Σ_g, {R_g^i, i = 1,...,|Ag|}⟩
```

Payoffs and optimal strategy — The payoffs for player i playing game Γ_g are given by the application-dependent payoff function R_g^i which associates to each joint opportunity and in every state of the game, a gain in terms of utility for player i. Each state of an APRSG can be seen locally as a two-player zero-sum static game. Consequently, the payoff function must satisfy the following constraint: $\forall s \in S_g, \forall \mathbf{o} \in \mathbf{O_g}(\mathbf{s}),\ \sum_{i \in Ag} R_g^i(s,\mathbf{o}) = 0$. A two-player zero-sum static game can be solved using the Minimax algorithm (Neumann 1928). Assuming that player 1 wants to maximize its payoff while player 2 wants to minimize player 1's payoff, the optimal strategy for player 1 (resp. player 2) is called the *maximin* (resp. *minimax*) strategy, and an equilibrium is reached if *maximin* = *minimax*. Such an equilibrium always exists in mixed strategy. Let $\mathbf{R_g}(\mathbf{s})$ be a $m \times n$ matrix where m (resp. n) is the number of opportunities available to player 1 (resp. player 2) in s, and $\mathbf{R_g}(\mathbf{s})_{i,j} = R_g^1(s, o_i^1, o_j^2)$, the mixed maximin strategy $x^* = (x_1^*,...,x_m^*)$ in s is obtained by solving the following linear program (Nisan et al. 2007):

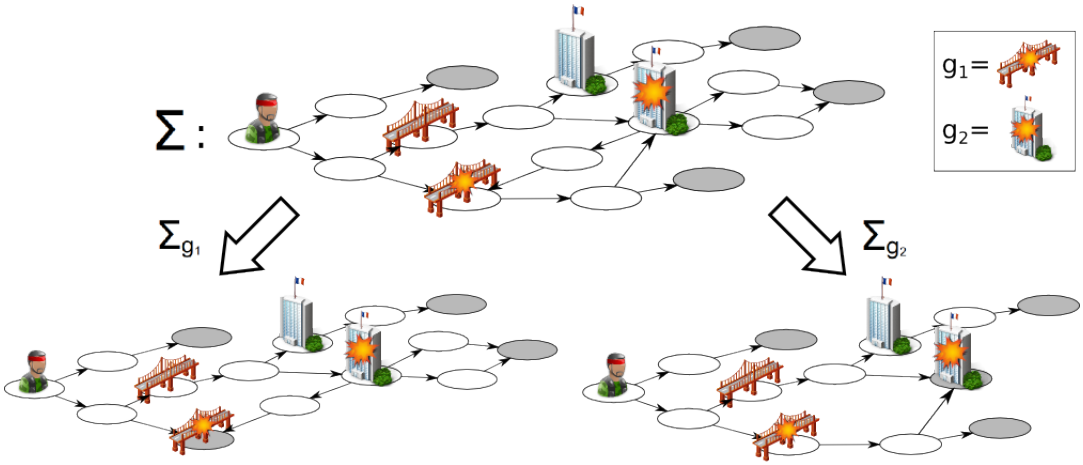


Figure 3: Illustration of the game generation procedure. In this example, the attacker has two possible goals: bombing the bridge (g_1) and/or attacking the embassy (g_2). Terminal states are represented as grey ellipses and do not necessarily corresponds to goal states. By defining terminal situations distinct from goal states, we allow an attacker to achieve a sequence of goals. From the projected situation Σ resulting from the situation projection procedure, the game generation algorithm derives two Markov games Σ_{g_1} and Σ_{g_2} associated to goal g_1 and g_2 respectively, by converting their respective goal states into terminal states and pruning unreachable successor states. The attacker can achieve the sequence of goal (*bombBridge, attackEmbassy*) by executing the optimal strategy π^*_{1,g_1}, then switch to strategy π^*_{1,g_2} once he reach the first goal state.

$$\begin{array}{rl} \textit{maximize} & v \\ \textit{s.t} & \sum_i x_i \, \mathbf{R_g}(\mathbf{s})_{i,j} \geqslant v, \quad for\ 1 \leqslant j \leqslant n \\ \textit{and} & \sum_i x_i = 1. \end{array}$$

The mixed minimax strategy $y^* = (y^*_1, ..., y^*_n)$ in state s is obtained by solving the analogous linear program:

$$\begin{array}{rl} \textit{minimize} & v \\ \textit{s.t} & \sum_i y_i \, \mathbf{R_g}(\mathbf{s})^T_{i,j} \leqslant v, \quad for\ 1 \leqslant j \leqslant m \\ \textit{and} & \sum_i y_i = 1. \end{array}$$

An APRSG Γ_g is played in discrete time steps. In each time step t, each $i \in Ag$ chooses an opportunity o^i_t from his set of available opportunities $O_i(s_t)$, where s_t is the current state of the game. Decisions are simultaneous and the joint opportunity $\mathbf{o_t} \in \mathbf{O_g}(\mathbf{s_t})$ is obtained. Each player i received a reward $R^i_g(s_t, \mathbf{o_t})$ and the players are transferred to the next state s_{t+1}. A policy $\pi_{i,g} : SO_g \to [0,1]$ for agent i defines for each state of Γ_g a mixed strategy, i.e. a probability distribution over the set of available opportunities.

We use Shapley's algorithm (Shapley 1953), an extension of the Value-Iteration algorithm to the case of stochastic games, to find one Nash equilibrium $\pi^*_g = (\pi^*_{1,g}, \pi^*_{2,g})$ in each Γ_g. Shapley's algorithm iteratively builds a normal form game $M(s)$ for each state $s \in S_g$ by using a value function which accounts for both the immediate utility of choosing an action in s and the long-term utility of playing the equilibrium starting from the next state. The optimal joint strategy $\pi^*(s)$ is then obtained by calculating the maximin and minimax strategies for the two-player zero-sum static game $M(s)$. Once a Nash equilibrium π^*_g is known for each Γ_g, we can finally build the set $\Pi^*_P = \{\pi^*_{1,\,g} | \forall g \in G\}$ of optimal plans for the attacker and the set $br(\Pi^*_P) = \{\pi^*_{2,\,g} | \forall g \in G\}$ of best responses for the defender given problem P.

Plan Recognition

The aim of the plan recognition module is to infer the current intent (goal and plan) of the attacker from observations of his actions. The past behaviors of both the attacker and the defender are represented as an observation history H_h which keeps record of the last h states visited by the players and of the joint opportunities that were played in these states: $H_h(t) = \{(s_{t-(h-1)}, \mathbf{o_{t-(h-1)}}), ..., (s_t, \mathbf{o_t})\}$.

According to the rationality principle underlying the generative plan recognition paradigm, if the attacker intends to reach the desired end-state $g \in G$, then he will follow the optimal policy $\pi^*_g \in \Pi^*_P$ to achieve g. Our plan recognition procedure acts by creating and updating at each time step t a belief state defined over the set Π^*_P of optimal plans for the attacker. This belief state is represented by a belief function $b_t : \Pi^*_P \longrightarrow [0,1]$ with $b_t(\pi^*_g) = P\left(\pi_t = \pi^*_g | H_h(t), s_t\right)$, where π_t is the policy of the attacker at time t. Hence $b_t(\pi^*_g)$ represents the belief of the defender that, at time t, the attacker is following policy $\pi^*_g \in \Pi^*_P$ given the observation history $H_h(t)$.

Proposition 1. *Let o_t be the opportunity played by the attacker in state s_t. Given history $H_h(t)$, the belief that the attacker is following policy π^*_g at t is given by:*

$$b_t(\pi^*_g) \propto P(o_t | \pi^*_g, s_t) \times \prod_{i=1}^{h-1} P(o_{t-i} | \pi^*_g, s_{t-i}) T_g(s_{t-i}, \mathbf{o_{t-i}}, s_{t-i+1}),$$

with the constraint that b_t is a probability distribution over Π^*_P.

Proof. From Bayes' rule:

$$P\left(\pi_g^*|H_h(t),s_t\right) = P\left(H_h(t)|\pi_g^*,s_t\right)P\left(\pi_g^*,s_t\right)/P\left(H_h(t),s_t\right),$$

Let $H_n = H_{h-n}(t)$ (i.e. the last $h-n$ observations at t), and $\tau = t-(h-1)$. The expression for $P\left(H_h(t)|\pi_g^*,s_t\right)$ results from the following recursive decomposition

$$P\left(H_n|\pi_g^*,s_{\tau+n}\right) = P\left(o_{\tau+n}|\pi_g^*,s_{\tau+n}\right) T_g\left(s_{\tau+n},\mathbf{o}_{\tau+\mathbf{n}},s_{\tau+(n+1)}\right)P\left(H_{n+1}|\pi_g^*,s_{\tau+(n+1)}\right),$$

with $H_{n+1} = H_n \setminus \{(s_{\tau+n},\mathbf{o}_{\tau+\mathbf{n}})\}$. Therefore, starting from state $s_{t-(h-1)}$ ($n=0$), we obtain

$$P\left(H_h(t)|\pi_g^*,s_t\right) = P(o_t|\pi_g^*,s_t)\times\prod_{i=1}^{h-1}P(o_{t-i}|\pi_g^*,s_{t-i}) T_g(s_{t-i},\mathbf{o_{t-i}},s_{t-i+1}).$$

□

The use of a finite length history allows us to handle the fact that the adversary may change his intent during the execution of the plan recognition procedure. By keeping only "up-to-date" information, we prevent old observations inconsistent with the new intent from deteriorating the solution of the inference process. Another important remark is that the plan recognition algorithm introduced in Proposition 1 does not require the plan library to contain optimal strategies, but can be used to recognize any type of plan, with the constraint however that these plans are represented as MDP policies.

Experimental results

Scenario description

We perform our experiments on the scenario depicted in Figure 4. In this scenario, the defender is the Blue force (bf) with capabilities "move to", "attack red force", and "do nothing", and the attacker is the Red force (rf) with capabilities "move to", "attack embassy", "attack civilians", "attack blue force" and "do nothing". Each capability is associated with an area of effect specifying the type of target which can be acted upon, the action range, and the probability of success of the action. The set of possible goals of the attacker is $G = \{g_i = destroyed(target_i)|i = 1...6\}$. Of course, the true goal of the attacker is hidden to the defender. The mission of the defender is to determine which target has been chosen and to protect this target by eliminating the attacker. In this particular set of experiments, we assume that the attacker has only one single goal (hence he will not try to achieve several goals sequentially) and therefore the set of terminal situations is $\mathscr{T} = G \cup \{destroyed(rf), destroyed(bf)\}$. We also define a simple application-dependent payoff function $R_g^i : S \to \mathbb{R}$ which associates immediate rewards to each state of a game given a goal $g \in G$:

$$R_g^i(s) = \begin{cases} = & 200, & \text{if } s \models g, \\ = & 50, & \text{if } s \models destroyed(bf), \\ = & -100, & \text{if } s \models destroyed(rf), \\ = & \alpha d(rf,target) - \beta d(rf,bf), & \text{otherwise,} \end{cases}$$

with $d(rf,target)$ the distance between the attacker and his target, $d(rf,bf)$ the distance between the attacker and the defender, and $\alpha+\beta = 1$.

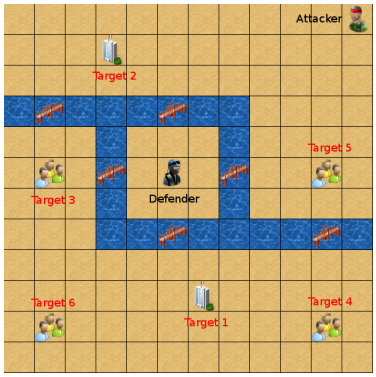


Figure 4: Initial situation of our urban warfare scenario.

Experiments We use the scenario defined in the previous section to evaluate the ability of our generative game theoretic framework to recognize the current plan of a rational adversary. We also verify that the recognition of the attacker's plan is rapid enough for the defender to select and implements the appropriate response before the attacker reaches his goal. With this in mind, we define two possible strategy selection policies for the defender. Let $b_t(\pi_g^*)$ be the belief state of the defender at time t:

WAIT : the defender waits until $\exists g \in G$ such as $b_t(\pi_g^*) = 1$, then executes the best response $br(\pi_g^*)$.

MAX : the defender executes the best response $br(\pi_g^*)$ with $g = argmax_{g' \in G}\ b_t(\pi_{g'}^*)$, and chooses randomly between goals with the same maximum likelihood.

Figure 5 shows the belief state $b_t(\pi_g^*)$ for $g = destroyed(target_6)$ as a function of time. As we can see from this plot, the true goal of the attacker is correctly recognized by our plan recognition algorithm at $t = 16$. This is not a surprise: since the attacker is assumed to be strictly rational, we are guaranteed to recognize his real goal in finite time (in the worst case when the attacker executes the final action leading to the achievement of the goal), unless a terminal situation such as $destroyed(bf)$ or $destroyed(rf)$ is achieved before the true goal has been recognized. The real question is therefore to know if our plan recognition engine is able to recognize the real goal of the attacker before it is fulfilled.

Policy	Goal 1	Goal 2	Goal 3	Goal 4	Goal 5	Goal 6
MAX (%)	98.9 ± 1.0	75.5 ± 2.6	100	80.4 ± 3.8	89.2 ± 3.8	100
WAIT (%)	100	42.1 ± 4.5	100	17.6 ± 2.7	100 ± 0	100

Table 2: Mean percentage of instances during which the real goal of the attacker was recognized before it was achieved.

Table 2 contains the results of the evaluation of the ability of our plan recognition method to infer the real goal of the adversary before he achieves this goal. For instance, when the true goal of the adversary is goal 1 and the strategy selection policy of the defender is **MAX**, our approach is able to recognize the goal of the attacker before the destruction of the target in 98.9% of cases (mean over 100 runs of 100

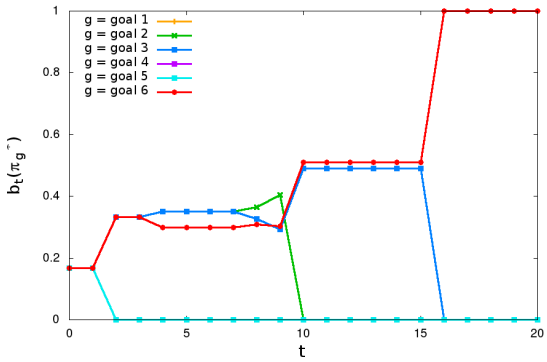


Figure 5: Evolution of the defender's belief about the real goal of the attacker.

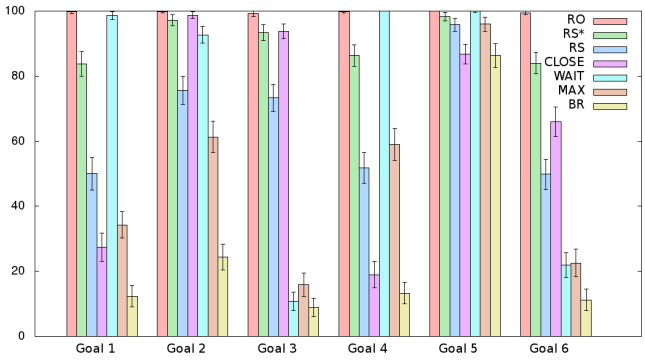


Figure 6: Mean attacker's percentage of success when using **MAX** and **WAIT**, compared to **BR** and other baselines.

scenario instances). Our method performs better on average when associated to the **MAX** selection policy, since it seems to lead the attacker in being less unclear about the goal he is chasing. This is particularly true for ambiguous goals such as goal 2 (whose optimal policy is very similar to those of goals 3 and 6) and goal 4 (similar to goal 5 and 1).

From Table 2, we saw that our plan recognition method is able to infer the real goal of an adversary before the achievement of this goal in most cases. But does this inference occurs soon enough for the defender to implement the appropriate response and defend the target? To evaluate this point, we define several strategy formulation policies which will serve as baseline values for comparison with the **WAIT** and **MAX** policies defined above:

RO : the defender selects an opportunity randomly in each state of the game (uniform distribution).

RS* : the defender selects a goal g randomly in each state of the game and executes the best response $br(\pi_g^*)$.

RS : the defender selects a goal g randomly at the beginning of the game and executes the best response $br(\pi_g^*)$.

CLOSE : the defender selects the goal g which is closest from the attacker in each state of the game and executes the best response $br(\pi_g^*)$.

BR : the defender always executes the best response $br(\pi_g^*)$ where g is the true goal of the attacker.

The quality of each policy is evaluated by averaging the number of scenario instances during which the attacker successfully achieved his goal over a total of 100 runs of 100 instances for each goal. From the results depicted in Figure 6, we can see that the **MAX** strategy formulation policy, which relies on our plan recognition engine, performs better on average than every other baseline (except **BR** of course). In a few cases however (goal 3 and 6), it may be preferable for the defender to wait to be certain about the real goal of the attacker before acting. But the disappointing results of the **WAIT** policy on the other 4 goals tend to confirm this famous quote from Prussian general and military theorist Carl Von Clausewitz

> The greatest enemy of a good plan is the dream of a perfect plan (Clausewitz 1832).

Conclusion and Future Work

In this paper, we presented PRESAGE, a generative game theoretic framework for adversarial plan recognition in stochastic multi-agent environments. This framework generates a library of optimal plans for a rational attacker from the definition of a planning problem. We proposed an algorithm for situation projection relying on a simple yet expressive contextual action model. We showed how a set of Markov games can be generated automatically from this projected situation to model the planning processes of both the defender and the attacker. By computing one Nash equilibrium for each one of these games, we built a library of optimal plans for the attacker and a set of best responses for the defender. This library was later exploited by a probabilistic plan recognition algorithm to infer the current plan of the adversary from observations of his behavior. Finally, we demonstrated the capability of our adversarial plan recognition to assist a decision-maker in selecting the most appropriate response to a given threat.

An interesting direction for future works would be to relax the attacker's rationality assumption and to adapt our plan recognition algorithm to the case of an adversary with bounded rationality. We also plan to extend our framework in order to deal with an adversary which would be actively hostile to the inference of his plans. This would require the ability to detect deceptive actions performed by the attacker. We would also have to relax the assumption of full observability of the opponent's actions, since he may use concealment. Currently, our APRSGs are defined as two-player zero-sum stochastic games and therefore, they can only model the strategic interactions between one defender and one attacker. Our intuition is that our framework can be generalized quite directly to the 1 vs N and N vs N cases.

Acknowledgments

We thank Bruno Zanuttini, assistant professor with habilitation at the University of Caen Basse-Normandie, for helpful comments and corrections.

References

Avrahami-Zilberbrand, D., and Kaminka, G. A. 2007. Incorporating observer biases in keyhole plan recognition (efficiently!). In *AAAI*, volume 7, 944–949.

Baker, C. L.; Saxe, R.; and Tenenbaum, J. B. 2009. Action understanding as inverse planning. *Cognition* 113(3):329–349.

Braynov, S. 2006. Adversarial planning and plan recognition: Two sides of the same coin. In *Secure Knowledge Management Workshop*.

Burkov, A., and Chaib-draa, B. 2008. Stochastic games. In *Markov Decision Processes in Artificial Intelligence*, volume 1. Wiley Online Library. 229–276.

Chen, G.; Shen, D.; Kwan, C.; Jr., J. B. C.; Kruger, M.; and Blasch, E. 2007. Game theoretic approach to threat prediction and situation awareness. *Journal of Advances in Information Fusion* 2(1):35–48.

Clausewitz, C. v. 1832. *On War*. Ed./trans. Michael Howard and Peter Paret. Princeton University Press, 1976, revised 1984.

Garcia, F., and Rachelson, E. 2008. MDPs: models and methods. In *Markov Decision Processes in Artificial Intelligence*, volume 1. Wiley Online Library. 1–38.

Geib, C. W., and Goldman, R. P. 2001. Plan recognition in intrusion detection systems. In *Proceedings of DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01.*, volume 1, 46–55. IEEE.

Geib, C. W., and Goldman, R. P. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence* 173(11):1101–1132.

Kott, A., and McEneaney, W. M. 2006. *Adversarial reasoning: computational approaches to reading the opponents mind*. CRC Press.

Lisỳ, V.; Píbil, R.; Stiborek, J.; Bosanskỳ, B.; and Pechoucek, M. 2012. Game-theoretic approach to adversarial plan recognition. In *ECAI*, 546–551.

Little, E. G., and Rogova, G. L. 2006. An ontological analysis of threat and vulnerability. In *9th International Conference on Information Fusion*, 1–8. IEEE.

Neumann, J. v. 1928. Zur theorie der gesellschaftsspiele. *Mathematische Annalen* 100(1):295–320.

Nisan, N.; Roughgarden, T.; Tardos, E.; and Vazirani, V. V. 2007. *Algorithmic game theory*, volume 1. Cambridge University Press Cambridge.

Ramırez, M., and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the 21st international joint conference on Artifical intelligence. Morgan Kaufmann Publishers Inc*, 1778–1783.

Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*.

Ramırez, M., and Geffner, H. 2011. Goal recognition over pomdps: Inferring the intention of a pomdp agent. In *IJCAI*, 2009–2014.

Shapley, L. S. 1953. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America* 39(10):1095.

Steinberg, A. N. 2005. An approach to threat assessment. In *8th International Conference on Information Fusion*, 1–8. IEEE.

Steinberg, A. N. 2007. Predictive modeling of interacting agents. In *10th International Conference on Information Fusion*, 1–6. IEEE.

Stoelinga, M. 2002. An introduction to probabilistic automata. *Bulletin of the EATCS* 78(176-198):2.

Combining off-line Multi-Agent Planning with a Multi-Agent System Development Framework

Rafael C. Cardoso and **Rafael H. Bordini**
FACIN-PUCRS
Porto Alegre - RS, Brazil
{rafael.caue@acad.pucrs.br, rafael.bordini@pucrs.br}

Abstract

Automated planning is an important capability to have in multi-agent systems. Extensive research has been done for single-agents, but so far it has not been fully explored in multi-agent systems mainly because of the computational costs of multi-agent planners. With the increasing availability of distributed systems, and more recently multi-core processors, there have been several novel multi-agent planning algorithms developed, such as the MAP-POP algorithm, which in this work we integrate with the JaCaMo multi-agent system framework. Our work provides off-line multi-agent planning capabilities as part of a multi-agent system development framework that supports the development of systems for complex multi-agent problems. In summary, our approach is to provide to developers an initial multi-agent system implementation for the target scenario, based on solutions found by the MAP-POP multi-agent planner, and on which the developer can work further towards a fully-fledged multi-agent system.

1 Introduction

In this paper, we provide an approach to combine off-line Multi-Agent Planning (MAP) algorithms with a Multi-Agent Systems (MAS) platform. Our work serves two purposes: it provides an execution stage for off-line MAP, and it provides an initial MAS on which developers can further work. This process is done with a translator, which receives as input the problem instances and the solution found by the multi-agent planner and generates a MAS program as output.

Automated planning is an interesting and desirable capability to have in intelligent agents and MAS, which so far has not been fully explored because of the computational costs of MAP algorithms (Jonsson and Rovatsos 2011; Crosby, Jonsson, and Rovatsos 2014). Recent algorithms have managed to improve performance, which was one of the main incentives for pursuing this topic.

Although there is an increase in interest in theoretical research on multi-agent planning, as evidenced in (Witwicki and Durfee 2011; Jr. and Durfee 2011; Planken, de Weerdt, and Witteveen 2010), current implemented multi-agent planning algorithms and planners are mostly application specific, such as in (Mao et al. 2007; van Leeuwen and Witteveen 2009).

A distributed multi-agent planning problem involves the development and execution of joint plans through cooperation (agents on the same team) and competition (agents on opposing teams) without centralised control. These off-line planning algorithms normally stop at the planning stage, providing a solution but with no means of executing it.

The term multi-agent planning has been used in a variety of contexts through the years, and as such, its concept can mean widely different things. For the purposes of this paper, we use the multi-agent planning definition found in (Durfee and Zilberstein 2013), which states that the planning process itself is multi-agent (i.e. multiple agents cooperatively generate plans), and the solution can be distributed across and acted upon by multiple agents. In other words, multi-agent planning by multiple agents and multi-agent planning for multiple agents.

We use the Multi-Agent Planning based on Partial-Order Planning (MAP-POP) (Lerma 2011; Torreño, Onaindia, and Sapena 2014a; 2014b; Sapena, Onaindia, and Torreño 2015) as an example of a multi-agent planner, and provide a basic grammar for it. This grammar is then used in the translation algorithms that we describe. These algorithms can be easily adapted to work with the grammars of other multi-agent planners.

For the MAS development platform we use the JaCaMo framework (Boissier et al. 2011). JaCaMo is composed of three technologies, each representing a different abstraction level that is required for the development of sophisticated MAS. JaCaMo is the combination of Jason, CArtAgO, and Moise, each of these technologies are responsible for a different programming dimension. Jason is used for programming the agent level, CArtAgO is responsible for the environment level, and Moise for the organisation level.

The translator takes as input the problem instances and the solution provided by a multi-agent planner, MAP-POP in this paper. As output, the translator generates a coordination scheme in Moise, followed by the respective agent plans in Jason, and CArtAgO artefacts for organisation control and environment representation. All of these come together to form an initial multi-agent system in JaCaMo.

Our goal with this work was to check if current general-purpose MAP algorithms could be easily integrated with a MAS framework in order to execute the solution, and if the resulting MAS in JaCaMo was complex enough to be of any help to developers. Our results are encouraging, the coordination aspect that is needed for the distributed planning stages of a MAP algorithm is a natural fit for the specification of a Moise organisation. The plans generated in Jason are parsed from the solution presented by each algorithm, and generally were a simple conversion from a step to a plan, maintaining its pre-conditions and effects.

The rest of this paper is structured as follows. In Section 2 we cover the background. Section 3 provides some of the related work. In Section 4 we describe the grammar that we made for the input and the solution of MAP-POP and the translation algorithms. Section 5 presents two case studies and we conclude in Section 6.

2 Background

It is common for MAP algorithms to use and improve upon single-agent planning techniques, as single-agent planning has been extensively researched over the years and has also been the focus of several International Planning Competitions (IPC). As a consequence, single-agent planners generally have an excellent performance. Usually, in distributed MAP algorithms, agents plan locally using adaptations of single-agent planners, which means that at some point they will need to exchange information to be able to arrive at a global solution plan. Therefore in order to properly exchange information the agents need some kind of coordination mechanism.

The input of a MAP algorithm refers to instances of the formalism chosen to represent MAP planning problems. Similarly to single-agent planners, problem formalisms have also been extensively researched over the years. PDDL for example has been the standard formalism to represent single-agent problems for quite some time, and it can be easily adapted to comply with the needs that arise when dealing with multi-agent planning problems. The output of a MAP algorithm is the solution it generates, and contrary to the input, the output does not have any standard representation. However, as we are dealing with multi-agent plans that can cause interference with each other, coordination constraints are needed to guarantee that during the execution stage the partial plans will be executed in the correct order so as to achieve the global goal.

The MAP-POP (Lerma 2011) planner builds upon the concept of refinement planning, where agents propose successive refinements to a base plan until a solution is obtained. It uses the PDDL 3.1 formalism with some ad-hoc adaptations to make it work with their multi-agent planner. MAP-POP is based on partial-order planning, establishing partial order relations between the actions in the plan.

The MAP-POP algorithm starts with an initial communication stage in which the agents exchange some information on the planning domain, in order to generate data structures that will be useful in the subsequent planning process. The next step comprises of two different stages that are interleaved, they repeat themselves until a solution plan is found:

- **an internal planning process**, through which the agents refine the current base plan individually with an internal POP system. In order to guide the search the SUM heuristic is applied, it is based on the sum of the costs of the open goals found in the initial communication stage.
- **and a coordination process**, that allows agents to exchange the refinement plans that were generated in the previous stage and to select the next base plan, using the SUM heuristic to estimate the quality of a refinement. A leadership baton is passed among the agents, following a round-robin order. If the current base plan does not have any open goals it is a solution, and if not the baton agent selects the next most costly open goal to be solved. With a new subgoal to be solved, the next internal planning stage starts.

According to (Wooldridge 2002), an agent is a computer system that is capable of autonomous action in the environment that it is situated in order to meet its objectives. In other words, agents receive perceptions through sensors in the environment, and respond to these events with actions that affect the environment. Systems that require the use of the Agent Model will seldom need only a single-agent. Albeit obvious, a MAS then is composed of multiple agents.

Many agent-oriented programming languages have been developed over the years. Some examples of these include Jason (Bordini, Wooldridge, and Hübner 2007), JACK (Busetta et al. 1999), 2APL (Dastani 2008), GOAL (Hindriks et al. 2000), and more recently ALOO (Ricci 2014).

Several studies indicate that Jason has an excellent performance when compared with other agent-oriented programming languages. For example, Jason is included in a qualitative comparison of features alongside with Erlang and Java (Jordan et al. 2011); in a universal criteria catalog for agent development artefacts (Braubach, Pokahr, and Lamersdorf 2008); in a quantitative comparison between Jason and two actor-oriented programming languages (Erlang and Scala) using a communication benchmark (Cardoso, Hübner, and Bordini 2013); and finally a performance evaluation of several benchmarks between agent programming languages (Jason, 2APL, and GOAL) and actor programming languages (Erlang, Akka, and ActorFoundry) (Cardoso et al. 2013). In those cases where performance was considered, Jason typically showed excellent results.

A JaCaMo[1] (Boissier et al. 2011) MAS (i.e. a software system programmed in JaCaMo) is defined by an agent organisation programmed in Moise, responsible for the organisation of autonomous agents programmed in Jason. Those agents work in a shared distributed artefact-based environment programmed in CArtAgO. JaCaMo integrates these three platforms by defining a semantic link among concepts of the different programming dimensions (agent, environment, and organisation) at the meta-model and programming levels, in order to obtain a uniform and consistent programming model that simplifies the combination of those dimensions for the development of MAS.

[1] `http://jacamo.sourceforge.net/`.

Jason (Bordini, Wooldridge, and Hübner 2007) focuses on the agent programming level, it is a programming language for the development of MAS based on the BDI (Belief-Desire-Intention) model, inspired by the AgentSpeak language (Rao 1996). In Jason an agent is an entity composed of a set of beliefs — agent's current state and knowledge about the environment in which it is situated; a set of goals — tasks the agent has to achieve; a set of intentions — tasks the agent is committed to achieve; and a set of plans — courses of actions triggered by events (can be related to changes in either the agent's belief base or its goals).

CArtAgO (Ricci et al. 2009) is a framework and infrastructure for environment programming and execution in multi-agent systems. In CArtAgO the environment is used as a first-class abstraction for designing MAS, a computational layer encapsulating functionalities and services that agents can explore at runtime. These software environments can be designed and programmed as a dynamic set of computational entities called artefacts, that are collected into several workspaces, possibly distributed among various nodes of a network.

Finally, the Moise (Hübner, Sichman, and Boissier 2007) model is used to program the organisational dimension. This approach includes an organisation modelling language, an organisation management infrastructure, and support for organisation-based reasoning mechanisms at the agent level. The organisation model is divided into three layers: the structural specification, where the groups, roles, and links between roles are specified; the functional specification, where the schemas are specified, containing a group of goals and missions, along with information on which goals will be executed in parallel and which will be executed in sequence; and the normative specification, where obligations and permissions towards certain missions are assigned to certain roles.

3 Related Work

A survey (Meneguzzi and De Silva 2013) presents a collection of recent techniques used to integrate automated planning in BDI-based agent-oriented programming languages. It focuses mostly on efforts to generate new plans at runtime, while as with our work we translate the output of MAP algorithms into a MAS that is then able to execute the solution plan, i.e. the MAP algorithms are not involved during runtime. There are at least two other surveys on multi-agent planning, they can be found in (Weerdt, Mors, and Witteveen 2005; de Weerdt and Clement 2009).

In (Mao et al. 2007), decommitment penalties and a Vickrey auction mechanism are proposed to solve a multi-agent planning problem in the context of an airport — deicing and anti-icing aircrafts during winter — where the agents are self-interested and often have conflicting interests. The experiments showed that the former ensures a fairer distribution of delay, while the latter respects the preferences of the individual agents. Both mechanisms outperformed a first come, first served mechanism, but were specifically tailored to the airport problem.

CANPLAN2 (Sardiña and Padgham 2007) is a BDI-based formal language that incorporate an HTN planning mechanism. This approach was further extended in (Sardiña and Padgham 2011) to address previous limitations such as failure handling, declarative goals, and lookahead planning. It is important to note that the CAN family are not implemented programming languages, although its features could be used to augment some BDI-based Agent Oriented Programming (AOP) languages.

The TAEMS framework (Decker 1996) provides a modelling language for describing task structures — the tasks that the agents may perform. Such structures are represented by graphs, containing goals and sub-goals that can be achieved, along with methods required to achieve them. Each agent has its own graph, and tasks can be shared between graphs, creating relationships where negotiation or coordination may be of use. Coordination in TAEMS is identified using the language's syntax, and then the developer choose or create an ad-hoc coordination mechanism by using the commitment constructs that are available. The TAEMS framework does no explicit planning, its focus is on coordinating tasks of agents where specific deadlines may be required. Its development has been discontinued since 2006.

In (Clement, Durfee, and Barrett 2007), multi-agent planning algorithms and heuristics are proposed to exploit summary information during the coordination stage in order to speed up planning. The key idea is to annotate each abstract operator with summary information about all of its potential needs and effects. That often resulted in an exponential reduction in planning time compared to a flat representation. This approach depends on some specific conditions and assumptions, and therefore cannot be used in all domains.

4 Combining MAP with MAS

In order to allow the JaCaMo framework to execute the solution generated by the MAP algorithm, we define a grammar for MAP-POP and a set of algorithms for a translator. The translator is used to help bridge the planning and execution stages of multi-agent planning problems. Off-line MAP algorithms usually ignore the execution stage of planning, ending up with just a set of plans that has to be implemented by the user. On the other hand, we have AOP languages and MAS development frameworks that usually have some kind of planning capabilities available during runtime (online), but provide no sophisticated way to solve complex multi-agent planning problems.

The translator needs as input the definition of a multi-agent planning problem and the solution for the problem found by a MAP algorithm. It then provides as output a MAS specified in JaCaMo that is able to execute the solution found during the planning stage. If the MAP algorithm being used during the planning stage also supports single-agent planning, then the translator should still be able to provide a valid output, but it will not use all of the abstraction levels that JaCaMo provides, such as Moise organisations, which are not necessary in single-agent systems.

A standard input would be ideal for the translation process, but in this case it means that we would need to change the source code of the MAP algorithms. If we develop a standard input, each new algorithm would need to be adapted to accept this new input, while if we choose to use the inputs of

the MAP algorithms, we then have to adapt the grammar and algorithms of the translator to accept them. Unfortunately, at the time of writing there is no standard formalism for the representation of multi-agent planning problems that is accepted by the MAP community, therefore we chose to adapt the translator to accept multiple inputs.

The input depends on which MAP algorithm is used, as each multi-agent planner usually makes its own adaptations to a planning problem formalism. The MAP-POP algorithm uses its own extension of PDDL 3.1 for multi-agent planning. We use this input from the PDDL files of the MAP-POP algorithm to define the name of the agents in the JaCaMo project file and the roles in the Moise organisation. We use the PDDL problem file to build CArtAgO artefacts that represent the initial state of the environment.

The output of MAP algorithms consists of a solution that solves the global goal of the problem. The MAP-POP algorithm requires coordination constraints alongside the actions in order to establish the partial order in which the actions should be executed. This resulted in MAP-POP providing a solution that allows the organisation in Moise to use the coordination constraints in order to construct a MAS with parallel execution of plans.

Both the input and output of the MAP algorithm are given as input for the translator. The translator then generates a MAS specified in JaCaMo, containing: JaCaMo project file, Jason agent's files, Moise XML specifications, and Java codes for the CArtAgO artefacts. This standard output provides generic classes that can be used to integrate new MAP algorithms. In order to integrate new MAP algorithms, one would have to develop input and solution grammars (similar to what we made for MAP-POP), and simply adapt our translation algorithms accordingly.

A summary of how the translator works is available in the diagram of Figure 1. The solution provided by the MAP algorithm is translated into AgentSpeak plans, and added to the respective agent's plan library in Jason. Because plan representation and action theory in Jason differs from the basics of the planning formalisms used by the MAP algorithms (STRIPS-like), we had to use simple transitions, that is, every action in the solution would translate to a plan in Jason with the preconditions at the context, and the effects of the action at the body of the plan. After executing the action, the effects will change the environment, i.e. the CArtAgO artefacts.

Due to space constraints, in this paper we show only the grammar of the problem file for MAP-POP. For the full grammar, all the algorithms, the domain, problem, solution, and resulting MAS of the case studies present in the next section check `http://bit.ly/1DFqveG`.

In Listing 1, we present a simplified BNF grammar based on the official PDDL 3.1 definition, which can be found in `http://bit.ly/1BRcTC8`. Each single quote pair encloses a string that is expected to appear in the file, brackets are optional, and the rest are non-terminal symbols. For example the non-terminal symbol `name` represents a terminal string of characters $a..z|A..Z$.

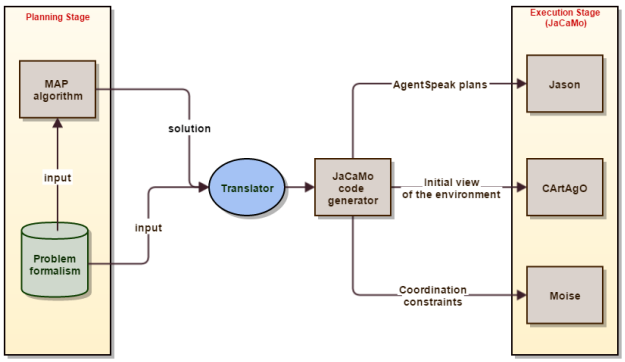


Figure 1: An overview of the translation process.

Listing 1: Initial lines from the grammar for the problem file.

```
problem ::= '((define (problem' name ')'
              '(:domain name ')'
               objectsDef
               [sharedData]
               initDef
               globalGoals ')' ;

objectsDef ::= '(:objects' typedList+ ')' ;

sharedData ::= '(:shared-data' pf+ '- (
    either' name+ ') )' ;

pf ::= predicate | func ;

initDef ::= '(:init' literal* ')' ;

literal ::= term | '(not' term ')' ;

term ::= ( '(' litName first* name* ')' ) |
    ( '(= (' litName first ')' name ')' ) ;

litName ::= name ;

first ::= name ;

globalGoals ::='(:global-goal (and' literal*
     ') )' ;
```

Similarly, for the translation we show only the main translation algorithm in Algorithm 1. The translation function receives as parameters the information contained in the domain, problem, and solution files, which are in accordance with their respective grammar. For example, the notation in `DomainSpec.domain.typesDef.typedList` means that we look in the domain information and inside typesDef for any typedList, as specified in the domain grammar. The translation starts by getting the agent types from the PDDL domain file, and the agents names from the PDDL problem file. With this information it then calls the rest of the algorithms, starting with the algorithm for the translation of the organisation, the algorithm for the plans in Jason, and finally returning and calling the algorithm for the CArtAgO artefacts.

To demonstrate part of the translation process consider

Algorithm 1 Main translation algorithm.

```
1: function TRANSLATE(DomainSpec, ProblemSpec, SolutionSpec)
2:     for (n:name, t:type) in DomainSpec.domain.typesDef.typedList do
3:         if t = 'agent' then
4:             agentsTypes ← agentsTypes ∪ n
5:         end if
6:     end for
7:     for t1 in agentsTypes do
8:         for (n:name, t2:type) in ProblemSpec.problem.objectsDef.typedList do
9:             if t1 = t2 then
10:                agents ← agents ∪ (n, t2)
11:            end if
12:        end for
13:    end for
14:    organisation ← organisation ∪ createOrg(SolutionSpec, agentsTypes, agents)
15:    agentCode ← agentCode ∪ createAgentCode(SolutionSpec, agents)
16:    artefacts ← artefacts ∪ createEnv(DomainSpec, ProblemSpec)
17:    return (agents ∪ organisation ∪ agentCode ∪ artefacts)
18: end function
```

Listing 2 and Listing 3, a step (action) from the solution and its translation to a plan in Jason. The parameters from the step of the solution are used in the context of the resulting plan in Jason — these parameters are used to access and update the artefacts, and are also used to check preconditions. The context (note that the context of a plan starts after the colon) contains the information to access the necessary artefacts, all subsequent lines are each a precondition specified in that step of the solution. Preconditions that involves only predicates pertaining the agent that is responsible for executing that plan can be checked directly in that agent's belief base. The remaining preconditions access the artefacts and make the necessary tests.

Finally, at the body of a Jason plan (the body starts after the left arrow), the effects of the step are translated into Jason actions. The translation can generate two types of actions: an action that can change the belief base of the agent that is running that action — this happens if the predicate in question involves only that same agent; or an action can result in a change in the environment — i.e. an update to observable properties of the artefacts that represent the environment.

A simple print mechanism is added using the syntax for detecting plan failure in Jason, `-!`, that provides basic feedback on which plans failed. If a plan fails and it has any subsequent dependent plans in the Moise organisation schema, the organisation will prevent the execution of those plans as the previous goals were not achieved. If there were no errors during the translation process, then these plans should never fail. However, they may fail because of two different reasons: new plans were added or translated plans were edited by the developer; or there may be other agents that may cause some kind of interference during execution, resulting in plan failure. Regardless, the mechanism for handling plan failure is present only to inform the user of the failure, it is not possible for the translator to call for replanning mechanisms as the MAP algorithms do not have any kind of interaction with the execution stage, this is part of future work.

Listing 2: A step from the solution of a driverlog problem.

```
3 // step id
driver2 // agent executing this step
Action: board
Parameters: driver2 truck1 street0
Precond:
pos truck1
street0

at driver2
street0

empty truck1
true
Effect:
at driver2
truck1

empty truck1
false
```

Listing 3: A Jason translated plan for a driverlog problem.

```
+!board1: V1 = ``truck1'' & V2 = ``street0''
     & id(V1,Id1) & id(V2,Id2) & at(V2) &
    pos(L)[artifact_id(Id1)] & processList(L
    ,V2) & empty(E)[artifact_id(Id1)] & E
          <- -at(V2);
             +at(V1);
             updateEmpty(false)[artifact_id(
    Id1)].
-!board1 <- .print(``Plan board1 failed,
    check solution plan.'').
```

The roles of the organisation are acquired from the instances of the formalism used to represent the problem, which in this case with MAP-POP are the PDDL files. By checking for agent types in Listing 4, and then checking the objects that use those types in Listing 5, the translator generates the roles present in Listing 6. The coordination constraints from the solution found by the MAP-POP algorithm are instantiated in a Moise specification file as a new schema to be followed by the agents. The plans for adopting this schema are also added to each agent's plan library. The conversion of coordination constraints into schemas is exemplified in the next section, along with the descriptions of the driverlog do main and problem that were used as examples.

Listing 4: Types of the driverlog domain.

```
(:types location truck obj - object
        driver - agent)
```

Listing 5: Objects from the problem file that use types in the driverlog domain.

```
(:objects
   driver1 driver2 - driver
   truck1 truck2 - truck
   package1 package2 - obj
   s0 s1 s2 p1-0 p1-2 - location
)
```

Listing 6: Example of translated PDDL types into Moise roles.

```
<role-definitions>
   <role id="driver" />
      <role id="driver1"> <extends role="
    driver"/> </role>
      <role id="driver2"> <extends role="
    driver"/> </role>
   </role-definitions>
```

Now for the environment, we obtain the names of the artefacts by looking at the objects that are not of type agent in the problem file, for example, in Listing 5 they are truck, obj, and location. Next, we create one artefact for each of these types. Information about these objects are stored in observable properties — when an agent focuses an artefact, the observable properties of that artefact will be directly represented as beliefs in that agent's belief base. In Listing 7 the truck object has two observable properties, `empty` (whether or not there are packages inside) and `pos` (where the truck is). For each observable property the artefact also has an operation that allows agents to execute it as an action in order to update its value.

Listing 7: Example of a CArtAgO artefact representing a truck object from the driverlog domain.

```
  defineObsProperty("empty");
  defineObsProperty("pos");

  @OPERATION public void updateEmpty(Boolean
     newEmpty) {
    ObsProperty opEmpty = getObsProperty("
    empty");
    opEmpty.updateValue(newEmpty);
```

At the end of this process all files necessary for the execution stage are available and the user can run the system as any normal JaCaMo system, by running the MAS project file that was also generated during the translation.

5 Case Studies

In this section we describe two multi-agent adaptations of single-agent planning problems from previous IPCs: the driverlog domain and the depots domain. We also discuss the solution and coordination constraints found by the MAP-POP algorithm and the output of the translation process, that is, the MAS that was generated as output.

Performance is not an issue discuss here since there is no purpose in benchmarking the translation, as the planning stage is separated from the execution stage. Instead, we focus on a more qualitative evaluation, analysing the input and output during the planning and execution stages.

Driverlog Domain

The Driverlog domain is a simple problem of logistics. There are several streets and passageways that may contain packages, trucks, and drivers. A driver cannot directly walk through streets, it can only walk through passageways that have paths between a street and a passageway. When driving a truck, a driver can then drive through streets that are linked with each other.

In this domain we only have one type of agent, the driver, as it is the only object that can perform actions. The agent can perform the following actions:

- **load truck**: loads a package from a location into a truck;
- **unload truck**: unloads a package from a truck into a location;
- **board truck**: the driver enters the truck at a location;
- **disembark truck**: the driver leaves the truck at a location;
- **drive truck**: the driver drives the truck from a street to another;
- **walk**: the driver walks from a location that contains a path to another location.

The initial state of the problem can be observed in Figure 2. All the streets are linked, but note that only a truck can move through the linked streets. The drivers, when not driving a truck, can only move through passageways that have paths to streets. The global goal is to have driver1 at s1, and t1 at s1. That is, driver1 and truck1 should be at street1.

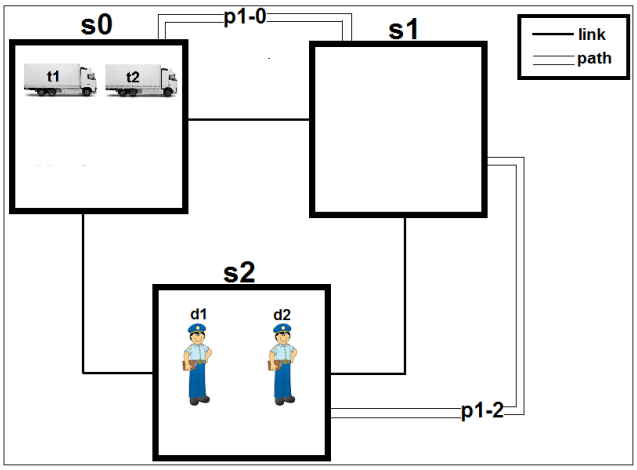


Figure 2: Initial state of the problem for the Driverlog domain.

The solution found by MAP-POP for the Driverlog problem contained the following steps:

- Id 0 — Initial Step
- Id 1 — Final Step
- Id 2 — agent driver2: drive t1 to s1
- Id 3 — agent driver2: board t1 at s0

- Id 4 — agent driver2: walk from p1-0 to s0
- Id 5 — agent driver2: walk from s1 to p1-0
- Id 6 — agent driver2: walk from p1-2 to s1
- Id 7 — agent driver2: walk from s2 to p1-2
- Id 8 — agent driver1: walk from p1-2 to s1
- Id 9 — agent driver1: walk from s2 to p1-2

The partial order in which these steps need to be executed can be obtained from the ordering constraints, also provided in the solution. The ordering constraints are represented in pairs of Ids, the first Id is the step that must come before the second, e.g. 0 — 1 means that the step with Id 0 must come before the step with Id 1. We can also use this order to set the plan operators, i.e. if it will be executed in parallel or sequentially, in the Moise schema. The execution order for the solution of this problem is (numbers between commas can be executed in parallel): 0 — 7,9 — 6,8 — 5 — 4 — 3 — 2 — 1.

Depots Domain

The Depots domain is more complex than the previous domain, as it involves different types of agents. In this domain trucks are used to transport crates between warehouses, with the help of hoists that are present in each warehouse.

There are two types of agents: trucks and locations. Note here that a location (depots or distributors) is a type of agent, since each location has control over a hoist. A truck can perform the following actions:

- **drive**: move the truck from a place to another;
- **load**: loads a crate that a hoist has into the truck;
- **unload**: unloads a crate from the truck to a hoist.

A location can perform the following control actions with its hoist:

- **liftP**: lifts a crate that is on top of a pallet;
- **liftC**: lifts a crate that is on top of another crate;
- **dropP**: drops a crate on top of a pallet;
- **dropC**: drops a crate on top of another crate.

The initial state of the problem can be observed in Figure 3. Truck `t1` is located at `depot0`, and truck `t2` is located at `distributor1`. A truck agent is able to move freely between any of the locations. The global goal is to have c0 on p2, and c1 on p1, i.e. crate0 must be moved to distributor1 and crate1 must be moved to distributor0.

Next we have the solution found by MAP-POP for the Depots domain:

- Id 0 — Initial Step
- Id 1 — Final Step
- Id 2 — agent ditributor1: drop c0 on p2 at distributor1
- Id 3 — agent distributor0: drop c1 on p1 at distributor0
- Id 4 — agent distributor0: lift c0 from p1 at distributor0
- Id 5 — agent truck2: unload c0 to h2 at distributor1
- Id 6 — agent truck2: load c0 from h1 at distributor0

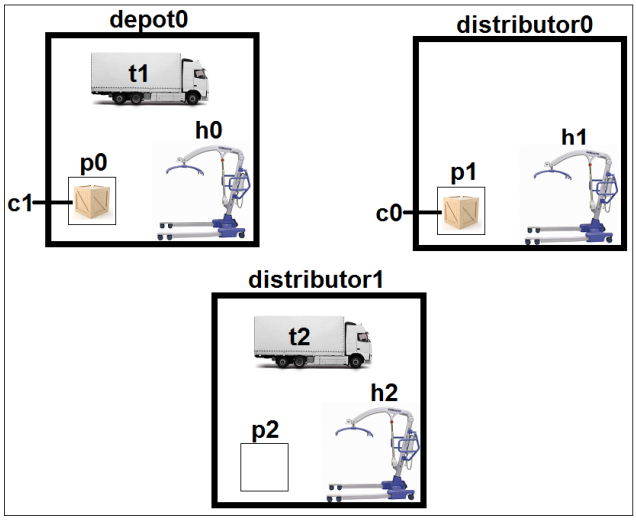


Figure 3: Initial state of the problem for the Depots domain.

- Id 7 — agent truck1: unload c1 to h1 at distributor0
- Id 8 — agent truck1: load c1 from h0 at depot0
- Id 9 — agent truck2: drive from distributor0 to distributor1
- Id 10 — agent depot0: lift c1 from p0 at depot0
- Id 11 — agent truck1: drive from depot0 to distributor0
- Id 12 — agent truck2: drive from distributor1 to distributor0

Once again, we find the partial order of actions by retracing all the ordering constraints, resulting in the order: 0 — 4,10,12 — 6,8 — 9,11 — 5,7 — 2,3 — 1.

Translation

The translator extracts from the solution the steps and the ordering constraints. Each agent directly represents a role in the Moise organisation, e.g. objects driver1 and driver2 are translated as roles that extend a driver role in the Moise specification file under the structural specification. For future work we expect to implement, for example, only the role of driver with a maximum cardinality of 2. In the Moise functional specification we translate steps into goals, with the plan operators (sequence or parallel) that were extracted from the ordering constraints.

Every role (agent) has its mission, and that mission contains all the goals that need to be executed by that particular role. Links and formation constraints, two Moise features, are not considered in our translation algorithms, but they could be expressed by making a few adaptations in the planning formalism. However, since we are using the default input of the MAP algorithms we chose not to make use of these features.

As for the environment, the translator checks the initial state provided by the input of the MAP algorithms. If one of the variables in an initial state is an agent, then that state will be represented as a belief in that particular agent's belief

base. If not, then it will be stored as an observable property in its respective artefact.

The information about initial states is also used to instantiate initial values for the observable properties, that are defined by the predicates and functions of the problem domain. An artefact is created for each type declared as an object in the domain file, to represent the initial state of the environment. For the driverlog domain we have artefacts for `location`, `truck`, and `obj`. For the depots domain we have artefacts for `hoist` and `surface`. When an agent executes the operation of an artefact, it updates the observable properties of the artefact that is involved by using the effects of that particular action.

For the Jason plans, each step is converted to a plan that is added to the agent's plan library, with that step's respective preconditions and effects. In the end of this process we obtain a MAS that can execute the solution for a driverlog problem and a MAS for a depots problem.

6 Conclusion

We integrated a multi-agent planner into JaCaMo through the use of a translator. JaCaMo provided practical solutions for some of the problems that appeared in the execution stage, such as the coordination of agents using Moise organisations, representation of the environment with CArtAgO artefacts, and execution of the solution using Jason agents.

The execution stage of planning is often overlooked when dealing with off-line planning. Our work tries to bridge this gap by using translation algorithms to create a MAS, using the input and output of a multi-agent planner. Our goal with this work was to provide the developers with an initial multi-agent system implementation for a target scenario, based on the solutions found by the multi-agent planner, and to provide a basis for extending other MAP algorithms to work with JaCaMo.

We are investigating two other possible choices of MAP algorithms to be integrated with JaCaMo, the Planning-First (Nissim, Brafman, and Domshlak 2010) and the MAD-A* (Nissim and Brafman 2012). Planning-First is a general, distributed multi-agent planning algorithm that uses Distributed Constraint Satisfaction Problem to coordinate the agents. The MAD-A* is an adaptation for multi-agent planning of one of the best known heuristic search algorithm, A*.

During our work we identified a few downsides:

- although the translation can be used to fill the gap between planning and execution stages, it is not a seamless transition such as the one present in online planning;
- plans in Jason are different from the PDDL formalism used by the three MAP algorithms, which resulted in a simplified conversion of steps to plans;
- the performance was strictly dependent on the performance of the MAP algorithm used during the planning stage.

For future work we would like to use JaCaMo agents not only during the execution stage, but also during the planning stage, which would allow most of the translation to be done directly, and make the transition between planning and execution stages much more seamless. For example, an HTN planner would be able to provide agents in Jason with much more robust plans than previously, and also allows it to make use of current plans present in the agent's plan library prior to the planning stage. This may lead to performance gains and possibly some kind of planning and/or replanning during runtime.

Another line for future work includes the standardisation of input used by the algorithms, so that the translator accepts a standard input file. This input could be a completely new formalism or, for example, the PDDL 3.1 Multi-Agent extension introduced in (Kovacs 2012). This extension allows planning for agents in temporal, numeric domains and copes with many of the already discussed open problems in multi-agent planning, such as the exponential increase in the number of actions, but it also approaches new problems such as the constructive and destructive synergies of concurrent actions. This would also make the process of including a new MAP algorithm easier and at the same time promote a standard formalism to represent domains and problems in multi-agent planning, which at the time of writing does not exist.

Finally, we would also like to test our work on real world applications, such as robotics. Specifically, the scenario we have in mind is the use of Unmanned Aerial Vehicles (UAVs) to monitor, control, and mitigate flash flood occasioned by heavy rain when associated with severe thunderstorms. The planner could be used to generate possible trajectories in flash flood locations, while JaCaMo is used to coordinate the UAVs and reason about possible courses of actions.

7 Acknowledgments

We are grateful for the support given by CAPES (grant number 23038.006826/2014-64) and by CNPq (grant number 308095/2012-0).

References

Boissier, O.; Bordini, R. H.; Hübner, J. F.; Ricci, A.; and Santi, A. 2011. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*.

Bordini, R. H.; Wooldridge, M.; and Hübner, J. F. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons.

Braubach, L.; Pokahr, A.; and Lamersdorf, W. 2008. A universal criteria catalog for evaluation of heterogeneous agent development artifacts. In Jung, B.; Michel, F.; Ricci, A.; and Petta, P., eds., *From Agent Theory to Agent Implementation (AT2AI-6)*, 19–28.

Busetta, P.; Ronnquist, R.; Hodgson, A.; and Lucas, A. 1999. JACK Intelligent Agents - Components for Intelligent Agents in Java. AgentLink News, Issue 2.

Cardoso, R. C.; Zatelli, M. R.; Hübner, J. F.; and Bordini, R. H. 2013. Towards Benchmarking Actor- and Agent-Based Programming Languages. In *AGERE! @ SPLASH 2013*.

Cardoso, R. C.; Hübner, J. F.; and Bordini, R. H. 2013. Benchmarking Communication in Agent- and Actor-Based Languages. In *Proceedings of the EMAS '13, held with AAMAS-2013*, 81–96.

Clement, B. J.; Durfee, E. H.; and Barrett, A. C. 2007. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research (JAIR)* 28:453–515.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A single-agent approach to multiagent planning. In *21st European Conf. on Artificial Intelligence (ECAI'14)*.

Dastani, M. 2008. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems* 16(3):214–248.

de Weerdt, M., and Clement, B. 2009. Introduction to Planning in Multiagent Systems. *Multiagent Grid Syst.* 5(4):345–355.

Decker, K. 1996. TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. *Foundations of Distributed Artificial Intelligence, Chapter 16* 429–448.

Durfee, E. H., and Zilberstein, S. 2013. Multiagent planning, control, and execution. In Weiss, G., ed., *Multiagent Systems 2nd Edition*. MIT Press. chapter 11, 485–545.

Hindriks, K. V.; de Boer, F. S.; van der Hoek, W.; and Meyer, J.-J. C. 2000. Agent Programming with Declarative Goals. In *Proceedings of the 7th International Workshop on Agent Theories, Architectures, Boston, MA, USA*, 228–243. Springer.

Hübner, J. F.; Sichman, J. S.; and Boissier, O. 2007. Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. *Int. J. Agent-Oriented Software Engineering* 1(3/4):370–395.

Jonsson, A., and Rovatsos, M. 2011. Scaling Up Multiagent Planning: A Best-Response Approach. In *Procs. ICAPS 2011*, 114–121. AAAI Press.

Jordan, H.; Botterweck, G.; Huget, M.-P.; and Collier, R. 2011. A feature model of actor, agent, and object programming languages. In *Proceedings of the SPLASH '11 Workshops*, 147–158. New York, NY, USA: ACM.

Jr., J. C. B., and Durfee, E. H. 2011. Distributed algorithms for solving the multiagent temporal decoupling problem. In *AAMAS 2011, Taipei, Taiwan*, 141–148.

Kovacs, D. L. 2012. A Multi-Agent Extension of PDDL3.1. In *Proceedings of the 3rd Workshop on the International Planning Competition (IPC)*, ICAPS-2012, 19–27.

Lerma, A. T. 2011. Design and implementation of a Multi-Agent Planning system. Master's thesis, Polytechnic University of Valencia, Valencia, Spain.

Mao, X.; Mors, A.; Roos, N.; and Witteveen, C. 2007. Coordinating Competitive Agents in Dynamic Airport Resource Scheduling. In *Proceedings of the 5th German conference on Multiagent System Technologies*, 133–144.

Meneguzzi, F., and De Silva, L. 2013. Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review* FirstView:1–44.

Nissim, R., and Brafman, R. I. 2012. Multi-Agent A* for Parallel and Distributed Systems. In *Proceedings of the Heuristics for Domain-Independent Planning Workshop, held with ICAPS 12*.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A General, Fully Distributed Multi-Agent Planning Algorithm. In *AAMAS 10*, 1323–1330.

Planken, L.; de Weerdt, M.; and Witteveen, C. 2010. Optimal temporal decoupling in multiagent systems. In *AAMAS 2010, Toronto, Canada*, 789–796.

Rao, A. S. 1996. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world*, MAAMAW '96, 42–55.

Ricci, A.; Piunti, M.; Viroli, M.; and Omicini, A. 2009. Environment programming in CArtAgO. In *Multi-Agent Programming: Languages, Tools and Applications*, Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer. chapter 8, 259–288.

Ricci, A. 2014. From actor event-loop to agent control-loop: Impact on programming. In *AGERE! '14*, 121–132. New York, NY, USA: ACM.

Sapena, O.; Onaindia, E.; and Torreño, A. 2015. FLAP: applying least-commitment in forward-chaining planning. *AI Commun.* 28(1):5–20.

Sardiña, S., and Padgham, L. 2007. Goals in the context of BDI plan failure and planning. In *AAMAS 2007, Honolulu, Hawaii, USA*.

Sardiña, S., and Padgham, L. 2011. A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems* 23(1):18–70.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014a. A flexible coupling approach to multi-agent planning under incomplete information. *Knowl. Inf. Syst.* 38(1):141–178.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014b. FMAP: distributed cooperative multi-agent planning. *Appl. Intell.* 41(2):606–626.

van Leeuwen, P., and Witteveen, C. 2009. Temporal decoupling and determining resource needs of autonomous agents in the airport turnaround process. In *Proceedings of the International Conference on Intelligent Agent Technology, IAT 2009, Milan, Italy*, 185–192.

Weerdt, M. D.; Mors, A. T.; and Witteveen, C. 2005. Multi-agent planning: An introduction to planning and coordination. Technical report, Handouts of the European Agent Summer.

Witwicki, S. J., and Durfee, E. H. 2011. Towards a unifying characterization for quantifying weak coupling in dec-POMDPs. In *AAMAS 2011, Taipei, Taiwan*, 29–36.

Wooldridge, M. 2002. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 1st edition.

Narrative Planning Agents Under a Cognitive Hierarchy

Josef Hájíček and **Antonín Komenda**
hajicjo1@fel.cvut.cz,komenda@agents.fel.cvut.cz
Department of Computer Science, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic

Abstract

Narrative planning, as an approach to automated storytelling, uses techniques of automated planning to emulate intelligent behavior of story characters, which helps to generate believable tales.

Following the idea of interactive storytelling, where the characters are represented by agents, we propose to use multiagent planning by means of plan merging to generate believable stories. As the story characters represent people, we bound rationality of the planning agents by the concept of a Cognitive Hierarchy, which we use on decisions described as plans. Not only the Cognitive Hierarchy improves believability of the characters, it also increases efficiency of the planning process as sub-optimal solutions might be used. With help of a classical planner and novel compilation for narrative multiagent planning, we experimentally show the efficiency is comparable with the state of the art. Moreover, we show how adjustment of character's cognitive level produces believable simple-minded behavior.

Introduction

Narrative as an inseparable part of many artistic forms plays an important role in human entertainment. From the most fundamental medium which literature surely is, over movies, television, to computer games. Storytelling as a form of narrative can be automated by techniques of artificial intelligence. In order to generate a believable story, we need to order story events in an appropriate sequence. Symbolically, such sequences can be understood as trajectories in a space of possible states of the story (Porteous and Cavazza 2009), bringing us to the area of automated planning, or more particularly—*narrative planning*.

The logical causal progression of plot and character believability were key elements of work of (Riedl and Young 2010). Both elements played an important role in the general understandability of the generated stories. The work introduced a concept of *intent*-based generation of a story, where each character was, in contrast to the previous work, framed by commitments generated based on the required results of the story. This principle allowed better description of motivations of the characters in the story and improved overall understandability of the stories. The process depended on a specialized Intent-based Partial Order Causal Link (IPOCL) planner using heuristics favoring plans, thus stories, with higher character believability.

We do not use multiagent planning directly for generation of the story, but for one agent's internal prediction of behavior of other agents. Such principle, if optimal, represents a planning-based search for an optimal equilibrium among the agents' behavior. In narrative planning, we are however looking for human-like behavior which is usually not optimal. Therefore we employ the game theoretical concept of a Cognitive Hierarchy (CH) (Camerer, Ho, and Chong 2004) causing the agents to create plans as responses to a limited behavior (on lower levels of the hierarchy), not to perfect behavior as in the optimal case. The bounded rationality of the agents causes more human-like decisions, thus more believable story characters.

Humans create strategies to reach their goals, despite contradictory goals of others, by predicting behavior of other entities and then suppose their strategy is the most sophisticated (Camerer, Ho, and Chong 2004). These predictions include opponents' predictions, but these recursive predictions are usually done only to a limited depth (cognitive level). Our system is based on this principle.

The example of the limited recursion is the "beauty contest" game, in which players are asked to pick numbers from 0 to 100, and the player whose number is closest to $\frac{2}{3}$ of the average wins a prize. Equilibrium theory predicts each contestant will reason as follows: "Even if all the other players guess 100, I should guess no more than $\frac{2}{3}$ times 100, or 67. Assuming that the other contestants reason similarly, however, I should guess no more than 45..." and so on, finally concluding that the only rational and consistent choice for all the players is zero (Camerer, Ho, and Chong 2004).

When the game is played by humans, the average guest is typically between 20 and 35. Only, when the game is repetitively played in the same group, the average is approaching to 0.

We will show that the CH is applicable to more complex situations than one-shot games and that the limited cognitive level improves efficiency of the story generation. We analyse complexity of our approach theoretically and practically and demonstrate its competitiveness with the narrative compilation for classical planning proposed by (Haslum 2012).

Additionally, our another objective was to design story-telling system that can handle replacing any subset of artificial agents by human controlled agents without any disruption of the generated story.

Solution Principle and Formalization

Our approach to narrative planning conceptually follows the interactive storytelling principle of each agent representing one character and the idea of multiagent planners based on merging of agents' plans. As in our case the agents *are* the characters, not only act in their roles, the story is induced by pursuing of agents' own goals which are typically antagonistic and naturally weak (only a subset of goals can be fulfilled eventually). Such behavior necessarily causes conflicts among the agents. Both causal progression and character believability stems from the sequential solving of these conflicts by the agents, where the intents of the characters are encoded in the goals.

We consider a set of n possibly competitive agents $\mathcal{A} = \{\alpha_i\}_{i=1}^{n}$ acting synchronously in a shared, fully-observable, deterministic environment. The planning process will be described from perspective of one agent $\alpha_i \in \mathcal{A}$, therefore in unambiguous cases we will use α instead of α_i or instead of all agents $\alpha \in \mathcal{A}$ one by one. Similarly, β will be used as $\beta \in \{\alpha_j | \alpha_j \in \mathcal{A}\, s.t.\, j \neq i\}$, or instead of all agents α_j one by one, as in the next sentence. The agents β will be denoted as *opponents* of α.

The narrative planning problem $\Pi^\alpha = \langle P, A^\alpha, I, G^\alpha, c^\alpha, r^\alpha \rangle$ of an agent α consists of a set of shared propositions P, where each proposition $p \in P$ represents one fact about the shared environment all agents act in. A state of the environment is described by a subset of facts $s \in 2^P$. We also allow for negative preconditions and negative goal conditions. A negative fact will be denoted $\neg p$ and a set of all negative facts $\neg P = \{\neg p | p \in P\}$. The agents act in the environment using their actions $a \in A^\alpha$. The actions follow classical STRIPS (Fikes and Nilsson 1971) definition with negative preconditions $a = \langle \mathrm{pre}(a), \mathrm{add}(a), \mathrm{del}(a) \rangle$, where the sets $\mathrm{pre}(a) \subseteq P \cup \neg P$, $\mathrm{add}(a) \subseteq P$, and $\mathrm{del}(a) \subseteq P$ represent preconditions, addition effects and deletion effects of a respectively. We will use superscripts $^+$ and $^-$ to select positive or negative facts from a set $P \cup \neg P$, thus $\mathrm{pre}^+(a) = \{p | p \in \mathrm{pre}(a), p \in P\}$ and $\mathrm{pre}^-(a) = \{p | \neg p \in \mathrm{pre}(a), \neg p \in \neg P\}$. An action a is *applicable* in a state s iff $\mathrm{pre}^+(a) \cap s = \mathrm{pre}^+(a)$ and[1] $\mathrm{pre}^-(a) \cap s = \emptyset$, whereas it changes the environment to a new state $s' = (s \setminus \mathrm{del}(a)) \cup \mathrm{add}(a)$ if applied. Application of an action a has a positive *cost* $c^\alpha(a)$ for agent α. Each action set A^α contains a no-op action $\epsilon = \langle \emptyset, \emptyset, \emptyset \rangle, c(\epsilon) = 0$. The environment begins in a distinct initial state $I \in 2^P$. Each agent α pursues goals defined as facts $p \in G^\alpha \subseteq P \cup \neg P$, valued by a non-negative *reward* $r^\alpha(p)$. As we consider competitive agents, the goals can be antagonistic, that is for an agent α an opponent β can exists such that

$$G^{\alpha+} \cap G^{\beta-} \neq \emptyset.$$

A solution to Π^α is a plan π^α, which is a sequence of agent's α actions $\pi^\alpha = (a_1, \ldots, a_k)$ such that the actions are successively applicable beginning from the state I inducing intermediate states s_i for $i \in 1, \ldots, k$, and maximizing net-benefit to find a subset of goals (Smith 2004) given as

$$u(\pi^\alpha) = \sum_{p \in (G^{\alpha+} \cap s_k) \cup (G^{\alpha-} \setminus s_k)} r^\alpha(p) - \sum_{a \in \pi^\alpha} c^\alpha(a).$$

A *multiagent narrative planning problem* Φ is defined as a set of narrative agent planning problems $\Phi = \{\Pi^{\alpha_1}, \ldots, \Pi^{\alpha_n}\}$ with a shared environment described by facts P and a common initial state I. Agents' intentions are described by the goals $G^{\alpha_1}, \ldots, G^{\alpha_n}$ with their worthiness for the agents in form of rewards $r^{\alpha_1}, \ldots, r^{\alpha_n}$. Acting of the agents is described by their actions $A^{\alpha_1}, \ldots, A^{\alpha_n}$ with related costs $c^{\alpha_1}, \ldots, c^{\alpha_n}$.

An optimal solution to Φ is a parallel multiagent plan (Nissim, Brafman, and Domshlak 2010) ϕ^* which is a sequence of agents' actions executed in parallel $\phi^* = (\{a_1^{\alpha_1}, \ldots, a_1^{\alpha_n}\}, \ldots, \{a_k^{\alpha_1}, \ldots, a_k^{\alpha_n}\})$, where the actions $\bar{a}_i = \{a_i^{\alpha_1}, \ldots, a_i^{\alpha_n}\}$ in each step i are not in a *conflict* pairwise, formally actions a_g and a_h are not in conflict $\neg conf(a_g, a_h)$ iff

$$\begin{aligned} \bigcup_{\alpha \in \mathcal{A}} \mathrm{pre}^+(a_g^\alpha) \cap \bigcup_{\alpha \in \mathcal{A}} \mathrm{del}(a_h^\alpha) &= \emptyset, \\ \bigcup_{\alpha \in \mathcal{A}} \mathrm{pre}^-(a_g^\alpha) \cap \bigcup_{\alpha \in \mathcal{A}} \mathrm{add}(a_h^\alpha) &= \emptyset, \\ \bigcup_{\alpha \in \mathcal{A}} \mathrm{add}(a_g^\alpha) \cap \bigcup_{\alpha \in \mathcal{A}} \mathrm{del}(a_h^\alpha) &= \emptyset, \end{aligned}$$

all agents' actions a_1^α are applicable in I and the net-benefit over all agents' plans $\pi^\alpha(\phi^*) = (a_1^\alpha, \ldots, a_k^\alpha)$ s.t. $a_i^\alpha \in \bar{a}_i \in \phi^*$ is maximized

$$\sum_{\alpha \in \mathcal{A},} u(\pi^\alpha(\phi^*)). \tag{1}$$

An optimal solution ϕ^* represents a story plan to Φ with perfectly rational agents. Such solutions are impractical for story generation because (i) all characters are unrealistic "foreseers" with respect to all possible behaviors of the other characters, (ii) characters are giving up if they see they cannot reach their goals and (iii) because of high computational complexity of the

[1] Rendering any action with $\mathrm{pre}^+(a) \cap \mathrm{pre}^-(a) \neq \emptyset$ inapplicable.

All characters are alive and want to stay alive. Characters can kill themselves. A man and a woman can get married. Parents can stop a marriage of their children.

There was a king, his daughter princess and a [mighty] **prince. Prince wants to marry the princess and princess wants to marry the prince. The king does not want the princess to marry the prince.**

The king kills himself [because he knows he cannot win a fight with the prince and both the prince and the princess want to get married]**. The prince marries the princess. The end.**

Table 1: A transcription of a story based on an optimal multiagent plan. Comments on the story are in brackets [], explaining the underlying reasoning of the agents.

planning process comparable with complexity needed to solve a fully observable perfect information game in extensive form representing the same problem (ϕ^* would be a trace through the optimal policies maximizing Equation 1). Such story is exemplified in Table 1.

In order to tackle these issues, we propose to bound the rationality of the agents using the concept of a Cognitive Hierarchy and enrich the repertoire of agent behaviors by additional actions representing an attempt to thwart an opponent's action, denoted as *contra-actions*. The bounded rationality of the agents causes the agents cannot foresee all (cognitive) levels of opponents' responses. If an agent α is required to plan on level l by the narrative planning system (the maximum level is externally defined) in the cognitive hierarchy, it responds only to $l-1$ level plans of its opponents. The semantics of a contra-action $\hat{a}[a^\alpha, a^\beta]$ is to allow an agent α to use its action a^α simultaneously against an action a^β of an opponent β. Where the action a^α would be in conflict with opponent's action a^β, the contra-action $\hat{a}[a^\alpha, a^\beta]$ can be used instead of both of them. Still, for sake of consistency of the resulting plan, only one of the actions a^α, a^β can be executed eventually. A *resolution function* $Res[a^\alpha, a^\beta] \mapsto a^\alpha$ or a^β prescribes which agent prevails in the conflict (e.g., based on skills of the story characters) and whether a^α or a^β is going to be executed. The resolution function can be chosen arbitrarily provided it realistically models results of possible conflicts among the story characters. We have experimented with "local" deterministic resolution functions[2], oblivious to the particular state the conflict is resolved in and deterministic as the resolved action is always the same for a particular pair a^α, a^β.

The story generation process is based on planning of a best *response* in form of a multiagent plan ϕ^α by which agent α reacts to behavior of agents β at previous cognitive level $l-1$. The behavior of the agents β is described in their (single-agent) plans π^β. The plan ϕ^α comprises the π^β plans as well. If α is generating a response plan ϕ^α_l at cognitive level l it reacts to plans π^β_{l-1} of βs at level $l-1$. Plans at $l=0$ are empty $\pi^\beta_0 = \emptyset$. A cognitive level l is completely planned if all agents $\alpha \in \mathcal{A}$ plan all response plans ϕ^α_l. The response plans are used for next cognitive level s.t. $\pi^\alpha(\phi_{l-1}) = \pi^\alpha_l$ (for definition of $\pi^\alpha(\phi)$ see Equation 1) for all agents α.

The response plans are generated using a state-of-the-art classical cost-optimal planner run on a problem compiled from Φ representing a planning problem of response generation of α to π^β_l of all opponents β. The compilation is described in details in the next section.

Response Plan Compilation

The problem of generating a response multiagent plan ϕ^α_l at cognitive level l of agent α to behavior of agents β at cognitive level $l-1$ will be modeled and solved as a classical planning problem.

In order to use classical planning, we need the compilation to cover

- (i) parallel multiagent planning as sequential planning,
- (ii) net-benefit selection of actions and goals of agent α,
- (iii) planning of agent's α response to opponents' behavior on cognitive level $l-1$, and
- (iv) planning of contra-actions and their resolution with opponent's action as *Res* defines.

The input of the compilation for agent α is a tuple $\left\langle \Phi, Res, k, \bigcup_{\beta \in \mathcal{A}\setminus\{\alpha\}} \{\pi^\beta_{l-1}\} \right\rangle$, where Φ is the narrative multiagent problem, *Res* is the resolution function, k is the lookahead (horizon) of agent α, and plans π^β_{l-1} represent behavior of all agent's α opponents $\beta \in \{\alpha_j | \alpha_j \in \mathcal{A}\, s.t.\, \alpha_j \neq \alpha\}$ at previous cognitive level $l-1$. If the plans π^β_{l-1} comprise actions a_g and a_h in a conflict, only the action $Res[a_g, a_h]$ is kept in the plan for future use in the compilation. The other action is replaced by ϵ. Thus the rest of the compilation treats the plans π^β_{l-1} as non-conflicting.

The output of the compilation is a classical planning problem Π' which solution plan π' can be converted to the response multiagent plan ϕ^α_l.

Solution plans of the compiled problem are constituted such that they consists of k action sub-sequences framing the parallel actions in a multiagent plan $(a^{\alpha_1}_1, \ldots, a^{\alpha_n}_1, \ldots, a^{\alpha_1}_k, \ldots, a^{\alpha_n}_k) \approx (\{a^{\alpha_1}_1, \ldots, a^{\alpha_n}_1\}, \ldots, \{a^{\alpha_1}_k, \ldots, a^{\alpha_n}_k\})$. In each such *time frame* $a^{\alpha_1}_i, \ldots, a^{\alpha_n}_i$ of step i we distinguish three phases:

1. (AO) acting of opponents $\{\beta_i | \beta_i \in \mathcal{A}\, s.t.\, \beta_i \neq \alpha\}$,
2. (AA) acting of the responding agent α, and
3. (GC) check of goal facts.

Each time frame follows an ordering scheme $a^{\beta_1}, \ldots, a^{\beta_{n-1}}, a^\alpha$ of agents' actions, fit to the phases

[2] A stochastic resolution function as well as stationary $Res[s, a^\alpha, a^\beta]$ or non-stationary $Res[\pi, a^\alpha, a^\beta]$ resolution functions are interesting alternatives left for the future work.

$\mathsf{AO}, \mathsf{AA}, \mathsf{GC}$, where GC does not use any of regular agents' actions. A formal description of the time frame scheme will follow a detailed description of propositions P', actions A' with a cost function c', an initial state I' and goal conditions G' of the compiled problem Π'.

Propositions

The propositions of the compiled planning problem are

- facts P of the input problem Φ,
- t_i for all $1 \leq i \leq k$ denoting which time frame the planned actions are in,
- $\mathsf{phAO}, \mathsf{phAA}, \mathsf{phGC}$ marking what phase of the time frame the actions are planned in,
- phAO^β for all opponents β which already acted in the AO phase,
- $\mathsf{oap}_{a^\beta,i}$ (opponent action proposition) representing opponent's β action $a^\beta \in A^\beta$ at step i in the input plan π^β_{l-1} ($\mathsf{OAP} = \bigcup_{a^\beta \in \pi^\beta_{l-1} \text{ at } i} \{\mathsf{oap}_{a^\beta,i}\}$),
- g_p for all $p \in G$, where $G = \bigcup_{\alpha \in \mathcal{A}} G^\alpha$, holding if the goal condition p is satisfied to allow finish the planning without satisfying all goals, and
- fin denoting end of the planning process if all goals are known to be either satisfied or unsatisfied.

The complete set of propositions is formally

$$\begin{aligned} P' &= P \cup \bigcup_{1 \leq i \leq k} \{\mathsf{t}_i\} \cup \{\mathsf{phAO}, \mathsf{phAA}, \mathsf{phGC}, \mathsf{fin}\} \\ &\quad \cup \bigcup_{\beta \in \mathcal{A} \setminus \{\alpha\}} \{\mathsf{phAO}^\beta\} \cup \mathsf{OAP} \cup \bigcup_{p \in G} \{\mathsf{g}_p\}. \end{aligned}$$

Initial State and Goal Conditions

The compilation initializes the planning to the original initial state I, 0-th time frame, first phase AO, and as all opponents' actions has to be tackled, all oaps hold

$$I' = I \cup \{\mathsf{t}_0, \mathsf{phAO}\} \cup \mathsf{OAP}.$$

The goal condition is straightforward

$$G' = \{\mathsf{fin}\}.$$

Operators

The compiled actions will be defined in form of parameterized operators. The operators form three groups following the frame phases: opponents' acting (AO); agent's α acting (AA); and phase switching, time progression, treatment of goal conditions and termination (GC). Let us recall the scheme of an action $a = \langle \mathrm{pre}(a), \mathrm{add}(a), \mathrm{del}(a) \rangle$. For definition of the compilation operators, we will use the same form with possible additional parameters of the operator par_i written as $o[par_1, ..., par_m]$.

The initial state I' indicates by the propositions phAO that the opponents act firstly in the time frame. Opponents' acting is prescribed by the actions $a^\beta \in \pi^\beta_{l-1}$ at an i-th step. The opponent acting operator for action $a^\beta \in A^\beta$ is defined as follows

$$\begin{aligned} \mathsf{aopp}[a^\beta, i] &= \langle \mathrm{pre}(a^\beta) \cup \{\mathsf{t}_i, \mathsf{phAO}, \neg\mathsf{phAO}^\beta, \mathsf{oap}_{a^\beta,i}\}, \\ &\quad \mathrm{add}(a^\beta) \cup \{\mathsf{phAO}^\beta\}, \\ &\quad \mathrm{del}(a^\beta) \rangle, \\ c'(\mathsf{aopp}[a^\beta, i]) &= 0. \end{aligned}$$

The preconditions additionally limit usage to an action represented by $\mathsf{oap}_{a^\beta,i}$, only if the opponent β did not act yet in the AO phase of the current time frame i. As the action a^β at i can be inapplicable, because of planned actions of the agent α in previous time frames, an alternative operator representing impossibility to act of β is defined for all propositions $p \in \mathrm{pre}(a^\beta)$ as

$$\begin{aligned} \mathsf{acopp}[a^\beta, p, i] &= \langle \{\neg p, \mathsf{t}_i, \mathsf{phAO}, \neg\mathsf{phAO}^\beta, \mathsf{oap}_{a^\beta,i}\}, \\ &\quad \{\mathsf{phAO}^\beta\}, \\ &\quad \emptyset \rangle, \\ c'(\mathsf{acopp}[a^\beta, p, i]) &= 0. \end{aligned}$$

Since each action induced by acopp contains one negation $\neg p$ of a fact from $\mathrm{pre}(a^\beta)$, the agent α has to *either* use the regular opponent's action a^β by $\mathsf{aopp}[a^\beta, i]$ or one of the acopp actions. Hence the agent can act against opponents' actions in previous time frames by obstructing one of ps, but cannot ignore opponents' actions freely and use only a "no-op" variant acopp.

As long as all opponents acted, the current phase is switched to phAA by

$$\begin{aligned} \mathsf{aAO} \triangleright \mathsf{AA}[i] &= \langle \{\mathsf{t}_i, \mathsf{phAO}\} \cup \bigcup_{\beta \in \mathcal{A} \setminus \{\alpha\}} \mathsf{phAO}^\beta, \\ &\quad \{\mathsf{phAA}\}, \\ &\quad \{\mathsf{phAO}\} \cup \bigcup_{\beta \in \mathcal{A} \setminus \{\alpha\}} \mathsf{phAO}^\beta \rangle, \\ c'(\mathsf{aAO} \triangleright \mathsf{AA}[i]) &= 0. \end{aligned}$$

In phase phAA, acting of agent α is straightforward and uses action $a^\alpha \in A^\alpha$. The compiled actions are described as the operator

$$\begin{aligned} \mathsf{a}[a^\alpha, i] &= \langle \mathrm{pre}(a^\alpha) \cup \{\mathsf{t}_i, \mathsf{phAA}\} \cup \neg\mathcal{C}(a^\alpha, i), \\ &\quad \mathrm{add}(a^\alpha) \cup \{\mathsf{phGC}\}, \\ &\quad \mathrm{del}(a^\alpha) \cup \{\mathsf{phAA}\} \rangle, \\ c'(\mathsf{a}[a^\alpha, i]) &= c^\alpha(a^\alpha), \end{aligned}$$

where the action cannot be used if it is in conflict with one of opponents actions in the same time frame $\mathcal{C}(a^\alpha, i) = \{\mathsf{oap}_{a^\beta,i} | conf(a^\alpha, a^\beta), a^\beta \in \pi^\beta_{l-1} \text{ at } i \; s.t. \; \beta \in \mathcal{A} \setminus \{\alpha\}\}$.

In the last phase GC (of the last time frame), the goal conditions and termination are checked. Termination depends on either positive or negative fulfillment of all goal conditions g_p. Two operators describing positive and negative fulfillment of a goal condition follows

$$\begin{aligned} \mathsf{agp}[p] &= \langle \{\mathsf{t}_k, \mathsf{phGC}, p\}, \{\mathsf{g}_p\}, \emptyset \rangle, c'(\mathsf{agp}[p]) = 0, \\ \mathsf{agn}[p] &= \langle \{\mathsf{t}_k, \mathsf{phGC}\}, \{\mathsf{g}_p\}, \emptyset \rangle, c'(\mathsf{agn}[p]) = r^\alpha(p). \end{aligned}$$

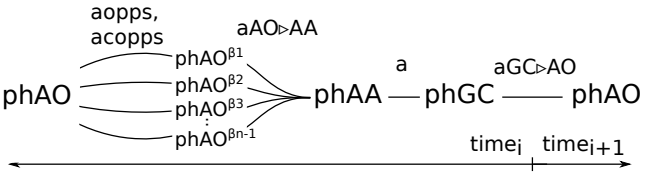


Figure 1: Ordering of the actions of the opponents $\beta_1, \ldots \beta_{n-1}$ and the agent α with the phase markers $\mathsf{phAO}, \mathsf{phAO}^{\beta}, \mathsf{phAA}, \mathsf{phGC}$ in one time frame i. All aopp, acopp actions of opponents β and an action a of agent α within the t_i frame represent one set of parallel actions $\{a_i^{\alpha_1}, \ldots, a_i^{\alpha_n}\}$ of the resulting multiagent plan.

The preconditions limit usage of the actions to the last time frame t_k as we do not want to skip possible later opponents' actions. The cost of $\mathsf{agn}[p]$ is a penalty for not fulfilling the goal equal to the reward of the goal $r^{\alpha}(p)$. The problem is solved, thus the planning process terminates if the proposition fin is reached by action

$$\mathsf{afin} = \left\langle \{\mathsf{t}_k, \mathsf{phGC}\} \cup \bigcup_{p \in G} \{\mathsf{g}_p\} \cup, \{\mathsf{fin}\}, \emptyset \right\rangle, c'(\mathsf{afin}) = 0,$$

which can be used only if all goal conditions were already treated either the positive or the negative way.

If the goal check did not succeed the time progresses and the first phase phAO starts again by

$$\begin{aligned} \mathsf{aGC} \triangleright \mathsf{AO}[i] &= \langle \{\mathsf{t}_i, \mathsf{phGC}\}, \\ &\quad \{\mathsf{t}_{i+1}, \mathsf{phAO}\}, \\ &\quad \{\mathsf{t}_i, \mathsf{phGC}\} \rangle, \\ c'(\mathsf{aGC} \triangleright \mathsf{AO}[i]) &= 0. \end{aligned}$$

The set of compiled actions A' is induced by grounding of all presented operators for all parameter instances: time frames $0 \leq i \leq k$, actions $a^{\alpha} \in A^{\alpha}$ and $a^{\beta} \in A^{\beta}$ for all opponents, and all goal conditions $p \in G$. The compiled problem is thus defined as $\Pi' = \langle P', A', I', G', c' \rangle$. Figure 1 depicts how the compilation constraints actions in one time frame.

Proposition 1. *The complexity of the compilation process of* $\left\langle \Phi, Res, \bigcup_{\beta \in \mathcal{A} \setminus \{\alpha\}} \{\pi_{l-1}^{\beta}\} \right\rangle$ *to* Π' *is polynomially bounded in the size of the input.*

Proof. The dominant term in the number of compiled propositions of P' is number of actions $a^{\beta} \in A^{\beta}$ times k for the set $\bigcup_{a^{\beta} \in \pi_{l-1}^{\beta} \text{ at } i} \{\mathsf{oap}_{a^{\beta},i}\}$, all other added predicates are linear or constant in their size in Φ. The size of the initial and goal state is number of the predicates $\mathsf{oap}_{a^{\beta},i}$ as well. The number of actions in A' is bounded either by the number of actions $a^{\beta} \in A^{\beta}$ times number of predicates $p \in P$ times the length k by the operator $\mathsf{acopp}[a^{\beta}, p, i]$ or by the number of actions $a^{\alpha} \in A^{\alpha}$ times the length k times the size of $\mathcal{C}(a^{\alpha}, i)$ which is maximally the number of the $\mathsf{oap}_{a^{\beta},i}$ predicates representing the opponents' plans π_{l-1}^{β} on the input. □

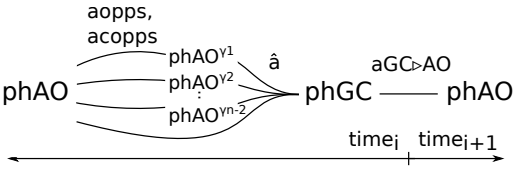


Figure 2: Ordering of the actions of the opponents $\gamma_1, \ldots \gamma_{n-2}$, and a contra-action $\hat{\mathsf{a}}$ used by the agent α against the opponent β. Action framing and phase markers follow Figure 1.

The presented compilation does not allow α to act against an assumed action $a \in \pi_{l-1}^{\beta}$ of an opponent β on previous cognitive level $l-1$ in one time frame. We will deal with this drawback in the next section.

Contra-actions

A contra-action describes an optimistic intention of an agent α that if an action a^{α} is played in parallel against a conflicting action a^{β} of an opponent β, the agent α prevail and its action's effects take place in the resulting plan, formally $Res[a^{\alpha}, a^{\beta}]$ returns a^{α}. A "degree of optimism" is described as a cost multiplication factor f used to increase the price of the contra-action as far as the agent knows its action a^{α} fails by the resolution function $Res[a^{\alpha}, a^{\beta}]$.

The set of contra-actions is described by one additional operator which can be used together with the compilation of Π' explained in previous sections. The definition follows

$$\begin{aligned} \hat{\mathsf{a}}[a^{\alpha}, a^{\beta}, i] &= \langle \mathrm{pre}(a^{\alpha}) \cup \{\mathsf{t}_i, \mathsf{phAO}, \neg\mathsf{phAO}^{\beta}, \mathsf{oap}_{a^{\beta},i}\} \\ &\quad \cup \bigcup_{\gamma \in \mathcal{A} \setminus \{\alpha,\beta\}} \mathsf{phAO}^{\gamma} \cup \neg\hat{\mathcal{C}}(a^{\alpha}, \beta, i), \\ &\quad \mathrm{add}(a^{\alpha}) \cup \{\mathsf{phGC}\}, \\ &\quad \mathrm{del}(a^{\alpha}) \cup \{\mathsf{phAO}\} \cup \bigcup_{\gamma \in \mathcal{A} \setminus \{\alpha,\beta\}} \mathsf{phAO}^{\gamma} \rangle. \end{aligned}$$

The definition subsumes $\mathsf{a}[a^{\alpha}, i]$, $\mathsf{acopp}[a^{\beta}, p, i]$, and phase switch from AO to GC as depicted in Figure 2. As a contra-action describes a conflict only between a pair of agents, the rest of the agents denoted γ has not to be in a conflict, thus $\hat{\mathcal{C}}(a^{\alpha}, \beta, i) = \{\mathsf{oap}_{a^{\gamma},i} | conf(a^{\alpha}, a^{\gamma}), a^{\gamma} \in \pi_{l-1}^{\gamma} \text{ at } i \; s.t. \; \gamma \in \mathcal{A} \setminus \{\alpha, \beta\}\}$.

The cost of a contra-action formally describes the optimism of the agent acting against the opponent β

$$c'(\hat{\mathsf{a}}^{\alpha}[a^{\alpha}, a^{\beta}, i]) = \begin{cases} c^{\alpha}(a^{\alpha}) & Res[a^{\alpha}, a^{\beta}] = a^{\alpha}. \\ f \cdot c^{\alpha}(a^{\alpha}), f > 1 & otherwise \end{cases}$$

The first branch represents successful contra-acting for cost $c^{\alpha}(a^{\alpha})$ and the other branch represents optimistic (presumably unsuccessful) higher cost $f \cdot c^{\alpha}(a^{\alpha})$ acting. The higher cost acting can be still worth using if reaching the effects of a^{α} exceeds the cost $f \cdot c^{\alpha}(a^{\alpha})$.

Similarly as in the compilation without contra-actions, the added operator induces a set of grounded contra-actions $\hat{A}'$. A compiled problem with contra-actions is then defined as $\hat{\Pi}' = \left\langle P', A' \cup \hat{A}', I', G', c' \right\rangle$.

Proposition 2. *The complexity of the compilation process with contra-action of* $\left\langle \Phi, Res, \bigcup_{\beta \in \mathcal{A} \setminus \{\alpha\}} \{\pi^{\beta}_{l-1}\} \right\rangle$ *to* $\hat{\Pi}'$ *is polynomially bounded in the size of the input.*

Proof. Follows proof of Proposition 1. The number of the contra-actions in $\hat{A}'$ is another polynomial term as it is the number of actions $a^{\alpha} \in A^{\alpha}$ times the number of actions $a^{\beta} \in A^{\beta}$ times the length k times the number of conflict actions $\hat{\mathcal{C}}(a^{\alpha}, \beta, i)$. The number of actions in conflict follows the same argumentation as in the proof of Proposition 1 without one agent (opponent β). □

The polynomial bounds result from interactions of agents limited only to a pair of agents (in case of the contra-actions). More general contra actions among larger groups would imply exponential dependence on number of agents[3].

Narrative Multiagent Planning

With the help of the compilation from the previous sections, we can define the algorithm for narrative multiagent planning. It is outlined in Algorithm 1.

As an input, the algorithm takes a problem Φ, a resolution function *Res*, the length of prediction plans k, a set of non-zero positive integer cognitive levels l^{α} for all agents α and limit on the length of the story m. The output is a multiagent plan representing the story as actions of the agents.

Following the well-proven principle of uncertainty planning by re-planning (e.g., in (Yoon, Fern, and Givan 2007)) the outermost loop (lines 2–19) successively re-plans the problem. The agents decide always in the last reached state s_i of the environment, regardless unknown future caused by behavior of the opponent agents on various cognitive levels and (not necessarily deterministic) resolution function *Res*. The agents α act using the first action a of their response plans $\pi^{\alpha}_{l^{\alpha}}$ on the required cognitive level l^{α} (lines 14–16). A conflict of actions $a^{\alpha} \in \bar{a}_i$ and $a^{\beta} \in \bar{a}_i$ is resolved (on line 15) by keeping only the $Res[a^{\alpha}, a^{\beta}]$ action and replacing the others by ϵ.

According to the algorithm scheme for single-action decisions under the Cognitive Hierarchy from (Camerer, Ho, and Chong 2004), the inner while loop (lines 5–13) gradually builds plans π^{α}_{l} of all agents α from the lowest cognitive level $l = 0$, for which the plans are initialized as empty (line 3), to the maximal required cognitive level $\max_{\alpha \in \mathcal{A}}\{l^{\alpha}\}$ (line 5).

[3]Efficient tackling of general contra-actions is left for the future work.

Algorithm 1 Narrative multiagent planning.

Input: multiagent narrative planning problem Φ,
resolution function *Res*,
a set of cognitive levels l^{α} for all agents α, and
maximal length of the story m
length of prediction k

Output: story of all agents in form of a parallel multiagent plan ϕ

1: $i \leftarrow 0, s_0 \leftarrow I$
2: **while** $i \leq m$ **do**
3: all $\pi^{\alpha}_0 \leftarrow \emptyset$ for all $\alpha \in \mathcal{A}$
4: $l \leftarrow 1$
5: **while** $l \leq \max_{\alpha \in \mathcal{A}}\{l^{\alpha}\}$ **do**
6: **for** all $\alpha \in \mathcal{A}$ **do**
7: $\hat{\Pi}' \leftarrow$ compile $\left\langle \Phi, Res, k, \bigcup_{\beta \in \mathcal{A} \setminus \{\alpha\}} \{\pi^{\beta}_{l-1}\} \right\rangle$
8: $\hat{\pi}' \leftarrow$ optimally solve $\hat{\Pi}'$ // FastDownward
9: $\phi^{\alpha}_{l} \leftarrow$ decompile multiagent plan from $\hat{\pi}$'
10: $\pi^{\alpha}_{l} \leftarrow \pi^{\alpha}(\phi^{\alpha}_{l})$ // select α's plan
11: **end for**
12: $l \leftarrow l + 1$
13: **end while**
14: $\bar{a}_i \leftarrow \bigcup_{\alpha \in \mathcal{A}}\{a | a \text{ is first action of } \pi^{\alpha}_{l^{\alpha}}\}$
15: $\bar{a}_i \leftarrow$resolve conflicts in $\bar{a}_i$ by *Res*
16: $s_{i+1} \leftarrow (s_i \setminus \bigcup_{a \in \bar{a}_i} \mathrm{del}(a)) \cup \bigcup_{a \in \bar{a}_i} \mathrm{add}(a)$
17: $I \leftarrow s_{i+1}$, where I is in Φ
18: $i \leftarrow i + 1$
19: **end while**
20: **return** $(\bar{a}_1, \ldots, \bar{a}_m)$

The innermost loop (lines 6–11) generates the best response plans for each agent α (line 6), similarly as proposed by (Jonsson and Rovatsos 2011). First, the described compilation is used to generate a classical multiagent plan $\hat{\Pi}'$ representing agent's α response problem with contra-actions $\left\langle \Phi, Res, k, \bigcup_{\beta \in \mathcal{A} \setminus \{\alpha\}} \{\pi^{\alpha}_{l-1}\} \right\rangle$ to opponents' plans from $l-1$ level $\{\pi^{\alpha}_{l-1}\}$ (line 7). Second, FastDownward planner (Helmert 2006) is used to find a cost-optimal solution to the compiled problem $\hat{\Pi}'$ (line 8) and the solution is decompiled to a parallel multiagent plan ϕ^{α}_{l} (line 9) such that in each time frame, the actions defined in the input problem Φ are used as parallel actions and contra-actions are replaced according to the resolution function *Res*. Finally, the plan of agent α for level l is extracted (line 10) and remembered as π^{α}_{l} for later use.

On the line 20, the algorithm returns the while story, consisting of ordered story events (characters' actions).

Experiments

We have implemented the narrative multiagent planning algorithm including the response plan compilation with contra-actions. For practical reasons, the compilation of $\mathsf{acopp}[a^{\beta}, p, i]$ was modified such that if an action of an opponent β is once inapplicable, the opponent is marked by a proposition and no longer allowed to use

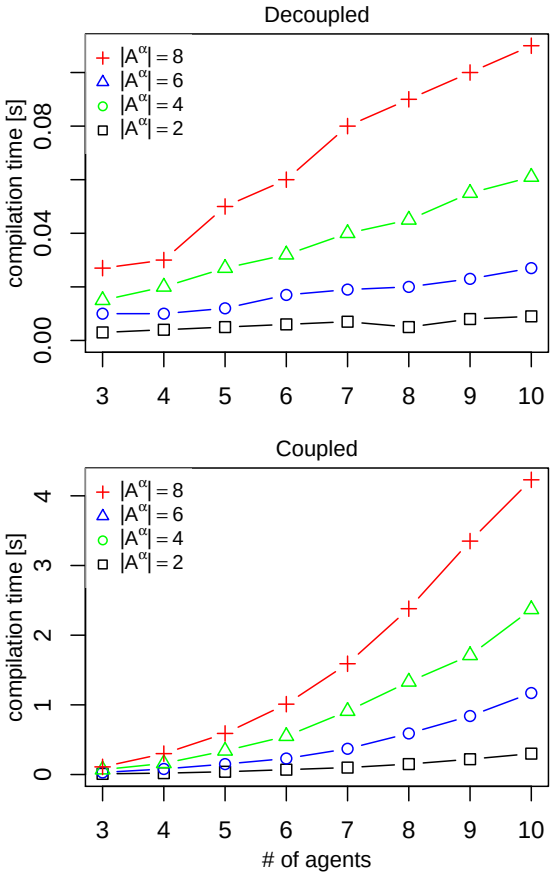


Figure 3: Compilation time for increasing number of agents and different problem sizes (number actions in A^α of each agent in $\mathcal{A}$) at cognitive level $l^\alpha = 2$.

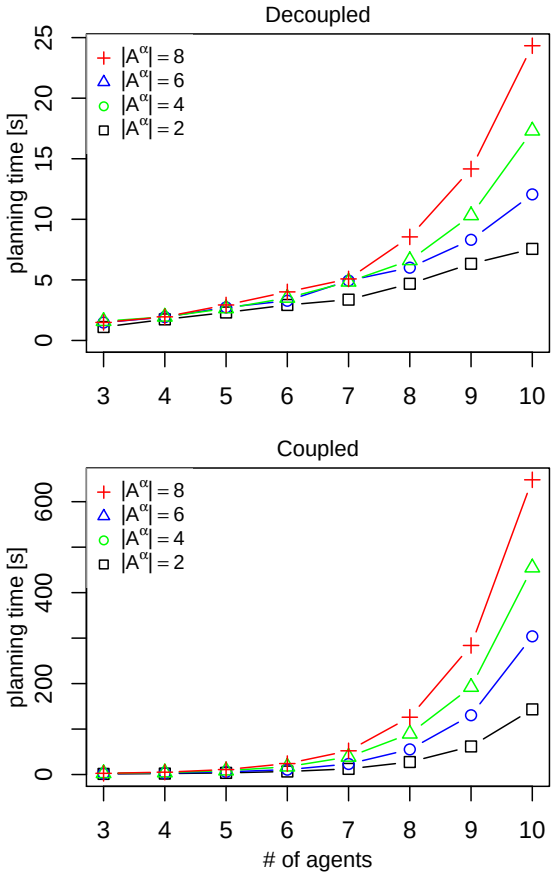


Figure 4: Runtime of the multiagent narrative planning for increasing number of agents and different problem sizes (number of actions in A^α of each agent) at $l^\alpha = 2$.

$\mathsf{aopp}[a^\beta, i]$. The opponent is also marked if a contra-action $\hat{\mathsf{a}}[a^\alpha, a^\beta, i]$ was successfully used against it. The compilation as described formally subsumes such behavior and it does not affect the complexity results. The target length of the story plan m as well as the prediction horizon k was set to 10 steps. All experiments were performed as a single thread process on i5 processor at 2.9 Ghz with 3GB allowed memory.

To verify tractability of the compilation and efficiency of the proposed algorithm, we have prepared a synthetic planning domain. The domain allowed us to variably change the number of agents and/or coupling of their actions. Multiagent planning in general gets (exponentially) harder with increasing coupling as proven by (Brafman and Domshlak 2008), therefore our experiments show the results both for coupled and decoupled problem variations.

The relation of compilation time and increasing number of agents is depicted in Figure 3. The trends agree with Proposition 2 and show that the compilation time is roughly linear or polynomial to number of agents in decoupled or coupled problems respectively. As the size of the problem increases, the compilation time increases accordingly, following the argumentation in the proof of Propositions 1 and 2.

The efficiency of the proposed algorithm was analyzed for increasing number of agents as well. Figure 4 shows how is the planning runtime dependent on the number of agents. In the decoupled case, the trends show rather exponential growth with exception of the smallest problem with 2 actions per agent. In the coupled problems, the growth is clearly exponential which is an expected behavior considering results of (Brafman and Domshlak 2008).

For the final experiment, we used the narrative planning domain and problem by (Riedl and Young 2010) for which (Haslum 2012) proposed an efficient compilation to classical planning. The results in Figure 5 show that the narrative multiagent planning algorithm generates a story faster for all cognitive levels $l^\alpha \leq 7$. A believable story by (Riedl and Young 2010) presented in Table 2 is generated for all cognitive levels $l^\alpha \geq 3$. The key point in the story is that the king realizes the dragon will try to kill the king as the dragon knows the king needs the magic lamp and will plan to kill the dragon.

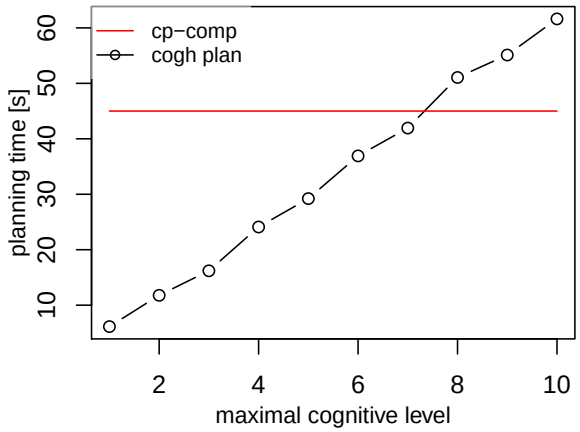


Figure 5: Varying cognitive level in our approach (cogh plan) compared with planning runtime by (Haslum 2012) (cp-comp) on problem of (Riedl and Young 2010).

All characters are alive and want to stay alive. Characters can kill themselves. A man and a woman can get married if they both want to. A love spell from a magic lamp can change someones mind about wanting to marry someone.

There was a king, his [mighty] **knight, and a princess, all in a castle. There was a magic lamp held by a dragon in mountains. The king wants to marry the princess. The princess does not want to marry the king. The dragon does not want to give the lamp to anybody.**

The king and his knight travel to the mountains [as the king knows the dragon would kill him, he travels with the knight]**. The knight tries to kill the dragon by king's order, the dragon tries to kill the knight** [as the dragon knows he wants the lamp]**, the dragon is killed by the knight** [the dragon's contra-action failed based on a resolution function preferring king's "mighty" knight]**. The king orders the knight to take the lamp from the dragon. The king takes the lamp from the knight. The king returns to the castle and plays the love spell on the princess. The king marries the princess. The end.**

Table 2: A transcription of a story generated by the narrative multiagent planning algorithm on cognitive levels $l^{\alpha} \geq 3$ in problem by (Riedl and Young 2010). The agents are the king and the dragon characters. The story plan is: *travel*(king, castle, mountains), {*kill-by-knight*(king, dragon, mountains), *kill*(dragon, king, mountains)}, *take-lamp-by-knight-from*(king, dragon, mountains), *take-lamp-from-knight*(king, mountains), *travel*(king, mountains, castle), *love-spell*(king, princess, castle), *marry*(king, princess, castle).

Base facts follow the story in Table 2.

There was a king, his [mighty] **knight, and a princess, all in a castle. There was a magic lamp protected by a dragon in mountains. The king wants to marry the princess. The princess does not want to marry the king. The dragon does not want to give the lamp to anybody.**

The king travels to the mountains [to get the magic lamp, not realizing the dragon knows he wants to kill him because of the lamp; knight is not ordered because of his cost]**. The king tries to kill the dragon, the dragon tries to kill the king** [as the dragon knows he wants to kill him as well]**, the dragon kills the king** [dragon's contra-action prevailed based on resolution function preferring the dragon]. **The end.**

Table 3: A transcription of a story generated by the narrative multiagent planning algorithm with the agents on cognitive levels $l^{\mathsf{king}} = 2$ and $l^{\mathsf{dragon}} \geq 3$ in problem by (Riedl and Young 2010). The agents are the king and the dragon characters. The story plan is: *travel*(king, castle, mountains), {*kill*(king, dragon, mountains), *kill*(dragon, king, mountains)}.

This realization by the king does not happen if the king on cognitive level $l^{\mathsf{king}} = 2$ plans the dragon's plan $\pi^{\mathsf{dragon}}_{l-1}$. On this level, dragon's plan $\pi^{\mathsf{dragon}}_{l=1}$ does not contain any reaction to kings (empty) behavior $\pi^{\mathsf{king}}_{l=0} = \emptyset$. If it is so, the king tries to take the lamp regardless possible dragon's actions, which ends not well for the king as demonstrated by a story in Table 3. Such cognitive bound can be utilized to model simple-minded characters.

Conclusion & Future Work

Practical requirements of narrative planning do not need optimal (fully combinatorial) multiagent planning of all agents competing with each other in parallel. We have utilized this fact to propose a polynomially bounded compilation of a multiagent narrative planning problem into a sequence of classical re-planning.

We have experimentally shown that our solution is practically usable and comparable with the state-of-the-art single-agent narrative planning algorithm from perspective of runtime and believability of the story.

Interesting research questions kept for future work are how could more elaborate conflict resolution functions improve generated stories e.g., by modeling dynamically changing skills of the characters. How to efficiently extend current pairwise contra-actions to larger groups of agents. And finally, an interesting practical experiment would be, as the proposed technique allows to replace some agents by human players, to play the story as an interactive game and evaluate the believability of the characters by the human players.

Acknowledgments

This research was supported by the Czech Science Foundation (grant no. 15-20433Y).

References

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

Camerer, C. F.; Ho, T. H.; and Chong, J.-K. 2004. A cognitive hierarchy model of games. *The Quarterly Journal of Economics* 119(3):861–898.

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

Haslum, P. 2012. Narrative planning: Compilations to classical planning. *J. Artif. Int. Res.* 44(1):383–395.

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

Jonsson, A., and Rovatsos, M. 2011. Scaling Up Multiagent Planning: A Best-Response Approach. In *Procs. ICAPS 2011*, 114–121. AAAI Press.

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

Porteous, J., and Cavazza, M. 2009. Controlling narrative generation with planning trajectories: The role of constraints. In Iurgel, I.; Zagalo, N.; and Petta, P., eds., *Interactive Storytelling*, volume 5915 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 234–245.

Riedl, M. O., and Young, R. M. 2010. Narrative planning: Balancing plot and character. *J. Artif. Int. Res.* 39(1):217–268.

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 393–401. AAAI.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, 352. AAAI.

Planning Over Multi-Agent Epistemic States: A Classical Planning Approach (Amended Version) *

Christian Muise*, Vaishak Belle†, Paolo Felli*, Sheila McIlraith†
Tim Miller*, Adrian R. Pearce*, Liz Sonenberg*

*Department of Computing and Information Systems, University of Melbourne
†Department of Computer Science, University of Toronto
*{christian.muise,paolo.felli,tmiller,adrianrp,l.sonenberg}@unimelb.edu.au, †{vaishak,sheila}@cs.toronto.edu

Abstract

Many AI applications involve the interaction of multiple autonomous agents, requiring those agents to reason about their own beliefs, as well as those of other agents. However, planning involving nested beliefs is known to be computationally challenging. In this work, we address the task of synthesizing plans that necessitate reasoning about the beliefs of other agents. We plan from the perspective of a single agent with the potential for goals and actions that involve nested beliefs, non-homogeneous agents, co-present observations, and the ability for one agent to reason *as if* it were another. We formally characterize our notion of planning with nested belief, and subsequently demonstrate how to automatically convert such problems into problems that appeal to classical planning technology. Our approach represents an important first step towards applying the well-established field of automated planning to the challenging task of planning involving nested beliefs of multiple agents.

1 Introduction

AI applications increasingly involve the interaction of multiple agents – be they intelligent user interfaces that interact with human users, gaming systems, or multiple autonomous robots interacting together in a factory setting. In the absence of prescribed coordination, it is often necessary for individual agents to synthesize their own plans, taking into account not only their own capabilities and beliefs about the world but also their beliefs about other agents, including what each of the agents will come to believe as the consequence of the actions of others. To illustrate, consider the scenario where Larry and Moe plan to work together on an assembly task. Each knows what needs to be done and can plan accordingly. Unbeknownst to Moe, Larry decides to start the job early. Larry believes that Moe believes that assembly has not yet commenced. As a consequence, Larry's plan must include a communication action to inform Moe of the status of the assembly when Moe arrives.

In this paper, we examine the problem of synthesizing plans in such settings. In particular, given a finite set of agents, each with: (1) (possibly incomplete and incorrect) beliefs about the world and about the beliefs of other agents; and (2) differing capabilities including the ability to perform actions whose outcomes are unknown to other agents; we are interested in synthesizing a plan to achieve a goal condition. Planning is at the belief level and as such, while we consider the execution of actions that can change the state of the world (ontic actions) as well as an agent's state of knowledge or belief (epistemic or more accurately doxastic actions, including communication actions), all outcomes are with respect to belief. Further, those beliefs respect the KD45$_n$ axioms of epistemic logic (Fagin et al. 1995). Finally, we take a *perspectival view*, planning from the viewpoint of a single agent. We contrast this with traditional multi-agent planning which generates a coordinated plan to be executed by multiple agents (e.g., (Brenner and Nebel 2009)).

We focus on computational aspects of this synthesis task, leaving exploration of interesting theoretical properties to a companion paper. To this end, we propose a means of encoding a compelling but restricted subclass of our synthesis task as a classical planning problem, enabling us to exploit state-of-the-art classical planning techniques to synthesize plans for these challenging planning problems. Our approach relies on two key restrictions: (1) we do not allow for disjunctive belief; and (2) the depth of nested belief is bounded. A key aspect of our encoding is the use of *ancillary conditional effects* – additional conditional effects of actions which enforce desirable properties such as epistemic modal logic axioms (cf. Section 3), and allow domain modellers to encode conditions under which agents are mutually aware of actions (cf. Section 4). By encoding modal logic axioms as effects of actions, we are using our planner to perform epistemic reasoning. As such, our planning machinery additionally supports answering queries involving the nested beliefs of agents (cf. Section 5): e.g, "Does Agent 1 believe that Agent 2 believes they can achieve the goal?".

Computational machinery for epistemic reasoning has historically appealed to theorem proving or model checking (e.g., (van Eijck 2004)), while epistemic planning, recently popularized within the *Dynamic Epistemic Logic* (DEL) community, has largely focused on theoretical concerns (e.g., (Löwe, Pacuit, and Witzel 2011)). The work presented here is an important first step towards leveraging state-of-the-art planning technology to address rich epistemic planning problems of the sort examined by the DEL

*A version of this paper also appears in the Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI-15).

community. Indeed, we can readily solve existing examples in the DEL literature (cf. Section 6). We further discuss the relationship of our work to other work in epistemic reasoning and planning at the end of this paper.

Example 1 (Grapevine)**.** We will use a common example to explain the concepts introduced throughout the paper. Consider a scenario where a group of agents each have their own secret to (possibly) share with one another. Each agent can move freely between a pair of rooms, and broadcast any secret they currently believe to everyone in the room. Initially they only believe their own unique secret. Goals we might pose include the universal spread of information (everyone believes every secret), misconception (an agent holds a false belief about someone else's belief), etc. We will use $1, 2, \cdots$ to represent the agents, and $s_1, s_2, \cdots$ to represent their secrets, respectively.

2 Specification

The general aim of this work is to address problems similar to DEL planning (Bolander and Andersen 2011) using the computational machinery of automated planning. We use DEL to formally specify our planning system. Our presentation below is terse, and we refer interested readers to van Ditmarsch, van der Hoek, and Kooi (2007) for a more comprehensive overview.

Let $\mathcal{P}$, $\mathcal{A}$, and Ag respectively be finite sets of propositions, actions, and agents. The set of well-formed formulae, $\mathcal{L}$, for DEL is obtained from the following grammar:

$$\phi ::= p \mid \phi \wedge \phi' \mid B_i\phi \mid [\alpha]\phi \mid \neg\phi$$

in which $p \in \mathcal{P}$, $\alpha \in \mathcal{A}$, and $i \in Ag$. $B_i\phi$ should be interpreted as "agent i believes ϕ." The semantics is given using *Kripke structures* (Fagin et al. 1995). Given a world w, standing for some state of affairs, such a structure determines (by means of an *accessibility relation*) the worlds that an agent considers possible when at w. (That is, the agent is unsure which world it is truly in). A *model* M is the set of all worlds, an accessibility relation between these worlds for each agent i, and a function specifying which propositions are true in each world. Informally, the meaning of formulas wrt a pair (M, w) is as follows: p holds if it is true in w, $\phi \wedge \psi$ holds if both ϕ and ψ hold, $\neg\phi$ holds if ϕ does not hold at (M, w), $B_i\phi$ if ϕ holds in all worlds agent i considers possible at w, and $[\alpha]\phi$ holds if ϕ holds after *applying* action α to (M, w). The semantics is defined formally in terms of $\models$, where $M, w \models \phi$ means that ϕ holds in world w for model M.

As discussed by Fagin et al. (1995), constraints on Kripke structures lead to particular properties of belief. If the Kripke structure is *serial*, *transitive*, and *Euclidean* we obtain (arguably) the most common properties of belief:

K	$B_i\phi \wedge B_i(\phi \supset \psi) \supset B_i\psi$	(Distribution)
D	$B_i\phi \supset \neg B_i\neg\phi$	(Consistency)
4	$B_i\phi \supset B_iB_i\phi$	(Positive introspection)
5	$\neg B_i\phi \supset B_i\neg B_i\phi$	(Negative introspection)

These axioms collectively form the system referred to as $KD45_n$, where n specifies that there are multiple agents in the environment. From the axioms, additional theorems can be derived. For example, in this work, we use the following theorems for reducing neighbouring belief modalities involving the same agent into a single belief modality:

$$B_iB_i\phi \equiv B_i\phi \qquad B_i\neg B_i\phi \equiv \neg B_i\phi$$
$$\neg B_i\neg B_i\phi \equiv B_i\phi \qquad \neg B_iB_i\phi \equiv \neg B_i\phi$$

We can now define a planning problem as follows:

Definition 1. Multi-Agent Epistemic Planning Problem

A ***m**ulti-agent **e**pistemic **p**lanning* (MEP) problem $\mathcal{D}$ is a tuple of the form $\langle \mathcal{P}, \mathcal{A}, Ag, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{P}$, $\mathcal{A}$, and Ag are as above, $\mathcal{I}$ is the *initial theory*, and $\mathcal{G}$ is the *goal condition.* Each $a \in \mathcal{A}$ is assumed to be of the form $\langle \pi, \{(\gamma_1, l_1), \ldots, (\gamma_k, l_k)\} \rangle$, where π is called the *precondition* of a, γ_i is called the *condition* of a *conditional effect*, and l_i is called the *effect* of a conditional effect. Finally, we assume $\mathcal{G}$, $\mathcal{I}$, π, γ_i, and l_i are all well-formed formulae over $\mathcal{L}$, excluding the $[\alpha]$ modality.

Following Reiter (2001) and van Ditmarsch, van der Hoek, and Kooi (2007), the above action formalization can be expressed as standard *precondition* and *successor state axioms*, which would then define the meaning of $[\alpha]\phi$ in DEL. By extension, we say that given a domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A}, Ag, \mathcal{I}, \mathcal{G} \rangle$, the sequence of actions $a_1, \ldots, a_k$ *achieves* $\mathcal{G}$ iff for any (M, w) such that $M, w \models \mathcal{I}$, we have $M, w \models [a_1] \ldots [a_k]\mathcal{G}$. Thus, the *plan synthesis task* is one of finding a sequence of actions $\vec{a}$ that achieves the goal condition $\mathcal{G}$.

Not surprisingly, reasoning (and planning) in these logical frameworks is computationally challenging (Fagin et al. 1995; Aucher and Bolander 2013). In this work, we limit our attention to a planning framework described using a fragment of epistemic logic. First, we consider reasoning from the perspective of a single *root agent*; this is a perspectival view of the world. Second, we do not allow disjunctive formulae as a belief. Following Lakemeyer and Lespérance (2012), we define a *restricted modal literal* (RML) as one obtained from the following grammar:

$$\phi ::= p \mid B_i\phi \mid \neg\phi$$

where $p \in \mathcal{P}$ and $i \in Ag$. The depth of an RML is defined as: $depth(p) = 0$ for $p \in \mathcal{P}$, $depth(\neg\phi) = depth(\phi)$ and $depth(B_i\phi) = 1 + depth(\phi)$. We will view a conjunction of RMLs equivalently as a set, and denote the set of all RMLs with bounded depth d for a group of agents Ag as $\mathcal{L}_{RML}^{Ag,d}$.

We define a ***r**estricted **p**erspectival* multi-agent epistemic planning problem (RP-MEP problem) for depth bound d and the root agent $\star \in Ag$ as a MEP problem with the additional restrictions that: (1) every RML is from the perspective of the root agent – i.e., it is from the following set:

$$\{B_\star\phi \mid \phi \in \mathcal{L}_{RML}^{Ag,d}\} \cup \{\neg B_\star\phi \mid \phi \in \mathcal{L}_{RML}^{Ag,d}\},$$

and (2) there is no disjunctive belief: the initial theory, goal specification, and every precondition are sets of positive RMLs (i.e., no negated belief), every effect is a single RML, and every effect condition is a set of RMLs.

We focus on the class of RP-MEP problems with an aim to extend our work to the more general class in the future.

We address the planning problem from the view of an acting agent, where the designated root agent $\star$ is the one for which we plan. Intuitively, this means that conditional effects are formulated in the context of the root agent; e.g., we would have a conditional effect of the form $(\{B_\star\gamma\}, B_\star l)$ for action a in a RP-MEP problem to capture the fact that the root agent will believe l if it believed γ before a occurred.

This admits a rich class of planning problems; e.g., it is reasonable to assume that the root agent's view of the world differs from what a particular agent i believes, and so another conditional effect of a might be $(\{B_\star\gamma\}, B_\star B_i\neg l)$ – even though the root agent believes doing a would make l true if γ holds, the root agent believes that i will believe $\neg l$ if γ holds. In particular, this is easily shown to generalize a standard assumption in the literature (Liu and Wen 2011) that all agents hold the same view of what changes after actions occur.

In the next section, we show how restricted perspectival multi-agent epistemic planning problems can be represented as a classical planning problem, where the key insight is to encode reasoning features (such as deduction in $KD45_n$) as *ramifications* realized using ordinary planning operators.

3 A Classical Encoding

In this section, we present our model for planning with nested belief in a classical planning setting. We assume that the state of the world represents the mental model of a particular agent, perhaps an omniscient agent, that perceives an environment that includes all other agents. As a result, all reasoning is from the perspective of this single agent. The fluents that are true in a state correspond to the RMLs that the agent believes, while the fluents that are false correspond to the RMLs that the agent does not believe. Action execution, then, is predicated on the agent *believing* that the preconditions are satisfied. Similarly, the mental model of the agent is updated according to the effects of an action. Note that we do not need to enforce a separation of ontic and epistemic effects – the same action can update belief about propositions as well as RMLs. This is due to the interpretation that the state of the world represents the mental model of a given agent: every effect is epistemic in this sense.

The remainder of the section will proceed as follows:

1. We present the framework for our encoding of an RP-MEP problem into classical planning.
2. We specify how the state of the world is updated to maintain the deductive closure of the agent's belief.
3. We describe how to address the situation when an agent is uncertain if a conditional effect fires (e.g. due to lack of knowledge), and how the agent removes the corresponding beliefs from its knowledge base.

Items 2 and 3, in particular, enable the introspection capabilities and the change in beliefs after actions in standard epistemic frameworks (Aucher and Bolander 2013).

3.1 Encoding RP-MEP

We begin by providing a quick background on the classical planning formalism we use. A classical planning problem consists of a tuple $\langle F, I, G, O\rangle$, where F is a set of fluent atoms, I is a complete setting of the fluents describing the initial state, G is a set of fluents describing the goal condition, and O is a set of operators. A *state* s is a subset of the fluents F with the interpretation that atoms not in s are false. Every operator $o \in O$ is a tuple $\langle Pre_o, \mathit{eff}^+_o, \mathit{eff}^-_o\rangle$, and we say that o is applicable in s iff $Pre_o \subseteq s$. The set eff^+_o (resp. eff^-_o) contains conditional effects describing the fluent atoms that should be added (resp. removed) from the state when applying the operator. Finally, every conditional effect in eff^+_o or eff^-_o is of the form $(C \to l)$ where C is the condition for the effect and l is a fluent that is the result of the effect. The condition C consists of a tuple $\langle C^+, C^-\rangle$ where C^+ is the set of fluents that must hold and C^- the set of fluents that must not hold. A conditional effect $(\langle C^+, C^-\rangle \to l)$ *fires* in state s iff $C^+ \subseteq s$ and $C^- \cap s = \emptyset$. Assuming o is applicable in s, and $\mathit{eff}^+_o(s)$ (resp. $\mathit{eff}^-_o(s)$) are the positive (resp. negative) conditional effects that fire in state s, the state of the world s' after applying o is defined as follows:

$$s' = s \setminus \{l \mid (C \to l) \in \mathit{eff}^-_o(s)\} \cup \{l \mid (C \to l) \in \mathit{eff}^+_o(s)\}$$

Our account of classical planning mirrors the standard representation (see, for example, (Ghallab, Nau, and Traverso 2004)), with the exception that we make explicit the fluent atoms that are added, deleted, required to be in, or required to be absent from the state of the world. This simplifies the exposition when we encode nested beliefs as a classical planning problem. Intuitively, every RML in $\mathcal{L}^{Ag,d}_{RML}$ will correspond to a single fluent in F (e.g., both $B_1 p$ and $\neg B_1 p$ will become fluents), and the operators will describe how the mental model of our root agent should be updated. Formally, we define the classical encoding of a RP-MEP problem as follows:

Definition 2. Classical Encoding of RP-MEP

Let $\mathcal{B}_i$ and $\mathcal{N}_i$ be functions that map i's positive (resp. negative) belief from a set of RMLs KB to the respective fluents:

$$\mathcal{B}_i(KB) = \{l_\phi \mid B_i\phi \in KB\}$$
$$\mathcal{N}_i(KB) = \{l_\phi \mid \neg B_i\phi \in KB\}$$

Given a RP-MEP problem, $\langle \mathcal{P}, \mathcal{A}, Ag, \mathcal{I}, \mathcal{G}\rangle$ and a bound d on the depth of nested belief we wish to consider, we define the classical encoding as the tuple $\langle F, I, G, O\rangle$ such that:

$$F \stackrel{\text{def}}{=} \{l_\phi \mid \phi \in \mathcal{L}^{Ag,d}_{RML}\} \qquad I \stackrel{\text{def}}{=} \mathcal{B}_\star(\mathcal{I}) \qquad G \stackrel{\text{def}}{=} \mathcal{B}_\star(\mathcal{G})$$

and for every action $\langle \pi, \mathit{effects}\rangle$ in $\mathcal{A}$, we have a corresponding operator $\langle Pre_o, \mathit{eff}^+_o, \mathit{eff}^-_o\rangle$ in O such that:

$$Pre_o \stackrel{\text{def}}{=} \mathcal{B}_\star(\pi)$$
$$\mathit{eff}^+_o \stackrel{\text{def}}{=} \{(\langle \mathcal{B}_\star(\gamma_i), \mathcal{N}_\star(\gamma_i)\rangle \to l_\phi) \mid (\gamma_i, B_\star\phi) \in \mathit{effects}\}$$
$$\mathit{eff}^-_o \stackrel{\text{def}}{=} \{(\langle \mathcal{B}_\star(\gamma_i), \mathcal{N}_\star(\gamma_i)\rangle \to l_\phi) \mid (\gamma_i, \neg B_\star\phi) \in \mathit{effects}\}$$

3.2 Maintaining the Deductive Closure

Because of the direct correspondence, we will use the RML notation and terminology for the fluent atoms in F. The encoding, thus far, is a straight-forward adaptation of the RP-MEP definition that hinges on two properties: (1) there is

a finite bound on the depth of nested belief; and (2) we restrict ourselves to representing RMLs and not arbitrary formulae. Crucially, however, we wish to maintain the assumption that the agents are internally consistent with respect to KD45$_n$. To accomplish this, we define a closure procedure, Cl, that deduces a new set of RMLs from an existing one under KD45$_n$:

Definition 3. RML Closure

Given an RML l, we define $Cl(l)$ to be the set of KD45$_n$ logical consequences of l computed as follows:

1. Rewrite l into *negation normal form* (NNF) (Bienvenu 2009), which is the equivalent formula in which negation appears only in front of propositional variables. For this we introduce an operator F s.t. $F_i\phi \equiv \neg B_i \neg \phi$.
2. Repeatedly apply the D axiom ($B_i\psi \supset F_i\psi$) to the NNF, resulting in the set of all RMLs that follow logically from ϕ using the D axiom. This can be done by simply replacing all combinations of occurrences of B_i with F_i; e.g., for the RML $B_iF_jB_kp$, the resulting set would be $\{F_iF_jB_kp,\ B_iF_jF_kp,\ F_iF_jF_kp\}$.
3. Invert the NNF by replacing all instances of $F_i\psi$ with $\neg B_i \neg \psi$ and eliminating double negation.

Note that to calculate the closure, we do not apply the positive (4) or negative (5) introspection actions of KD45$_n$, due to the equivalences in KD45$_n$ mentioned in Section 2. Proving the completeness of our RML closure is beyond the scope of this paper, but the soundness follows directly from the K and D axioms of KD45$_n$, and from the sound NNF rewriting rule (Bienvenu 2009). Further details on maintaining KD45$_n$ consistency can be found in Muise et al. (2015).

Along with the requirement that an agent should never believe an RML and its negation, we have the following state constraints for the encoded planning problem:

$$\phi \in s \Rightarrow \neg\phi \notin s$$
$$\phi \in s \Rightarrow \forall \psi \in Cl(\phi), \psi \in s$$

The enforcement of such state constraints can either be achieved procedurally within the planner, or representationally. We choose the latter, appealing to a solution to the well-known ramification problem (e.g., (Pinto 1999; Lin and Reiter 1994)), representing these state constraints as *ancillary conditional effects* of actions that enforce the state constraints. The correctness of the resulting encoding is predicated on the assumption that the domain modeller provided a consistent problem formulation. The ancillary conditional effects for operator o are as follows:

$$(C \rightarrow l) \in eff_o^+ \Rightarrow (C \rightarrow \neg l) \in eff_o^- \quad (1)$$
$$(C \rightarrow l) \in eff_o^+ \Rightarrow \forall l' \in Cl(l),\ \ (C \rightarrow l') \in eff_o^+ \quad (2)$$

Example 2. Returning to our example, consider the effect of agent 1 telling secret s_1 to agent 2. Assuming there is no positive or negative condition for this effect to fire, the effect would be $(\langle\emptyset,\emptyset\rangle \rightarrow B_2s_1) \in eff^+$. Using (1) would create $(\langle\emptyset,\emptyset\rangle \rightarrow \neg B_2s_1) \in eff^-$ and (2) would create $(\langle\emptyset,\emptyset\rangle \rightarrow \neg B_2\neg s_1) \in eff^+$. Subsequently, (1) would fire again creating $(\langle\emptyset,\emptyset\rangle \rightarrow B_2\neg s_1) \in eff^-$. We can see already, with this simple example, that effects may cascade to create new ones.

3.3 Uncertain Firing and Removing Beliefs

To complete the faithful transformation of a RP-MEP problem to a classical problem, we must also consider the axioms that hold when updating the state due to the occurrence of an action. In particular, the following two issues remain: (1) the frame problem is solved only partially when we use the procedure listed above for updating a state in the encoded domain, and (2) removing belief about an RML may invalidate the state from being closed under KD45$_n$ (e.g., removing $\neg B_i\neg p$ while B_ip currently holds).

For the first issue, we appeal to a common technique in planning under uncertainty (e.g., (Petrick and Levesque 2002; Palacios and Geffner 2009)): when the conditions of a positive conditional effect are not believed to be false, the negation of the effect's result can no longer be believed. Intuitively, if an agent is unsure whether a conditional effect fires then it must consider the condition's effect possible, and thus no longer believe the negation of the effect. We create the following additional conditional effects for operator o:

$$(\langle C^+, C^-\rangle \rightarrow l) \in eff_o^+ \Rightarrow$$
$$(\langle\emptyset, \{\neg\phi \mid \phi \in C^+\} \cup C^-\rangle \rightarrow \neg l) \in eff_o^- \quad (3)$$

The second issue is to ensure the state remains closed under KD45$_n$. If we remove an RML l, we should also remove any RML that could be used to deduce l. To compute the set of such RMLs, we use the contrapositive: $\neg l'$ will deduce l if and only if $\neg l$ deduces l' (i.e., $l' \in Cl(\neg l)$). We thus have the following additional conditional effects for operator o:

$$(C \rightarrow l) \in eff_o^- \Rightarrow \forall l' \in Cl(\neg l),\ (C \rightarrow \neg l') \in eff_o^- \quad (4)$$

Example 3. Consider a conditional effect for the action of agent 1 sharing their secret that stipulates if we, the root agent, think agent 1 is trustworthy (denoted as t_1), then we would believe agent 1's secret: $(\langle\{t_1\},\emptyset\rangle \rightarrow s_1) \in eff^+$. Using (3), we would derive the new negative effect $(\langle\emptyset,\{\neg t_1\}\rangle \rightarrow \neg s_1) \in eff^-$. Intuitively, if we are unsure about agent 1's trustworthiness, then we are unsure about their secret being false. On the other hand, consider the effect of an action informing us that we should no longer believe that agent 1 does not believe agent 2's secret: $(\langle\emptyset,\emptyset\rangle \rightarrow \neg B_1s_2) \in eff^-$. Using (4), we would have the additional effect $(\langle\emptyset,\emptyset\rangle \rightarrow B_1\neg s_2) \in eff^-$. If $B_1\neg s_2$ remained in our knowledge base, then so should $\neg B_1s_2$ assuming that our knowledge base is deductively closed.

With these extra conditional effects, we have a faithful encoding of the original RP-MEP problem.

Theorem 1. Our encoding is sound and complete with respect to RP-MEP. That is, a plan $\vec{o}$ will be found for a goal G from initial state I using our encoding if and only if $M, w \models \mathcal{I}$ implies $M, w \models [\vec{a}]\mathcal{G}$ for any (M, w), where M satisfies $KD45_n$ and $\vec{a}$ is the action sequence corresponding to $\vec{o}$.

Space precludes a formal proof, but this result follows from (1) the correctness of the newly added conditional effects, (2) the soundness and completeness of the Cl procedure, and (3) the use of a solution to the ramification problem to compile the properties into conditional effects.

4 Conditioned Mutual Awareness

Our specification of a RP-MEP problem and the subsequent encoding into classical planning allow us to specify a rich set of actions. Unlike traditional approaches that compile purely ontic action theories into ones that deal with belief (e.g., the work on conformant planning by Palacios and Geffner (2009)), we allow for arbitrary conditional effects that include nested belief both as conditions and as effects.

While expressive, manually encoding effects with nested belief can be involved due to the cascading of ancillary conditional effects. Here, we extend the scope of ancillary conditional effects to safely capture a common phenomenon in planning with nested belief: that of agents being mutually aware of the effects of actions.

Example 4. In our running example, if an agent enters a room, then we realize this as an effect: e.g., $(\langle\emptyset,\emptyset\rangle \rightarrow at\_1\_loc1) \in eff^{+}$. In many applications, other agents may also be aware of this: e.g., $(\langle\emptyset,\emptyset\rangle \rightarrow B_2at\_1\_loc1) \in eff^{+}$. Perhaps we wish to predicate this effect on the second agent believing that it is also in this room: e.g., $(\langle\{B_2at\_2\_loc1\},\emptyset\rangle \rightarrow B_2at\_1\_loc1) \in eff^{+}$. It is this kind of behaviour of conditioned mutual awareness that we would like to capture in a controlled but automated manner.

By appealing to ancillary conditional effects, we will create new effects from existing ones. We have already demonstrated the ancillary conditional effects required for a faithful encoding to adhere to the axioms and state constraints we expect from our agent. We extend this idea here to capture the appealing property of *conditioned mutual awareness*. For simplicity, we describe conditioned mutual awareness in terms of the encoded problem, but assume the conditions for mutual awareness are optionally provided with a RP-MEP problem.

Definition 4. Condition for Awareness

We define $\mu_i^o \in F$ to be the condition for agent i to be aware of the effects of operator o. If need be, μ_i^o may be a unique fluent that is either always believed or never believed.

Intuitively, we want to assume that agent i is aware of every conditional effect of o only when agent i believes μ_i^o.

For a given set of fluents T, we define the shorthand $B_iT = \{B_il \mid l \in T\}$ and $\neg B_iT = \{\neg B_il \mid l \in T\}$ and model conditioned mutual awareness through the following two encoding rules for every agent $i \in Ag$ to derive new conditional effects:

$$(\langle C^+, C^-\rangle \rightarrow l) \in eff_o^+ \Rightarrow (\langle B_iC^+ \cup \neg B_iC^- \cup \{B_i\ \mu_i^o\ \},\emptyset\rangle \rightarrow B_il) \in eff_o^+ \quad (5)$$

$$(\langle C^+, C^-\rangle \rightarrow l) \in eff_o^- \Rightarrow (\langle B_iC^+ \cup \neg B_iC^- \cup \{B_i\ \mu_i^o\ \},\emptyset\rangle \rightarrow \neg B_il) \in eff_o^+ \quad (6)$$

Note that each form of ancillary conditional effect adds a new positive conditional effect. In the positive case, we believe that the agent i has a new belief B_il if we believe that agent i had the prerequisite belief for the effect to fire. In the negative case, we would believe that the agent no longer holds the belief, but because we take a perspectival view, it is encoded as a positive conditional effect – i.e., we would believe $\neg B_il$. For instance, the ancillary conditional effect from our working example says that we should no longer believe the negation of agent 1's secret if we do not believe agent 1 is untrustworthy (see Example 3), which would create the following ancillary conditional effect:

$$(\langle\emptyset,\{\neg t_1\}\rangle \rightarrow \neg s_1) \in eff^- \Rightarrow (\langle\{\neg B_2\neg t_1\},\emptyset\rangle \rightarrow \neg B_2\neg s_1) \in eff^+.$$

We restrict the application of the above rules by applying them only if the following two conditions are met: (1) every RML in the newly created effect has a nested depth smaller than our bound d; and (2) if we are applying the above rule for agent i to a conditional effect $(C \rightarrow l) \in eff_o^-$, then $l \notin \{B_il', \neg B_il'\}$. The first restriction bounds the number of conditional effects while the second prevents unwanted outcomes from introspection. To see why this exception is required, consider the example of a pair of conditional effects for an action where we discover agent 1 may or may not believe s_2 (i.e., we should forget any belief about what agent 1 believes regarding s_2). Omitting μ_1^o for clarity, we have the following *negative* conditional effects:

$$(\langle\emptyset,\emptyset\rangle \rightarrow \neg B_1s_2) \quad (\langle\emptyset,\emptyset\rangle \rightarrow B_1s_2)$$

If we were to apply the above rules with agent 1, we would add two *positive* ancillary conditional effects:

$$(\langle\emptyset,\emptyset\rangle \rightarrow \neg B_1\neg B_1s_2) \quad (\langle\emptyset,\emptyset\rangle \rightarrow \neg B_1B_1s_2)$$

which subsequently would simplify to the following conditional effects (given that we combine successive modalities of the same agent index under $KD45_n$):

$$(\langle\emptyset,\emptyset\rangle \rightarrow B_1s_2) \quad (\langle\emptyset,\emptyset\rangle \rightarrow \neg B_1s_2)$$

Thus, the resulting effects would indicate that the agent reaches an inconsistency with its own belief. To avoid this issue, we apply rule (6) only when the effect is not a belief (negative or positive) of the corresponding agent.

Because we can assume that the specification of conditioned mutual awareness is given and computed in the original RP-MEP specification, Theorem 1 continues to hold.

5 Projection to Reason As Others

It is natural that an agent may want to reason about what another believes or may come to believe, allowing queries such as, "Does Sue believe that Bob believes that Sue believes a plan exists?". We construe the term *virtual agent* to be the list of agents we wish to have the root agent reason as. Here, [Bob, Sue] is the virtual agent assuming that Sue is the root agent: i.e., we want Sue to reason as if she was Bob reasoning as if he was Sue.

To reason as a virtual agent, we require two items: (1) the assumed mental model of the virtual agent's initial state, and (2) the virtual agent's view of the operators. We assume that the goal, set of agents, and operator preconditions remain the same. To create the new initial state for the virtual agent, we use projection:

Definition 5. Agent Projection

Given a state s, the *agent projection* of s with respect to a vector of agents $\vec{Ag}$, denoted as $Proj(s, \vec{Ag})$, is defined as:

$$\begin{cases} \{\phi \mid B_i\phi \in s\} & \text{if } \vec{Ag} = [i] \\ Proj(Proj(s, [i]), \vec{Ag}') & \text{if } \vec{Ag} = [i] + \vec{Ag}' \end{cases}$$

Essentially, agent projection repeatedly filters the set of RMLs according to the appropriate agent and strips the belief modality from the front of the RML. When projecting a planning problem, we project the initial state using agent projection – giving us the believed mental state of the virtual agent – and additionally project the effects of every operator. Because we allow for heterogeneous agents with respect to their view on operator effects, we first must decide which conditional effects to keep for the projection. For a particular agent i, these are the effects *uniform in* i.

Definition 6. Uniform Conditional Effect

We say that an RML is uniform in i if the RML is a condition for awareness or it begins with either the modality B_i or $\neg B_i$. A set of fluents is uniform in i iff every RML in the set is uniform in i. Finally, the set of conditional effects uniform in i for operator o are all those $(\langle C^+, C^- \rangle \rightarrow l) \in eff_o^+$ such that C^+ is uniform in i, l is uniform in i, and $C^- = \emptyset$.

The projection of an operator for agent i will retain all those conditional effects that are uniform in i. Note that this discards all negative conditional effects. Once we have the set of uniform conditional effects, we project each effect $(\langle C^+, \emptyset \rangle \rightarrow l)$ in the set for the agent i to be defined as:

$$(\ \langle\ \{\phi \mid B_i\phi \in C^+\},\ \{\phi \mid \neg B_i\phi \in C^+\}\ \rangle \rightarrow l'),$$

where $l = B_i l'$ and the projected effect is in eff_o^+ or $l = \neg B_i l'$ and the projected effect is in eff_o^-.

The intuition behind our definition of conditional effects uniform for agent i, is that we consider only those effects that we (the current agent) believe agent i will reason with. If a conditional effect has a negative condition (i.e., C^- is non-empty), then that is a condition that involves our own lack of belief and not the lack of belief for agent i (the latter would exist as an RML starting with $\neg B_i$ in C^+). Similarly, negative conditional effects describe how *we* remove belief, and not how agent i would update their belief. Paired with the ability to add conditioned mutual awareness, the projection of effects for a particular agent can target precisely those effects we want to keep.

We generate a new initial state for a particular virtual agent using the agent projection procedure, and we generate a new set of operators by repeatedly applying the above procedure for operator projection. Additionally, because we assume that the nestings of modalities of the same agent index are combined (cf. Section 2), we process the operator preconditions and goal slightly to accommodate for the new root agent perspective assumed for the final agent i in the virtual agent list: B_i is removed from the start of precondition and goal RMLs while any RML of the form $\neg B_i\phi$ is converted to a negative precondition or goal RML ϕ. The strength in projecting away effects is that it can simplify the domain greatly – any conditional effect not uniform in the projected agent will be pruned from the domain prior to planning. Combining ancillary conditional effects and projection allows us to answer a complex suite of queries for the nested belief of agents.

6 Preliminary Evaluations

We implemented the scheme above to convert a RP-MEP planning problem into a classical planning problem, which can be subsequently solved by any planner capable of handling negative preconditions and conditional effects. The compiler consumes a custom format for the RP-MEP problems and can either simulate the execution of a given action sequence or a classical planner built using the SIW$^+$ and BFS$_f$ planners (Lipovetzky and Geffner 2012) found in the LAPKT planning library (Ramirez and Lipovetzky 2015). The source code, benchmark problems, and demo of the compilation process can be found online at:

`http://www.haz.ca/research/pdkb`

We have verified the model of the pre-existing *Thief* problem, and all of the existing queries considered in the previous literature posed to demonstrate the need for nested reasoning (e.g., those found in (Löwe, Pacuit, and Witzel 2011)) are trivially solved in a fraction of a second. As a more challenging test-bed, we modelled a setting that combines the *Corridor* problem (Kominis and Geffner 2014) and the classic *Gossip* problem (Entringer and Slater 1979). In the new problem, *Grapevine*, there are two rooms with all agents starting in the first room. Every agent believes their own secret to begin with, and the agents can either move between rooms or broadcast a secret they believe. Movement is always observed by all, but through the use of conditioned mutual awareness the sharing of a secret is only observed by those in the same room. This problem allows us to pose a

Problem	$\lvert Ag\rvert$	d	$\lvert F\rvert$	$\lvert\vec{o}\rvert$	**Time (s)**	
					Plan	**Total**
Corridor	3	1	70	5	0.01	0.11
	7	1	150	5	0.01	0.21
	3	3	2590	5	0.05	6.85
Grapevine	4	1	216	10	0.04	0.27
	3	2	774	4	0.09	1.84
	4	2	1752	4	0.70	6.61

Table 1: Results for various Corridor and Grapevine problems. Ag, d, F, and $\vec{o}$ are as above. Plan time is the time spent solving the encoded problem, while Total time additionally includes the encoding and parsing phases.

variety of interesting goals ranging from private communication (similar to the Corridor problem) to goals of misconception in the agent's belief (e.g., $G = \{B_a s_b, B_b \neg B_a s_b\}$).

As a preliminary investigation, we varied some of the discussed parameters and report on the results in Table 1 (the first Corridor problem corresponds to the one presented by Kominis and Geffner). The largest bottleneck stems from the depth of nested knowledge, as the number of newly introduced fluents is exponential in d. The planning process is typically fast, and moving forward we hope to reduce the compilation time by only generating fluents and conditional effects that are relevant to achieving the goal.

7 Related Work

There is a variety of research related to the ideas we have presented, and we cover only the most closely related here. Research into DEL (van Ditmarsch, van der Hoek, and Kooi 2007), and more recently DEL planning (e.g., (Bolander and Andersen 2011)), deals with how to reason about knowledge or belief in a setting with multiple agents. Focus in this area is primarily on the logical foundation for updating an epistemic state of the world according to physical (ontic) and non-physical (epistemic) actions, as well as identifying the classes of restricted reasoning that are tractable from a theoretical standpoint (Löwe, Pacuit, and Witzel 2011). While DEL techniques are more expressive than our approach in terms of the logical reasoning that an agent can achieve in theory, this expressiveness increases the computational complexity of the reasoning. In particular, the practical synthesis of DEL plans remains an unsolved problem.

On the other end of the spectrum, a range of techniques exist for planning with partial observability (e.g., (Brafman and Shani 2012; Bonet and Geffner 2014)). These techniques typically represent the individual knowledge of facts about the world (as opposed to belief): the agent can "know p holds" (i.e., Kp), "know p does not hold" (i.e., $K\neg p$), or "not know the value of p" (i.e., $\neg Kp \wedge \neg K\neg p$), but do not extend to the multi-agent case. The use of a knowledge modality was extended to be predicated on assumptions about the initial state, leading to effective techniques for conformant and contingent planning (Palacios and Geffner 2009; Albore, Palacios, and Geffner 2009).

Others have investigated restricted classes of syntactic knowledge bases for tractable reasoning, including the 0-approximation semantics (Baral and Son 1997), and the notion of *Proper Epistemic Knowledge Bases* (PEKBs) (Lakemeyer and Lespérance 2012). Like ours, these works restrict the type of knowledge that is stored about a single agent, such as not permitting disjunctive knowledge (e.g., the agent "knows either p or q holds"). PEKBs extend this to the multi-agent setting, and in a sense the preconditions, goals, and states of our work can be viewed as PEKBs.

The work most related to ours is that of Kominis and Geffner (2014). They share the same general motivation of bridging the rich fields of epistemic reasoning and automated planning by using classical planning over multi-agent epistemic states. However, the two approaches are fundamentally different and as a result each comes with its own strengths and shortcomings. The largest difference is our choice to focus on belief rather than knowledge – for us, modelling the possibility of incorrect belief is essential. Kominis and Geffner assume that all agents start with common initial knowledge, and most strongly that all action effects are commonly known to all agents (while we can model this, it is not required). Conversely, they are able to handle arbitrary formulae, including disjunctive knowledge, while we are restricted to reasoning with RMLs. Moving forward, we hope to explore how we can combine ideas from both approaches.

8 Concluding Remarks

We have presented a model of planning with nested belief, and demonstrated how a syntactically restricted subclass of this expressive problem can be compiled into a classical planning problem. Despite the restricted form, we are able to model complex phenomena such as public or private communication, commonly observed action effects, and non-homogeneous agents (each with their own view of how the world changes). Our focus on belief (as opposed to knowledge) provides a realistic framework for an agent to reason about a changing environment where knowledge cannot be presumed. We have additionally demonstrated how to pose queries *as if* we were other agents, taking our belief of the other agents into account. To solve this expressive class of problems, we appeal to existing techniques for dealing with ramifications, and compile the problem into a form that classical planning can handle.

In future work, we hope to expand the work in two key directions. First, we would like to explore other forms of ancillary conditional effects similar to the conditioned mutual awareness to give the designer greater flexibility during modelling (e.g., with concepts such as teamwork protocols or social realities). Second, we want to formalize the connection between general multi-agent epistemic planning and the syntactic restriction that we focus on encoding. We hope to provide an automated sound (but incomplete) approximation of an arbitrary MEP problem into a RP-MEP problem.

Amendments This paper differs from the published AAAI work in the following ways: (1) a new classical planner was used and the implementation improved to obtain the new results reported in Table 1; (2) reference to the online repository and demo was added; (3) reference to our followup work concerning the theoretical properties of belief consistency was added; and (4) minor formatting issues were fixed.

Acknowledgements This research is partially funded by Australian Research Council Discovery Grant DP130102825, *Foundations of Human-Agent Collaboration: Situation-Relevant Information Sharing*, and by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *IJCAI 2009, Proceedings of the 21st International Joint Confer-*

ence on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009, 1623–1628.

Aucher, G., and Bolander, T. 2013. Undecidability in epistemic planning. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*.

Baral, C., and Son, T. C. 1997. Approximate reasoning about actions in presence of sensing and incomplete information. In *Logic Programming, Proceedngs of the 1997 International Symposium, Port Jefferson, Long Island, NY, USA, October 13-16, 1997*, 387–401.

Bienvenu, M. 2009. Prime implicates and prime implicants: From propositional to modal logic. *Journal of Artificial Intelligence Research* 36(1):71–128.

Bolander, T., and Andersen, M. B. 2011. Epistemic planning for single and multi-agent systems. *Journal of Applied Non-Classical Logics* 21(1):9–34.

Bonet, B., and Geffner, H. 2014. Belief tracking for planning with sensing: Width, complexity and approximations. *Journal of Artificial Intelligence Research (JAIR)* 50:923–970.

Brafman, R. I., and Shani, G. 2012. Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research (JAIR)* 45:565–600.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems* 19(3):297–331.

Entringer, R. C., and Slater, P. J. 1979. Gossips and telegraphs. *Journal of the Franklin Institute* 307(6):353–360.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning about knowledge*, volume 4. MIT press Cambridge.

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

Kominis, F., and Geffner, H. 2014. Beliefs in multiagent planning: From one agent to many. In *Workshop on Distributed and Multi-Agent Planning*, 62–68.

Lakemeyer, G., and Lespérance, Y. 2012. Efficient reasoning in multiagent epistemic logics. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, 498–503.

Lin, F., and Reiter, R. 1994. State constraints revisited. *J. Log. Comput.* 4(5):655–678.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In *20th European Conference on Artificial Intelligence*, 540–545.

Liu, Y., and Wen, X. 2011. On the progression of knowledge in the situation calculus. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 976–982.

Löwe, B.; Pacuit, E.; and Witzel, A. 2011. DEL planning and some tractable cases. In *Logic, Rationality, and Interaction - Third International Workshop, LORI 2011, Guangzhou, China, October 10-13, 2011. Proceedings*, 179–192.

Muise, C.; Miller, T.; Felli, P.; Pearce, A.; and Sonenberg, L. 2015. Efficient reasoning with consistent proper epistemic knowledge bases. In *The International Conference on Autonomous Agents and Multiagent Systems*.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research (JAIR)* 35:623–675.

Petrick, R. P. A., and Levesque, H. J. 2002. Knowledge equivalence in combined action theories. In *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, 303–314.

Pinto, J. 1999. Compiling ramification constraints into effect axioms. *Computational Intelligence* 15:280–307.

Ramirez, M., and Lipovetzky, N. 2015. Lightweight Automated Planning ToolKiT. `http://lapkt.org/`. Accessed: 2015-01-17.

Reiter, R. 2001. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*, volume 16. MIT press Cambridge.

van Ditmarsch, H.; van der Hoek, W.; and Kooi, B. P. 2007. *Dynamic epistemic logic*, volume 337. Springer.

van Eijck, J. 2004. Dynamic epistemic modelling. *Manuscript, CWI, Amsterdam*.

Cooperative Epistemic Multi-Agent Planning With Implicit Coordination

Thorsten Engesser
Institut für Informatik
Albert-Ludwigs-Universität
Freiburg, Germany
engesset@cs.uni-freiburg.de

Thomas Bolander
DTU Compute
Technical University of Denmark
Copenhagen, Denmark
tobo@dtu.dk

Robert Mattmüller
Institut für Informatik
Albert-Ludwigs-Universität
Freiburg, Germany
mattmuel@cs.uni-freiburg.de

Bernhard Nebel
Institut für Informatik
Albert-Ludwigs-Universität
Freiburg, Germany
nebel@cs.uni-freiburg.de

Abstract

Epistemic Planning has been used to achieve ontic and epistemic control in multi-agent situations. We extend the formalism to include perspective shifts, allowing us to define a class of cooperative problems in which both action planning and execution is done in a purely distributed fashion, meaning coordination is only allowed implicitly by means of the available epistemic actions. While this approach can be fruitfully applied to model reasoning in some simple social situations, we also provide some benchmark applications to show that the concept is useful for multi-agent systems in practice.

1 Introduction

One important task in Multi-Agent Systems is to collaboratively reach a joint goal with multiple autonomous agents. The problem is particularly challenging in situations where the knowledge required to reach the goal is distributed among the agents. Most existing approaches therefore apply some centralized coordinating instance from the outside, strictly separating the stages of communication and negotiation from the agents' internal planning and reasoning processes. In contrast, building upon the epistemic planning framework by Bolander and Andersen (2011), we propose a decentralized planning notion in which each agent has to individually reason about the entire problem and autonomously decide when and how to (inter-)act. For this, both reasoning about the other agents' possible contributions and reasoning about their capabilities of performing the same reasoning is needed. We achieve our notion of implicitly coordinated plans by requiring all desired communicative abilities to be modeled as epistemic actions which then can be planned alongside their ontic counterparts, thus enabling the agents to perform observations and coordinate at runtime. While this imposes certain restrictions on the problems that can be solved, it captures the intuition that communication clearly constitutes an action by itself and, more subtly, that even a purely ontic action can play a communicative role (e.g. indirectly suggesting follow-up actions to another agent). Thus, for many problems our approach appears quite natural. On the practical side, the epistemic planning framework allows a very expressive way of defining both the agents' physical and communicative abilities and thereby seems an ideal choice in our case.

Our work directly builds upon the framework introduced by Bolander and Andersen (2011) and Löwe, Pacuit, and Witzel (2011), who formulated the planing problem in the context of Dynamic Epistemic Logic (DEL) (van Ditmarsch, van der Hoek, and Kooi 2007). Andersen, Bolander, and Jensen (2012) extended the approach to allow strong and weak conditional planning in the single-agent case. Similar to Bolander and Andersen (2011), we use search in the space of epistemic states to find a solution. This is in contrast to compilation approaches inspired by Palacios and Geffner (2009), which are popular and successful in the AI planning community. These approaches map an epistemic (or doxastic) planning problem to a corresponding classical one allowing to solve the problem using a classical planner (Muise et al. 2015; Kominis and Geffner 2015). However, these approaches can only deal with bounded nesting of knowledge (or belief) and can produce only sequential plans. There is an important similarity between the work by Muise et al. (2015) and ours, though: In both approaches, it is possible to shift the perspective from agent to agent along the plan. In particular this possibility of perspective shifts distinguishes these approaches from more traditional multi-agent planning (Brenner and Nebel 2009). Recent work in this area by Nissim and Brafman (2014) proposes a search algorithm for multi-agent planning that allows private actions and a certain degree of decentralization that achieves efficiency at the cost of not supporting reasoning about knowledge of other agents or only implicitly.

2 Theoretical Background

2.1 Epistemic language and epistemic states

We first recapitulate the foundations of DEL, following the conventions of Bolander and Andersen (2011). This means that our version of DEL includes postconditions allowing for ontic/factual change, but postconditions are without loss of expressivity limited to conjunctions of literals (as in classical planning).

Definition 1. The *epistemic language* $\mathcal{L}_{\text{KC}}(P, \mathcal{A})$ with respect to a set of atomic propositions P and a finite set of agents $\mathcal{A}$ is

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid C\varphi$$

where $p \in P$ and $i \in \mathcal{A}$.

We read $K_i\varphi$ as "agent i knows φ" and $C\varphi$ as "it is common knowledge (among all agents) that φ". In the following, we will always use P to denote our set of atomic propositions, and $\mathcal{A}$ our set of agents. Formulas of the epistemic language are evaluated in epistemic models.

Definition 2. An *epistemic model* over $(P, \mathcal{A})$ is a triple $\mathcal{M} = \langle W, (R_i)_{i\in\mathcal{A}}, V\rangle$ where

- The *domain* W is a non-empty finite set of *worlds*.
- $R_i \subseteq W \times W$ is an equivalence relation called the *indistinguishability relation* for agent i.
- $V : P \to W$ assigns a *valuation* to each atomic proposition.

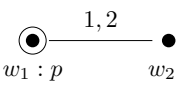
For $W_d \subseteq W$, the pair $(\mathcal{M}, W_d)$ is called an *epistemic state* (or simply *state*), and the worlds of W_d are called the *designated worlds*. An epistemic state is called *global* if W_d is a singleton. The designated world of a global state is called the *actual world*. In general, $(\mathcal{M}, W_d)$ can be thought of as the belief state $\{(\mathcal{M}, \{w\}) \mid w \in W_d\}$ over possible global states. An epistemic state $(\mathcal{M}, W_d)$ is called a *local state* for agent i if W_d is closed under R_i. A local state is *minimal* if W_d is a minimal set closed under R_i. Given an epistemic state $(\mathcal{M}, W_d)$, the *associated local state* of agent i, denoted $(\mathcal{M}, W_d)^i$, is $(\mathcal{M}, \{v \mid wR_iv, w \in W_d\})$.

Definition 3. Let $(\mathcal{M}, W_d)$ be an epistemic state where $\mathcal{M} = \langle W, (R_i)_{i\in\mathcal{A}}, V\rangle$. For $i \in \mathcal{A}$, $p \in P$ and $\varphi, \psi \in \mathcal{L}_{\text{KC}}(P, \mathcal{A})$, we define truth as follows:

$$
\begin{array}{ll}
(\mathcal{M}, W_d) \models \varphi & \text{iff } (\mathcal{M}, w) \models \varphi \text{ f.a. } w \in W_d \\
(\mathcal{M}, w) \models p & \text{iff } w \in V(p) \\
(\mathcal{M}, w) \models \neg\varphi & \text{iff } \mathcal{M}, w \not\models \varphi \\
(\mathcal{M}, w) \models \varphi \wedge \psi & \text{iff } \mathcal{M}, w \models \varphi \text{ and } \mathcal{M}, w \models \psi \\
(\mathcal{M}, w) \models K_i\varphi & \text{iff } \mathcal{M}, w' \models \varphi \text{ f.a. } w' \in W \text{ s.t. } wR_iw' \\
(\mathcal{M}, w) \models C\varphi & \text{iff } \mathcal{M}, w' \models \varphi \text{ f.a. } w' \in W \text{ s.t. } wR^*w'
\end{array}
$$

where R^* is the transitive closure of $\bigcup_{i\in\mathcal{A}} R_i$.

Example 1. A global epistemic state describes an epistemic situation from a global perspective, where the actual world has been pointed out. Consider the following global state $(\mathcal{M}, \{w_1\})$, where the nodes represent worlds and the edges represent the indistinguishability relations (reflexive edges left out):



We use ⦿ to denote the designated worlds, in this case only w_1. The actual world is w_1 where p holds, but for agent 1 (and 2) the actual world w_1 is indistinguishable from the world w_2 where p is false. Since agent 1 is ignorant about whether the actual world is w_1 or w_2, the model that represents his local view on the situation is $(\mathcal{M}, \{w_1, w_2\})$, which is exactly his associated local state of $(\mathcal{M}, \{w_1\})$. The point is that since agent 1 is unable to point out whether the actual world is w_1 or w_2, his internal state must consider both as candidates for being the actual world, and this is exactly what the model $(\mathcal{M}, \{w_1, w_2\}) = (\mathcal{M}, \{w_1\})^1$ does. We have $(\mathcal{M}, \{w_1\})^1 \not\models p$ and $(\mathcal{M}, \{w_1\})^1 \not\models \neg p$, corresponding to the fact that from agent 1's local perspective it can not be verified whether p holds or not.

2.2 Perspective shifts

In general, given an epistemic state $(\mathcal{M}, W_d)$, the associated local state $(\mathcal{M}, W_d)^i$ will represent agent i's internal perspective on that state. Going from $(\mathcal{M}, W_d)$ to $(\mathcal{M}, W_d)^i$ amounts to a *perspective shift*, where the perspective is shifted to the local perspective of agent i. Note that we have the following properties, where the third follows directly from the two first:

Lemma 1. *Let $(\mathcal{M}, W_d)$ be an epistemic state over $(P, \mathcal{A})$, $i \in \mathcal{A}$ and $\varphi \in \mathcal{L}_{\text{KC}}(P, \mathcal{A})$.*

1. *$(\mathcal{M}, W_d)^i \models \varphi$ iff $(\mathcal{M}, W_d) \models K_i\varphi$.*
2. *If $(\mathcal{M}, W_d)$ is local for agent i then $(\mathcal{M}, W_d)^i = (\mathcal{M}, W_d)$.*
3. *If $(\mathcal{M}, W_d)$ is local for agent i then $(\mathcal{M}, W_d) \models \varphi$ iff $(\mathcal{M}, W_d) \models K_i\varphi$.* □

Perspective shifts are of fundamental importance in multi-agent planning to allow an agent to reason about the other agents' possible contributions to a plan.

2.3 Dynamic language and epistemic actions

To model actions, we use the *event models* of DEL.

Definition 4. An *event model* over $(P, \mathcal{A})$ is a tuple $\mathcal{E} = \langle E, (Q_i)_{i\in\mathcal{A}}, \text{pre}, \text{post}\rangle$ where

- The *domain* E is a non-empty finite set of *events*.
- $Q_i \subseteq E \times E$ is an equivalence relation called the *indistinguishability relation* for agent i.
- $\text{pre} : E \to \mathcal{L}_{\text{KC}}(P, \mathcal{A})$ assigns a *precondition* to each event.
- $\text{post} : E \to \mathcal{L}_{\text{KC}}(P, \mathcal{A})$ assigns a *postcondition* to each event. For all $e \in E$, $post(e)$ is a conjunction of literals (atomic propositions and their negations).

For $E_d \subseteq E$, the pair $(\mathcal{E}, E_d)$ is called an *epistemic action* (or simply *action*), and the events in E_d are called the *designated events*. Similar to epistemic states, $(\mathcal{E}, E_d)$ is called a *local action* for agent i when E_d is closed under Q_i.

Each event of an epistemic action represents a different possible outcome. By using multiple events $e, e' \in E$ that are indistinguishable (i.e. eQ_ie'), it is possible to obfuscate the outcomes for some agent $i \in \mathcal{A}$, i.e. modeling partially observable actions. Using event models with $|E_d| > 1$, it is also possible to model sensing actions and nondeterministic actions (Bolander and Andersen 2011).

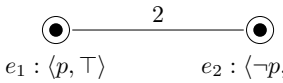
The *product update* is used to specify the successor state resulting from the application of an action in a state.

Definition 5. Let a state $(\mathcal{M}, W_d)$ and an action $(\mathcal{E}, E_d)$ over $(P, \mathcal{A})$ be given with $\mathcal{M} = \langle W, (R_i)_{i\in\mathcal{A}}, V\rangle$ and $\mathcal{E} = \langle E, (Q_i)_{i\in\mathcal{A}}, \text{pre}, \text{post}\rangle$. Then the *product update* is defined as $(\mathcal{M}, W_d) \otimes (\mathcal{E}, E_d) = (\langle W', (R'_i)_{i\in\mathcal{A}}, V'\rangle, W'_d)$ where

- $W' = \{(w, e) \in W \times E \mid \mathcal{M}, w \models \text{pre}(e)\}$
- $R'_i = \{((w, e), (w', e')) \in W' \times W' \mid wR_iw' \text{ and } eQ_ie'\}$
- $V'(p) = \{(w, e) \in W' \mid \text{post}(e) \models p \text{ or } (\mathcal{M}, w \models p \text{ and } \text{post}(e) \not\models \neg p)\}$
- $W'_d = \{(w, e) \in W' \mid w \in W_d, e \in E_d\}$.

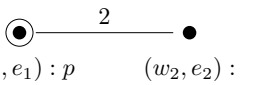
If both $(\mathcal{M}, W_d)$ and $(\mathcal{E}, E_d)$ are local for agent i, then so is $(\mathcal{M}, W_d) \otimes (\mathcal{E}, E_d)$ (Bolander and Andersen 2011).

Example 2. Consider the following epistemic action $(\mathcal{E}, \{e_1, e_2\})$, using the same conventions as for epistemic models, except each event e is labelled by $\langle \mathrm{pre}(e), \mathrm{post}(e) \rangle$:



It is a private sensing action for agent 1, where (only) agent 1 gets to know the truth value of p, since e_1 and e_2 are distinguishable to agent 1. Letting $(\mathcal{M}, \{w_1\})$ denote the state from Example 1, we get:

$(\mathcal{M}, \{w_1\}) \otimes (\mathcal{E}, \{e_1, e_2\}) =$



Hence $(\mathcal{M}, \{w_1\}) \otimes (\mathcal{E}, \{e_1, e_2\})$ is exactly as $(\mathcal{M}, \{w_1\})$ except the indistinguishability edge for agent 1 is removed. So the private sensing action reveals to agent 1 that p is true (without revealing it to agent 2). Before executing the action, agent 1 however does not know whether he will learn p or $\neg p$, which is signified by $(\mathcal{M}, \{w_1\})^1 \otimes (\mathcal{E}, \{e_1, e_2\})$ having both a designated p world and a designated $\neg p$ world (it differs from the model $(\mathcal{M}, \{w_1\}) \otimes (\mathcal{E}, \{e_1, e_2\})$ shown above exactly by (w_2, e_2) also being designated).

We extend the language $\mathcal{L}_{\mathrm{KC}}(P, \mathcal{A})$ into the *dynamic language* $\mathcal{L}_{\mathrm{DEL}}(P, \mathcal{A})$ by adding a modality $[(\mathcal{E}, e)]$ for each event model $\mathcal{E} = (E, (Q_i)_{i \in \mathcal{A}}, \mathrm{pre}, \mathrm{post})$ over $(P, \mathcal{A})$ and $e \in E$. The truth conditions are extended with the following standard clause from DEL:

$$(\mathcal{M}, w) \models [(\mathcal{E}, e)]\varphi \text{ iff}$$
$$(\mathcal{M}, w) \models \mathrm{pre}(e) \text{ implies } (\mathcal{M}, w) \otimes (\mathcal{E}, e) \models \varphi.$$

We define the following abbreviations: $[(\mathcal{E}, E_d)]\varphi := \bigwedge_{e \in E_d} [(\mathcal{E}, e)]\varphi$ and $\langle (\mathcal{E}, E_d) \rangle \varphi := \neg [(\mathcal{E}, E_d)] \neg \varphi$. We say that an action $(\mathcal{E}, E_d)$ is *applicable* in a state $(\mathcal{M}, W_d)$ if for all $w \in W_d$ there is an event $e \in E_d$ s.t. $(\mathcal{M}, w) \models \mathrm{pre}(e)$. Intuitively, an action is applicable in a state if for each possible situation (designated world), at least one possible outcome (designated event) is specified. Let $s = (\mathcal{M}, W_d)$ denote an epistemic state and $a = (\mathcal{E}, E_d)$ an action. Andersen (2015) shows that a is applicable in s iff $s \models \langle a \rangle \top$, and that $s \models [a]\varphi$ iff $s \otimes a \models \varphi$. We now define a further abbreviation: $(\!(a)\!)\varphi := \langle a \rangle \top \wedge [a]\varphi$. Hence:

$$s \models (\!(a)\!)\varphi \text{ iff } a \text{ is applicable in } s \text{ and } s \otimes a \models \varphi \quad (1)$$

Thus $(\!(a)\!)\varphi$ means that the application of a is possible and will (necessarily) lead to a state fulfilling φ.

3 Cooperative Planning

We will now consider different types of planning problems and solution concepts in the setting of DEL-based planning. In the simplest case, a planning problem $\langle s_0, A, \varphi_g \rangle$ over $(P, \mathcal{A})$ consists of an initial epistemic state s_0 over $(P, \mathcal{A})$, a set A of epistemic actions over $(P, \mathcal{A})$ and a goal formula φ_g of $\mathcal{L}_{\mathrm{KC}}(P, \mathcal{A})$. Informally, a (sequential) solution to such a planning problem is a sequence of actions $(a_1, \ldots, a_n)$ from A, such that executing the sequence in s_0 leads to a state satisfying φ_g. In the DEL-based setting, the state-transition function mapping a state-action pair (s, a) into the state resulting from executing a in s is given by $(s, a) \mapsto s \otimes a$ (when a is not applicable in s, the state-transition function is taken to be undefined on (s, a)). Hence, more formally, a solution to $\langle s_0, A, \varphi_g \rangle$ is a sequence of actions $(a_1, \ldots, a_n)$ from A such that for all $i = 1, \ldots, n$, the action a_i is applicable in $s_0 \otimes a_1 \otimes \cdots \otimes a_{i-1}$, and $s_0 \otimes a_1 \otimes \cdots \otimes a_n \models \varphi_g$. Note that by (1) above, these conditions are equivalent to simply requiring $s_0 \models (\!(a_1)\!)(\!(a_2)\!) \cdots (\!(a_n)\!)\varphi_g$.

This solution concept is equivalent to the one considered in (Bolander and Andersen 2011). In that paper, the initial state as well as all actions are supposed to be modelled from the perspective of one single planning agent, that is, be local to that agent. Such a setting provides a natural formal framework for a single agent acting alone in a multi-agent environment, but does not provide a systematic solution to the case where multiple agents are (inter)acting towards a joint goal. The latter situation is what we wish to consider in this paper.

For cooperative multi-agent planning towards a joint goal, we identify the following settings:

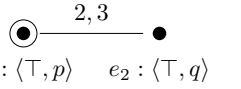
1. *Centralized planning*, meaning one instance (having complete or incomplete knowledge, e.g. as one of the agents) generates a plan in advance, which, if given to and executed by the cooperative agents, will lead to a goal state.
2. *Decentralized planning*. Here each agent does the planning process for himself. Usually this is done as part of a multi-agent architecture where the agents announce their plans, negotiate, solve conflicts, etc.
3. *Decentralized planning with implicit coordination*. In this scenario, all coordination is achieved implicitly through observing the effects of the actions of other agents. The rationale is, if all of the agents act to compatible individual plans (which may include assumptions on other agents' actions and plans), the goal condition can be reached without explicit coordination and commitments.

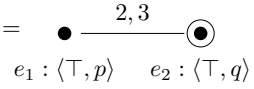
This paper focuses on the concept of decentralized planning with implicit coordination, which relies closely on the perspective-shifting capabilities of the epistemic planning framework and is, to the best of our knowledge, novel to this paper. Using our approach, it is possible to solve some multi-agent problems by just specifying a common goal directive for all the agents in a system, given some optimistic assumptions, namely common knowledge of the available actions, consistent internal models of the situation and perfect reasoning capacity of all agents. The agents can then take the necessary steps (deducing compatible plans and acting on them) in a generic, autonomous manner.

Let A denote a set of actions, and $\mathcal{A}$ a set of agents. In this paper we assume each action to be executable by a single agent, that is, we are not considering joint actions. For technical reasons, we wish each action to be executable by a *unique* agent, which we call the *owner* of the action. More precisely, an *owner function* is a mapping $\omega : A \to \mathcal{A}$, mapping each action to the agent who can execute it (who *owns* it). This approach is closely related to the one by Löwe, Pacuit, and Witzel (2011). Mapping each action to a unique

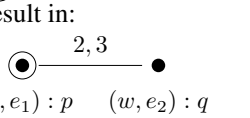
agent can be done without loss of generality, since semantically equivalent duplicates can always be added to the action set.

Example 3. As mentioned above, our framework is based on the assumption that there is common knowledge of the available actions in the action set A (so that agents can always correctly reason about the actions available to themselves and others). This however does not imply that agents are always aware of the actions executed by others. Consider the following two actions owned by agent 1:

$a_p =$ 

$a_q =$ 

The action a_p makes p true and a_q makes q true. If agent 1 executes a_p, agents 2 and 3 will still consider it possible that q was made true (due to the indistinguishability edge in a_p), and conversely for a_q. Let s_0 be a singleton model where no atoms are true: $s_0 =$  $w :$. Then a_p is applicable in s_0 and the execution will result in:

$s_0 \otimes a_p =$ 

We see that $s_0 \otimes a_p \models p$ from which we get $s_0 \models (\!(a_p)\!)\, p$, using (1). As s_0 is clearly local to all 3 agents, we can apply item 3 of Lemma 1 to conclude $s_0 \models K_i(\!(a_p)\!)\, p$ for $i = 1, 2, 3$. Note also that $s_0 \otimes a_p \models \neg K_j p$ for $j = 2, 3$, and hence $s_0 \models (\!(a_p)\!)\, \neg K_j p$ using again (1). Hence we get:

$$s_0 \models K_i(\!(a_p)\!)p \qquad \text{for } i = 1, 2, 3 \tag{2}$$

$$s_0 \models (\!(a_p)\!)\neg K_j p \qquad \text{for } j = 2, 3 \tag{3}$$

The intuition is this: All agents know that executing a_p leads to p, since they know the action set. This is captured by (2). However, even after a_p has been executed, agents 2 and 3 do not know p, since for them action a_p is indistinguishable from action a_q (they will not know whether a_p or a_q was chosen by agent 1). This is captured by (3).

The only situations where it makes sense to have both (2) and (3) above are when agents 2 and 3 can mistake the action a_p for another action. To only consider such sensible scenarios we will require our action sets A to satisfy the following closure property: If $(\mathcal{E}, E_d) \in A$ and e is an event in $\mathcal{E}$, then there exists a set E'_d with $e \in E'_d$ and $(\mathcal{E}, E'_d) \in A$. If A satisfies this property, it is called *closed*.

We are now finally ready to formally define our notion of a cooperative epistemic planning problem.

Definition 6. A *cooperative planning problem* (or simply a *planning problem*) $\Pi = \langle s_0, A, \omega, \varphi_g \rangle$ over $(P, \mathcal{A})$ consists of

- an initial epistemic state s_0 over $(P, \mathcal{A})$
- a finite, closed set of epistemic actions A over $(P, \mathcal{A})$
- an owner function $\omega : A \to \mathcal{A}$
- a goal formula $\varphi_g \in \mathcal{L}_{\text{KC}}(P, \mathcal{A})$

such that each $a \in A$ is local for $\omega(a)$. When s_0 is a global state, we call it a *global cooperative planning problem* (or simply a *global planning problem*). When s_0 is local for agent i, we call it a *cooperative planning problem for agent* i (or simply a *planning problem for agent* i). Given a global planning problem $\Pi = \langle s_0, A, \omega, \varphi_g \rangle$, the *associated planning problem for agent* i is $\Pi^i = \langle {s_0}^i, A, \omega, \varphi_g \rangle$.

Given a multi-agent system is facing a global planning problem Π, then each individual agent i is facing the planning problem Π^i (agent i cannot observe the global state s_0 directly, only the associated local state ${s_0}^i$).

In the following, we define sequential and conditional implicitly coordinated solution concepts for cooperative planning problems.

3.1 Sequential Plans

We first want to define our notion of an implicitly coordinated sequential solution to a planning problem. We wish every agent to plan for himself, but come up with cooperative plans involving also the required actions of the other agents. Intuitively, we want the planning agent to be able to both verify the validity of the plan itself (that it reaches the goal), and to verify that each of the involved agents can do the same for their respective subplans given their local information.

Definition 7. Let $\Pi = \langle s_0, A, \omega, \varphi_g \rangle$ be a cooperative planning problem. An *implicitly coordinated plan* (or simply *plan*) for Π is a sequence $(a_1, \ldots, a_n)$ of actions from A such that:

$$s_0 \models K_{\omega(a_1)}(\!(a_1)\!)K_{\omega(a_2)}(\!(a_2)\!) \cdots K_{\omega(a_n)}(\!(a_n)\!)\varphi_g \tag{4}$$

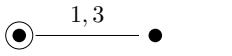
If Π is a planning problem for agent i, we call the plan a *plan for agent* i. A *plan for agent* i to a global planning problem Π is a plan for Π^i.

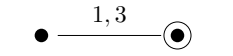
Note that a formula of the form $K_{\omega(a)}(\!(a)\!)\varphi$ expresses "the owner of action a knows that a is applicable and will lead to φ". Equation (4) hence expresses that a solution $(a_1, \ldots, a_n)$ should satisfy the following:

> *The owner of the first action a_1 knows that a_1 is initially applicable and will lead to a situation where the owner of the second action a_2 knows that a_2 is applicable and will lead to a situation where... the owner of the nth action a_n knows that a_n is applicable and will lead to the goal being satisfied.*

Example 4. Consider the cooperative planning problem $\Pi = \langle s_0, \{a_p, a_q\}, \omega, p \rangle$ over $(\{p, q\}, \{1, 2, 3\})$ where s_0, a_p and a_q are as in Example 3 and $\omega(a_p) = \omega(a_q) = 1$. Note that $\Pi = \Pi^i$ for all $i \in \{1, 2, 3\}$: The planning problem "looks the same" to all agents. The single-element sequence (a_p) is an (implicitly coordinated) plan for Π since $\omega(a_p) = 1$ and $s_0 \models K_1(\!(a_p)\!)\, p$ by (2). This is indeed a plan for all agents $i = 1, 2, 3$, since it is a plan for all Π^i. In other words, all agents can individually come up with the implicitly coordinated plan (a_p). They all know that all that has to be done is for agent 1 to execute a_p, and they know that agent 1 knows this himself.

Now consider extending the planning problem as follows. We add two more actions a_r and a_s given by:

$a_r =$ 

$a_s =$ 

Consider the cooperative planning problem $\Pi' = \langle s_0, \{a_p, a_q, a_r, a_s\}, \omega, r\rangle$ where ω is extended by $\omega(a_r) = \omega(a_s) = 2$. A successful plan is clearly (a_p, a_r) since $s_0 \models \langle\!\langle a_p \rangle\!\rangle \langle\!\langle a_r \rangle\!\rangle r$. However, it does not qualify as an implicitly coordinated plan for any of the agents, since it can be showed that $s_0 \not\models K_1 \langle\!\langle a_p \rangle\!\rangle K_2 \langle\!\langle a_r \rangle\!\rangle r$. The problem is that even if agent 1 starts out by executing a_p, agent 2 will not know he did (cf. Example 3), and hence agent 2 will not at runtime know that he can apply a_r to achieve r.

Consider adding a fifth action $p! := \textcircled{\bullet}\, e : \langle p, \top\rangle$ with $\omega(p!) = 1$. This is a public announcement of p by agent 1. Extending Π' with the action $p!$, all agents can again find an implicitly coordinated plan, namely $(a_p, p!, a_r)$: agent 1 first makes p true, then announces that he did, which makes agent 2 know that he can apply a_r to make r true.

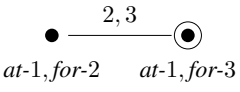
The following proposition gives a more structural characterization of implicitly coordinated plans. It thus becomes clear that such plans can be found by performing a breadth-first search over the set of successively applicable actions, shifting the perspective for each state transition to the owner of the respective action.

Proposition 2. *For a cooperative planning problem* $\Pi = \langle s_0, A, \omega, \varphi\rangle$*,*

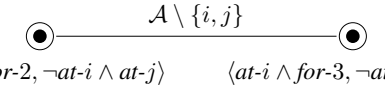
- $()$ *is an implicitly coordinated plan for* Π *iff* $s_0 \models \varphi$
- $(a_1, \ldots, a_n)$ *with* $n \geq 1$ *is an implicitly coordinated plan for* Π *iff* a_1 *is applicable in* ${s_0}^{\omega(a_1)}$ *and* $(a_2, \ldots, a_n)$ *is an implicitly coordinated plan for* $\langle {s_0}^{\omega(a_1)} \otimes a_1, A, \omega, \varphi\rangle$. □

The proof is simple and hence omitted (it relies on (1), Lemma 1 and (4)).

Example 5. As a more practical example, consider a situation with agents $\mathcal{A} = \{1, 2, 3\}$ where a letter is to be passed from agent 1 to one of the other two agents, possibly via the third agent. Mutually exclusive propositions *at*-1, *at*-2, *at*-3 $\in P$ are used to denote the current carrier of the letter, while *for*-1, *for*-2, *for*-3 $\in P$ denote the addressee. In our example, agent 1 has a letter for agent 3, so *at*-1 and *for*-3 are initially true.

$s_0 =$ 

In s_0, all agents know that agent 1 has the letter (*at*-1), but agents 2 and 3 do not know who of them is the addressee (*for*-2 or *for*-3). We assume that agent 1 can only exchange letters with agent 2 and agent 2 can only exchange letters with agent 3. We thus define the four possible actions $a_{12}, a_{21}, a_{23}, a_{32}$, with a_{ij} being the composite action of agent i publicly passing the letter to agent j and privately informing him about the correct addressee. I.e.

$a_{ij} =$ 

Given that the joint goal is to pass a letter to its addressee, the global planning problem then is $\Pi = \langle s_0, A, \omega, \varphi_g\rangle$ with

- $A = \{a_{12}, a_{21}, a_{23}, a_{32}\}$,
- $\omega = \{a_{12} \mapsto 1, a_{21} \mapsto 2, a_{23} \mapsto 2, a_{32} \mapsto 3\}$, and
- $\varphi_g = \bigwedge_{i \in \{1,2,3\}} (\textit{for-}i \to \textit{at-}i)$.

Consider the action sequence (a_{12}, a_{23}): Agent 1 passes the letter to agent 2, and agent 2 passes it on to agent 3. It can now be verified that

$$ {s_0}^1 \models K_1 \langle\!\langle a_{12} \rangle\!\rangle K_2 \langle\!\langle a_{23} \rangle\!\rangle \varphi_g $$
$$ {s_0}^i \not\models K_1 \langle\!\langle a_{12} \rangle\!\rangle K_2 \langle\!\langle a_{23} \rangle\!\rangle \varphi_g \quad \text{for } i = 2, 3 $$

Hence (a_{12}, a_{23}) is an implicitly coordinated plan for agent 1, but not for agents 2 and 3.

This is because in the beginning, agents 2 and 3 do not know for who of them the letter is intended and hence cannot verify that (a_{12}, a_{23}) will lead to a goal state. However, after agent 1's execution of a_{12}, agent 2 can distinguish between the possible addressees at runtime, and find his subplan (a_{23}), as contemplated by agent 1.

3.2 Conditional Plans

Sequential plans are often not sufficient to solve a given epistemic planning problem. In particular, as soon as nondeterministic action outcomes or splits on obtained observations come into play, we need *conditional* plans to solve such a problem. Consider for instance a problem where agents 1 and 2 need to cooperate as follows: Agent 1 starts out with an action a_1 that provides necessary information to agent 2 on how to continue, and depending on the new information, agent 2 needs to continue either with action a_2 or a_3 in order to achieve the joint goal. Unlike Andersen, Bolander, and Jensen (2012), who represent conditional plans as action trees with branches depending on knowledge formula conditions, we represent them as policy functions $(\pi_i)_{i \in \mathcal{A}}$, mapping minimal local epistemic states to actions for their respective observer agents.

First we define a few new pieces of notation. For the rest of this section, except in examples, we fix a set P of atomic propositions and a set $\mathcal{A}$ of agents. Hence all considered cooperative planning problems will be over the pair $(P, \mathcal{A})$ without this being mentioned explicitly. Given $i \in \mathcal{A}$, we use $S_i^{\min}$ to denote the set of minimal local states of agent i over $(P, \mathcal{A})$. We use S^{gl} to denote the set of global states over $(P, \mathcal{A})$. For any epistemic state $s = (\mathcal{M}, W_d)$ we let $\mathit{Globals}(s) = \{(\mathcal{M}, w) \mid w \in W_d\}$. Hence $\mathit{Globals}(s)$ is the set of global states "contained in" s.

We now define two different types of policies, *joint policies* and *global policies*, and later show them to be equivalent.

Definition 8 (Joint policy)**.** Let $\Pi = \langle s_0, A, \omega, \varphi_g\rangle$ be a cooperative planning problem. Then a *joint policy* $(\pi_i)_{i \in \mathcal{A}}$ consists of partial functions $\pi_i : S_i^{\min} \to A$ where for each $(s, a) \in \pi_i$, $\omega(a) = i$ and a is applicable in s.

Definition 9 (Global policy)**.** Let $\Pi = \langle s_0, A, \omega, \varphi_g\rangle$ be a cooperative planning problem. Then a *global policy* π is a mapping $\pi : S^{\text{gl}} \to \mathcal{P}(A)$ satisfying the requirements applicability (1), and uniformity (2), (3) below,

(1) For all $s \in S^{\text{gl}}$, $a \in \pi(s)$: a is applicable in $s^{\omega(a)}$.
(2) For all $s \in S^{\text{gl}}$, $a, a' \in \pi(s)$ with $a \neq a'$: $\omega(a) \neq \omega(a')$.
(3) For all $s, t \in S^{\text{gl}}, a \in \pi(s)$ s.t. $s^{\omega(a)} = t^{\omega(a)}$: $a \in \pi(t)$.

Proposition 3. *Any joint policy $(\pi_i)_{i\in\mathcal{A}}$ induces a global policy π given by*

$$\pi(s) = \left\{\pi_i(s^i) \mid i \in \mathcal{A} \text{ and } \pi_i(s^i) \text{ is defined}\right\}.$$

Conversely, any global policy π induces a joint policy $(\pi_i)_{i\in\mathcal{A}}$ given by

$$\pi_i(s^i) = a \quad \text{for all } (s, A') \in \pi,\ a \in A' \text{ with } \omega(a) = i.$$

Proof. First we prove that the induced mapping π as defined above is a global policy. Condition (1): If $a \in \pi(s)$ then $\pi_i(s_i) = a$ for some i, and by definition of joint policy this implies a is applicable in s^i. Condition (2): We prove the contrapositive. Assume $a, a' \in \pi(s)$ with $\omega(a) = \omega(a')$. By definition of π we have $\pi_i(s^i) = a$ and $\pi_j(s^j) = a'$ for some i, j. By definition of joint policy, $\omega(a) = i$ and $\omega(a') = j$. Since $\omega(a) = \omega(a')$ we get $i = j$ and hence $\pi_i(s^i) = \pi_j(s^j)$. This implies $a = a'$. Condition (3): Assume $a \in \pi(s)$ and $s^{\omega(a)} = t^{\omega(a)}$. By definition of π and joint policy, we get $\pi_i(s^i) = a$ for $i = \omega(a)$. Thus $s^i = t^i$, and since $\pi_i(s^i) = a$, we immediately get $\pi_i(t^i) = a$ and hence $a \in \pi(t)$. We now prove that the induced mappings $(\pi_i)_{i\in\mathcal{A}}$ defined above form a joint policy. Constraint (1) ensures the applicability property as required by Definition 8, while the constraints (2) and (3) ensure the right-uniqueness of each partial function π_i. □

By Proposition 3, we can identify joint and global policies, and will in the following move back and forth between the two. Notice that Definitions 8 and 9 allow a policy to distinguish between modally equivalent states. A more sophisticated definition avoiding this is possible, but is beyond the scope of this paper. Usually, a policy π is only considered to be a solution to a planning problem if it is closed in the sense that π is defined for all non-goal states reachable following π. Here, we want to distinguish between two different notions of closedness: one that refers to all states reachable from a centralized perspective, and one that refers to all states considered reachable when tracking perspective shifts. To that end, we distinguish between centralized and individual successor functions.

Definition 10. A successor function is a function $\sigma : S^{\text{gl}} \times A \to \mathcal{P}(S^{\text{gl}})$ mapping pairs of a global states s and applicable actions a to sets $\sigma(s, a)$ of possible successor states.

We can then define the *centralized* successor function as

$$\sigma_{cen}(s, a) = \mathit{Globals}(s \otimes a).$$

It specifies the global states that are possible after the application of a in s. If closedness of a global policy π based on the centralized successor function is required, then no execution of π will ever lead to a non-goal state where π is undefined. Like for sequential planning, we are again interested in the decentralized scenario where each agent has to plan and decide when and how to act by himself under incomplete knowledge. We achieve this by encoding the perspective shifts to the next agent to act in the *individual* successor function

$$\sigma_{ind}(s, a) = \mathit{Globals}(s^{\omega(a)} \otimes a).$$

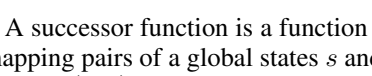
Unlike $\sigma_{cen}(s, a)$, $\sigma_{ind}(s, a)$ considers a global state s' to be a successor of s after application of a if *agent $\omega(a)$ considers s' possible* after the application of a, not only if *s' is actually possible from a global perspective*. Thus, $\sigma_{cen}(s, a)$ is always a (possibly strict) subset of $\sigma_{ind}(s, a)$, and a policy π_{ind} that is closed wrt. $\sigma_{ind}(s, a)$ must be defined for *at least* the states for which a policy π_{cen} that is closed wrt. $\sigma_{cen}(s, a)$ must be defined. This corresponds to the intuition that solution existence for decentralized planning with implicit coordination is a stronger property than solution existence for centralized planning. For both successor functions, we can now formalize what a strong solution is that can be executed collectively by the agents. Our notion satisfies the usual properties of strong plans (Cimatti et al. 2003), namely closedness, properity and acyclicity.

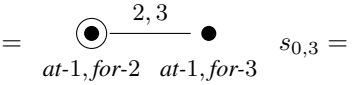
Definition 11 (Strong Policy)**.** Let $\Pi = \langle s_0, A, \omega, \varphi_g\rangle$ be a cooperative planning problem and σ a successor function. A global policy π is called a *strong policy* for Π with respect to σ if

(i) Finiteness: π is finite.
(ii) Foundedness: for all $s \in \mathit{Globals}(s_0)$,
  (1) $s \models \varphi_g$, or
  (2) $\pi(s) \neq \emptyset$.
(iii) Closedness: for all $(s, A') \in \pi$, $a \in A'$, $s' \in \sigma(s, a)$,
  (1) $s' \models \varphi_g$, or
  (2) $\pi(s') \neq \emptyset$.

Note that we do not explicitly require acyclicity, since this is already implied by a literal interpretation of the product update semantic that ensures unique new world names after each update. It then follows from (i) and (iii) that π is proper. We call strong plans with respect to σ_{cen} *centralized policies* and strong plans with respect to σ_{ind} *implicitly coordinated policies*.

Example 6. Consider again the letter passing problem introduced in Example 5. Let $s_{0,2}$ and $s_{0,3}$ denote the global states that are initially considered possible by agent 2.

$s_{0,2} =$ 

$s_{0,3} =$ 

With $s_{1,3} = s_{0,3} \otimes a_{12}$, a policy for agent 2 is given by

$$\pi_1 = \left\{s_{0,3} \mapsto a_{12}, s_{0,2} \mapsto a_{12}\right\}, \pi_2 = \left\{s_{1,3} \mapsto a_{23}\right\}.$$

After the contemplated application of a_{12} by agent 1 (in both cases), agent 2 can distinguish between $s_{1,2} = s_{0,2} \otimes a_{12}$, where the goal is already reached and nothing has to be done, and $s_{1,3}$, where agent 2 can apply a_{23}, leading directly to the goal state $s_{1,3} \otimes a_{23}$. Thus, π is an implicitly coordinated policy for Π^2. While in the sequential case, agent 2 has to wait for the first action a_{12} of agent 1 to be able to find its subplan, it can find the policy $(\pi_i)_{i\in\mathcal{A}}$ in advance by explicitly planning for a run-time distinction.

In general, strong policies can be found by performing an AND-OR search, where AND branching corresponds to branching over different epistemic worlds and OR branching corresponds to branching over different actions. By considering modally equivalent states as duplicates and thereby transforming the procedure into a graph search, space and time requirements can be reduced, although great care has to be taken to deal with cycles correctly.

It is easy to show that implicitly coordinated policies generalize implicitly coordinated plans.

Proposition 4. *Each implicitly coordinated plan* $(a_1, \ldots, a_n)$ *for* $\Pi = \langle s_0, A, \omega, \varphi_g \rangle$ *has a corresponding implicitly coordinated policy* π *for* Π.

Proof sketch. Let $s_i = s_0 \otimes a_1 \otimes \ldots \otimes a_i$. Then we can construct the policy π with $\pi(s'_i) = a_{i+1}$ for all $s'_i \in \mathit{Globals}(s_i)$ and all $i = 0, \ldots, n-1$. All three requirements of Definition 9 trivially hold. We further have to show that π is an implicitly coordinated policy for Π. Finiteness and foundedness are trivial. Closedness results from the correspondence to the recursive part of Proposition 2. □

4 Experiments

We implemented a planner that is capable of finding implicitly coordinated plans and policies, and conducted two experiments: one small case study of the Russian card problem (van Ditmarsch 2003) intended to show how this problem can be modeled and solved from an individual perspective, and one experiment investigating the scaling behavior of our approach on private transportation problems in the style of Examples 5 and 6, using instances of increasing size. Our planner is written in C++ and uses breadth-first search with an approximate bisimulation test that is used for state contraction and duplicate detection. All experiments were performed on a computer with a single Intel i7-4510U CPU core.

4.1 Russian Card Problem

In the Russian card problem, seven cards numbered $0, \ldots, 6$ are randomly dealt to three agents. Alice and Bob get three cards each, while Eve gets the single remaining card. Initially, each agent only knows its own cards. The task is now for Alice and Bob to inform each other about their respective cards using only public announcements, without revealing the holder of any single card to Eve. The problem was analyzed and solved from the global perspective by van Ditmarsch et al. (2006), and a given protocol was *verified* from an individual perspective by Ågotnes et al. (2010) before. We want to *solve* the problem from the individual perspective of agent Alice and find an implicitly coordinated policy for her. We only allow ontic announcements about hands, not about knowledge. To keep the problem computationally feasible, we impose some restrictions on the resulting protocol, namely that the first action has to be Alice announcing five possible alternatives for her own hand (one of which has to be her true hand), and that the second action has to be Bob announcing the card Eve is holding. Without loss of generality, we fix one specific initial hand for Alice, namely 012. From a plan for this initial hand, plans for all other initial hands can be obtained by renaming. For simplicity, we only generate *applicable* actions for Alice, i.e. announcements that include her true hand 012. This results in the planning problem having a total of 46376 options for the first action, and 7 for the second action. Still, the initial state s_0 consist of 140 worlds, one for each possible deal of cards. Agents can only distinguish worlds where their own hands differ. Alice's designated worlds in her associated local state of s_0 are those four worlds in which she holds hand 012.

Our planner, run in the conditional planning mode, needs 7282 seconds and approximately 630MB of memory to come up with a solution policy. In the solution, Alice first announces her hand to be one of 012, 034, 156, 236, and 245. It can be seen that each of the five hands other than the true hand 012 contains at least one of Alice's and one of Bob's cards, meaning that Bob will immediately be able to identify the complete deal. Since also every card occurs in exactly two of the five announced hands, of which at least one is considered possible by Eve, she stays unaware of the individual cards of Alice and Bob. Afterwards, Alice can wait for Bob to announce that Eve has either card 3, 4, 5 or 6 (which will not tell Eve anything new, either).

4.2 Mail Instances

Our second experiment concerns the letter passing problem from Examples 5 and 6. We generalized the scenario to allow an arbitrary number of agents with an arbitrary undirected neighborhood graph, indicating which agents are allowed to directly pass letters to each other. As neighborhood graphs, we used randomly generated Watts-Strogatz small-world networks (Watts and Strogatz 1998), exhibiting characteristics that can also be found in social networks. Watts-Strogatz networks have three parameters: The number N of nodes (determining the number of agents in our setting), the average number K of neighbors per node (roughly determining the average branching factor of a search for a plan), and the probability β of an edge being a "random shortcut" instead of a "local connection" (thereby influencing the shortest path lengths between agents). We only generate *connected* networks in order to guarantee plan existence.

We distinguish between the MAILTELL and the MAILCHECK benchmarks. To guarantee plan existence, in both scenarios the actions are modeled such as to ensure that the letter position remains common knowledge among the agents in all reachable states. The mechanics of MAILTELL directly correspond to those given in Example 5. There is only one type of action, publicly passing the letter to a neighboring agent while privately informing him about the final addressee. This allows for sequential implicitly coordinated plans. In the resulting plans, letters are simply moved along a shortest path to the addressee. In contrast, in MAILCHECK, an agent that has the letter can only check if he himself is the addressee or not using a separate action (without learning the actual addressee if it is not him). To ensure plan existence in this scenario, we allow an agent to pass on the letter only if it is destined for someone else. Unlike in MAILTELL, conditional plans are required here. In a solution (policy), the worst-case sequence of succes-

sively applied actions contains an action passing the letter to each agent at least once. As soon as the addressee has been reached, execution is stopped. Experiments were conducted for both scenarios with different parameters (see Tables 1, 2 and 3). For each set of parameters, 100 trials were performed. The tables show the average numbers of nodes that were created, expanded and discarded (because a bisimilar state was already considered somewhere else) in the search. For the conditional search in MAILCHECK, compound AND-OR nodes are used. In Table 1, *direct path* denotes the average shortest path length between sender and addressee, while in Tables 2 and 3, *full path* denotes the average length of a shortest path passing through all agents starting from the sender.

While the shortest path length between sender and addressee grows very slowly with the number of agents (due to the shortcut connections in the network), the shortest path passing through all agents roughly corresponds to the number of agents. Since these measures directly correspond to the minimal plan lengths, the observed exponential growth of space and time requirements with respect to them (and to the base K) is unsurprising.

Note also that in both scenarios, the number of agents determines the number of worlds (one for each possible addressee) in the initial state. Since the preconditions of the available actions are mutually exclusive, this constitutes an upper bound on the number of worlds per state throughout the search. Thus we get only a linear overhead in comparison to directly searching the networks for the relevant paths.

5 Conclusion and Future Work

We defined and studied an interesting new cooperative, decentralized planning concept without the necessity of explicit coordination or negotiation. Instead, by modeling all possible communication directly as plannable actions and relying on the ability of the autonomous agents to put themselves into each others shoes (using perspective shifts), some problems can be elegantly solved achieving implicit coordination between the agents. We briefly demonstrated an implementation of both the sequential and conditional solution algorithms and its performance on the Russian card problem and two letter passing problems.

An important starting point for further research concerns concrete problems (e.g. epistemic versions of established multi-agent planning problems) and the question of which kind of communicative actions the agents would need to solve these problems in an implicitly coordinated way. As seen in the MAILTELL benchmark, the dynamic epistemic treatment of a problem does not necessarily lead to more than linear overhead. It will be interesting to identify classes of tractable problems and see how agents cope in a simulated environment. Another issue that is relevant in practice concerns the interplay of the single agents' individual plans. In our setting, the agents have to plan individually and decide autonomously when and how to act. Also, when it comes to action application, there is no predefined notion of agent precedence. This leads to the possibility of *incompatible* plans, and in consequence to the necessity for agents having to *replan* in some cases. While our notion of implicitly coordinated planning explicitly forbids the execution of actions leading to *deadlock situations* (i.e. non-goal states where there is no implicitly coordinated plan for any of the agents), replanning can still lead to *livelocks*. Both the conditions leading to livelocks and individually applicable strategies to avoid them can be investigated. Since we need to be able to deal with replanning anyway, we can follow Andersen, Bolander, and Jensen (2013) and also investigate another successor function σ_{plaus} that maps only into the *most plausible* successor states. We expect this to lead to less branching and thus higher efficiency than the successor functions defined above.

agents	10	20	30	40	50
direct path	1.4	2.3	3.1	3.7	3.6
created	25	116	415	965	1023
expanded	7	36	142	347	359
discarded	2	32	166	455	443
time/s	0.02	0.14	0.65	2.38	5.02

Table 1: MAILTELL, $K = 4$, $\beta = 0.1$

agents	10	15	20	25	30
full path	10.4	16.1	21.7	27.6	33.2
created	402	2073	8065	35691	113481
expanded	361	1968	7771	34890	111582
discarded	552	3229	13126	59827	193555
time/s	0.02	0.18	1.14	8.03	38.76

Table 2: MAILCHECK, $K = 2$, $\beta = 0.1$

agents	7	9	11	13	15
full path	7.0	9.0	11.0	13.0	15.0
created	712	3161	13071	50104	188997
expanded	642	3027	12838	49739	188421
discarded	1859	9167	39528	154756	588582
time/s	0.03	0.21	1.14	5.90	27.30

Table 3: MAILCHECK, $K = 4$, $\beta = 0.1$

Acknowledgments

This work was partly supported by the DFG as part of the SFB/TR 14 AVACS. We thank Christian Becker-Asano for the fruitful discussions of earlier versions of this work.

References

Ågotnes, T.; Balbiani, P.; van Ditmarsch, H. P.; and Seban, P. 2010. Group announcement logic. *Journal of Applied Logic* 8(1):62–81.

Andersen, M. B.; Bolander, T.; and Jensen, M. H. 2012. Conditional epistemic planning. volume 7519 of *Lecture Notes in Computer Science*, 94–106.

Andersen, M. B.; Bolander, T.; and Jensen, M. H. 2013. Don't plan for the unexpected: Planning based on plausibility models. *To appear in Logique et Analyse, 2015*.

Andersen, M. B. 2015. *Towards Theory-of-Mind agents using Automated Planning and Dynamic Epistemic Logic*. Ph.D. Dissertation, Technical University of Denmark.

Bolander, T., and Andersen, M. B. 2011. Epistemic planning for single and multi-agent systems. *Journal of Applied Non-Classical Logics* 21(1):9–34.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems* 19(3):297–331.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1–2):35–84.

Kominis, F., and Geffner, H. 2015. Beliefs in multiagent planning: From one agent to many. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*. To appear.

Löwe, B.; Pacuit, E.; and Witzel, A. 2011. DEL planning and some tractable cases. In *LORI 2011*, volume 6953 of *Lecture Notes in Artificial Intelligence*, 179–192.

Muise, C.; Belle, V.; Felli, P.; McIlraith, S.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2015. Planning over multi-agent epistemic states: A classical planning approach. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*.

Nissim, R., and Brafman, R. I. 2014. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research (JAIR)* 51:293–332.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research (JAIR)* 35:623–675.

van Ditmarsch, H. P.; van der Hoek, W.; van der Meyden, R.; and Ruan, J. 2006. Model checking russian cards. *Electronic Notes in Theoretical Computer Science* 149(2):105–123.

van Ditmarsch, H. P.; van der Hoek, W.; and Kooi, B. 2007. *Dynamic Epistemic Logic*. Springer Heidelberg.

van Ditmarsch, H. P. 2003. The russian cards problem. *Studia Logica* 75(1):31–62.

Watts, D. J., and Strogatz, S. H. 1998. Collective dynamics of 'small-world' networks. *Nature* 393(6684):440–442.

Comparison of RPG-based FF and DTG-based FF Disrtibuted Heuristics

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

Abstract

Distributed computation of heuristic functions has recently become a hot topic in multiagent planning. One of the classical heuristics most often adapted for distribution is the FF heuristic. In this paper we compare two approaches used to distribute the FF heuristic, which were previously incomparable, as each was implemented in a different planning scheme, a greedy best-first search and a forward-chaining partial order planner.

Introduction

The vast majority of recent (deterministic, cooperative) multiagent planners rely more or less on a heuristic guidance, regardless of the particular planning paradigm used. One of the first such planners was the MAD-A* (Nissim and Brafman 2012) planner. MAD-A* introduced a scheme for multiagent A* search, later generalized to multiagent best-first search (Štolba and Komenda 2014; Nissim and Brafman 2014). In this state-space search scheme, each agent searches using only its own actions, but whenever a public action is used, the resulting state is sent to all other agents and consequently added to their open lists. In MAD-A*, the heuristic evaluation is computed only on the part of the problem known to the evaluating agent (a *projected problem*), no knowledge of other agents is involved. This approach is referred to as a *projected heuristic*.

The main benefits of the projected heuristic approach are i) all classical planning heuristics can be used without modification ii) no communication is necessary (which may also imply higher speed). The drawback is, that the heuristic quality can be significantly lowered. In fact, in (Štolba, Fišer, and Komenda 2015) was shown, that such heuristic estimate can be arbitrarily worse than a heuristic computed on the global problem.

A natural solution is to compute the heuristic in a distributed manner, in order to estimate the value on a global problem. Multiple classical planning heuristics were already treated this way. An inadmissible landmark heuristic (Maliah, Shani, and Stern 2014), an admissible landmark heuristic (Štolba, Fišer, and Komenda 2015) and various relaxation heuristics (Štolba and Komenda 2014). The heuristic computed distributively most often is the classical FF (Hoffmann and Nebel 2001) heuristic (Torreño, Onaindia, and Sapena 2012; Štolba and Komenda 2013; 2014; Torreño, Onaindia, and Sapena 2014), although the resulting heuristics are typically not equal to the original FF.

In this paper, we compare two approaches to the distribution of the FF heuristic. The first approach is based on the original method of FF computation based on Relaxed Planning Graphs (RPGs) and their effective implementation. This method was evaluated using multiagent greedy best-first state-space search in (Štolba and Komenda 2013; 2014). The second approach replaces the Relaxed Planning Graphs with Domain Transition Graphs (DTGs) (Helmert 2006) in order to reduce the communication among agents. The second method was evaluated using a multiagent forward-chaining plan-space search in the FMAP planner (Torreño, Onaindia, and Sapena 2014). Due to the very different planning paradigms, the two methods of distributed FF computation were never directly compared. To bridge this gap, we have re-implemented the DTG-based heuristic in a multiagent greedy best-first state-space search in order to evaluate it and compare it with the RPG-based heuristic.

MA-STRIPS

As the formal framework of the comparison we use the MA-STRIPS (Brafman and Domshlak 2008) formalism. We assume a set of n *cooperative* agents with common goals which search for a multiagent plan solving a planning problem in a *coordinated* fashion. The search is decoupled from the prospective execution of the plan, similarly as in classical (off-line) planning. A multiagent planning problem is formally defined as:

Definition 1. Let a quadruple $\Pi = \langle P, \{A_i\}_{i=1}^n, I, G\rangle$ be a MA-STRIPS planning problem for n agents $\mathcal{A} = \{\alpha_i\}_{i=1}^n$, where:

- P is a finite set of propositions describing facts about the world the agents act in, a state of the world will be denoted as $s \subseteq P$,
- A_i is a finite set of actions an agent α_i can perform
- each action $a \in A = \bigcup_{i=1}^{n} A_i$ has the standard STRIPS syntax and semantics $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$, where $\mathsf{pre}(a) \subseteq P, \mathsf{add}(a) \subseteq P, \mathsf{del}(a) \subseteq P$ represent preconditions, add effects, and delete effects respectively, an action a is applicable in state s iff $\mathsf{pre}(a) \subseteq s$ and the resulting state is $s' = s \cup \mathsf{add}(a) \backslash \mathsf{del}(a)$.
- the sets of actions are pairwise disjoint, that is $\forall i \neq j : A_i \cap A_j = \emptyset$,
- $I \subseteq P$ is the initial state of the world, and
- $G \subseteq P$ is the goal condition defining the goal (final) states of the problem; a state s is a goal state iff $G \subseteq s$.

In order to define a set of fact restricted to one agent, we firstly define a set $\mathsf{facts}(a) = \mathsf{pre}(a) \cup \mathsf{add}(a) \cup \mathsf{del}(a)$ which denote facts required and/or affected by an action a. Then all facts of an agent α_i are defined as $P_i = \bigcup_{a \in A_i} \mathsf{facts}(a)$.

MA-STRIPS provides a scheme for separation of private (internal) information of particular agents and public (common) information which is shared among all agents. Private facts of an agent α_i called α_i-internal are its facts which are not facts of any other agent, formally $P_i^{\mathsf{int}} = P_i \backslash \bigcup_{\alpha_j \in \mathcal{A} \backslash \alpha_i} P_j$. Public facts of an agent α_i are the complement $P_i^{\mathsf{pub}} = P_i \backslash P_i^{\mathsf{int}}$. In a similar manner, we define separation of private and public actions of the agents. Private actions of agent α_i are such actions that does not affect and are not affected by other agents via the public facts, formally

$$A_i^{\mathsf{int}} = \{a | a \in A_i, \mathsf{facts}(a) \subseteq P_i^{\mathsf{int}}\}.$$

Conversely, public actions are $A_i^{\mathsf{pub}} = A_i \backslash A_i^{\mathsf{int}}$.

States and actions restricted to a specific set of facts are projections on that specific set. If a projection is on a set of facts of an agent α_i and all public facts, formally $P_i^{\mathsf{proj}} = P_i \cup \bigcup_{i \neq j} P_j^{\mathsf{pub}}$, we will be using the term α_i-projection of a state, formally defined as $s^{\alpha_i} = s \cap P_i^{\mathsf{proj}}$. Similarly, an α_i-projection of an action $a \in A_i$ is defined as

$$a^{\alpha_i} = \left\langle \mathsf{pre}(a) \cap P_i^{\mathsf{proj}}, \mathsf{add}(a) \cap P_i^{\mathsf{proj}}, \mathsf{del}(a) \cap P_i^{\mathsf{proj}} \right\rangle.$$

FF Heuristic

The FF heuristic builds on the idea of delete relaxation. A relaxed problem Π^+ is obtained from problem Π by relaxing all actions, this means ignoring all delete effects. An action $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$ is transformed to $a^+ = \langle \mathsf{pre}(a), \mathsf{add}(a), \emptyset \rangle$. A solution of relaxed problem Π^+ is a relaxed plan (RP) π^+.

The heuristic itself progresses in two steps:

1. Reachability analysis
2. Relaxed plan extraction

The details of reachability analysis phase are not important for the topic of the paper. In theory it is done by building a Relaxed Planning Graph, a layered oriented graph where alternate layers of reachable facts and reachable actions. Edges are from a precondition fact to an action and from an action to an effect fact. In practice, more effective data structures are often used. The important information necessary for the FF heuristic is that for each fact f the action a which first achieves it (a is the *supporter* of f) is determined. If there are multiple such actions, the supporter is chosen using some tie-breaking mechanism.

The plan extraction phase works from the goal G backwards in the following steps:

1. Initialize $\pi^+ = \emptyset$ a set of unsupported facts U to contain all goal facts: $U \leftarrow G$, a set of supported facts S to contain all facts in the current state: $S \leftarrow s$.
2. Move an unsupported fact p from U to a set of supported facts S and add its supporter a to π^+.
3. Mark all preconditions of a as unsupported if not supported already: $U \leftarrow U \cup (\mathsf{pre}(a) \setminus S)$.
4. Loop 1-3 until all facts in U are either supported or in the current state: until $U \setminus S = \emptyset \vee U \subseteq s$.

The returned heuristic estimate for the state s is either the cost or length of π^+.

DTG-based Heuristic

An alternative solution for the reachability analysis is to use the Domain Transition Graphs, introduced to multiagent planning by the FMAP planner mainly to exploit its properties promising for the distributed variant. To introduce DTGs, we first need to shift from the STRIPS representation to a Finite Domain Representation (FDR) (Helmert 2006).

In FDR, the state space is represented by a set of variables V, each variable $v \in V$ having a finite domain of possible values D_v. In a partial state, some of the variables have assigned a value from their respective domains. A state is a partial state where all variables from V have assigned some value. Assigning a value to a variable can be seen as forming a fact, which is a tuple $\langle v, d \rangle$ where $v \in V$ and $d \in D_v$ representing $v = d$. A partial state can alternatively be seen as a set of facts, such that no two facts contain the same variable. For the sake of simplicity, we can reformulate the definition of actions such that an action $a = \langle \mathsf{pre}(a), \mathsf{eff}(a) \rangle$ where $\mathsf{pre}(a)$ and $\mathsf{eff}(a)$ are partial states. An action a is applicable in a state s if for each $\langle v, d \rangle \in \mathsf{pre}(a)$, the value of v in s is d. For the state s' resulting from the application of a to s holds, that for each $\langle v, d' \rangle \in \mathsf{eff}(a)$, the value of v in s' is d', for variables v' not assigned in the effect of a the value of v' in s' is the same as in s.

For a variable $v \in V$, the DTG_v is a graph where nodes are the values $d \in D_v$. An edge is between two nodes d and d' iff there is some action a s.t. $\langle v, d \rangle \in \mathsf{pre}(a)$ and $\langle v, d' \rangle \in \mathsf{eff}(a)$. The edge $d \rightarrow d'$ is labeled with a set $dd'v$ of all such actions.

To compute a heuristic estimate utilizing the DTG graphs, a modified plan extraction routine can be used:

1. Initialize $\pi^+ = \emptyset$ a set of unsupported facts U to contain all goal facts: $U \leftarrow G$, a set of supported facts S to contain all facts in the current state: $S \leftarrow s$.
2. Select a $\langle v, d_U \rangle \in U$ and find the shortest path from any d_S s.t. $\langle v, d_S \rangle \in S$ to d_U in the DTG_v.
   (a) For each edge $d \rightarrow d'$ on the path, choose an action $a \in dd'v$.
   (b) Add a to π^+, $\mathsf{eff}(a)$ to S and all facts in $\mathsf{pre}(a) \setminus S$ to U.
3. Loop 1-2 until all facts in U are either supported or in the current state: until $U \setminus S = \emptyset \vee U \subseteq s$.

We can see, that the algorithm follows a similar high-level scheme. The difference is, that the reachability is not assessed using a supporter relation based on a RPG, but instead by an existence of a path in the respective DTG. The relaxation here is not achieved by ignoring delete effects, but by accumulating the variable-value pairs in S. Note, that this is in accordance to the most common interpretation of delete relaxation in FDR, which is an accumulating semantics (variables are accumulating values instead of switching). The difference is, that the accumulating semantics is exhibited only in the set of supported facts S, but the DTGs are unaffected and keep their switching semantics.

RPG-based Distributed FF

For comparison, we choose the Lazy FF (Štolba and Komenda 2013; 2014) distribution of the FF heuristic, as the algorithm is much similar to the DTG-based heuristic and differs mainly in the reachability analysis part. The Lazy FF heuristic suffers from over-counting of actions in the relaxed plans (some actions are counted multiple times). In this article, we introduce a new technique how to handle the over-counting in Lazy FF. We take inspiration from the Set-Additive variation of the FF heuristic (Keyder and Geffner 2008), where instead of cost of reaching a fact p, each fact p is associated with a relaxed plan π_p^+ solving a relaxed planning problem where p is the goal $G = \{p\}$. The overall relaxed plan π^+ is then constructed by computing a set unions of the respective fact relaxed plans

$$\pi^+ = \bigcup_{p \in P} \pi_p^+, \tag{1}$$

which is possible as the order of the actions in a relaxed plan can be arbitrary and using any action more than once is redundant. The new set-additive variant of distributed FF heuristic will be denoted SA Lazy FF.

The algorithm for SA Lazy FF follows:

1. The agent α_i initiating the estimation locally computes a projected relaxed plan π^{α_i+} which is a solution of a relaxed projected problem Π^{α_i+}.
2. For a projected action $a^{\alpha_i+} \in \pi^{\alpha_i+}$, the initiator agent α_i sends a request to the action's owner agent α_j. Upon receiving, α_j computes partial RP π^{α_j} from the current state $s \cap P_j^{\mathsf{proj}}$ to the private preconditions of a^+, that is $\mathsf{pre}(a^+) \cap P_j^{\mathsf{int}}$ and sends π^{α_j} to the initiator agent as a reply. The partial relaxed plan π^{α_j} may of course contain projected actions of other agents $\alpha_{k \neq j}$ and also of agent α_i.
3. The initiator agent merges the received relaxed plan π^{α_j} with the initial RP $\pi^{\alpha_i+} \leftarrow \pi^{\alpha_i+} \cup \pi^{\alpha_j+}$ and continues with step 2 until there are no projected actions in π^{α_i+}.
4. The initiator agent returns the cost of π^{α_i+}: $\sum_{a \in \pi^{\alpha_i+}} \mathsf{cost}(a)$.

In step 3, the received action may contain a projection of a public action of agent α_i. If the action was not present in π^{α_i+} it has to be treated as a projection and α_i requests the satisfaction of its private preconditions from itself.

It may happen, that the agent α_j cannot find a plan to satisfy preconditions of the requested action. In that case it returns an empty plan.

For applications requiring "real" private knowledge preservation as proposed in (Borrajo 2013), SA Lazy FF can use hashes as a way of obfuscation (and also communication load reduction) for the communicated private actions. It is questionable, whether some useful information can be extracted from the obfuscated relaxed plans, but if so, this approach may not be suitable for applications requiring such degree of privacy protection.

DTG-based Distributed FF

In the distributed DTG-based FF heuristic, the DTGs of some variables span over multiple agents. For example in a logistics domain, the variable representing the location of a package spans over all agents which can load and unload the package. To preserve privacy, each agent sees only its part of the DTG, the parts of other agents are reduced to a special value $\perp$.

Unlike the original implementation in FMAP, we obtain the $\perp$ value simply from the projections of other agents' public actions. If a projected action requires some value d, there is an edge $d \rightarrow \perp$ added to the DTG, if a projected action produces d, edge $d \leftarrow \perp$ is added. The edges $d \rightarrow \perp$ and $d \leftarrow \perp$ are labeled with the name of the action's owner $\alpha \in d\perp v$ and $\alpha \in dv$ respectively instead of the action itself.

The distributed heuristic estimation algorithm follows the high-level scheme of the centralized version (but operating on the modified DTGs), modified to treat the $\perp$ value in the following way:

1. Initialize $\pi^+ = \emptyset$ a set of unsupported facts U to contain all goal facts: $U \leftarrow G$, a set of supported facts S to contain all facts in the current state: $S \leftarrow s$.

			DTG-based			RPG-based		
Domain	problems	agents	cent.	proj.	dist.	cent.	proj.	dist.
blocksworld	35	4	35	35	35	35	35	35
depot	20	5-12	11	12	11	17	11	**18**
driverlog	20	2-8	18	18	**20**	17	19	16
elevators08	30	4-5	**30**	5	29	11	2	9
logistics00	20	3-7	**20**	11	19	14	**20**	**20**
ma-blocks	24	3-6	19	13	13	**21**	15	**15**
openstacks	30	2	**30**	**30**	21	6	6	6
rovers	18	2-8	16	10	**18**	16	**18**	**18**
rovers-large	20	6-14	6	0	**20**	5	**20**	**20**
satellites	18	2-5	**18**	7	**18**	17	**18**	**18**
satellites-hc	15	5-8	6	0	**13**	6	**13**	8
ma-sokoban	10	2-4	8	8	9	**10**	**10**	**10**
woodworking08	30	7	12	11	8	**30**	24	28
zenotravel	17	2-5	**17**	6	**17**	**17**	**17**	15
Total	307		246	166	**251**	222	228	236

Table 1: Coverage of the DTG-based and RPG-based FF heuristics in the centralized variant (cent.), projected variant (proj.) and distributed variant (dist.)

2. Select a $\langle v, d_U \rangle \in U$ and find the shortest path from any d_S s.t. $\langle v, d_S \rangle \in S$ to d_U in the DTG_v. If there is no such d_S, find the shortest path from $\langle v, \bot \rangle$.
   (a) For each edge $d \to d'$ on the path, choose an action $a \in dd'v$.
   (b) Add a to π^+, $\mathsf{eff}(a)$ to S and all facts in $\mathsf{pre}(a) \setminus S$ to U.
   (c) If $\bot$ is on the path, i.e. $d \to \bot \to d'$, send a request $R = \langle v, d, d', s, S_v \cap P^{\mathsf{pub}} \rangle$, where S_v is S restricted only to the values of v, to all agents $\alpha \in d'v$.
   (d) If $\bot$ is at the beginning of the path, i.e. $\bot \to d'$, send a request $R = \langle v, \bot, d', s, S_v \cap P^{\mathsf{pub}} \rangle$ to all agents $\alpha \in d'v$.
   (e) When all replies received, add the minimum to the heuristic estimate and continue.
3. Loop 1-2 until all facts in U are either supported or in the current state: until $U \setminus S = \emptyset \vee U \subseteq s$.

The agent α' receiving the request R computes a limited version of the heuristic estimate. It searches for the shortest path in the DTG_v of the requested variable v from d to d', or from any value in the received $S_v \cap P^{\mathsf{pub}}$ to d', if d was not in the request. If α' encounters the $\bot$ value again, the respective agents are requested recursively. If the recursion would cycle back to any of already visited agents, the recursion ignores such agent assigning its part of the path the cost of 0. This is a difference from the original FMAP version, where such situation would be penalized with the cost of 1000, effectively treating such state as a dead-end (never expanded).

All agents in the recursion report only the length of the shortest path, ignoring all preconditions of the visited actions. The private parts of the state s sent via the requests can be obfuscated in order to preserve privacy. The shortest paths can be cached in order to speed up the computation and reduce communication.

Experimental Comparison

The experimental comparison of the described heuristics was performed on a set of benchmarks commonly used in the MA planning literature, derived from the classical IPC benchmarks. Each run (per problem) of the planner was limited to 30 min. and 8GB of memory (total for all agents) on a 16 core machine.

There are several infrequently used domains. First, the ma-blocks (Maliah, Shani, and Stern 2014) domain is the classical blocksworld with multiple hands as agents, but with a major difference. There are multiple locations on the table and not all agents can reach all of them. The ma-sokoban domain is a straightforward extension of sokoban by adding multiple robots. In order to do so, completely new problems were designed. Last of all, the rovers-large and satellites-hc are larger instances of the classical domains.

The coverage on the set of domains is shown in Table 1. We first focus on the centralized versions of the heuristics (which were measured using a centralized greedy best-first search). Overall, the DTG-based heuristic solves more problems than the classical FF. The main difference is in the elevators08 and openstacks domains, where the DTG-based heuristic solves significantly more problems. This may be caused by the fact, that the DTG-heuristic ignores costs, while the RPG-based does not. The RPG-based heuristic solves significantly more problems in the woodworking08 domain.

Next, we look on the situation where we compute the centralized versions of the heuristics restricted to each agent's projected problem, similarly as was done in the MAD-A* planner (Nissim and Brafman 2012) (measured using a distributed greedy best-first search).

Here, the projected RPG-based heuristic solves significantly more problems, especially in the domains with extensive private knowledge such as logistics00, rovers, satellites and their large variants. It also solves more problems than the centralized version of the RPG-based heuristic, which is caused mostly by the rovers-large and satellites-hc domains which are hard and extremely suitable for multi-agent factorization.

Finally, we proceed to the main topic of the paper and that is the comparison of the distributed versions of the DTG-based and RPG-based FF heuristics. We measured the performance of the distributed versions of the heuristics using a distributed greedy best-first search. In this setting, the search using the DTG-based heuristic solves about 5% more problems than the search using its RPG-based counterpart.

The results follow a pattern very similar to the results of the centralized versions. The DTG-based heuristic is significantly better in the elevators08 and openstacks domains, again probably because it ignores the costs of actions. Again, in the woodworking08 domain the RPG-based heuristic solves significantly more problems.

Domain	prob.	DTG-based	RPG-based
blocksworld	35	35	35
depot	20	11	**16**
driverlog	20	**20**	16
elevators08	30	29	**30**
logistics00	20	19	**20**
ma-blocks	24	13	**15**
openstacks	30	21	**30**
rovers	18	18	18
rovers-large	20	20	20
satellites	18	18	18
satellites-hc	15	**13**	8
ma-sokoban	10	9	**10**
woodworking08	30	8	**22**
zenotravel	17	**17**	15
Total	307	251	**273**

Table 2: Coverage of the DTG-based and RPG-based FF heuristics in the distributed variant, both ignoring costs of actions.

To make the comparison more fair, we have tested the RPG-based heuristic also if the action costs are ignored. The results for coverage are shown in Table 2. The results confirms the hypothesis that the bad results of the RPG-based heuristic in the elevators08 and openstacks domains were caused by the use of action costs instead of simple plan length. In the case of unit costs, the RPG-based heuristic performs significantly better in terms of coverage.

We compare not only the coverage, but also the search speed using the distributed variants of the heuristics. Multiple patterns can be observed on various domains. In the blocksworld domain, neither of the heuristics dominate, although in one of the problems,

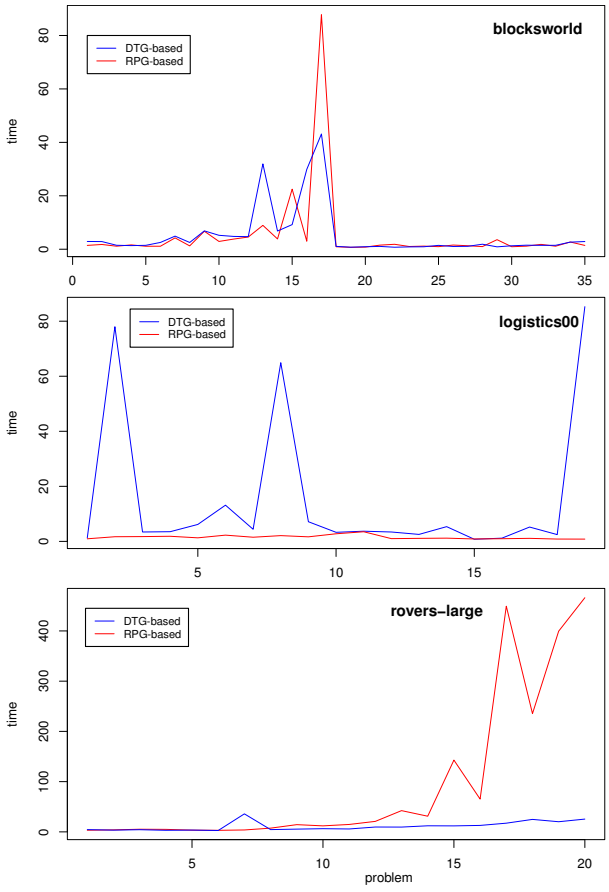


Figure 1: Comparison of solution time (s) on each problem solved by both heuristics.

the RPG-based heuristic takes significantly longer time to find the solution. In the logistics00 domain, the RPG-based heuristic finds solution faster on all problems and scales better, but the worse performance of the DTG-based heuristic does not have an effect on the coverage. In the contrary, the RPG-based heuristic scales significantly worse in the rovers-large domain, where the effect of scaling is not apparent in the coverage, but the trend suggests that unlike the DTG-based heuristic, the RPG-based heuristic would not be able to solve even larger problems.

Conclusion

We have re-implemented the DTG-based variant of distributed FF heuristic originally used in the FMAP partial-order planner in order to compare it with a classical RPG-based distributed FF heuristic using a multi-agent distributed greedy best-first search.

In most domains, both heuristics have very similar results in terms of coverage. The DTG-based heuristic solves more problems in the elevators08 and openstacks domains, which may be caused by the fact that the DTG-based heuristic ignores the costs of the actions, which gives it an advantage especially in those domains.

In the woodworking08 domain, the situation is inverse and the RPG-based heuristic dominates. Altogether, the DTG-based heuristic solves slightly more problems. But if the RPG-based variant ignores the action costs, the coverage results shift in favor of the RPG-based heuristic.

In terms of solution time, the DTG-based heuristic seems to exhibit shorter times and better scaling behavior, except for some domains, where the solution time is worse than the solution time of the RPG-based heuristic, but the scaling behavior still does not seem to be prohibitive.

Similar pattern can be observed in the results of the centralized versions of the heuristics in a centralized greedy best-first search. What is notable is, that both distributed heuristics outperform their centralized counterparts. This is most probably caused by the inclusion of some very factorization-friendly domains. Also, the distributed planner was not communicating over network, but using an in-process communication, which might have given it some advantage.

We have also tested the distributed search using projected versions of the heuristics. The results show, that unlike the RPG-based heuristic, the DTG-based one is not very suitable for use as a projected heuristic.

Acknowledgments This research was supported by the Czech Science Foundation (grant no. 15-20433Y) and by the Grant Agency of the CTU in Prague (grant no. SGS14/202/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

Borrajo, D. 2013. Plan sharing for multi-agent planning. In *Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13)*, 57–65.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, 28–35.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

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

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, 588–592.

Maliah, S.; Shani, G.; and Stern, R. 2014. Privacy preserving landmark detection. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, 597–602.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*, 1265–1266.

Nissim, R., and Brafman, R. 2014. Distributed heuristic forward search for multi-agent planning. *JAIR* 51:293–332.

Štolba, M., and Komenda, A. 2013. Fast-forward heuristic for multiagent planning. In *Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13)*, 75–83.

Štolba, M., and Komenda, A. 2014. Relaxation heuristics for multiagent planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, 298–306.

Štolba, M.; Fišer, D.; and Komenda, A. 2015. Admissible landmark heuristic for multi-agent planning. In *To appear at (ICAPS'15)*.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014. FMAP: Distributed cooperative multi-agent planning. *Applied Intelligence* 41(2):606–626.

Torreño, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, 762–767.

Leveraging FOND Planning Technology to Solve Multi-Agent Planning Problems

Christian Muise, Paolo Felli, Tim Miller, Adrian R. Pearce, Liz Sonenberg
Department of Computing and Information Systems, University of Melbourne
{christian.muise, paolo.felli, tmiller, adrianrp, l.sonenberg}@unimelb.edu.au

Abstract

Single-agent planning in a multi-agent environment is challenging because the actions of other agents can affect our ability to achieve a goal. From a given agent's perspective, actions of others can be viewed as non-deterministic outcomes of that agent's actions. While simple conceptually, this interpretation of planning in a multi-agent environment *as* non-deterministic planning is challenging due to the non-determinism resulting from others' actions, and because it is not clear how to compactly model the possible actions of others in the environment. In this paper, we cast the problem of planning in a multi-agent environment as one of Fully-Observable Non-Deterministic (FOND) planning. We extend a non-deterministic planner to plan in a multi-agent setting, given the goals and possible actions of other agents. We use the other agents' goals to reduce their set of possible actions to a set of *plausible* actions, allowing non-deterministic planning technology to solve a new class of planning problems in first-person multi-agent environments. We demonstrate our approach on new and existing multi-agent benchmarks, demonstrating that modelling the other agents' goals reduces complexity.

1 Introduction

Synthesising a plan for an agent operating in a multi-agent domain is a challenging and increasingly important problem (Bolander and Herzig 2014). When an agent acts in an environment with other agents, it must be aware of the possible actions of others and how they might affect the continued feasibility of achieving a goal. In this work, we focus on a version of the multi-agent planning problem where the goals and possible actions of all agents are known, and the state of the world is fully observable. These restrictions capture a wide class of multi-agent problems, including many collaborative and competitive games. In Fully Observable Non-Deterministic (FOND) planning, an agent must synthesize a policy to achieve a goal with some guarantee. The actions in a FOND problem are *non-deterministic*; that is, any one of a number of outcomes may occur when the agent executes the action. The world is fully observable, so the agent knows the outcome of the action after execution.

In this paper, we cast the first-person view of a multi-agent planning (MAP) problem as a FOND planning problem, taking advantage of recent advances in that field. The key to our approach is that the choice of action by other agents can be viewed as a non-deterministic effect of our own action. Any valid sequence of actions from other agents can potentially result in a different state of the world, which we encode as the non-deterministic outcomes of our own action. As an example, consider the partially played game of Tic-Tac-Toe in Figure 1 where we are playing **X** and it is our turn. Playing the right column / middle row can subsequently lead to four possible states where it is our turn again: these are the non-deterministic outcomes of our move.

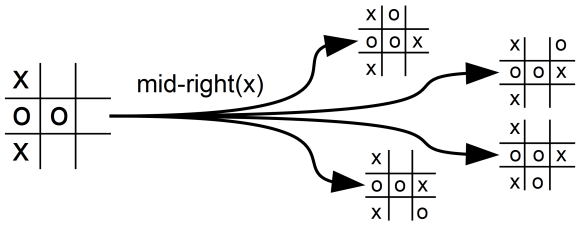


Figure 1: Tic-Tac-Toe example from **X**'s perspective

A number of recent advances in FOND planning have improved the scalability of the solvers significantly (Fu et al. 2011; Alford et al. 2014; Muise, McIlraith, and Beck 2012; Ramirez and Sardina 2014). We wish to take advantage of these improvements for solving MAP problems, but achieving this is not simply a matter of encoding the problem in a form that FOND planners can solve. If we naively encode the problem as input to a FOND planner, then the encoding must enable the planner to take account of the context in which other agents execute their actions: in particular to respect that for any particular state of the world, there may be only a small subset of actions that are applicable. As a result, to capture the behaviour of other agents as non-deterministic action effects must involve either: (1) enumerating the actions in some way to find those that are applicable; or (2) fully specifying the actions and effects to account for every situation that the other agents may encounter. Both options result in a combinatorial explosion on the size of the encoded FOND problem. Our approach takes an alternative direction and modifies an existing solver to support consideration of applicable actions, allowing us to keep the encoding of the MAP problem compact and naturally expressed.

Considering all of the applicable actions for other agents can result in a prohibitively large amount of non-

determinism. To mitigate this, we consider restricting the possible actions of another agent to a set of *plausible* actions, which are those that lead to states with the highest heuristic value for the agent given their goal. We do not consider how the goals of others are known, but in many scenarios the goal is known apriori or can be inferred; e.g., using goal recognition (Ramírez and Geffner 2010). This gives us a general means to focus on the worst case scenario, if we are considering agents with conflicting or opposing goals, or to focus on the expected scenarios if we are working towards the same goal as other agents (e.g., on the same team). Note that we do not presume authority over our teammates actions – rather, we wish to plan for what they would plausibly do given that they share a goal similar or the same as our own.

To realize our approach, we modified the state-of-the-art FOND planner PRP (Muise, McIlraith, and Beck 2012). We evaluate the approach on three multi-agent domains adapted from existing problems. The evaluation addresses our strategies for restricting non-determinism, and demonstrates the capability to solve MAP problems with existing FOND planning technology. By using a FOND planner at the core of our approach, which subsequently uses a classical planner at its core, we take advantage of every new advancement in either FOND or classical planning. We also discuss the issues surrounding the interpretation of MAP as FOND.

Next we describe the background concepts and notation required for our approach. Following this, we describe our approach for solving MAP problems as FOND, and how to heuristically reduce the amount of non-determinism in the domain. Next, we provide details on an evaluation of our approach, and we conclude with a discussion of related work and future directions.

2 Background

2.1 (FOND) Planning Notation

We describe briefly here the requisite planning background and notation (cf. Ghallab et al. (2004) for a full treatment). A *Fully Observable Non-Deterministic* (FOND) planning problem P consists of a tuple $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$; $\mathcal{F}$ is a set of fluents, and we use $\mathcal{S}$ as the set of all possible states; $\mathcal{I} \subseteq \mathcal{F}$ is the initial state; $\mathcal{G} \subseteq \mathcal{F}$ characterizes the goal to be achieved; and $\mathcal{A}$ is the set of actions. An action $a \in \mathcal{A}$ is a tuple $\langle \text{Pre}_a, \text{Eff}_a \rangle$ where $\text{Pre}_a \subseteq \mathcal{F}$ is the precondition (i.e., the fluents that must hold for a to be executable) and Eff_a is a set of one or more outcomes. An action with only one outcome is *deterministic*; otherwise it is *non-deterministic*. Each $e \in \text{Eff}_a$ contains a set of positive and negative effects that update the state of the world, and we use $Prog(s, a, e)$ to denote the state reached when action a is executed with outcome e in state s. After executing an action, *exactly one* outcome is used to update the state of the world. The planning agent does not know which outcome in advance, and so must plan for every contingency.

Following Cimatti et al. (2003), we consider three types of solution to a FOND planning problem: *weak*, *strong*, and *strong cyclic*. The representation for all three is in the form of a policy P that maps the state of the world to an action: $P : \mathcal{S} \rightarrow \mathcal{A}$. We say that a state s' is *reachable* from s by the plan P if it is equal to s or if there exists some other state s'' reachable from s such that s' is a possible successor to the execution of $P(s'')$ in state s''.

P is a *weak plan* if there is some state that is reachable from $\mathcal{I}$ where $\mathcal{G}$ holds. P is a *strong cyclic plan* if for every state s that is reachable from $\mathcal{I}$, there is another state reachable from s where $\mathcal{G}$ holds. Finally, P is a *strong plan* if it is a strong cyclic plan and no state s reachable from the initial state is reachable from the possible successors of s (i.e., a reachable cycle is impossible). The distinction between strong and strong cyclic plans is, as the terms imply, the presence of cycles in the reachable state space of the plan P. Finally, the *all-outcomes determinization* of a FOND problem $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ is the classical planning problem $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A}' \rangle$ where $\mathcal{A}'$ is defined as: $\mathcal{A}' = \{\langle \text{Pre}_a, [e] \rangle \mid a \in \mathcal{A} \text{ and } e \in \text{Eff}_a\}$

Solving the all-outcomes determinization is a technique used to compute weak plans, as any classical plan for the all-outcomes determinization represents a weak plan for the original FOND problem.

2.2 A First-person View of Multi-agent Planning

There are a variety of multi-agent planning (MAP) formalisms (e.g., see (Brafman and Domshlak 2008; Brenner 2003; Kovacs 2012)), however, we consider a first-person view of planning in the simplified setting where the world is fully known and is observable to all agents. Actions may be non-deterministic, but outcomes are known immediately by all agents in the domain. This setting is most related to the NAPL syntax for multi-agent planning introduced in (Jensen and Veloso 2000).

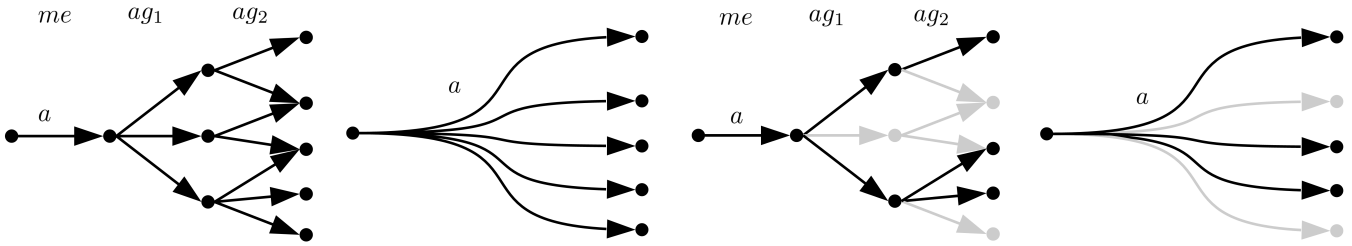
As input to the planner, we have a collection of agents, each of which has its own set of actions, $\mathcal{A}_i$, and possibly its own goal $\mathcal{G}_i \subseteq \mathcal{F}$. The planning agent's task in our MAP setting is to synthesize a policy that maps states to actions given the uncertainty of what other agents will do, and the analogies to FOND solutions are direct. Ideally the goal state of the planning agent should be guaranteed, but if this is not possible the agent should strive to achieve robust policies that achieve the goal with a high probability of success. While we do not compute optimal solutions, we do evaluate the policies based on this notion of solution quality.

3 Approach

First, we present our general approach for solving fully observable MAP problems using FOND planning, and discuss the issues that would arise if we instead used FOND as a black box. We then describe how to heuristically restrict the problem's non-determinism in cases where the goals of other agents in the domain are given.

3.1 MAP as FOND

Our method of solving MAP problems as FOND is straightforward on the surface, but powerful in practice: *we view the set of possible states that result from other agents conducting actions as non-deterministic outcomes to the planning agent's own choice of action*. Figure 2a shows how, in a setting with 3 agents, {*me*, ag_1, ag_2}, *me* choosing action a can



(a) Full MAP execution (b) Full FOND interpretation (c) Plausible MAP execution (d) Plausible FOND outcomes

Figure 2: Example execution for agents ag_1 and ag_2 after *me* executes action a. Subfigures (c) and (d) show plausible outcomes.

lead to 5 different states depending on the actions that ag_1 and ag_2 decide to execute on their turn. Figure 2b shows the conceptual interpretation of our decision to perform action a. Intuitively, we care about only the states that we may arrive in after a is executed, so we can therefore view the collective decisions of other agents as non-determinism in the world if we choose action a.

For simplicity, we assume agents execute in round-robin fashion according to the sequence $\overrightarrow{ag}$, although this is not strictly necessary to apply our approach.[1] The planning agent, labelled *me* and corresponding to agent ag_0 in $\overrightarrow{ag}$, acts first followed by every agent from $\overrightarrow{ag}$; and then the process repeats. Thus, for every action $a \in \mathcal{A}_i$ in agent i's action set, the precondition of a is augmented with a fluent stating that agent i is the current acting agent, and the effect is augmented with a proposition stating that agent i+1 mod $|\overrightarrow{ag}|$ is the next acting agent. Initially, *me* is the current actor. We use $App(s, ag)$ for $s \in \mathcal{S}$ and $ag \in \overrightarrow{ag}$ to signify the actions that are applicable by agent ag in state s. $App(s, ag)$ is empty if ag is not the current acting agent for state s. Figure 2a shows how one action a executed by *me* can be followed by three possible actions of ag_1 and then potentially seven different actions by ag_2 before agent *me* has a turn again.

Not all FOND planners are built using the same solving technique, but many approaches (e.g., NDP (Alford et al. 2014), FIP (Fu et al. 2011), and PRP (Muise, McIlraith, and Beck 2012)) build a policy by exploring the reachable state space and augmenting the policy with new plan fragments when encountering a previously unseen state. Adapted from the description of PRP (Muise, McIlraith, and Beck 2012), Algorithm 1 gives a high level view of computing a policy.

The GenPlanPairs($\langle \mathcal{F}, s, s_*, \mathcal{A} \rangle, P$) algorithm attempts to find a weak plan from s to the goal *assuming that we are omniponent* (i.e., we control the actions of others). It does this by simply merging actions from all agents into a single set and planning as normal using this set; that is $\mathcal{A} = \bigcup_{i \in \overrightarrow{ag}} \mathcal{A}_i$,. This is the result of using the all-outcomes determinization, which assumes that the world is deterministic and we can choose any outcome of a non-deterministic action.

The key distinction between Algorithm 1 and PRP's approach is that we generalize line 11. In this context, the original PRP algorithm defines GenerateSuccessors(s, a) to be: $\{Prog(s, a, e) \mid e \in \text{Eff}_a\}$.

[1] If joint actions are required, we need only to be able to enumerate the different possible action combinations given the state of the world and a single action choice for our agent.

Algorithm 1: Generate Strong Cyclic Plan

Input: FOND planning task $\Pi = \langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$
Output: Partial policy P

```
1  Initialize policy P
2  while P changes do
3      Open = {I}; Seen = {};
4      while Open ≠ ∅ do
5          s = Open.pop();
6          if Satisfies(s, G) ∧ s ∉ Seen then
7              Seen.add(s);
8              if P(s) is undefined then
9                  GenPlanPairs(⟨F, s, G, A⟩, P);
10             if P(s) is defined then
11                 for s' ∈ GenerateSuccessors(s, P(s)) do
12                     Open.add(s');
13     ProcessDeadends();
14 return P;
```

Given the agent sequence $\overrightarrow{ag}$, GenerateSuccessors(s, a) enumerates the *applicable* options for each agent in turn, including the non-deterministic effects of their possible actions. Algorithm 2 outlines this procedure in detail.

Line 6 is critical because it determines the options to consider for another agent, and later in the paper we present alternatives to $App(s', ag)$ to improve the problem efficiency.

In addition to modifying GenerateSuccessors(s, a), we also account for the multi-agent setting. For example, during the computation of partial states for the policy and deadend detection, the variable representing the current acting agent is maintained in all cases. While not strictly necessary for soundness, it helps the planner to avoid focusing on unreachable parts of the state space.

Another modification was made to handle deadends: when PRP detects a deadend, it creates a *forbidden state-action pair* that avoids actions that have at least one outcome leading to the deadend. We modified this to cycle through the list of agents in reverse to prevent the planning agent from executing actions that could lead to deadends. For example, consider Figure 1. One omnipotent strategy may be to play in the top middle spot, as it brings us close to win-

Algorithm 2: GENERATESUCCESSORS(s, a)

Input: State s, action a
Output: Set of successor states S

1 $S = \{Prog(s, a, e) \mid e \in \text{Eff}_a\}$;
2 **for** $i = 1 \ldots |\overrightarrow{ag}|$ **do**
3 $\quad ag = \overrightarrow{ag}[i]$;
4 $\quad S' = \emptyset$;
5 $\quad$ **for** $s' \in S$ **do**
6 $\quad\quad$ **for** $a' \in App(s', ag)$ **do**
7 $\quad\quad\quad S' = S' \cup \{Prog(s', a', e) \mid e \in \text{Eff}_{a'}\}$;
8 $\quad S = S'$;
9 **return** S;

ning with the top row. However, doing so means that O has a winning move at the middle right spot, which is a deadend for us. The modified version for the forbidden state-action procedure will create rules that forbid every move other than playing the one shown that blocks O from winning.

We now turn our attention to various considerations when solving MAP problems with FOND technology.

The value of doing nothing When planning in a multi-agent environment, it is important to consider whether or not to use *noop* actions. These actions simply change the current acting agent into the next agent in sequence without changing the state of the world. The advantage of allowing noop actions for the other agents is that the computed policies can be far more compact: if other agents cannot interfere with our plan, assigning noop actions to them will produce a compact policy. Another benefit of using noop actions is that for the acting agent ag in state s, the set $App(a, ag)$ will always be non-empty.

On the other hand, including a noop action can increase the applicable action space, causing the planner to work harder to compute a solution. This is evident in the Tic-Tac-Toe domain with the goal of drawing the game — solving the problem to completion without noop is roughly an order of magnitude faster than with. Ultimately, the decision to include noop actions should be made on a per-domain basis.

(Un)Fair non-determinism Typically, FOND planning assumes fairness (Cimatti et al. 2003): if an action is executed infinitely many times, every non-deterministic outcome will occur infinitely often. Agents are, of course, not always fair. By using a FOND planner, we are synthesizing policies under the assumption of fairness, although we evaluate the policies without this assumption, as discussed in Section 4.

While this assumption is not ideal, it does serve as a natural relaxation of the full multi-agent setting. Ultimately, it would be safer to produce strong plans rather than strong cyclic plans. There are encoding techniques that we do not discuss here that force all solutions to be strong plans, but it is crucial to observe that *many domains are inherently acyclic*. This means that a FOND planner, even with the assumption of fairness, will produce a strong plan. Examples include any game or setting where an agent can never return to the same state (e.g., Tic-Tac-Toe, board games, etc). In these settings, the fairness assumption does not play a role: an action will never occur infinitely often.

Winning -vs- not losing In settings such as Tic-Tac-Toe, where draws are possible, there is no distinction between losing by a wide margin and drawing a game: if we cannot win, then the state is a deadend. This falls under a generalization of goal achievement where we would like to satisfy a hierarchy of objectives: e.g., I would like to win, and if I cannot win I would like to draw.

Although the goal could be "any final state that is not a loss", as we did for one of the Tic-Tac-Toe problems,this falls short of the goal to create a policy that wins whenever possible. It is unclear how to evaluate the quality of a policy when multiple objectives are at play. For example, is a policy that wins more often but loses some of the time better than a policy that wins less often but never loses? We leave this question as part of future work.

Issues with FOND as a black box We considered a variety of approaches for encoding MAP problems directly to use FOND planners in an off-the-shelf manner. However, this leads to complications, making the approach unmanageable. Whether we use a monolithic action that captures all of the possibilities for an agent, or many individual actions that are all considered before the agent's turn is up (with only one being accepted), we would need heavy use of conditional effects. The only state-of-the-art FOND planner capable of solving problems with conditional effects (Muise, McIlraith, and Belle 2014) would be hampered by the range of conditions on the encoded actions — all savings due to leveraging state relevance would be lost, and as a result the size of the policies would grow exponentially.

In general, encoding the options for how other agents can act as part of the FOND problem itself comes with a variety of issues that range from prohibitively many actions to overly-restricted conditional effects. We sidestep these obstacles by modifying the FOND planner directly, as outlined in 3.3.

3.2 Reducing Non-determinism

Though we avoided the difficulties discussed above by modifying a FOND planner directly, non-determinism can still be unwieldy. If each agent has an average of k actions applicable in an arbitrary state of the world, the number of non-deterministic successors is on the order of $O(k^{|\overrightarrow{ag}|})$. Here, we consider restricting the reasoning to only a subset of the applicable actions, thus making k a low constant.

At Line 6 of Algorithm 2, rather than considering all actions in $App(s, ag)$, we use a *plausibility function* Γ, where $\Gamma(s, ag) \subseteq App(s, ag)$. Some examples include:

- $\Gamma_{full}(s, ag) = App(s, ag)$: The original set of all applicable actions for agent ag.
- $\Gamma_{rand_k}(s, ag) = \text{RANDOM}(\Gamma_{full}(s, ag), k)$: A random subset of (maximum) size k which is drawn from the set of applicable actions for agent ag.

Figures 2c and 2d show an example of using plausible action for agents ag_1 and ag_2, after agent *me* executes action a. Notice that from the non-deterministic perspective, the three plausible outcomes are entirely disassociated from the agents that lead us there. When using a random subset of the actions as the plausibility function, we will indeed reduce the amount of non-determinism in the encoded domain. However, this will clearly ignore some important actions.

Goal-based plausibility In many multi-agent settings, the goals of the agents are known or can be inferred. Examples include any competitive or collaborative game where the goal is to win, and teamwork scenarios where the goal is common among all agents. If not known, we can try to infer the goals (e.g., see (Ramírez and Geffner 2010)). For this work, we assume that we have every agents' goal.

We consider a heuristic for calculating the plausible actions based on these goals. Given the goals of the other agents, we limit the set of plausible actions to those that work towards the goal using any state heuristic function. We extend the plausibility function to include the goal function $\mathcal{G}$, which maps an agent to its goal:

$$\Gamma(s, ag, \mathcal{G}) \subseteq App(s, ag)$$

Considering an agent's goal enables a wide variety of plausibility functions, such as actions at the start of a plan for the other agent's goal, and nesting the reasoning of other agents so that the next agent considers the goal of the following agent. In this paper, we consider only one possibility: we select the top k actions based on successor state quality using a given state heuristic function.

Definition 1. Best Successor Plausibility Function

Let h be a state heuristic function that maps a state and goal to a value. Given the state s, current agent ag, and goal function $\mathcal{G}$, we define $\Gamma_{h_k}(s, ag, \mathcal{G})$ to be the function that returns the top k actions from $App(s, ag)$ ranked by the following scoring function:

$$Score(s, a, ag, \mathcal{G}) = \max_{e \in \text{Eff}_a} h(Prog(s, a, e), \mathcal{G}(ag))$$

We can use any existing state heuristic in place of h. For our current implementation, we use a variant of the FF heuristic where successors are sorted first based on whether or not the action was a "helpful operator" (Hoffmann and Nebel 2001), and then subsequently based on the estimated distance to the goal. We prioritize helpful operators to encourage the planner to consider the actions that seem helpful for the other agent. As a plausibility function, this may work well in some domains and poorly in others. Nonetheless, it represents a reasonable first step for evaluating the potential to reduce the non-determinism in the problem.

3.3 Modified FOND planner

We made a number of modifications to the planner PRP to improve the efficiency of non-deterministic planning. In Section 3.1 we discussed the key modifications that we made to account for the multi-agent setting. We also changed other aspects of the planner including: (1) parsing the various goals of the agents and planning for the appropriate goal as discussed in Section 3.2; (2) always keeping the fluent that represents the active agent in the partial states for the policy; (3) disabling the feature that attempts to repair a policy locally before planning for the goal; and (4) changing the processing of unhandled states from a depth-first to a breadth-first manner (i.e., the *Open* data structure in Algorithm 1 was made into a queue as opposed to a stack). The latter three changes are not strictly required, but did improve the efficiency of the problems that we tested by a fair margin. Additionally, we were forced to disable PRP's "strong cyclic detection" feature, because conceptually it would need significant changes in order to work within the multi-agent setting.

4 Evaluation

As our approach opens FOND planning to a new class of problems, there are no publicly available benchmarks to evaluate on as far as we are aware. Instead, we provide new benchmark problems for three domains: Blocksworld, Tic-Tac-Toe and Sokoban. We use these to evaluate our proposed strategies for mitigating non-determinism, and the general ability of the planner to solve fully-observable MAP problems.

Experiment design We ran the generated policies against 1000 simulation trials, in which the moves of other agents were selected by taking the best applicable action measured using monte carlo rollouts, with the stopping condition of achieving the agent's goal. Policies were generated by running the planner for a maximum of 30 minutes and with a limit of 2Gb memory. If the planner did not complete in the time limit, the best policy found so far was returned. For each problem, we report on the following:

1. Success rate: The percentage of the 1000 trials that ended in success for the planning agent.
2. Policy size: The number of partial state-action pairs in the best found policy. Note that because of relevance analysis, the policy size typically is far smaller than the number of actual states that it can handle.
3. Planning time: The number of seconds required to compute the policy. **30m** indicates that the best policy found by the 30-minute mark was used. ✘ indicates that the planner ran out of memory, and no plan was returned.

Results

Table 1 show the results for each of the problems tested, and we discuss the results for each domain in turn.

Blocksworld The Blocksworld domain includes the first 10 problems from the FOND benchmark set of the 2008 International Planning Competition (IPC), with one additional agent added. We set the goal of the second agent to be the same as the acting agent, so this is a collaborative task. Note that in this domain, the actions for either agent may be non-deterministic (e.g., blocks may slip out of the hand).

		Blocksworld										Tic-Tac-Toe				Sokoban				
	N	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p1	p2	p3	p4	p1	p2	p3	p4	p5
Success	1	38	63	31	20	0	4	0	19	12	11	0	1	45	48	100	99	97	67	28
rate (%)	2	62	66	61	73	17	22	47	81	18	39	0	2	47	100	100	98	96	98	100
	3	92	97	90	96	49	62	81	90	84	76	0	4	47	100	100	98	96	97	100
	4	100	100	100	100	95	70	89	100	98	86	16	3	47	100	100	✘	✘	99	100
	∞	100	100	100	100	100	68	85	100	100	100	100	100	79	100	100	✘	✘	✘	✘
	Rnd	100	100	100	100	100	68	88	100	100	100	39	81	52	100	100	71	48	37	✘
Policy	1	32	27	31	111	49	25	68	63	43	30	42	47	14	7	11	27	27	138	906
size	2	48	77	78	316	141	99	175	217	73	83	88	107	26	15	11	26	31	5990	7891
	3	125	114	157	576	298	246	445	459	126	164	121	260	19	15	11	26	31	10952	10223
	4	337	236	593	932	757	973	892	830	593	526	871	250	22	15	11	✘	✘	10312	9270
	∞	550	286	631	1113	1149	785	987	699	867	818	1358	651	69	15	11	✘	✘	✘	✘
	Rnd	586	444	671	1045	1076	662	1006	763	745	818	827	606	27	12	11	11	11	11	✘
Planning	1	0.01	0.01	0.01	0.02	0.02	0.01	0.02	0.02	0.02	0.01	5	1	0.01	0.01	0.06	0.08	0.08	0.46	195
time (s)	2	0.02	0.04	0.04	0.14	0.06	0.04	0.08	0.12	0.02	0.04	155	11	0.26	0.01	0.06	0.08	0.10	30m	30m
	3	0.06	0.06	0.06	0.32	0.24	0.16	0.56	0.36	0.04	0.10	658	40	0.50	0.01	0.06	0. 10	0.12	30m	30m
	4	0.24	0.14	0.62	1.02	1.10	1.08	0.98	0.90	0.68	0.54	978	39	7.34	0.01	0.06	✘	✘	30m	30m
	∞	0.14	0.06	0.26	0.44	0.46	0.30	0.40	0.22	0.32	0.48	1765	114	39.32	0.01	0.06	✘	✘	✘	✘
	Rnd	0.20	0.12	0.28	0.44	0.44	0.28	0.44	0.26	0.28	0.36	30m	130	0.01	0.01	0.08	0.06	0.08	0.08	✘

Table 1: Success rate over 1000 simulated trials, generated policy size, and the time to synthesize a plan. ✘ indicates a memory violation, 30m indicates the solving was capped at 30 minutes, and N indicates the level of restricted non-determinism (∞ meaning no restriction and Rnd meaning a random subset of 3 applicable actions).

We found that in general, the policies were easy to compute for any level of restricted non-determinism, and the quality of the policies that restrict possible actions to 3 or 4 achieved near perfect performance in all problems. The notable exceptions are problems p6 and p7. For these, the loss in quality is a direct result of the lack of fairness in this domain. For many simulations, both agents simply "did nothing" expecting that the other would make the first move. This reveals the need for some form of deadlock-breaking mechanism in modelling problems that involve multiple agents.

The good performance of the random outcomes is to be expected in such a collaborative setting. However, because the search is not focused on the other agents plausible actions, the subsequent size of the policies increase.

Sokoban For the Sokoban domain, each player must push a box to a goal location, while an opposing agent attempts to push it to a different location. Across the four problems, the second agent starts closer to both our starting location and the block (i.e., in problem p1 the other agent cannot interfere, while in problem p5 the other agent can interfere to a large extent). The domain is inherently competitive.

We found an interesting threshold for the construction of policies in this domain. If the exploration considers any behaviour of the opponent that attempts to thwart our approach to the goal, the planner becomes overwhelmed handling the deadends. Partially, this is due to the type of deadends that occur in Sokoban, and which the underlying planner does not detect easily. Table 1 shows the effect of this as either the maximum time limit (**30m**) or memory limit (✘).

There is an interesting distinction between plans that consider only a few outcomes (which is quite effective) and those that scrutinize all outcomes (which run out of memory in 4 of the 5 problems). When focusing on the actions that help the opponent to achieve its goal, the planner can find a highly successful solution. It struggles, however, when considering actions that serve no other purpose for the opponent than to prevent its own goal, as these include actions that push the block to a position that is no use for any agent.

Tic-Tac-Toe For Tic-Tac-Toe, problems p1 and p2 start with an empty board and our goal is to draw while the opponent's goal is to win. In problem p2, we do not allow for noop actions, and as discussed earlier the performance improvement is substantial. The low success rate of the reduced non-determinism is a result of how poorly our heuristic function approximates the simulated behaviour of the other agent. Randomly selecting 3 actions for the opponent (i.e., roughly half of the applicable actions), on the other hand, covers a wider set of states and thus improves the success rate. However, our solver was able to find the perfect strategy to ensure a draw, when given enough time.

In problems p3 and p4, both players' goal is to win. Problem p3 starts with an open board that has no guaranteed strategy to win, and problem p4 starts with one move each (beginning in the bottom corners) so that a perfect strategy exists. In the latter case, we find that very few of the outcomes are required to generate a highly successful policy, and the perfect strategy is computed by the planner in a fraction of a

second.

Problem p3 is the typical starting configuration for Tic-Tac-Toe, and it poses an interesting challenge for FOND technology. State relevance, deadend detection and avoidance, effective search heuristics, etc., all play an important role in producing a successful and compact policy. Further, because no state is repeatable in the game (aside from the potential of noops), the assumption of fairness is not a concern.

5 Summary and Related Work

In this work, we presented a novel application of FOND planning in multi-agent environments based on the intuition that actions taken by others in the world can be viewed as non-deterministic outcomes of our own choices. This is in contrast with treating the environment and the other agents as an explicit adversary, which is the idea behind game structures used in verification (Piterman, Pnueli, and Sa'ar 2006), which in turn are at the base of ATL interpretation structures (Alur, Henzinger, and Kupferman 2002), in which a successor state is selected depending on the action that each agent performs. The latter approach is the one captured, in a planning setting, by the notion of joint state-action pairs in Bowling et al. (2003). Although conceptually different, these two approaches allow us to model actions whose effects are not completely determined by the state of the world.

The work of Bowling et al. (2003) considers a setting similar to ours where the goals of the other agents are known. The distinction, however, is that they use the model of agents' goals to devise a game-theoretic notion of equilibria for the agents, whereas we use the information to improve the efficiency of reasoning.

The main contribution of our work is the realization of the above intuition to leverage the recent advances in non-deterministic planning for solving problems in a multi-agent setting. A second key contribution is a means to reduce the non-determinism in the domain by restricting the set of *possible* actions for other agents to those that are *plausible* given their goal. We discussed some issues that arise when using our approach, and demonstrated its ability to solve multi-agent problems on a new suite of benchmarks that include both collaborative and competitive tasks.

The connection that we make between multi-agent planning and FOND planning presents an exciting initial step towards a range of more sophisticated techniques. The generality of a plausibility function opens the door to techniques ranging from nested simulations of agent behaviour to online learning methods that model other agents in the domain. It also provides an avenue to incorporate UCT and sample-based approaches (e.g., the PROST planner (Keller and Eyerich 2012)) with the more systematic search used by determinization-based planners such as PRP. As evidenced by the running example in this paper, our approach lends itself naturally to competitive games: we hope to apply this work to general game playing in the near future.

An important step forward is to bring this approach together with our recent work on planning over multi-agent epistemic states (Muise et al. 2015b). In that work, state is represented using a belief base with syntactic restrictions (Muise et al. 2015a), in which beliefs can be about the world or about other agents' belief, including their belief about us, etc.; so called *nested belief*. The formalism supports ontic actions: actions that modify the state of the world; and *deontic* actions: actions that modify the knowledge or belief of other agents. We encode these multi-agent epistemic planning problems as classical planning problems. However, the modelled actions can only be performed by the single planning agent. Bringing the multi-agent planning as FOND work from this paper together with multi-agent epistemic planning will enable us to solve a rich set of problems in which the planner considers both the actions others can take, the beliefs they have, and how these two interact.

Acknowledgements This research is partially funded by Australian Research Council Discovery Grant DP130102825, *Foundations of Human-Agent Collaboration: Situation-Relevant Information Sharing*

References

Alford, R.; Kuter, U.; Nau, D.; and Goldman, R. P. 2014. Plan aggregation for strong cyclic planning in nondeterministic domains. *Artificial Intelligence* 216:206–232.

Alur, R.; Henzinger, T. A.; and Kupferman, O. 2002. Alternating-time temporal logic. *J. ACM* 49(5):672–713.

Bolander, T., and Herzig, A. 2014. Group attitudes and multi-agent planning: overview and perspectives. Technical report.

Bowling, M. H.; Jensen, R. M.; and Veloso, M. M. 2003. A formalization of equilibria for multiagent planning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 1460–1462.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, 28–35.

Brenner, M. 2003. A multiagent planning language. In *Proceedings of the Workshop on PDDL, ICAPS*, volume 3, 33–38.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1):35–84.

Fu, J.; Ng, V.; Bastani, F. B.; and Yen, I.-L. 2011. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *Proceedings of the 22nd International Joint Conference On Artificial Intelligence (IJCAI)*, 1949–1954.

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

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

Jensen, R. M., and Veloso, M. M. 2000. OBDD-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research (JAIR)* 13:189–226.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 119–127. AAAI Press.

Kovacs, D. L. 2012. A multi-agent extension of pddl3.1. 19.

Muise, C.; Miller, T.; Felli, P.; Pearce, A.; and Sonenberg, L. 2015a. Efficient reasoning with consistent proper epistemic knowledge bases. In Bordini; Elkind; Weiss; and Yolum., eds., *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Muise, C.; Belle, V.; Felli, P.; McIlraith, S.; Miller, T.; Pearce, A.; and Sonenberg, L. 2015b. Planning over multi-agent epistemic states: A classical planning approach. In *The 29th AAAI Conference on Artificial Intelligence*.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-deterministic Planning by Exploiting State Relevance. In *The 22nd International Conference on Automated Planning and Scheduling*, The 22nd International Conference on Automated Planning and Scheduling.

Muise, C.; McIlraith, S. A.; and Belle, V. 2014. Non-deterministic planning with conditional effects. In *The 24th International Conference on Automated Planning and Scheduling*.

Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2006. Synthesis of reactive(1) designs. In *VMCAI*, 364–380.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

Ramirez, M., and Sardina, S. 2014. Directed fixed-point regression-based planning for non-deterministic domains. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.