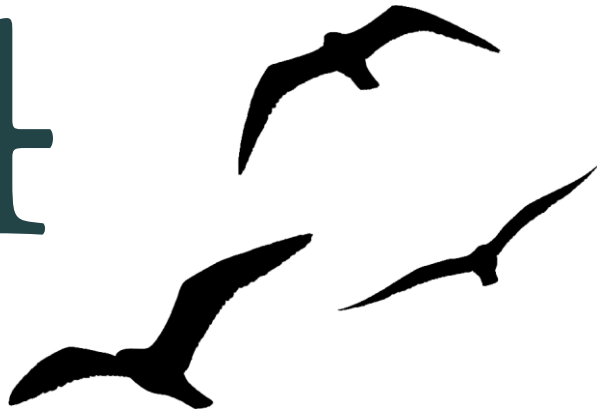
ICAPS 2014



Proceedings of the 1st Workshop on

Models and Paradigms for Planning under Uncertainty: a Broad Perspective

Edited By:
Andrey Kolobov, Ugur Kuter and Florent Teichteil-Königsbuch.

Portsmouth, New Hampshire, USA - June 22, 2014

Organizing Committee

Andrey Kolobov
MSR Redmond, USA
Ugur Kuter
SIFT, USA
Florent Teichteil-Königsbuch
ONERA, France

Program Committee

Alexandre Albore, ONERA, France
Daniel Bryce, SIFT, USA
Juergen Dix, Clausthal University of Technology, UK
Malik Ghallab, LAAS, France
Andrey Kolobov, Microsoft Research, USA
Ugur Kuter, SIFT, USA
Steven Schockaert, Cardiff University, UK
Guy Shani, Ben-Gurion University of the Negev, Israel
Florent Teichteil-Königsbuch, ONERA, France
Paolo Traverso, IRST, Italy

Foreword

Great strides have been made in automated AI planning under uncertainty in recent years, including symbolic and compact representations of planning problems and very efficient techniques for solving them. The effectiveness of these methods has been demonstrated in the past International Planning Competitions, and to some extent, in real-world applications such as navigation tasks, space operations, railway control, and rescue/evacuation tasks.

However, there are several remaining challenges for developing uncertainty models for the planning systems. When immersed in the real world, these systems often face a constantly changing environment, whose evolution is not deterministic. In addition to the environmental dynamics, planning systems also must deal with the partial knowledge about their surroundings, their models of the environment, and their goals in that environment.

So far, the ICAPS community has mainly focused on probabilities as the means of characterizing these types of uncertainty. At the same time, there are many other frameworks for describing uncertainty that have been studied in Artificial Intelligence in general (e.g., possibilities, fuzzy logics, imprecise probabilities, Dempster-Shafer's belief function), and other communities confronted with real-world problems like risk management use these models in their decision-making systems. We believe a cross-pollination of research approaches and methodologies from these relevant, yet mostly distant, paradigms will be very beneficial for the ICAPS community, leading to new ideas for tackling planning problems.

We are very pleased to welcome Dr. Ronen I. Brafman as our invited speaker. Dr. Brafman's talk will focus on *Planning under uncertainty: Reductions, Re-planning, Simplifications*. In AI planning, the focus has been on stochastic models (e.g., MDPs, POMDPs), and non-probabilistic but non-deterministic models, as in contingent and conformant planning. An important distinction has to do with the nature of the goals – satisfying some goal conditions in methods derived from classical planning vs. accumulating reward in POMDPs. Reduction techniques between different kinds of goals and planning problems have been proposed, and in his talk, Dr. Brafman will review why these techniques are useful and what they are good for.

Table of Contents

Factored Markov Decision Process with Imprecise Transition Probabilities

Karina V. Delgado
EACH-University of Sao Paulo
Sao Paulo - Brazil

Leliane N. de Barros
IME-University of Sao Paulo
Sao Paulo - Brazil

Scott Sanner
NICTA-ANU
Camberra - Australia

Fabio Cozman
POLI-University of Sao Paulo
Sao Paulo - Brazil

Abstract

This paper presents a short survey of the research we have carried out on planning under uncertainty where we consider different forms of imprecision on the probability transition functions. Our main results are on efficient solutions for Markov Decision Process with Imprecise Transition Probabilities (MDP-IPs), a generalization of a Markov Decision Process where the imprecise probabilities are given in terms of credal sets. Noting that the key computational bottleneck in the solution of MDP-IPs is the need to repeatedly solve an optimization problem, our goal is to minimize the number of calls to the optimizer. Our results show how to target approximation techniques to drastically reduce the computational overhead of the optimization solver while producing bounded, approximately optimal solutions.

Introduction

Markov Decision Processes (MDP) (Puterman 1994) have become the *de facto* standard model for decision-theoretic planning problems and a great deal of research in recent years has aimed at proposing efficient solutions to tackle large and more realistic problems. Formally, an MDP is defined by the tuple $\mathcal{M} = \langle S, A, P, R, T, \gamma \rangle$, where S is a finite set of fully observable states; A is a finite set of actions; $P(s'|s,a)$ is the conditional probability of reaching state $s' \in S$ when action $a \in A$ is taken from state $s \in S$; $R : S \times A \rightarrow \mathbb{R}$ is a fixed reward function associated with every state and action; T is the time horizon (number of stages-to-go) for decision-making; and $\gamma = [0, 1)$ is a *discount factor*. An important research topic in this area is how to exploit structure in order to compactly represent and efficiently solve factored MDPs (Boutilier, Hanks, and Dean 1999; Hoey et al. 1999; St-Aubin, Hoey, and Boutilier 2000; Guestrin et al. 2003). In many MDPs, it is often natural to think of the state as an assignment to multiple state variables and a transition function that compactly specifies the probabilistic dependence of variables in the next state on a subset of variables in the current state. Such an approach naturally leads us to define a *Factored MDP* (Boutilier, Hanks, and Dean 1999), where $S = \{\vec{x}\}$. Here, $\vec{x} = (x_1, \ldots, x_n)$ where each state variable $x_i \in \{0, 1\}$.

The reward can simply be specified as $R(\vec{x}, a)$. The transition probabilities in a factored MDP are encoded using *Dynamic Bayesian Networks (DBNs)* (Dean and Kanazawa 1990). A DBN is a *directed acyclic graph (DAG)* with two layers: one layer represents the variables in the current state and the other layer represents the next state (Figure 3a). Nodes x_i and x'_i refer to the respective current and next state variables. The connection between these two layers defines the dependences between state variables w.r.t. the execution of an action $a \in A$. Directed edges are allowed *from* nodes in the first layer *into* the second layer, and also between nodes in the second layer (these latter edges are termed *synchronic arcs*). We denote by $pa_a(x'_i)$ the parents of x'_i in the graph for action a and $P(x'_i|pa_a(x'_i), a)$, the conditional probability table (CPT). The graph encodes the standard Bayes net conditional independence assumption that a variable x'_i is conditionally independent of its nondescendants given its parents, which incidentally for a DBN also encodes the Markov assumption (the current state is independent of the history given the previous state). The use of a DBN leads to the following factorization of transition probabilities:

$$P(\vec{x}'|\vec{x}, a) = \prod_{i=1}^{n} P(x'_i|pa_a(x'_i), a). \quad (1)$$

However, in many real-world problems, it is simply impossible to obtain a precise representation of the transition probabilities in an MDP. This may occur for many reasons, including (a) imprecise or conflicting elicitations from experts, (b) insufficient data from which to estimate reliable precise transition models, or (c) non-stationary transition probabilities due to insufficient state information.

For example, in an automated navigation system, it can be difficult to estimate the probability of reaching some location after a move. The probabilities may change throughout time due the environment conditions (such weather and road conditions) and can make the navigation more difficult and subject to failures. In general, it is hard to accurately model all these changes since they can have many external dependencies. In this case, it is better to have a policy optimized over a range of possible probabilities in order to be robust against transition uncertainty.

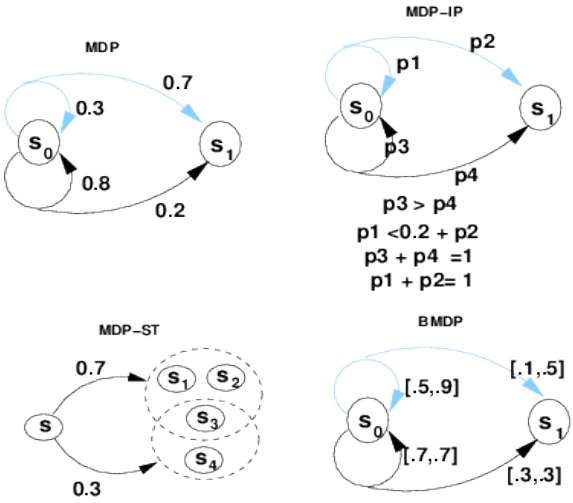


Figure 1: Examples of MDP, MDP-IP, MDP-ST and BMDP.

To accommodate optimal models of sequential decision-making in the presence of strict uncertainty over the transition model, the MDP with imprecise transition probabilities (MDP-IP) was introduced (Satia and Lave Jr. 1970; White III and El-Deib 1994) where the imprecise probabilities are represented by probabilistic parameters p_i and a set of constraints over them which define a credal set, that is explained in Definition 1. While the MDP-IP poses a robust framework for the real-world application of decision-theoretic planning, its general solution requires the use of computationally expensive optimization routines that are extremely time-consuming in practice.

A particular sub-classes of MDP-IP is the *Bounded-parameter Markov Decision Process (BMDP)* (Givan, Leach, and Dean 2000), where the probabilities are specified by intervals. Givan proposes the Interval Value Iteration algorithm (Givan, Leach, and Dean 2000) that can find an optimal policy without requiring expensive optimization techniques. Recent solutions to BMDPs include extensions of real-time dynamic programming (RTDP) (Buffet and Aberdeen 2005) and LAO* (Cui et al. 2006; Yin, Wang, and Gu 2007) that search for the best policy under the worst model. However, a problem with general linear constraints over the probability parameters, for example $\{0.3 \leq p_j \leq 0.5, p_j \geq p_i\}$ cannot be solved by these solutions. Another sub-class of MDP-IPs is the *Markov Decision Process with Set-valued Transitions (MDP-ST)* (Trevizan, Cozman, and de Barros 2007), where probability distributions are given over finite sets of states. Examples of MDP-IP, BMDP and MDP-ST are given in Figure 1.

In Figure 2 we show the relationship among different types of planning under uncertainty which includes as special cases the deterministic and nondeterministic planning problems. Notice that BMDP and MDP-ST do not have the same representational power, i.e., some MDP-ST problems can not be reduced to BMDP, and vice versa. Note also that since BMDPs and MDP-STs are special cases of MDP-IPs, we can represent them as MDP-IPs, thus the algorithms for

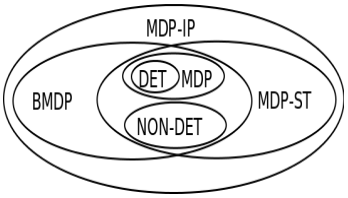


Figure 2: Relationship among MDP-IP and its sub-classes.

MDP-IPs clearly apply to both BMDPs and MDP-STs.

To address the computational deficiency of solutions for MDP-IPs, first we extended the factored MDP model by replacing the usual Dynamic Bayesian Network (DBN) (Dean and Kanazawa 1990) used in factored MDPs with Dynamic Credal Network (DCNs) (Delgado et al. 2009; Delgado, Sanner, and de Barros 2011) to support compact factored structure in the imprecise transition model of factored MDP-IPs. Second, we have proposed efficient, scalable algorithms for solving these factored MDP-IPs based on two different approaches: dynamic programming (Delgado, Sanner, and de Barros 2011) and multilinear programming (Delgado et al. 2011).

MDPs with Imprecise Transitions

An *MDP with imprecise transition probabilities (MDP-IP)* is simply an extension of the MDP where the transition probabilities can be imprecisely specified. That is, instead of a probability measure $P(\cdot|s,a)$ over the state space S, we have a *set* of probability measures. that is referred to as a *credal set* (Cozman 2000).

Definition 1. Transition credal set. *A credal set containing conditional distributions over the next state s', given a state s and an action a, is referred to as a* transition credal sets *(Cozman 2000) and denoted by $K(s'|s,a)$. Thus, we have $P(\cdot|s,a) \in K(\cdot|s,a)$ to define imprecisely specified transition probabilities.*

We assume that all credal sets are closed and convex, an assumption that is often used in the literature, and that encompasses most practical applications (Walley 1991). We further assume stationarity for the transition credal sets $K(s'|s,a)$; that is, they do not depend on the stage t. While $K(s'|s,a)$ is non-stationary, we note that this does not require $P(s'|s,a)$ to be stationary in an MDP-IP: distributions $P(s'|s,a)$ may be selected from the corresponding credal sets in a time-dependent manner (Nilim and El Ghaoui 2005).

Formally, an MDP-IP is defined by $\mathcal{M}_{IP} = (S, A, K, R, T, \gamma)$. This definition is identical to the MDP $\mathcal{M}$, except that the transition distribution P is replaced with a transition credal set K. We will represent K implicitly as the set of transition probabilities consistent with a set of side linear inequality constraints C over the probability parameters.

There are several optimization criteria that can be used to define the value of a policy in an MDP-IP. In the context of the discounted infinite horizon setting focused on this work, there is always a deterministic stationary policy

that is *maximin* optimal (Satia and Lave Jr. 1970) (i.e., no other policy could achieve greater value under the assumption that Nature's selects $P(s'|s,a)$ adversarially to minimize value); moreover, given the assumption that A is finite and the credal set K is closed, this policy induces an optimal value function that is the unique fixed-point solution of

$$V^*(s) = \max_{a \in A} \min_{P \in K} \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right\}. \quad (2)$$

There are various algorithms for solving enumerated state MDP-IPs based on dynamic programming (Satia and Lave Jr. 1970; White III and El-Deib 1994). In this work, we build on a value iteration solution to MDP-IPs (Satia and Lave Jr. 1970):

$$V^t(s) = \max_{a \in A} \min_{P \in K} \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{t-1}(s') \right\} \quad (3)$$

Value iteration for MDP-IPs is the same as for MDPs except that now for *every* state s, we optimize our action choice $a \in A$ w.r.t. the *worst-case* distribution $P \in K$ that minimizes the future expected value. Thus we ensure that the resulting value function and policy are robust to the worst outcome that Nature could choose in light of the future value $V^{t-1}(s')$ that we expect to achieve.

The Bellman equation can be also solved through a multilinear program (Shirota et al. 2007):

$$\begin{aligned} \min_{V^*, P} \quad & : \quad \sum_s V^*(s) \quad (4) \\ \text{s.t.} \quad & : \quad V^*(s) \geq R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s'), \\ & \qquad \forall s \in S, a \in A, P(s'|s,a) \in K(s'|s,a). \end{aligned}$$

Notice that the constraints force $V^*(s)$ to be greater than or equal to $\max_{a \in A} \min_{P \in K} \{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s')\}$, considering all $a \in A$, and then minimizing $\sum_s V^*(s)$ enforces that the minimal $V^*(s)$ is obtained.

Factored MDP-IPs

The definitions of MDP-IP given in the previous section models an enumerated MDP-IP (also called flat MDP-IP), where states are seen as black boxes.

We have extended the factored MDP representation (Boutilier, Hanks, and Dean 1999) to compactly represent MDP-IPs. This requires modifying the DBN transition representation to account for uncertainty over the exact transition probabilities.

Like the previous definition of an enumerated state MDP-IP, the set of all legal transition distributions for a factored MDP-IP is defined as a *credal set* K. The challenge then was to specify such transition credal sets in a factored manner that is itself compact. For this, we have proposed to use *dynamic credal networks (DCNs)*, a special case of credal networks (Cozman 2000; 2005), as an appropriate language to express factored transition credal sets.

Definition 2. Factored transition credal set. *A credal set containing conditional distributions over the values of a variable x_i, given the values of $pa_a(x_i)$ (the parents of x_i in the graph for action a), is referred to as a* factored transition credal set *and denoted by $K_a(x_i|pa_a(x_i))$.*

Definition 3. Dynamic credal network. *A Dynamic credal network (DCN) is a generalization of a DBN. Different from the definition of a DBN, in a DCN each variable x_i is associated with factored transition credal sets $K_a(x_i|pa_a(x_i))$ for each value of $pa_a(x_i)$. We assume that a DCN represents a* joint credal set *(Cozman 2005; 2000) over all of its variables consisting of all distributions that satisfy the factorization in Equation (1), where each CPT distribution $P(x'_i|pa_a(x'_i), a)$ is an element of the transition credal set $K_a(x'_i|pa_a(x'_i))$ associated with the DCN, i.e. $P(x'_i|pa_a(x'_i), a) \in K_a(x'_i|pa_a(x'_i))$.*

A DCN example is shown in Figure 3a. For each variable x'_i in a DCN, we have a *conditional probability table (CPT)* with imprecise probabilities. If we examine the CPTs in Figure 3b, we note that entries are specified by probability parameters p_{ij} (i for variable x'_i and j for the jth parameter in the CPT for x'_i). Furthermore, we note that we have a set of side linear constraints on these p_{ij} (shown in the boxes below the CPT, collectively call this constraint set C). We use $\vec{p}$ to denote a vector containing all parameter values that are free to vary within the given credal sets (i.e., that satisfy the probability constraints C of the DCN).

We note that the joint transition probability may be nonlinear in the probability parameters $\vec{p}$. However, we explicitly introduce the following restriction to prevent exponents exceeding 1: a parameter p_{ij} may only appear in the CPT for x'_i. This restriction prevents the multiplication of p_{ij} by itself when CPTs for each x'_i are multiplied together to determine the joint transition distribution in the DCN. This subset of nonlinear expressions, where the exponent of each p_{ij} is either 0 or 1, is referred to as a *multilinear* expression. To see the multilinearity of the transition probability in Figure 3, we observe $P(x'_1 = 1, x'_2 = 1 | x_1 = 1, x_2 = 1, notreboot) = p_{11} p_{21}$.

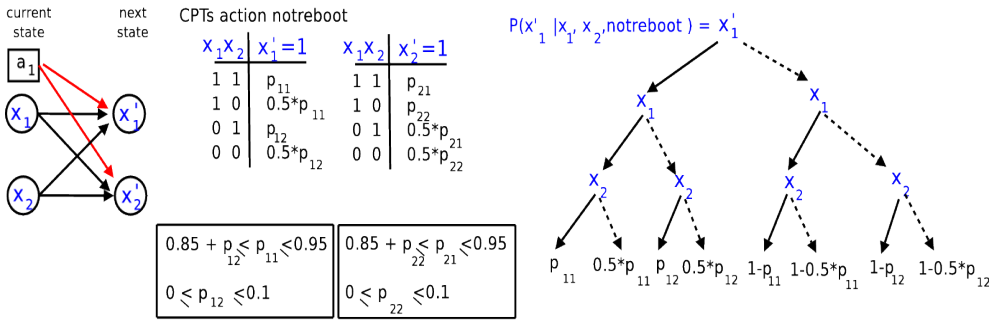
Exact Solution for Factored MDP-IPs

SPUDD-IP

SPUDD (Hoey et al. 1999) is an efficient factored version of Value Iteration for MDPs that represents CPTs, rewards and value functions as *algebraic decision diagrams (ADDs)* (Bahar et al. 1993). Inspired on that we have proposed *SPUDD-IP* (Delgado, Sanner, and de Barros 2011) to solve factored MDP-IPs.

ADDs compactly represent *context-specific independence* (CSI) (Boutilier et al. 1996) that are not evident in the DBNs. In order to compactly represent a CSI and shared function structure in the CPTs for an MDP-IP, we have proposed a novel extension of ADDs called *parameterized ADDs (PADDs)* (Delgado, Sanner, and de Barros 2011) since the leaves are parameterized expressions as shown in Figure 3c. PADDs do not only allow us to compactly represent the CPTs for factored MDP-IPs, but they also enable efficient computations for factored MDP-IP value iteration operations.

We begin by expressing MDP-IP value iteration from (3)



a) b) c)

Figure 3: a) DCN for action a. b) Conditional probability table for the state variables $x'_1 = 1$ and $x'_2 = 1$ and the constraints related to the probabilities. c) The Parameterized ADD representation of $P(x'_1|x_1, x_2, a)$. Solid lines indicate the true (1) branch of a variable test and dashed lines indicate the false (0) branch.

in the following factored form using the transition representation of (1) and operations on decision diagrams:[1]

$$V^t_{DD}(\vec{x}) = \max_{a \in A} \left\{ R_{DD}(\vec{x}, a) \oplus \gamma \min_{\vec{p}} \boxed{\sum_{\vec{x}'} \bigotimes_{i=1}^{n} P_{DD}(x'_i | pa_a(x'_i), a) V^{t-1}_{DD}(\vec{x}')} \right\} \quad (5)$$

Because the transition CPTs in the MDP-IP DCN contain parameters $\vec{p}$, these CPTs must be represented in decision diagram format as PADDs ($P_{DD}(x'_i|pa_a(x'_i), a) \in K_a(x'_i|pa_a(x'_i))$). The reward $R_{DD}(\vec{x}, a)$ can be represented as an ADD since it contains only constants (for the purpose of operations, recall that ADDs are special cases of PADDs). Although it may appear that the form of $V^t_{DD}(\vec{x})$ is a PADD, we note that the parameters $\vec{p}$ are "minimized"-out w.r.t. the side constraints on $\vec{p}$ during the $\min_{\vec{p}} \square$ operation in (5) ($\min_{\vec{p}} \square$ is the *MinParameterOut* operation on PADDs, that performs the minimization over the parameters by calling a nonlinear solver for each leaf and returns an ADD). Thus the resulting $V^t_{DD}(\vec{x})$ computed from the $\max_{a \in A}$ has constant leaves and can be expressed as the ADD special case of PADDs.

To explain the efficient evaluation of (5) in more detail, we can exploit the variable elimination algorithm (Zhang and Poole 1994) in the marginalization over all next states $\sum_{\vec{x}'}$. For example, if x'_1 is not dependent on any other x'_i for $i \neq 1$, we can "push" the sum over x'_1 inwards to obtain:

$$V^t_{DD}(\vec{x}) = \max_{a \in A} \left\{ R_{DD}(\vec{x}, a) \oplus \gamma \min_{\vec{p}} \boxed{\sum_{x'_i (i \neq 1)} \bigotimes_{i=1 (i \neq 1)}^{n} P_{DD}(x'_i | pa_a(x'_i), a) \sum_{x'_1} P_{DD}(x'_1 | pa_a(x'_1), a) V^{t-1}_{DD}(\vec{x}')} \right\} \quad (6)$$

Then we can continue with x'_2, multiplying this result by the $P_{DD}(x'_2|pa_a(x'_2), a)$, summing out over x'_2, and repeating for all x'_i to compute $\square$. After this $\square$ does not contain anymore the variables x'_i, but only the variables x_i.

The SPUDD-IP value iteration solution to factored MDP-IPs returns a robust optimal policy and often yields an improvement over flat value iteration. Figure 4 shows that for large state spaces (2^{12} to 2^{22}) flat Value Iteration does not return a solution due to time and space limitations, while SPUDD-IP can solve the largest problems.

[1] We use DD for the functions represented by ADDs or PADDs.

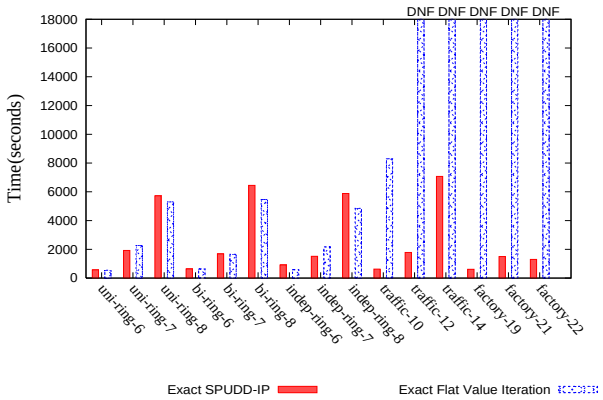


Figure 4: Time performance comparison for TRAFFIC, SYSADMIN and FACTORY problems using SPUDD-IP and Flat Value Iteration. The name includes the number of variables in each problem, so the corresponding number of states is $2^{\#variables}$.

Approximate Solutions for Factored MDP-IPs

As the number of state variables in a problem grows larger SPUDD-IP also becomes inefficient. Thus, in this section we describe three approximate solutions for MDP-IP: APRICODD-IP (Delgado, Sanner, and de Barros 2011), Objective-IP (Delgado, Sanner, and de Barros 2011), two approximate versions of Value Iteration for MDP-IP; and AMP (Delgado et al. 2011) an approximate Multilinear Programming algorithm for MDP-IP.

Approximate Value Iteration

Approximate value iteration (AVI) is one way to trade off time and space with error by approximating the value function after each iteration.

APRICODD-IP algorithm is an extension of the APRICODD algorithm (St-Aubin, Hoey, and Boutilier 2000) that

provides an efficient way of *approximating the ADD value representation* for a factored MDP, reducing its size and thus reducing computation time per iteration. This is done by a method that has two inputs: (1) a value function represented as an ADD and (2) an approximation error to merge the leaves. The output is a new ADD with the merged leaves. The algorithm first collects all leaves of the ADD and determines which can be merged to form new values without approximating more than *error*. The old values are then replaced with these new values creating a new (minimally reduced) ADD that represents the approximated value function. This approach immediately generalizes to MDP-IPs since the value function V^t_{DD} is also an ADD.

However, in solving (factored) MDP-IPs, the time is dictated less by the size of the value function ADD and more by the number of calls to the multilinear optimizer. Thus we also proposed a new approximate algorithm, Objective-IP, that attempts to make less calls to the solver by *approximating the objectives* (the min of the optimization call) in an attempt to avoid calling the solver altogether. Different from APRICODD-IP, Objective-IP approximates the leaves of the PADD just prior to carrying out the multilinear optimization. The approximation method takes as input a PADD and the maximum *error* and returns a new PADD with approximated leaves using the upper and lower bounds of the parameters p_i. Note that each leave is approximated independently, this can be done since each leaf corresponds to a different state (or set of states) and the system can only be in one state at a time. Furthermore, we can guarantee that no objective pruning at the leaves of the PADD incurs more than *error* after the multilinear optimization is performed.

In order to evaluate the policy returned by our approximate solutions, we compute the True Approximation Error (TAE) given by:

$$max_{\vec{x}}|V^*(\vec{x}) - V_{approx}(\vec{x})| \tag{7}$$

where $V_{approx}(\vec{x})$ is the value returned by the approximate solutions and $V^*(\vec{x})$ is the optimal value computed by SPUDD-IP.

In Figure 5 we show a comparison of the True Approximation Error (TAE) vs. running times for the bi-ring-6 problem of an imprecise version of the SysAdmin domain (Guestrin et al. 2003). The results echo one conclusion: Objective-IP consistently takes less time than APRICODD-IP to achieve the same approximation error and *up to one order of magnitude less time* than APRICODD-IP (Delgado, Sanner, and de Barros 2011). This time reduction can be explained by the decreased number of calls to the multilinear solver.

Approximate Multilinear Programming

Another approximate solution for Factored MDP-IPs we have proposed is the Approximate Multilinear Programming (AMP) (Delgado et al. 2011). We extend previous work (Guestrin et al. 2003) that obtain efficient approximate linear programming solutions for Factored MDPs. In this approximate solution we use the approximate value function denoted by $\widehat{V}(\vec{x})$. Given $\vec{x} \in \mathcal{S}$ and a set of basis functions $H = \{h_1, ..., h_k\}$, $V^*(\vec{x})$ is approximated using a linear combination:

$$\widehat{V}(\vec{x}) = \sum_{j=1}^{k} w_j h_j(\vec{x}). \tag{8}$$

We can use this approximate value function and replace it in the multilinear formulation (Problem (4)) of an MDP-IP so as to obtain the factored multilinear programming problem:

$$\min_{w,P} \quad \sum_{\vec{x}} \sum_{i=0}^{k} w_i h_i(\vec{x}) \tag{9}$$

$$\text{subject to} \quad : \quad \sum_{i=0}^{k} w_i h_i(\vec{x}) \geq R(\vec{x}, a) +$$

$$\gamma \sum_{\vec{x}' \in \mathcal{S}} P(\vec{x}'|\vec{x}, a) \sum_{i=0}^{k} w_i h_i(\vec{x}'), \forall \vec{x} \in S, a \in A$$

$$P(x'_i|pa(X'_i), a) \in K_a(X'_i|pa(X'_i)),$$

$$P(\vec{x}'|\vec{x}, a) = \prod_i P(x'_i|pa(X'_i), a).$$

To solve this optimization problem, we exploit the Factored MDP-IP structure to reduce the number of constraints generated and to compactly encode the remaining constraints that empirically leads to an exponential reduction in the number of constraints for some problems.

In Figure 5 we compare the three approximate solution methods, APRICODD-IP and Objective-IP and AMP. For the AMP solution, we used *simple* basis functions (one for each variable in the problem description) and *pairwise* basis functions (one for each pair of variables that have a common child variable in the DCN). When it does finish within a limit of ten hours, AMP takes only a few seconds to produce an approximate solution for each problem (except for the FACTORY domain for which it did not return a solution). Comparing the algorithms in terms of their true approximation error, we observe that in the bi-ring-6 problem, AMP with *pair* basis functions outperforms APRICODD-IP and obtains a solution 2-3$\times$ larger than the error of Objective-IP, but in significantly less time. This experiment and the results obtained in other domains lead us to conclude that Objective-IP consistently gives an error at least 2-3$\times$ lower than AMP and sometimes runs as fast as the AMP solution, while in other cases running slower. However, the bottleneck of the AMP solution is to define appropriate basis functions (Guestrin et al. 2003).

Concluding Remarks

In this work we made a short survey on our contribution for Markov Decision Processes with Imprecise Probabilities (MDP-IPs), a class of models that adds considerable flexibility and realism to probabilistic planning allowing the representation of imprecise transition probabilities. We first propose a compact Factored MDPIP model, which represents states throughout state variables and uses Dynamic Credal Networks to specify the imprecise transition probabilities, which can reveal the structure of an application domain and

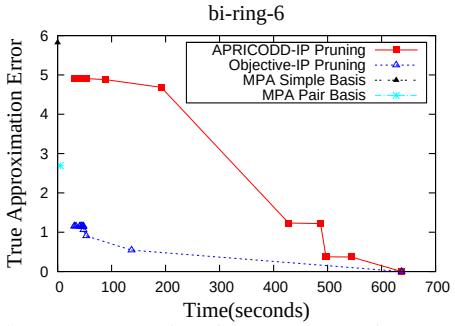


Figure 5: True Approximation Error vs. time required for APRICODD-IP, Objective-IP and AMP with simple basis and pairwise basis functions for a SYSADMIN problem with bidirectional-ring topology.

allows for the construction of efficient solutions. The proposed solution, exact or approximate were of two types: based on dynamic programming and based on multilinear programming. The first extends the factored approaches for MDPs, SPUDD and APRICODD, plus a new way of approximation that avoids calling the optimization solver. The second extends the approximate linear programming solution for MDPs, to the Approximate Multilinear Programming for MDP-IPs, that can be very efficient when we have appropriate basis functions.

Another work we are currently developing is an asynchronous dynamic programming algorithm, named RTDP-IP for MDP-IPs. The main challenges of this solution are: (i) how to sample the next state in a trial? (ii) how to ensure convergence of an asynchronous dynamic programming solution, having a range of possible probabilities in the state transition matrix? Additionally, we are currently working on how to extract an MDP-IP transition function from data.

References

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1993. Algebraic Decision Diagrams and their Applications. In *Proceedings of ICCAD*, 188–191. Los Alamitos, CA, USA: IEEE Computer Society Press.

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific Independence in Bayesian Networks. In *Proc. 12th UAI*, 115–123.

Boutilier, C.; Hanks, S.; and Dean, T. 1999. Decision-theoretic Planning: Structural Assumptions and Computational Leverage. *JAIR* 11:1–94.

Buffet, O., and Aberdeen, D. 2005. Robust Planning with LRTDP. In *Proc. of the IJCAI*, 1214–1219.

Cozman, F. G. 2000. Credal Networks. *Artificial Intelligence* 120:199–233.

Cozman, F. G. 2005. Graphical Models for Imprecise Probabilities. *International Journal of Approximate Reasoning* 39(2-3):167–184.

Cui, S.; Sun, J.; Yin, M.; and Lu, S. 2006. Solving Uncertain Markov Decision Problems: An Interval-Based Method. In *ICNC (2)*, 948–957.

Dean, T., and Kanazawa, K. 1990. A Model for Reasoning about Persistence and Causation. *Comput. Intell.* 5(3):142–150.

Delgado, K. V.; de Barros, L. N.; Cozman, F. G.; and Shirota, R. 2009. Representing and Solving Factored Markov Decision Processes with Imprecise Probabilities. In *6th ISIPTA*.

Delgado, K. V.; de Barros, L. N.; Cozman, F. G.; and Sanner, S. 2011. Using Mathematical Programming to Solve Factored Markov Decision Processes with Imprecise Probabilities. *Int. J. Approx. Reasoning* 52(7):1000–1017.

Delgado, K. V.; Sanner, S.; and de Barros, L. N. 2011. Efficient Solutions to Factored MDPs with Imprecise Transition Probabilities. *Artif. Intell.* 175(9-10):1498–1527.

Givan, R.; Leach, S.; and Dean, T. 2000. Bounded-parameter Markov Decision Processes. *Artificial Intelligence* 122:71–109(39).

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Efficient Solution Algorithms for Factored MDPs. *JAIR* 19:399–468.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic Planning using Decision Diagrams. In *Fifteenth Conference on Uncertainty in Artificial Intelligence*, 279–288. Morgan Kaufmann.

Nilim, A., and El Ghaoui, L. 2005. Robust Control of Markov Decision Processes with Uncertain Transition Matrices. *Oper. Res.* 53(5):780–798.

Puterman, M. L. 1994. *Markov Decision Processes*. Wiley Series in Probability and Mathematical Statistics. New York: John Wiley and Sons.

Satia, J. K., and Lave Jr., R. E. 1970. Markovian Decision Processes with Uncertain Transition Probabilities. *Operations Research* 21:728–740.

Shirota, R.; Cozman, F. G.; Trevizan, F. W.; de Campos, C. P.; and de Barros, L. N. 2007. Multilinear and Integer Programming for Markov Decision Processes with Imprecise Probabilities. In *5th ISIPTA*, 395–404.

St-Aubin, R.; Hoey, J.; and Boutilier, C. 2000. APRICODD: Approximate Policy Construction using Decision Diagrams. In *Proceedings NIPS*, 1089–1095. MIT Press.

Trevizan, F. W.; Cozman, F. G.; and de Barros, L. N. 2007. Planning under Risk and Knightian Uncertainty. In *IJCAI*, 2023–2028.

Walley, P. 1991. *Statistical Reasoning with Imprecise Probabilities*. London: Chapman and Hall.

White III, C. C., and El-Deib, H. K. 1994. Markov Decision Processes with Imprecise Transition Probabilities. *Operations Research* 42(4):739–749.

Yin, M.; Wang, J.; and Gu, W. 2007. Solving Planning Under Uncertainty: Quantitative and Qualitative Approach. In *IFSA (2)*, 612–620.

Zhang, N. L., and Poole, D. 1994. A Simple Approach to Bayesian Network Computations. In *Proc. of the Tenth Canadian Conference on Artificial Intelligence*, 171–178.

Monte-Carlo Tree Search: To MC or to DP?

Zohar Feldman and **Carmel Domshlak**
Technion—Israel Institute of Technology
Haifa, Israel
{zoharf@tx,dcarmel@ie}.technion.ac.il

Abstract

State-of-the-art Monte-Carlo tree search algorithms can be parametrized with any of the two information updating procedures: MC-backup and DP-backup. The dynamics of these two procedures is very different, and so far, their relative pros and cons have been poorly understood. Formally analyzing the dependency of MC- and DP-backups on various MDP parameters, we reveal numerous important issues that get hidden by the worst-case bounds on the algorithm performance, and reconfirm these findings by a systematic experimental test.

INTRODUCTION

Markov decision processes (MDPs) is a standard model for planning under uncertainty (Puterman 1994). An MDP $\langle S, A, \mathbb{P}, R\rangle$ is defined by a set of states S, a set of state transforming actions A, a stochastic transition function $\mathbb{P} : S \times A \times S \to [0, 1]$, and a reward function $R : S \times A \times S \to \mathbb{R}$. The states are fully observable and, in the finite horizon setting considered here, the rewards are accumulated over some predefined number of steps H. The objective of planning in MDPs is to sequentially choose actions so as to maximize the accumulated reward. The representation of large-scale MDPs can be either declarative or generative, but anyway concise, and allowing for simulated execution of all feasible action sequences, from any state of the MDP. In *online MDP planning*, the agent focuses on its current state s_0 only, deliberates about the set of possible policies from that state onwards and, when interrupted, chooses what action to perform next. In formal analysis of algorithms for online MDP planning, the quality of the action a, chosen for s_0 with H steps-to-go, is assessed in terms of the induced "simple regret", capturing the performance loss that results from taking a and then following an optimal policy π^* for the remaining $H - 1$ steps, instead of following π^* from the beginning (Bubeck and Munos 2010).

Many popular algorithms for online MDP planning constitute what is called *Monte-Carlo tree search (MCTS)* (Sutton and Barto 1998; Péret and Garcia 2004; Kocsis and Szepesvári 2006; Coquelin and Munos 2007; Cazenave 2009; Rosin 2011; Tolpin and Shimony 2012; Keller and Helmert 2013; Feldman and Domshlak 2013; 2014b), and adaptations of some of these algorithms are also popular in other settings of sequential decision making, including those with partial state observability and adversarial effects (Gelly and Silver 2011; Sturtevant 2008; Bjarnason, Fern, and Tadepalli 2009; Eyerich, Keller, and Helmert 2010; Browne et al. 2012). At a high level, all MCTS algorithms explore the state-space region around s_0 by iteratively (i) simulating an action/state trajectory from s_0, and (ii) using the outcome of that trajectory to update various action-value estimates related to the state-space region of interest, as well as to update the estimate of what action should be best applied at state s_0. In that respect, specific MCTS algorithms differ both in their trajectory rollout strategies, as well as in their rollout-based update strategies.

Recent work substantially advanced our understanding of how the performance of MCTS depends on the specifics of the rollout strategy, as well as on the choice of what pieces of information should be updated based on a given rollout (Kocsis and Szepesvári 2006; Coquelin and Munos 2007; Feldman and Domshlak 2012). Recently, however, Keller & Helmert (Keller and Helmert 2013) demonstrated empirically that the performance of MCTS also depends to a large extent on *how* the respective updates are being performed. Prior to the work of Keller & Helmert, updates in MCTS algorithms were all based on *MC-backups*, that is, sample averaging updates of the selected action-value estimates. Keller & Helmert showed that modifying a standard MCTS algorithm (such as UCT (Kocsis and Szepesvári 2006)), by replacing its MC-backups with dynamic programming estimates propagation *a la* Bellman backups in value iteration (Bellman 1957), can substantially affect the performance, and, at least in their experiments, *typically in favor of DP-backups.* Later on, Feldman & Domshlak (Feldman and Domshlak 2014b) showed that switching from MC-backups to such *DP-backups* preserves the order-of-magnitude convergence rates of MCTS instances that guarantee exponential rate performance improvement (such as BRUE (Feldman and Domshlak 2012)), and can even allow for proving *somewhat better convergence bounds*. Still, the relative pros and cons of MC- and DP-backups have not been systematically studied so far, and thus are still poorly understood.

This is precisely our contribution in this paper: Using

BRUE and MaxBRUE, a pair of state-of-the-art MCTS algorithms that, *ceteris paribus*, use MC-backups and DP-backups, respectively, we study the dynamics of MC- and DP-backups both formally and empirically. Starting with establishing a pair of comparable worst-case bounds on the convergence rates of these two algorithms, we use the analysis behind these bounds to examine specific dependencies of the two algorithms on various MDP parameters, namely the state and action branching factors, the shape of the reward function, and the entropy of the transition function. To our knowledge, our analysis is first of its kind, and it reveals numerous important issues that get hidden by the worst-case bounds due to certain deficiencies in the formal worst-case analysis of MC-backups. In particular, it suggests that MC-backups are less sensitive than DP-backups to the state branching factor, especially when the transition function concentrates on a small number of outcomes, as it is typically the case in practical applications. The various aspects of the analysis are then put on a systematic experimental test, which reconfirms its key findings.

BACKGROUND

In what follows, we adopt the notation and pseudo-code convention from (Feldman and Domshlak 2014b). In particular, when considering an MDP $\langle S, A, \mathbb{P}, R\rangle$, its state and action branching factors are respectively denoted by $K = \max_s |A(s)|$ and $B = \max_{s,a} |\{s' \mid \mathbb{P}(s'|s,a) > 0\}|$, $s\langle h\rangle$ denotes state $s \in S$ with h steps-to-go, and $A(s) \subseteq A$ denotes the actions applicable in state s. Some auxiliary notation: The operation of drawing a sample from a distribution $\mathcal{D}$ over set $\aleph$ is denoted by $\sim \mathcal{D}[\aleph]$, $\mathcal{U}$ denotes uniform distribution, and $[\![n]\!]$ for $n \in \mathbb{N}$ denotes the set $\{1, \ldots, n\}$. For a sequence of tuples ρ, $\rho[i]$ denotes the i-th tuple along ρ, and $\rho[i].x$ denotes the value of the field x in that tuple.

MCTS, a canonical scheme underlying various MCTS algorithms for online MDP planning, is depicted in Figure 1. MCTS explores the state space in the radius of H steps from the initial state s_0 by iteratively issuing simulated ROLLOUTs from s_0. Each such rollout ρ comprises a sequence of simulated steps $\langle s, a, r, s'\rangle$, where s is a state, a is an action applicable in s, r is an immediate reward collected from issuing the action a, and s' is the resulting state. Once generated, the rollout is used to UPDATE some variables of interest, typically including at least the action value estimators $\widehat{Q}\,(s\langle h\rangle, a)$ and the counters $n(s\langle h\rangle, a)$ that record the number of times the corresponding estimators $\widehat{Q}\,(s\langle h\rangle, a)$ have been updated. Once interrupted, MCTS uses the information collected throughout the exploration to recommend an action to perform at state s_0.

Instances of MCTS vary mostly along their ROLLOUT-ACTION policies, prescribing the action to apply in the current state of the rollout; and their UPDATE strategies, specifying (i) which of the maintained variables should be updated based on the rollout, as well as (ii) how those variables should be updated. The "how" aspect of the MCTS UPDATE procedures is of our focus here. By decoupling between the decisions of what to update and how to update, the emphasized text in Figures 1b and 1c shows the respective subroutines for MC-backup and DP-backup, the two alternatives for "how to update" that are in use these days by various MCTS algorithms.

- MC-backups are based on the principle of averaging random variable samples: Given a new value sample $\bar{r}$ for an action a at $s\langle h\rangle$, $\bar{r}$ updates the running sample average $\widehat{Q}(s\langle h\rangle, a)$, either knowing or just assuming that this way $\widehat{Q}(s\langle h\rangle, a)$ will eventually converge to the true Q-value of a at $s\langle h\rangle$.
- DP-backups implement dynamic programming style estimates propagation, resembling Bellman backups in value iteration. With DP-backups, action value estimates $\widehat{Q}(s\langle h\rangle, a)$ are updated by the weighted sum of the value estimates of the empirically best actions at the outcomes of a (discovered so far), with the weights being induced by the gradually learned parameters of the MDP's stochastic transition function.

Earlier MCTS algorithms, such as flat MC, ε-greedy, UCT, and their numerous variations (Browne et al. 2012), all reflected rather directly the algorithms for reinforcement learning-while-acting in multi-armed bandit problems (MAB) (Robbins 1952): Given a rollout ρ, update (the selected) action value estimates "by ρ", that is, by the actual rewards obtained along the rollout. Recent works on MCTS algorithms for online MDP planning examined the important differences between the (single state) MABs and (multi-state) general MDPs, leading to what was baptized as the principle of separation of concerns (Feldman and Domshlak 2012): Instead of updating "by ρ", update (the selected) action value estimates "along the trajectory of ρ" by some information that goes beyond, and possibly even has nothing to do with, the specific rewards achieved by ρ. In particular, one can use one of the following.

1. MC-updates along additional rollouts, issued from the states along ρ according to a special, update-oriented, "estimation" policy. Such an UPDATE procedure in particular gives rise to the BRUE algorithm (Feldman and Domshlak 2012), and it is depicted in Figure 1b, together with a general template for its ESTIMATE policy.
2. DP-updates along the trajectory of ρ, as depicted in Figure 1c. This procedure in particular gives rise to the MaxUCT (Keller and Helmert 2013) and MaxBRUE (Feldman and Domshlak 2014b) algorithms.

WORST-CASE GUARANTEES VS. REALISTIC EXPECTATIONS

Our comparison between MC-backups and DP-backups in MCTS is carried through two particular MCTS algorithms, BRUE (Feldman and Domshlak 2012) and MaxBRUE (Feldman and Domshlak 2014b), which guarantee exponential-rate reduction of simple regret. The only difference between BRUE and MaxBRUE is that the former employs MC-backups while the latter employs DP-backups. Hence, for ease of presentation, in what follows we refer to these two algorithms as MC and DP, respectively. Both MC and DP use uniform sampling for ROLLOUT-ACTION, and

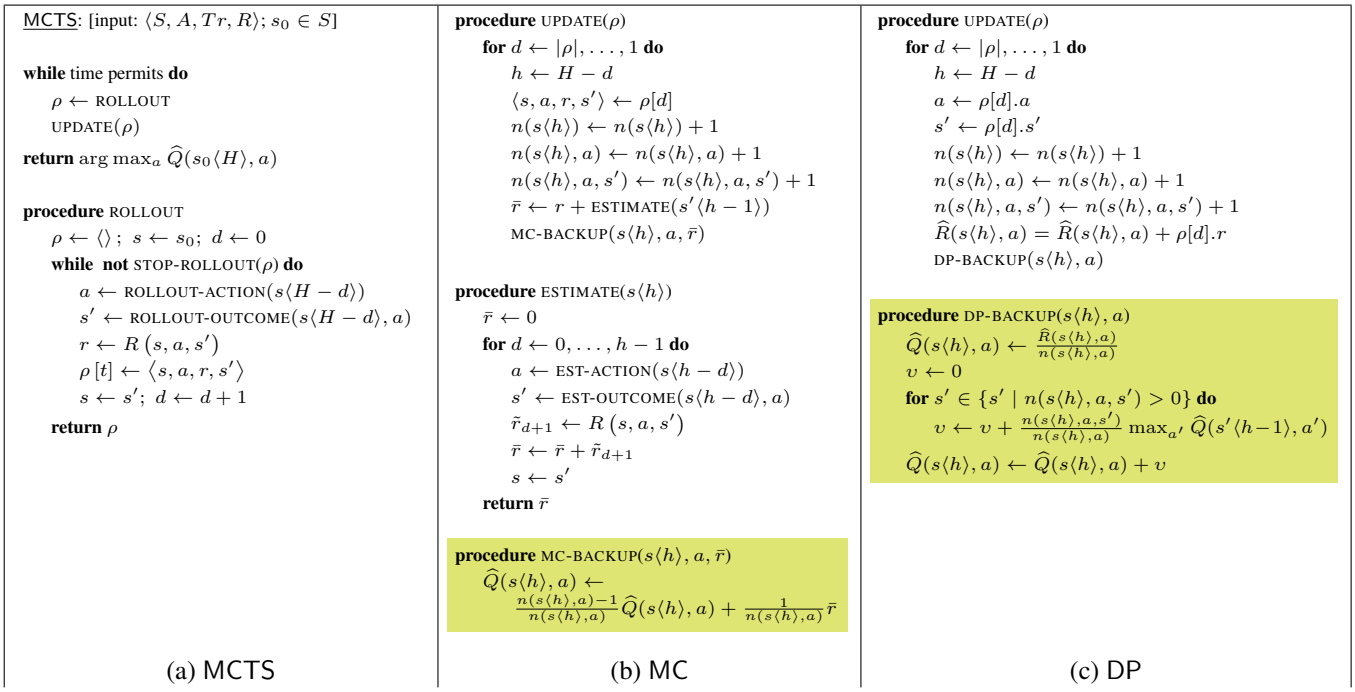


Figure 1: (a) Monte-Carlo tree search general scheme, with "separation of concerns" versions of (b) MC-backup and (c) DP-backup updates

both use the same ROLLOUT-OUTCOME that samples the provided generative model of the action's transition function. With UPDATE as in Figure 1c, that basically concludes the definition of DP. The UPDATE procedure of MC in Figure 1b, and in particular, its ESTIMATE subroutine, need one more choice to be made.

In analogy to DP-backup that propagates the value of the empirically best actions, MC in ESTIMATE makes the estimation rollouts along the empirically best actions (selected by EST-ACTION). Thus, in particular, no implementation choices are left open here. In contrast, for outcome selection along the estimation rollouts, two options are plausible, and both are viable in terms of consistency and performance guarantees. One option is to use the generative model of the action's transition function, same as in ROLLOUT-OUTCOME, while another option is to estimate the transition probabilities, similarly to DP, and draw samples from that empirical distribution.

The advantage of the second scheme is that the number of "oracle calls" to the generative model is similar to that of DP. In contrast, the first scheme performs a factor of $\frac{H}{2}$ more calls to the generative model. In applications where such oracle calls are expensive, due to, e.g., a need to simulate a complex physical model, this can be an important argument for using the second scheme. However, the advantage of the first scheme is that it is not affected by the errors in the estimation of the transition probabilities. In what follows, whenever we need to distinguish between the two options, we will use $\mathsf{MC_M}$ to refer to MC with EST-OUTCOME using the generative model, and $\mathsf{MC_P}$ to refer to MC with EST-OUTCOME using the estimated transition probabilities.

As we already mentioned, both DP and MC have been recently proven to reduce simple regret at exponential rate. However, the corresponding statements of the formal bounds in (Feldman and Domshlak 2012) and (Feldman and Domshlak 2014b) are somewhat involved, and this complicates the comparative analysis and discussion of DP and MC that we want to make here. Propositions 1 and 2 below provide much more accessible formal bounds on the performance of DP and MC, simplified by assuming $B, K \gg 0$, which allows keeping track only of the highest order factors of B and K, and replacing uniform ROLLOUT-ACTION with round-robin, which is equivalent in expectation but allows for simplifying the bounds further. We note that, while the bound for DP in Proposition 1 is qualitatively similar to this in (Feldman and Domshlak 2014b), the bound for MC in Proposition 2 actually improves over the result in (Feldman and Domshlak 2012). The proofs are delegated to a technical report (Feldman and Domshlak 2014a).

Proposition 1 *Let $\pi^B(s_0\langle H\rangle)$ be the action recommendation of* DP *after applying n iterations. Then,*

$$\mathbb{P}\left\{Q(s_0\langle H\rangle, \pi^*(s_0\langle H\rangle)) - Q(s_0\langle H\rangle, \pi^B(s_0\langle H\rangle)) \geq \delta\right\} \leq K(BK)^{H-1} e^{-\frac{\delta^2 n}{2K(BK)^{H-1}H^2}}$$

Proposition 2 *Let $\pi^B(s_0\langle H\rangle)$ be the action recommendation of* MC *after applying n iterations. Then,*

$$\mathbb{P}\left\{Q(s_0\langle H\rangle, \pi^*(s_0\langle H\rangle)) - Q(s_0\langle H\rangle, \pi^B(s_0\langle H\rangle)) \geq \delta\right\} \leq \left(\frac{8BK}{\delta^2}\right)^{H-1} (9BK)^{\frac{1}{2}(H-1)^2}(H-1)!^2 e^{-\frac{3\delta^2 n}{4K(9BK)^{H-1}H^2}}$$

Roughly speaking, the exponents in the bounds in Propositions 1 and 2 capture the reduction rate of the simple regret, while the multiplicative factors capture the length of the "cooling periods" after which the respective bounds become meaningful. In that respect, the convergence rates of DP and MC appear to be comparable, while the "cooling period" of MC appears to be much longer than that of DP. The latter suggests, even if only informally, that the empirical performance of DP should be expected to be more attractive than the empirical performance of MC. However, a deeper inspection of MC and DP below suggests a different perspective on the relative attractiveness of these two algorithms, and more generally, on the relative attractiveness of MC-backups and DP-backups in MCTS.

First, in (Feldman and Domshlak 2012) it was shown that the formal guarantees of MC can be improved by basing the action value estimators only on a fraction α of the most recent samples. This enhancement was referred in (Feldman and Domshlak 2012) as "learning by forgetting". For some specific values of α convenient for our discussion here, the bound for MC from Proposition 1 translates to a bound for the "learning by forgetting" MC(α) as in Proposition 3 below.

Proposition 3 *Let $\pi^B(s_0\langle H\rangle)$ be the action recommendation of* MC(α) *after applying n iterations with a steps-to-go-dependent averaging fraction $\alpha_h = \frac{3}{(9BK)^{h-1}(h-1)^2}$. Then,*

$$\mathbb{P}\left\{Q(s_0\langle H\rangle, \pi^*(s_0\langle H\rangle)) - Q(s_0\langle H\rangle, \pi^B(s_0\langle H\rangle)) \geq \delta\right\}$$
$$\leq \left(1 + \frac{4}{\delta^2}\right)^{H-1} KB^{H-1} e^{-\frac{3\delta^2 n}{4K(9BK)^{H-1}H^2}}$$

Comparing now the bounds for DP and MC(α), it seems that the deficiency of the latter in terms of the worst-case cooling period not only vanishes with sufficiently small values of α, but the dependence of the respective multiplicative factor on K becomes much better than that of DP. Moreover, this improvement seems to come at no cost in terms of convergence rate, expressed by the exponent. However, as we explain below, it appears that this significant improvement of the bound in Proposition 3 should be attributed mostly to the looseness of the bound for the standard setup of MC, and much less to the actual improvement of the performance measures. Indeed, adopting "learning by forgetting" leads to only minor empirical improvement, if at all.[1]

In general, two things should be noted with regards to the above formal bounds at this point. First, by definition, formal bounds capture the worst-case settings of the MDP parameters, that is, uniform transition probability functions, tree-structured state space, etc. As such, the bounds tend to blur certain advantages of one algorithm over another in solving MDPs with some specific (and possibly expected in practice) characteristics. Second, due to conceptual differences between the dynamics of MC- and DP-backups, derivation of the bounds in Propositions 1 and 2 is based on two very different types of analysis. Hence, unlike what often happens for conceptually close techniques (Bubeck, Munos, and Stoltz 2011; Feldman and Domshlak 2012), the value of formal bounds as indicators for the relative attractiveness of MC and DP is questionable. Having these two reservations in mind, in what follows we provide a more conceptual (aka less mathematically specific) comparative analysis of MC and DP by exploring several key features of MDP models, and reasoning about the (possibly different) effects of each of these features on the performance of the two algorithms.

Branching factors B and K (The size of the problem) In both Proposition 1 and Proposition 2, the basis of the analysis that gives rise to the formal guarantees is the fact that identifying the optimal action a^* at the root node $s_0\langle H\rangle$ requires that

(1) the value of a^* at $s_0\langle H\rangle$ is not too underestimated, and that

(2) the values of all the other, sub-optimal actions at $s_0\langle H\rangle$ are not too overestimated.

Both MC and DP ensure that these accuracy requirements are met, yet they differ in the way that these requirements recursively translate into requirements from the descendants of $s_0\langle H\rangle$.

In DP, to ensure that a sub-optimal action a is not too overestimated, all the applicable actions in all of the outcome states of a must not be too overestimated. Thus, the accuracy of estimating a sub-optimal action a in DP translates to accuracy requirements being posed to all the possible BK immediate action successors of a. In contrast, in MC, the likelihood that a sub-optimal action a will be too overestimated is negligible, and this is because the expected value of the samples that induce the estimate of a is upper-bounded by the true value of a, regardless of the estimates of the action successors of a. Thus, the accuracy of estimating a sub-optimal action a in MC translates to no accuracy requirements from the successors of a. In sum, in terms of "not overestimating sub-optimal actions", MC-backups seem to be clearly preferred to DP-backups.

Examining now the requirement of not underestimating the optimal action a^* too much, meeting this requirement in DP requires that the optimal actions at all of the outcome states of a^* are not too underestimated. Indeed, if the latter holds, then the maximal action values propagated to a^* from its outcome states are also not too underestimating. Thus, the requirement of "not too underestimating the optimal action" a^* at $s_0\langle H\rangle$ translates in DP into B similar requirements being posed to all of the state successors of $s_0\langle H\rangle$ via a^*.

When it comes to MC, the picture is somewhat more complicated. Not underestimating the optimal action a^* too much requires that, in expectation, each of the samples inducing the estimate $\widehat{Q}(s_0\langle H\rangle, a^*)$ does not underestimate too much the true value of a^*. This implies that all of the outcome states of a^* should "identify" their optimal actions, which in turn translates into accuracy requirements posed to all (both optimal and sub-optimal) actions applicable at

[1] This was observed both in our experiments for this work, as well as in (Feldman and Domshlak 2012).

these outcome states of a^*. As we just mentioned, the accuracy requirements from sub-optimal actions in MC are negligible, and thus the effective burden is only with the accuracy requirements from the optimal actions at the outcome states.

In sum, the requirement of not underestimating the optimal action translates to accuracy requirements from the optimal actions at the outcome states, *at different stages of planning*. In the lack of more effective proof methods, the accuracy of each estimation sample in the proof of Proposition 2 is considered in isolation, and this is precisely the point where the difference between the bound and the actual performance may inflate. Indeed, the accuracy of an action estimate correlates with the accuracy of the same action estimate at subsequent points in time, yet this correlation is not factored into the bounds.

Importantly, the analysis of MC(α) in that respect is not any different: partial averaging does not offer a way to factor this correlation, but only reduces the bound by imposing accuracy requirements on fewer samples. Clearly, as the number of iterations increases, the correlation increases as well. The question, however, is when can we expect to have higher correlation at earlier stages. For instance, when the probability mass of the transition function concentrates on a small number of outcomes, the effective action branching becomes smaller than the nominal action branching B. In such cases, one should expect to have more samples based on overlapping rollouts, and thus to have a higher correlation. In any case, when the correlation is high, MC becomes equivalent to DP in terms of the accuracy requirement on the optimal action.

In summary, the dependency of DP on B and K is of order $(BK)^H$, whereas, depending on the correlation, the dependency of MC on B and K can be of order as small as B^H. Therefore, MC can be expected to be less sensitive than DP to K, especially when the effective action branching is relatively low, and thus the correlation is relatively high.

The shape of the reward function If right from the first steps, the immediate rewards of the optimal actions appear more attractive than these of the suboptimal actions, then identifying the optimal action a^* at $s_0\langle H\rangle$ is somewhat a simpler problem. The more challenging cases in that respect are when the discriminative rewards are pushed down the search tree, similarly to what happens in goal-driven MDPs. In such cases, identifying the optimal action a^* at $s_0\langle H\rangle$ requires properly identifying optimal actions far from the root, where samples are much sparser. Relating this point to the previous discussion on the size of the problem, it can be expected that the advantage of MC in the more challenging cases becomes more dependent on the effective action branching being relatively low.

The entropy of the transition function For all the bounds, the factor $\frac{1}{B}^H$ in the exponent results from the worst-case transition probability function, which, for each action, induces a uniform distribution over its outcomes. Clearly, as the entropy of the transition functions decreases, the better the bounds and performance of both DP and MC would be. However, here as well, some differences are expected depending on the update scheme. Since DP and MC$_\mathsf{P}$ use in their UPDATE the estimated transition function, their value estimations would be skewed towards the value of the more probable outcomes. Although this skew decreases with the number of samples, this decrease is slower at the deeper nodes since they are sampled less frequently. MC$_\mathsf{M}$, on the other hand, is free from this type of inaccuracy and therefore has certain advantage in that respect.

EXPERIMENTAL STUDY

In what follows, we put the qualitative comparison above into an empirical test. In previous work, the empirical effectiveness of online MDP planning algorithms was typically examined on a set of specific MDP problems, such as, e.g., the benchmark suites of planning competitions (IPPC). These benchmarks, however, are problematic to use if one wants to examine the marginal impact of various parameters of the MDPs on the effectiveness of the algorithms, because these parameters simply cannot be controlled. In fact, almost all of these benchmarks are too large to compute the actual value of different actions at a state, and without that, assessing simple regret of different algorithms is impossible. Taking that on board, we devised a parametric MDP model from which one can select MDP instances with (i) arbitrary set of action values at the initial state, and (ii) arbitrary setting of the parameters discussed in the previous section. This allows us to experiment with large MDPs, for which otherwise it would be impossible (in reasonable time and computational resources) to compute the value function, and based on it, assess simple regret of different algorithms.

For ease of presentation, in what follows we refer to nodes $s\langle h\rangle$ simply by s; the steps-to-go component of the nodes remains clear from the text. In our base MDP setup, the horizon is set to $H = 10$, there are exactly $K = 20$ actions applicable at every node, and each node/action pair induces exactly B equiprobable outcome nodes, exclusive to that node/action pair, i.e, the induced state-space is tree-structured, and the transition probability functions are all uniform. For a node s, an applicable action a, and a possible outcome s', the immediate reward is set to $R(s,a,s') = \frac{Q(s,a)}{h}$, and the value of the outcome node receives the remainder $V(s') = Q(s,a) - R(s,a,s')$. At any node s, there is exactly one optimal action, the value of which equals the value $V(s)$ of the node, whereas all other actions have identical values of $\epsilon V(s)$, for some $\epsilon \in (0,1)$. The choice of ϵ plays an important role here. If, for instance, ϵ is set equally for all nodes, then basing the value updates in both MC and DP on random (and not empirically best) action successors will surface the optimal action at s_0, and this because all the actions will be underestimated by a similar magnitude. Therefore, in our setup, for all nodes reachable by the optimal policy from s_0, we set $\epsilon = 0.6$, while for all nodes off the optimal policy, we set $\epsilon = 0.8$. The only node that is not properly covered by this categorization is the actual initial node s_0, and there we also set $\epsilon = 0.8$. In this setup, the estimates induced by random updates would not preserve the right order of the action values, imposing a harder challenge

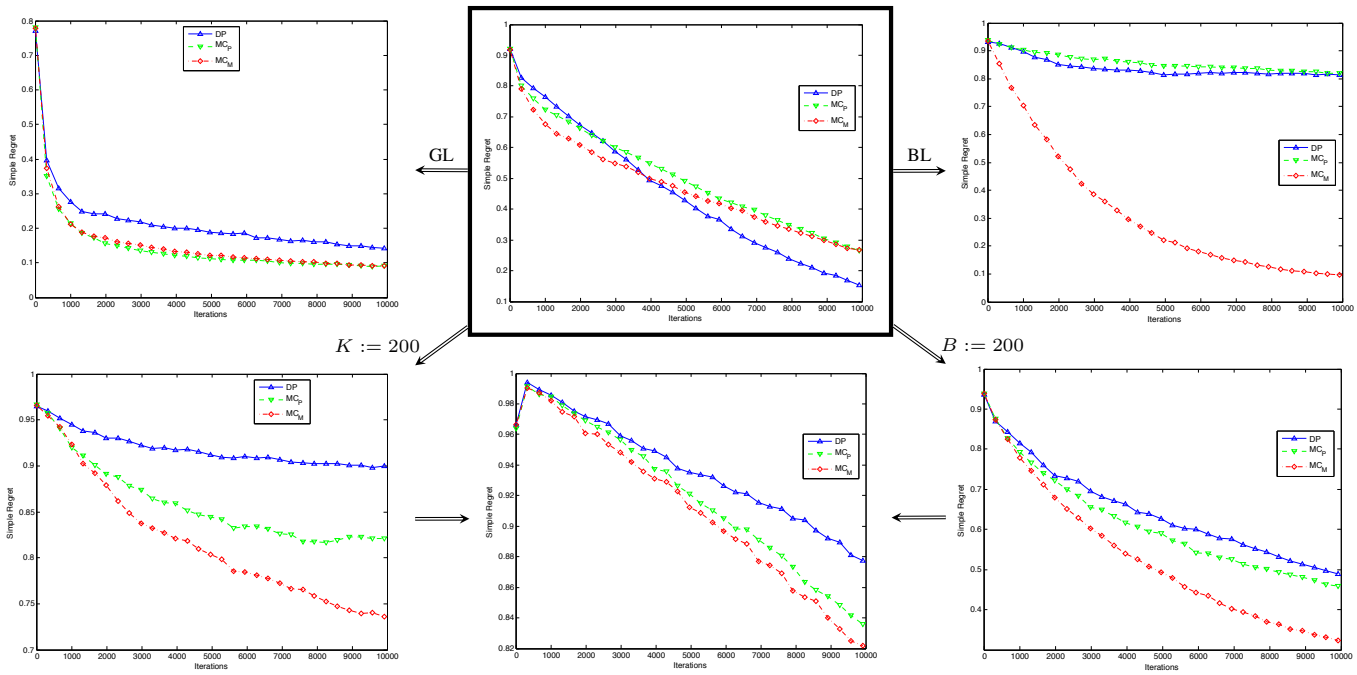


Figure 2: Experimental results on the base setup with $K = 20$, $B = 20$, and all action outcomes being equiprobable (top-center), as well as on the variants with ($\swarrow$) $K = 200$, ($\searrow$) $B = 200$, ($\downarrow$) $K = 200$ and $B = 200$, ($\leftarrow$) "good likely" transition functions, and ($\rightarrow$) "bad likely" transition functions.

on the algorithms.

The emphasized plot in the top-center of Figure 2 depicts the simple regret obtained by the three examined algorithms, DP, $\mathsf{MC_P}$, and $\mathsf{MC_M}$, on the base setup as above, as a function of the number of iterations. While MC here appear slightly better at the start, DP quickly catches up and gradually outperforms MC. However, the picture changes when the base setup is modified in several different ways. First, when scaling up the problem by increasing K to 200 (bottom-left), or by increasing B to 200 (bottom-right), or both (bottom-center), MC performs much better than DP at all times, with $\mathsf{MC_M}$ being the clear dominator. Second, the top-left and top-right plots in Figure 2 depict the results for two setups that deviate from the base only by altering the entropy of the transition probability functions. In both setups, for each node s and each applicable action a, one outcome s' is substantially more likely than all other, with $\mathbb{P}(s' \mid s, a) = 0.9$ and, for all outcomes $s'' \neq s$, $\mathbb{P}(s'' \mid s, a) = \frac{0.1}{B-1}$. The difference between the two setups is the relative value of the more likely outcome s': In the "good likely" setup (GL), the more likely outcome is also more valuable, i.e., $R(s, a, s') + V(s') > R(s, a, s'') + V(s'')$ for all $s'' \neq s'$, and in the "bad likely" setup (BL), it is the other way around. In either case, all action-outcome values are set such that (1) they reflect the value of the action, i.e. $\sum_{s'} \mathbb{P}(s' \,|\, s, a) \left(R(s, a, s') + V(s')\right) = Q(s, a)$, and (2) each action-outcome value is neither smaller than half of the action value, nor higher than the maximal immediate reward ($= 1$, in our experiments), times the number of steps-to-go.

In both "good likely" and "bad likely" variants, the reduction in the entropy of the transition function basically reduces the effective state branching of the actions, and thus the correlation between the successive samples in MC is expected to grow, getting more in line with the optimistic assumption on MC's dependence on B and K. The results depicted in Figure 2 support this expectation. Moving from the base setup, the performance of MC improves in both "good likely" and "bad likely" setups, and in fact, in both setups, MC outperforms DP. Likewise, importantly, while in "good likely" there is effectively no difference between $\mathsf{MC_M}$ and $\mathsf{MC_P}$, in "bad likely", $\mathsf{MC_M}$ is performing much better than $\mathsf{MC_P}$, with the latter meeting the very poor performance of DP. Basically, the "bad likely" setup demonstrates how dramatic can be the implications of establishing value estimation on the estimated transition probabilities. Here, the underestimation of the values by DP and $\mathsf{MC_P}$ results in their very poor performance. To recap Figure 2, it appears that MC outperforms DP except for on MDPs of relatively small size, and $\mathsf{MC_M}$ being justifiably more robust than $\mathsf{MC_P}$.

Another important aspect that we examined in our experiments pertains to the dependence of the algorithm performance on the shape of the rewards as a function of the node depth. In the base setup, *at any node*, the actions are rewarded proportionally to their actual value, and thus, in particular, optimal actions have higher immediate rewards than the sub-optimal actions.

Figure 3 shows the results for two setups that deviate from the base setup in that aspect as follows. (Both these setups are more challenging than the base, and thus the x-axis in Figure 3 goes up to 10^5 iterations, and not to 10^4 iterations like in Figure 2.) In "first equal" (top-center),

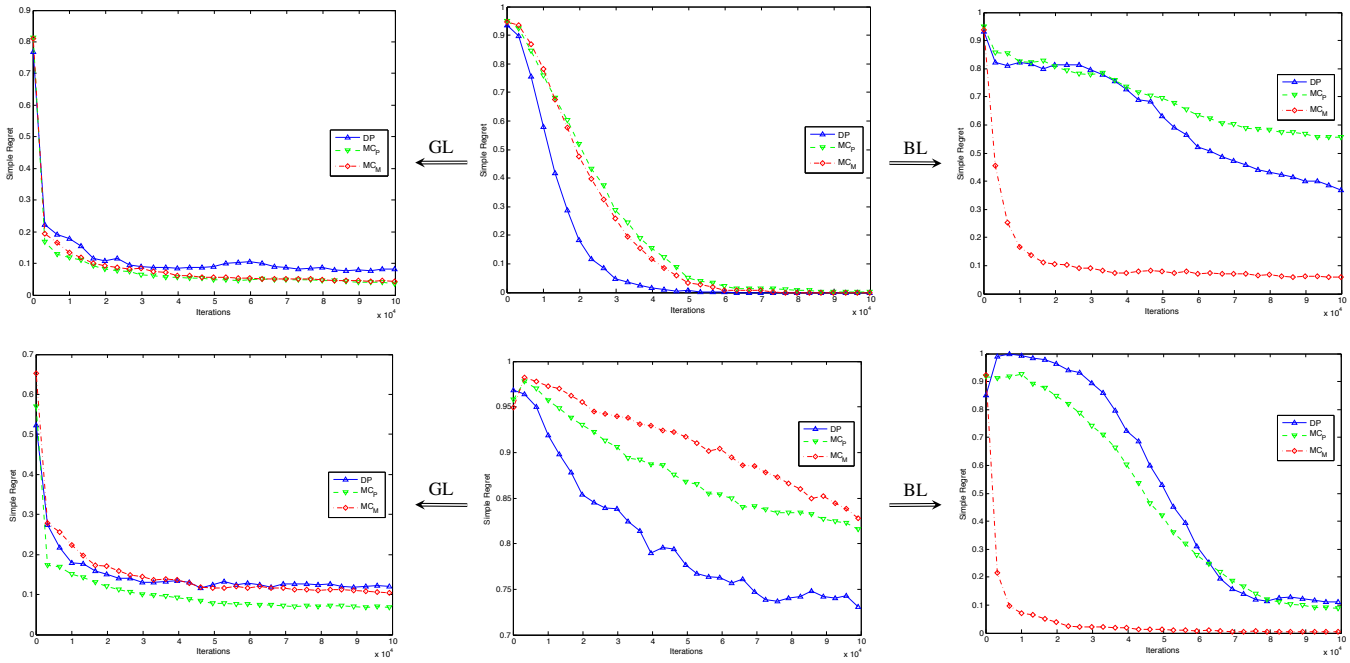


Figure 3: Experimental results on the "first equal" (top-center) and "first few equal" (bottom-center) modifications of the base setup, as well as on their variants with "good likely" ($\leftarrow$) and "bad likely" ($\rightarrow$) transition functions.

the immediate rewards differ from the base setup only at the root, where, instead of rewarding the optimal action higher than the sub-optimal actions, all the actions have the same reward of 0.5, independently of the outcome. The results for the "good likely" and "bad likely" variants of "first equal" are depicted in top-left and top-right corners of Figure 3. In the even more challenging setup "first few equal" (bottom-center), the immediate rewards are set to the minimum between 0.5 and the action-outcome value, that is $R(s, a, s') = \min\{0.5, Q(s, a)\}$; in the "good likely" (bottom-left) and "bad likely" (bottom-right) variants, the appropriate factor is added to $Q(s, a)$.

Comparing the results for the variants of the base setup in Figure 2 with the results for "first equal" and "first few equal" in Figure 3, the qualitative relative performance of DP, $\mathsf{MC_P}$, and $\mathsf{MC_M}$ remains the same, with the absolute performance of all algorithm decreasing, as expected, from the base setup to "first equal", and from "first equal" to "first few equal". It should also be noted that here, in contrast to the base setup, the advantage of DP over MC under equiprobable action outcomes was observed also when K and B were higher than 20. This goes in line with the dependence of MC's performance on the correlation between the successive samples, because pushing the discriminative rewards down the tree delays the correlation. Finally, we also experimented with various graph-structured (in contrast to tree-structured) variants of our MDP model. As expected, the performance of all three algorithms improved with the degree of the multi-connectedness of the nodes, but the improvements were of the same magnitude for all three algorithms. In sum, based on our experiments and in line with our previous analysis, DP appear more effective than MC as long as the size of the problem is sufficiently small, but otherwise, MC outperforms DP even under most challenging conditions, especially if the probability mass of the transition functions concentrates on very few outcomes.

References

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Bjarnason, R.; Fern, A.; and Tadepalli, P. 2009. Lower bounding Klondike Solitaire with Monte-Carlo planning. In *ICAPS*.

Browne, C.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte-Carlo tree search methods. *IEEE Trans. on Comp. Intell. and AI in Games* 143.

Bubeck, S., and Munos, R. 2010. Open loop optimistic planning. In *COLT*, 477–489.

Bubeck, S.; Munos, R.; and Stoltz, G. 2011. Pure exploration in finitely-armed and continuous-armed bandits. *Theor. Comput. Sci.* 412(19):1832–1852.

Cazenave, T. 2009. Nested Monte-Carlo search. In *IJCAI*, 456–461.

Coquelin, P.-A., and Munos, R. 2007. Bandit algorithms for tree search. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 67–74.

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-quality policies for the Canadian Traveler's problem. In *AAAI*.

Feldman, Z., and Domshlak, C. 2012. Simple regret optimization in online planning for Markov decision processes. *CoRR* arXiv:1206.3382v2 [cs.AI].

Feldman, Z., and Domshlak, C. 2013. Monte-Carlo planning: Theoretically fast convergence meets practical efficiency. In *UAI*.

Feldman, Z., and Domshlak, C. 2014a. Monte-Carlo tree search: To MC or to DP? Technical Report IE/IS-2014-03, Technion.

Feldman, Z., and Domshlak, C. 2014b. On MABs and separation of concerns in Monte-Carlo planning for MDPs. In *ICAPS*.

Gelly, S., and Silver, D. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *AIJ* 175(11):1856–1875.

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In *ICAPS*, 135–143.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *ECML*, 282–293.

Péret, L., and Garcia, F. 2004. On-line search for solving Markov decision processes via heuristic sampling. In *ECAI*, 530–534.

Puterman, M. 1994. *Markov Decision Processes*. Wiley.

Robbins, H. 1952. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.* 58(5):527535.

Rosin, C. D. 2011. Nested rollout policy adaptation for Monte Carlo tree search. In *IJCAI*, 649–654.

Sturtevant, N. 2008. An analysis of UCT in multi-player games. In *CCG*, 3749.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Tolpin, D., and Shimony, S. E. 2012. MCTS based on simple regret. In *AAAI*.

To Share or Not to Share? The Single Agent in a Team Decision Problem

Ofra Amir and **Barbara J. Grosz**
School of Engineering and Applied Sciences
Harvard University
oamir@seas.harvard.edu, grosz@seas.harvard.edu

Roni Stern
Department of Information Systems Engineering
Ben Gurion University of the Negev
roni.stern@gmail.com

Abstract

This paper defines a new decision-making challenge for multi-agent systems, the "Single Agent in a Team Decision" (*SATD*) problem. SATD differs from prior multi-agent communication problems in the assumptions it makes about teammates' knowledge of each other's plans and their possible observations. The SATD assumptions are more appropriate for the decision-making settings of multi-agent groups comprising people and computer agents, for which obtaining people's complete plans or anticipating all possible events is infeasible. The paper proposes a novel integrated logical-decision-theoretic approach to solving SATD problems, called MDP-PRT. Evaluation of MDP-PRT shows that it outperforms a previously proposed communication mechanism that did not consider the timing of communication and compares favorably with a coordinated Dec-POMDP solution that can exploit its full knowledge of all possible observations.

Introduction

This paper defines a new decision-making challenge for multi-agent systems, the "Single Agent in a Team Decision" (*SATD*) problem, which may be described informally as follows: An individual collaborating in a multi-agent team obtains new information, unanticipated at planning time. This (single) agent has incomplete knowledge of others' plans. It must decide whether to communicate this new information to its teammates, and if so, to whom, and at what time. SATD differs from previously studied multi-agent communications decisions problems (Pynadath and Tambe 2002; Xuan, Lesser, and Zilberstein 2001; Roth, Simmons, and Veloso 2005, inter alia) in that it does not assume complete knowledge of other agents' plans or policies nor that all observations are knowable in advance. It assumes instead that agents have some knowledge of each other's intentions and plans which can be used to reason about information sharing decisions.

SATD arises from the ways in which effective teamwork decomposes complex activities into constituent tasks and delegates responsibility for those tasks to team members with appropriate expertise and capabilities. Human teammates typically make only general plans and allocate tasks at a high level of abstraction. They do not necessarily know each other's plans nor consider together all contingencies of all possible plans for doing those tasks. Their cognitive load is lowered significantly as a result. To make appropriate decisions about sharing information, they must reason with only uncertain knowledge of their teammates' plans. Agents participating in mixed networks comprising people and computer agents or supporting people in team settings also must be capable of such reasoning. The SATD problem can also arise in purely computer-agent teamwork settings. For example, in ad hoc teamwork (Stone et al. 2010), an agent joining an existing team only after planning time lacks significant information about other agents' plans, but might still need to reason about which observations to share with teammates.

We are investigating SATD in the context of developing computer agents to support care teams for children with complex conditions (Amir et al. 2013). Agents able to support health-care teams by identifying the information to share, with whom and when have the potential to substantially improve health outcomes. Care teams for children with complex conditions typically involve many caregivers – a primary care provider, specialists, therapists, and non-medical care givers. The care team defines a high-level care plan that describes the main care goals, but there is no centralized planning mechanism that generates a complete plan for the team or that can ensure coordination. Caregivers are unaware of their collaborators' complete plans, yet their individual plans often interact. Communicating relevant information among team members is crucial for care to be coordinated and effective, but doing so is costly and often insufficient in practice.

To address SATD, the paper proposes a novel, integrated Belief-Desire-Intention (BDI) and decision-theoretic (DT) representation that builds on the strengths of each approach. In particular, our approach integrates the Probabilistic Recipe Trees (PRT) representation of an agent's beliefs about another agent's plans (Kamar, Gal, and Grosz 2009) with a Markov Decision Process (MDP) to support a collaborating group in their execution of a plan. We call this integrated representation MDP-PRT.

We evaluated an agent using MDP-PRT to solve the SATD problem in an abstract setting using the ColoredTrails framework (Gal et al. 2010). Results show that it outperforms the *inform algorithm* proposed by Kamar et al. (2009).

In addition, we compared the MDP-PRT agent's performance with that of Dec-POMDP policy that was informed about all possible observations. The MDP-PRT agent obtains results close to those obtained by this Dec-POMDP policy despite lacking such complete knowledge and thus a coordinated policy that was derived using full information about all possible observations.

The paper makes three contributions. First, it formally defines the SATD communication problem and contrasts it with previously studied communication problems in multi-agent settings. Second, it proposes a new representation (MDP-PRT) that enables agents to reason about and solve the SATD communication problem. Third, it demonstrates the usefulness of this representation and analyzes the effect of agents' uncertainties and of communication cost on the performance of a team.

Problem Definition

In this section, we formally define the SATD problem. SATD arises in the context of a group activity of a team of agents. We assume the group's plan meets the SharedPlans specification for collaborative action (Grosz and Kraus 1996). Two particular properties of SharedPlans are important: plans – which decompose into constituent tasks – may be only partially specified and agents may not know the details of constituent tasks for which they are not responsible. An instance of the SATD problem is represented by a tuple $\langle \mathbf{a_i}, \mathbf{o}^*, \mathbf{A_{-i}}, \mathbf{b_{SP}}, \mathbf{V}, \varphi_{\mathbf{comm}}, \mathbf{C} \rangle$:

- $\mathbf{a_i}$: the agent that observes new information.
- $\mathbf{o}^*$: the new information a_i obtained.
- $\mathbf{A_{-i}}$: the other agents that are part of the team.
- $\mathbf{b_{SP}}$: a_i's beliefs about the SharedPlan of the team. a_i knows its own plans but he is uncertain about others' plans.
- $\mathbf{V}$: the utility function; its value is the utility of completed constituent tasks.
- $\varphi_{\mathbf{comm}}$: a function that produces a modified b'_{SP} under the assumption that the agents in A_{-i} (or a subset of them) are informed about o^*.
- $\mathbf{C}$: the cost of communicating o^*.

SATD is the problem of a_i determining whether to communicate o^* to agents in A_{-i} and if so, at what time. It has two components: determining which agents are candidates for receiving the information and deciding whether, and when, to send information to (some or all) candidates. This paper focuses on the constituent problem of deciding whether and when to send information.

Related Work

In this section we discuss prior work on communication in multi-agent systems from BDI and decision theory literature and distinguish SATD problem from previously studied communication problems. Theories of teamwork and collaboration (Grosz and Kraus 1996; Cohen and Levesque 1990; Sonenberg et al. 1992) emphasize the key role of communication in teamwork. BDI approaches to multi-agent planning often base their communication mechanisms on these theories. For example, the joint intentions model (Cohen and Levesque 1990) defines conditions for communications, such as communicating to establish joint intentions, communicating the achievement of a goal, and communicating when learning that a goal cannot be achieved. These ideas were used in the STEAM multi-agent framework (Tambe 1997): In this framework agents use a decision tree to determine whether to communicate information about operation termination by considering their belief about the joint intentions of the team and weighing (the domain specified) costs and risks of not communicating. Other works have developed teamwork programming languages that include explicit rule-based communication mechanisms (Weerasooriya, Rao, and Ramamohanarao 1995; Pokahr, Braubach, and Lamersdorf 2005). These approaches do not reason about probabilities and utilities, and therefore it is hard to quantify and evaluate their performance (Pynadath and Tambe 2002).

Prior work on decision theoretic approaches to multi-agent communication can generally be classified into two types: approaches that reason about communication during planning, and approaches that reason about communication during execution. The DEC-POMDP-COM model (Goldman and Zilberstein 2003) and the COM-MTDP model (Pynadath and Tambe 2002) provide a theoretical model for reasoning about communication during planning and include communication actions in the agents' policies. Spaan et al. (Spaan, Gordon, and Vlassis 2006) developed a model in which the information to be communicated is included in the actions vectors of agents, and is then incorporated into the observation vectors received by agents at the next time step. Such offline approaches assume that all possible observations that agents may receive are known during planning and that there is a centralized planning process that generates the policies. In contrast, SATD models situations in which there is no centralized planning and not all observations are known in advance.

Other approaches reason about communication during execution time (Xuan, Lesser, and Zilberstein 2001; Oliehoek, Spaan, and Vlassis 2007; Emery-Montemerlo et al. 2005). Such approaches aim to identify situations in which communication would improve the group's performance, based on observations obtained by agents. This approach reduces computational complexity since agents do not need to consider in advance what to communicate for each possible scenario, but rather only reason about communication given their actual observations. For example, Roth et al. (Roth, Simmons, and Veloso 2005) reason about communication by growing a tree of the possible joint beliefs of the team. This work has been extended to consider not only when to communicate, but also what subset of observations to communicate to other team members (Roth, Simmons, and Veloso 2006). These approaches assume a known joint policy, such that agents can determine what action each agent in the team would take following communication. Wu et al. (Wu, Zilberstein, and Chen 2011) propose an algorithm for online planning and communication that merges observation histories based on their similarity in terms of cho-

sen actions. They then use these merged histories to reason about history incompatibility, which determines communication. This model does not include communication cost, but instead tries to maximize utility while minimizing the amount of communication. In SATD, reasoning about communication is also done during execution. However, SATD differs from the above works in that it does not assume knowledge about the possible observations that may occur, nor a coordinated policy generated by a centralized process. Instead, SATD assumes that agents know the intentions of other agents and can therefore reason about how their plans will be affected without having complete knowledge of their policy.

There are few prior works that have combined BDI concepts with decision theoretic approaches to reason about communication. Inspired by BDI teamwork, Kwak et al. (Kwak et al. 2011) define trigger points in which communication should be considered in the context of a DEC-POMDP model. These trigger points occur when there is an ambiguity in the mapping from a joint policy of an agent to its action. When such ambiguities arise, agents reason about the possible gains from communication, where an agent can communicate either by *asking* for information or by *telling* other agents about their observations. This model still requires knowledge of all possible observations that might be obtained during execution. Kamar et al. (Kamar, Gal, and Grosz 2009) developed the PRT representation of collaborative activities which we discuss in the approach section. Interactive POMDPs (Gmytrasiewicz and Doshi 2005) also integrate beliefs about other agents in an MDP framework. In I-POMDPs, beliefs are over agent "types"; types are distinguished by an agent's optimality criteria and other canonical POMDP components. In contrast, in MDP-PRT the beliefs are over collaborating agents' plans and are compactly represented in PRTs.

Approach

This section proposes a solution to the 2-agent SATD problem $\langle a_1, a_2, b_{SP}, V, o^*, \varphi_{comm}, C\rangle$, in which an agent a_1 learns new information o^* and needs to reason about whether and when to communicate o^* to a_2. We discuss extensions to settings with more agents in the Discussion section. To solve the 2-agent SATD communication problem, we use an MDP in which the states explicitly represent a_1's beliefs about a_2's plans. The choice of representation for the agent's beliefs b_{SP} is key, as it affects the way b_{SP} can be revised and therefore the computational efficiency of solving the MDP. Our approach uses a PRT to represent b_{SP}. Henceforth we refer to the integrated MDP-PRT representation as "MDP-PRT".

The PRT representation extends the SharedPlans (Grosz and Kraus 1996) formalization by introducing decision-theoretic notions of uncertainty and utilities to the hierarchical plan representation for recipes. Formally, a PRT for a complex action defines a probability distribution over the possible recipes (plans) for accomplishing that action. The recipes are represented in an AND/OR tree. AND nodes represent complex actions that need to be carried out. Probabilities over OR branches represent the agent's belief about its partners' possible alternatives for carrying out these complex actions. We chose PRTs because they are a compact representation and their decomposable structure allows reasoning about non-dependent activities separately.

The MDP-PRT $\langle A, S, s_0, R, Tr\rangle$ is defined by the following:

- A: the set of actions
- S: the set of states
- s_0: the initial state
- R: the reward function
- Tr: the transition function

A includes two actions: *inform* (communicating o^*) and $\neg$*inform* (not communicating o^*). Each state in S encompasses a_1's beliefs about a_2's plans (i.e., the PRT corresponding to b_{SP}).[1] We denote a state by b_{SP}. The initial state b_{SP_0} corresponds to a_1's initial beliefs about the SharedPlan. The reward function is a function of V and C: the reward for a state b_{SP} is the value of the constituent tasks completed in the last time step minus the cost of communication if a_1 chose to inform a_2. The transition function, $Tr(b_{SP}{}', a, b_{SP})$, defines the probability of reaching state $b_{SP}{}'$, when taking action a in state b_{SP}. a_1's belief may change for two different reasons. First, if a_1 communicates o^* to a_2, then b_{SP} changes to $\varphi_{comm}(b_{SP}, o^*)$. Second, as a_2 executes actions in its constituent plans, a_1 may "observe" a_2's actions or results of those actions and learn more about a_2's plans. To reflect this reasoning, we define an additional function, $\varphi_{obs}(b_{SP})$. This function takes as input b_{SP} and returns the set of next expected beliefs $\mathbf{b_{SP}^{next}}$ and their probabilities $\mathbf{Pr}(\mathbf{b_{SP}^{next}})$.

To illustrate the MDP-PRT components, we consider the care coordination problem. The primary care provider (PCP) has beliefs about the neurologist's treatment options and the likelihood each might be chosen. When the PCP considers whether to share new information about the patient (e.g. that the child had a seizure), she reasons about the effect of this information on the neurologist's plans (φ_{comm}). Whether or not she decides to inform the neurologist, her beliefs also evolve with time as she learns about actions the neurologist has executed (φ_{obs}). For example, if she learns that the neurologist ordered certain lab tests, she might revise her beliefs if the lab tests are consistent with some plans but not with others.

Algorithm 1 gives the pseudo-code for $Tr(b_{SP}{}', a, b_{SP})$. First, if a_1 chooses to inform, b_{SP} is updated using φ_{comm} (lines 2-3). If $b_{SP}{}'$ is one of the possible next states according to φ_{obs}, its probability based on φ_{obs} is returned (lines 4-6). If it's not included in the set of next possible beliefs the transition probability is 0 (line 7). We note that although each state encompasses a probability distribution over plans, the state space of the MDP is not continuous because the set of considered possible beliefs is discrete and finite.

An optimal single agent communication policy can be computed using any MDP solver, e.g. value iteration (Sondik 1971), which was used in our implementation.

[1] b_{SP} also includes a_1's own plans but a_1 does not need to reason about them in the context of information sharing.

Algorithm 1: The transition function.

Input: $b_{SP}{}', a, b_{SP}$
$Pr(b_{SP}{}') \leftarrow 0$
if $a = inform$ **then**
  $b_{SP} = \varphi_{comm}(b_{SP}, o^*)$
$\langle \mathbf{b_{SP}^{next}}, \mathbf{Pr}(\mathbf{b_{SP}^{next}}) \rangle = \varphi_{obs}(b_{SP})$
if $b_{SP}{}'$ *in* $\mathbf{b_{SP}^{next}}$ **then**
  return $Pr(b_{SP}{}')$ *according to* $\mathbf{Pr}(\mathbf{b_{SP}^{next}})$
return 0

This policy computation is performed when a new observation o^* is obtained by an agent a_i at time t. During subsequent rounds, the agent uses the computed policy to decide whether to inform a_2 of o^*.

The MDP-PRT can be viewed as a decision-theoretic representation that supports the monitoring and execution of a SharedPlan. While we focused on the question of sharing new information, a similar approach can be taken to reason about whether to help a teammate or ask teammates for information (Kamar, Gal, and Grosz 2009). We also note that while in our empirical domain agents could observe each other's actions, the MDP-PRT can also be used if not all actions are observable, by modifying φ_{obs} to consider the possibility of not observing an action. We discuss possible extensions of the MDP-PRT to larger groups of agents in the discussion section.

Empirical Methodology

We tested the MDP-PRT agent using a modified version of the Colored Trails (CT) game used by Kamar et al. (2009). An example configuration of the game is shown in Figure 1 The game is played by two agents, the *Partner* (PAR) and the *Observer* (OBS), on a 4×4 board. Each square in the board is of one of four colors. At the onset of the game, *PAR* is located on a board square ($(0, 0)$ in Figure 1). It possesses chips of the four colors and uses them to move on the board: it can move to an adjacent square (vertically or horizontally) by giving up a chip of the same color as the destination square. Its task is to reach the goal square.

Each square has some probability that a trap will appear on it based on its color. Traps are represented by the "bomb" icon. When an agent moves into a trap square, it cannot move further and the game ends. PAR has uncertain knowledge about the distribution of traps (i.e. the probability a trap will appear on each color), but does not know their exact locations. OBS learns the exact locations of traps at the onset of the game. It also knows the chips that PAR possesses and its knowledge regarding trap distributions.

The game is cooperative and the score (utility) is calculated as follows: the agents receive 100 points if PAR reaches the goal and 5 points for each remaining chip it holds at the end of the game. If PAR does not reach the goal, agents are penalized according to its distance from the goal (10 times Manhattan distance from the goal).

Each turn in the game consists of two phases: In the first phase, the communication phase, OBS can choose to perform an *inform* action that will reveal trap locations to PAR. Informing, however, incurs a cost that is deducted from the agents' final score. OBS therefore needs to reason about the utility of an inform action by weighing the benefits and costs entailed by it. In the second, movement, phase, PAR moves using its chips. The game ends when one of the following occurs: PAR reaches the goal or moves to a trap location, or after a maximal fixed number of turns passes.

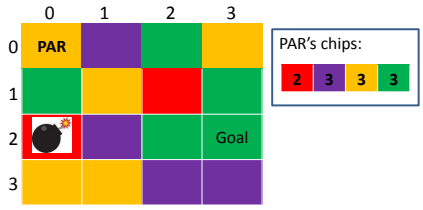


Figure 1: An example CT game configuration.

This game setting encompasses some of the key characteristics of the healthcare domain and enables studying them more abstractly: (1) PAR's alternative plans and the uncertain knowledge of OBS about these plans correspond to caregivers' alternative treatment plans and their incomplete knowledge of others' treatment plans; (2) OBS seeing PAR's location corresponds to a caregiver observing other caregivers' actions by reading notes in the medical record; (3) learning about traps corresponds to caregivers detecting problems with a treatment plan after observing a change in patient status, and (4) communication is costly due to caregivers' limited time and many responsibilities.

MDP-PRT for the CT Domain

We implemented OBS as an MDP-PRT agent that makes the decision of whether to share trap locations (o^*) with PAR. PAR's task of getting as close as possible to the goal represents its constituent task in the collaborative activity of OBS and PAR. At the onset of the game, when OBS learns the true locations of traps, value iteration is run to generate a communication policy. The initial PRT, representing OBS's beliefs about PAR's possible plans (b_{SP}), is derived from the board and from OBS's knowledge of PAR's knowledge of trap distributions. For example, if OBS knows PAR has no knowledge about trap locations, it assigns equal probabilities to all shortest paths that PAR can take.

φ_{comm} revises OBS's beliefs assuming it communicates the trap locations to PAR. It computes new shortest paths assuming that PAR will choose one of the shortest paths towards the goal that do not pass through any traps. For example, in the configuration shown in Figure 1, and assuming that PAR has no knowledge of trap locations, OBS's initial b_{SP} will assign equal probabilities to all shortest paths from PAR's location $(0, 0)$ to its goal $(2, 3)$. When considering the "inform" action in this state, the modified belief according to φ_{comm} will assign equal probabilities to all shortest paths that do not pass through $(2, 0)$.

Whether or not OBS chooses to inform PAR, the generation of the next possible states $b_{SP}{}'$ also takes into account PAR's next possible movements (which OBS will observe during execution). The states correspond to a_1's modified

beliefs according to φ_{obs}. That is, OBS knows it will observe PAR's movements at execution time and reasons about the possible changes in the probabilities of different paths. For each possible movement, φ_{obs} generates a new state $b_{SP}{}'$ by eliminating plans that do not include that movement and re-normalizing the probabilities of the remaining paths.

During the game, OBS revises its belief (b_{SP}) at each turn by eliminating paths that do not include the observed movements. It decides whether to communicate based on its computed policy.

Results

To evaluate the MDP-PRT agent we ran several types of experiments. The first experiment compared the performance of the MDP-PRT agent with that of a PRT agent using the inform algorithm of Kamar et al. (2009). The algorithm decides whether to inform at the onset of the game by comparing the expected utility of the PRT representing OBS's initial beliefs with the expected utility of a revised PRT describing the belief about the plans if trap locations are shared (φ_{comm}). If the increase in utility is greater than the communication cost, the agent communicates. This approach is myopic as it does not consider the possibility of waiting to learn more about PAR's plans and possibly communicating at a later stage.

We used 6 different board configurations varying the trap distribution in different colors. We generated 6 board instances for each trap distribution and varied the communication cost between runs. We ran each combination of board instance and communication cost 10 times. In all games, PAR knew the trap distribution but not the actual trap locations; its moves were chosen randomly based on its expected utility (e.g. it chose between shortest paths weighing trap probability). OBS knew that PAR knew only the trap distribution. PAR randomly chose one of its shortest paths and moved accordingly. Once OBS communicated the trap locations, PAR randomly chose a shortest path that did not pass through any traps.

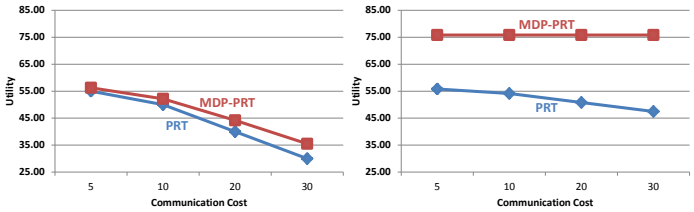
Table 1 shows the average utility achieved by the agents in experiments with different communication costs. Results are averaged across all board instances and the 10 runs of each instance. The maximal utility that could be obtained in the game ranged between 100 and 125, depending on PAR's chips and the goal location. Thus a communication cost of 10 is about 10% of the maximal possible utility. As expected, the MDP-PRT agent outperforms the PRT agent across all configurations and communication costs. All reported differences are statistically significant ($P < 0.01$).

Comm. Cost	5	10	20	30
PRT	67.67	65.17	60.17	55.17
MDP-PRT	**77**	**75.57**	**72.83**	**70.03**

Table 1: The average utility for each agent and communication cost, averaged over all board instances.

The MDP-PRT agent saves unnecessary communication and thus outperforms the PRT agent. To illustrate this difference, we consider the board configuration shown in Figure 1. In the first round of the game, PAR is 2 squares away from the trap. When the PRT agent decides whether to communicate, it considers the distribution over possible paths that PAR might choose. One of these paths goes through the trap; so there is some probability that PAR will reach the trap location. The OBS (PRT) agent will decide to communicate if the difference between the expected utility of the updated PRT after informing and the expected utility of the PRT representing PAR's plans without informing is higher than the communication cost. In contrast, the MDP-PRT agent would also take into account the possibility of informing at a later stage: Since PAR is two squares away from the trap, the MDP-PRT agent will choose not to inform at this turn, since if PAR would move right in the next turn, then it is unlikely it chose a plan that passes through a trap location.

Figure 2(a) shows the average utilities achieved by the agents in boards with probability of 0.15 for traps to appear on any board square; Figure 2(b) shows the average utilities in boards with probability of 0.15 for traps to appear on any red or green square. (There are more traps on the board configurations in (a) than in (b).) As can be seen in the figure, the MDP-PRT agent outperformed the baseline PRT agent in both. The difference in utilities grows with communication cost. When communication cost increases, the MDP-PRT agent benefits more from avoiding unnecessary communication. In the configuration shown in Figure 2(a) we also observe a higher decrease in utilities for the MDP-PRT agent. This is a result of the combination of boards that have more traps together with the high communication costs.



(a) All 0.15 configuration **(b)** RG 0.15 configuration

Figure 2: Average utilities obtained by the agents: (a) boards with probability of 0.15 for traps to appear on any color; (b) boards with probability 0.15 for traps to appear on any red or green squares.

Reasoning about the future, however, has a cost in terms of computation time. The MDP-PRT agent needs to generate and solve the MDP for the given game configuration, which requires more computation than the one-shot decision made by the PRT agent at the onset of the game. On average, decision time was about two times slower when using the MDP-PRT agent (mean = 29.82 msec), as compared to the PRT agent's (mean = 13.71 msec).

Comparison with Dec-POMDP

Key aspects of SATD are that agents cannot anticipate a_1 observing o^* nor form a coordinated policy. This limits the possible utility achievable in SATD, as a coordinated policy that takes into account knowledge of possible observations during planning time can lead to better performance. We illustrate this using the board instance shown in Figure 1. As described above, the MDP-PRT agent can delay its decision

about communication to the second round, after it observes the new location of PAR. If PAR moves to (1,0), the agent might decide to communicate. A Dec-POMDP with prior knowledge of the possible observations can do better. Had the agents known in advance that traps might appear only on the two red squares and that OBS will learn the true trap locations, they could have agreed on a communication policy, allowing PAR to learn something about the trap locations even when OBS does not communicate. For example, they could have agreed that OBS will communicate if the trap is in location $(1, 2)$ but not if it is on $(2, 0)$. Then, if OBS does not communicate, PAR learns that there is a trap on $(2, 0)$.

To compute a policy for the setting in which agents *anticipate* that OBS will learn about trap locations, we solve a Dec-POMDP that has information about the trap distributions, and knows that OBS will learn the true locations of traps. A state in the Dec-POMDP for this game includes the true location of the traps, the location of PAR, its chips and its knowledge of the trap distribution. For example, if traps can appear only on red squares, a board configuration with two red squares such as the board shown in Figure 1 will induce four possible initial worlds states: no traps on the board; 2 traps on the board, one on (2,0) and one on (1,2); a trap on (1,2) but not on (2,0); a trap on (2,0) but not on (1,0) (as in the world state shown in the figure). These four initial states also describe the location of PAR ($(0, 0)$ in the example), its chips and its knowledge of the trap distribution. At the onset of the game, OBS observes the true state of the world (i.e. the true trap locations).

Solving the Dec-POMDP results in an optimal joint policy that achieves the maximal utility. For PAR, the policy specifies movement actions. For OBS, the policy specifies whether or not to inform PAR of the trap locations at each communication stage. Note that the Dec-POMDP policy can always perform at least as well as the MDP-PRT agent, as it has more knowledge about the world. .

As a result of the computational complexity of the Dec-POMDP, we restricted the experiments to boards that had at most two possible trap locations. We varied the probability that a possible trap location actually includes a trap, with values $P_t = 0.2, 0.5, 0.8$, and generated 6 board configurations for each of these values. P_t determines the probability distribution over world states. For example, in the board shown in Figure 1, the two red squares are possible trap locations. If $P_t = 0.2$, then the probability of the world state shown in Figure 1 is $P_t \cdot (1 - P_t) = 0.2 \cdot 0.8 = 0.16$ because (2,0) has a trap while (1,2) does not.

A Dec-POMDP policy maximizes the expected utility over all possible world states while MDP-PRT is run on a particular world state (e.g. a particular board with traps already determined). Thus, for each board configuration and trap distribution, we ran MDP-PRT on boards representing all possible world states and compute a weighted average of their utility based on the distribution over world states. For example, the board instance in Figure 1 is one of four instances for a board configuration with possible traps on squares (2,0) and (1,2). The other three instances include a board with no traps, a board with a trap in (1,2) but not in (2,0) and a board with traps on both locations. We ran the MDP-PRT agent on each of these boards to compare it with the Dec-POMDP policy.

The first two rows in Table 2 show the utility obtained by the MDP-PRT agent and the expected utility of a joint policy generated by a Dec-POMDP. These utilities are averaged over all 6 board configurations and trap probabilities.

	5	10	25	50
Dec-POMDP	**101.8**	**101.46**	**100.2**	**98.13**
MDP-PRT (accurate)	100.52	99.08	94.38	87.5
MDP-PRT (inaccurate)	99.75	97.46	92.13	83.67

Table 2: The performance of MDP-PRT (with accurate and inaccurate beliefs) and an optimal Dec-POMDP approach.

As expected, the Dec-POMDP always performs better as a result of its additional information at planning time. The difference is statistically significant ($P < 0.01$). As can be seen in the table, increasing communication cost leads to larger differences between the Dec-POMDP and MDP-PRT utilities, a result of the additional communication required by the MDP-PRT agent. When communication cost is up to 10% of the utility of reaching the goal (100 points), MDP-PRT compares well with the Dec-POMDP policy (less than 3% difference). When communication cost rises to 25% or 50% of goal utility, the difference grows, but the average utility achieved by MDP-PRT is still within 15% of the optimal utility. Increasing the trap probability decreases the utility obtained by both agents; it also increases the gap between MDP-PRT and the Dec-POMDP because world states that include more traps and require more communication are more likely.

Inaccurate Beliefs About Others' Plans

The MDP-PRT agent requires knowledge of the probability distributions over the plans of other agents and a way to update this distribution given new information. In practice, it is possible (and likely) that agents will estimate these probabilities inaccurately. To examine the possible effects of such inaccuracies on agents' performance, we ran additional experiments. In these experiments OBS knew a probability distribution over PAR's chips, but did not know the actual chips PAR had. Specifically, it assumed a uniform distribution between 1 to 4 chips of each color. Consequently, the initial PRT and updates to the PRT could be inaccurate in their assignment of probabilities to possible plans.

The last row of Table 2 shows the performance of the MDP-PRT agent using this inaccurate PRT estimation. As expected, inaccurate beliefs lead to a decrease in utility ($P < 0.05$). In these settings, however, the average decrease in utility was small (less than 5%). Figure 3 shows the performance of three different agents on a particular board configuration that included only one possible trap location with probability 0.5 of a trap on that location. In this configuration, the Dec-POMDP never communicates and thus always obtains the maximal possible utility. Compared to the MDP-PRT with an accurate PRT representing PAR's possible plans, the inaccurate model communicated more often, resulting in lower utility. Its inaccurate probability estimation assigned higher probability to paths that pass through traps than their actual probability. In general, wrong esti-

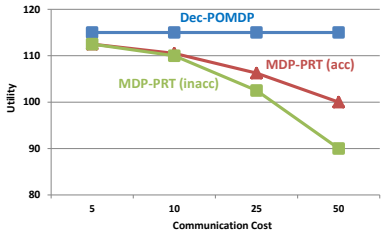


Figure 3: Utility for board configurations with one possible trap.

mation of the PRT can lead to communicating too much or communicating too little.

Discussion and Future Work

This paper defines the Single Agent in a Team Decision Problem, SATD, which concerns decisions made by a single agent that is part of a group during collaborative activities, decisions that affect the performance of the group. We focus on the problem of communication where an agent obtains new, unanticipated, information during a collaborative activity and needs to reason about whether and when to share this information with its teammates. This formulation of the problem differs from previously addressed multi-agent communication problems in that it does not assume that all observations are known and considered in advance and does not assume complete knowledge of other agents' plans or policies. Instead, it assumes that agents have some knowledge of other agents' intentions and plans that enables them to reason about the effect of sharing information.

Defining and solving the SATD problem is a first step in our effort to develop computer agents that would support the care team of children with complex conditions. Our goal is to develop agents that would remove some of the communication burden from caregivers by identifying what information should be shared and with whom. In mixed network teams of humans and computational agents and in dynamic domains with frequent changes, such as this health-care domain, it is unlikely that teams can or will produce a complete long-term joint plan or be able to anticipate all possible events. As a result, agents supporting people in such settings cannot directly apply existing decision-theoretic approaches to multi-agent communication. It may, however, be possible to elicit the goals, intentions and partial plans of team members and use those in conjunction with domain knowledge to reason about the effect of new information on a collaborative activity. We thus introduce the assumption that agents' have a way to update their beliefs about others' plans based on new information and the world state. While in this paper we consider agents reasoning about the utility of plans, it is also possible to consider weaker notions of the effect of information sharing on other agents' plans. For example, we might consider the probability that the current plan would fail or the probability that a collaborating agent would change its plan given the new information.

To solve the SATD problem, we propose a combined BDI/Decision-theoretic approach that augments MDPs by adding an explicit representation of an agent's belief about its partners' plans as part of the state representation and including a belief update function that estimates the possible plans resulting from sharing new information with teammates. We chose to use PRTs to model this belief, because they offer a compact and decomposable representation of multi-agent activities: in settings in which different parts of plans are independent of each other, PRTs enable modifying only that part of the plan that is affected by new information. The SATD definition, however, is general, and beliefs about plans can be represented in different ways, e.g. MDP policies or BDI plans.

The assumption that agents can reason about the plans of their teammates raises the question of how to obtain such knowledge. There are several possibilities we intend to explore in future work: First, in some settings, the team makes an initial partial plan together and allocates responsibilities. From these partial plans, agents can infer an initial belief about their teammates' plans. Further, the planning process may reveal not only agents' possible plans, but also some of their intentions and goals. The (possibly partial) knowledge about its partners' intentions and goals, together with domain knowledge, can enable an agent to reason about the alternative plans that its partners' may consider based on new information. For example, in a path-finding domain such as our evaluation domain, knowing the goal location and that shortest paths result in higher utilities enables the Observer to infer the new paths the Partner is likely to choose given new information about traps. In the medical domain, there are often guidelines for treating different conditions which can be used to reason about alternative plans of other care providers.

In addition, agents may be able to observe some or all of their teammates' actions and refine their beliefs about their plans based on those observations using plan recognition techniques (Amir and Gal 2011; Geib and Goldman 2001). For instance, physician notes in an electronic medical record provides doctors with information about the plans, actions and rationales of other members of the care team. For teams collaborating over a long time horizon, including health-care teams for chronically ill patients, learning could be used to better estimate team members' plans and the ways they change with new information.

In future work we intend to extend the MDP-PRT approach to larger teams and investigate efficient ways to reason about determining the agents with which to share information. A trivial approach is for an agent to consider each of the other agents separately. However, this approach is intractable and disregards interactions between other agents' plans. Therefore, agents will need to utilize the decomposability of the team's plan to focus on relevant agents and the relevant aspects of their plans. We will also implement MDP-PRT agents that would support people, where the agents will focus on reducing the communication burden of people while people will focus on executing and revising their plans given the information they receive. We further plan to test our agents in more complex domains that simulate additional aspects of care coordination such as uncertainty about the utilities of plans and possibly conflicting goals. Finally, we plan to investigate ways to obtain and update beliefs about other agents' plans as described above.

Acknowledgements

We thank Lee Sanders, Ece Kamar and Kobi Gal for helpful discussions. The research reported in this paper was supported in part by a grant from the Nuance Foundation.

References

Amir, O., and Gal, Y. 2011. Plan recognition in virtual laboratories. In *Proceedings of the 22nd international joint conference on Artificial Intelligence*, 2392–2397.

Amir, O.; Grosz, B. J.; Law, E.; and Stern, R. 2013. Collaborative health care plan support. In *Proceedings of the 12th international conference on Autonomous agents and multi-agent systems*, 793–796.

Cohen, P., and Levesque, H. 1990. Intention is choice with commitment. *Artificial intelligence* 42(2):213–261.

Emery-Montemerlo, R.; Gordon, G.; Schneider, J.; and Thrun, S. 2005. Game theoretic control for robot teams. In *IEEE International Conference on Robotics and Automation*, 1163–1169.

Gal, Y.; Grosz, B.; Kraus, S.; Pfeffer, A.; and Shieber, S. 2010. Agent decision-making in open mixed networks. *Artificial Intelligence* 174(18):1460–1480.

Geib, C. W., and Goldman, R. P. 2001. Plan recognition in intrusion detection systems. In *DARPA Information Survivability Conference*, volume 1, 46–55.

Gmytrasiewicz, P. J., and Doshi, P. 2005. A framework for sequential planning in multi-agent settings. *J. Artif. Intell. Res.(JAIR)* 24:49–79.

Goldman, C. V., and Zilberstein, S. 2003. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 137–144.

Grosz, B., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86(2):269–357.

Kamar, E.; Gal, Y.; and Grosz, B. 2009. Incorporating helpful behavior into collaborative planning. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, 875–882.

Kwak, J.-y.; Yang, R.; Yin, Z.; Taylor, M. E.; and Tambe, M. 2011. Robust execution-time coordination in dec-pomdps under model uncertainty. In *Sixth Annual Workshop on Multiagent Sequential Decision Making in Uncertain Domains (MSDM-2011)*, 39.

Oliehoek, F. A.; Spaan, M. T.; and Vlassis, N. 2007. Dec-pomdps with delayed communication. In *The 2nd Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains*.

Pokahr, A.; Braubach, L.; and Lamersdorf, W. 2005. Jadex: A bdi reasoning engine. In *Multi-agent programming*. Springer. 149–174.

Pynadath, D. V., and Tambe, M. 2002. The communicative multiagent team decision problem: analyzing teamwork theories and models. *Journal of Artificial Intelligence Research* 16(1):389–423.

Roth, M.; Simmons, R.; and Veloso, M. 2005. Reasoning about joint beliefs for execution-time communication decisions. In *Proceedings of the 4th international joint conference on Autonomous agents and multiagent systems*.

Roth, M.; Simmons, R.; and Veloso, M. 2006. What to communicate? execution-time decision in multi-agent POMDPs. *Distributed Autonomous Robotic Systems 7* 177–186.

Sondik, E. J. 1971. The optimal control of partially observable markov decision processes. *PhD the sis, Stanford University*.

Sonenberg, E.; Tidhar, G.; Werner, E.; Kinny, D.; Ljungberg, M.; and Rao, A. 1992. Planned team activity. *Artificial Social Systems* 890.

Spaan, M. T.; Gordon, G. J.; and Vlassis, N. 2006. Decentralized planning under uncertainty for teams of communicating agents. In *the fifth international joint conference on Autonomous agents and multiagent systems*, 249–256.

Stone, P.; Kaminka, G. A.; Kraus, S.; Rosenschein, J. S.; et al. 2010. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI*.

Tambe, M. 1997. Agent architectures for flexible practical teamwork. *AAAI* 97(1):997.

Weerasooriya, D.; Rao, A.; and Ramamohanarao, K. 1995. Design of a concurrent agent-oriented language. In *Intelligent Agents*. Springer. 386–401.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence* 175(2):487–511.

Xuan, P.; Lesser, V.; and Zilberstein, S. 2001. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the fifth international conference on Autonomous agents*, 616–623.

Computing Contingent Plans via Fully Observable Non-Deterministic Planning*

Christian Muise and **Vaishak Belle** and **Sheila A. McIlraith**
Department of Computer Science
University of Toronto, Toronto, Canada.
{cjmuise,vaishak,sheila}@cs.toronto.edu

Abstract

Planning with sensing actions under partial observability is a computationally challenging problem that is fundamental to the realization of AI tasks in areas as diverse as robotics, game playing, and diagnostic problem solving. Recent work on generating plans for partially observable domains has advocated for online planning, claiming that offline plans are often too large to generate. Here we push the envelope on this challenging problem, proposing a technique for generating conditional (aka contingent) plans offline. The key to our planner's success is the reliance on state-of-the-art techniques for fully observable non-deterministic (FOND) planning. In particular, we use an existing compilation for converting a planning problem under partial observability and sensing to a FOND planning problem. With a modified FOND planner in hand, we are able to scale beyond previous techniques for generating conditional plans with solutions that are orders of magnitude smaller than previously possible in some domains.

1 Introduction

An agent *planning* to achieve a goal in a *partially observable* environment with *sensing* actions (PPOS) has the option of choosing how to act *online* by interleaving planning, sensing, and acting; or choosing how to act *offline* by generating a plan with decision points predicated on sensing outcomes. In this paper we investigate the latter. In particular, we examine the problem of generating conditional plans for planning problems with incomplete information about the initial state, deterministic actions, and sensing actions. In this work, we use the terms "offline planning" and "conditional planning" interchangeably to always refer to the offline generation of contingent plans; the online variant will be referred to as "online contingent planning." Our focus here is on a particular class of problems where the initial state specification includes a set of state constraints called *state invariants*. These are commonly used to model the behaviour of a device, or physical environments with which the agent is interacting. We further assume that uncertainty decreases monotonically, *i.e.* once a property of the world is known, it can change but cannot become unknown again.

There are merits and shortcomings to both online and offline planning in this context. Online contingent plans are generally easier to compute since integrating online sensing with planning eliminates the need to plan for a potentially exponential (in the size of relevant unknown facts) number of contingencies. In the absence of deadends, online contingent planning can be fast and effective. Recent advances include CLG and CLG+ (Albore, Palacios, and Geffner 2009; Albore and Geffner 2009), *K*-Planner (Bonet and Geffner 2011), and SDR (Brafman and Shani 2012).

In contrast, planning offline constructs conditional plans with decision points for sensing outcomes and guarantees that the goal will be achieved if it is possible to do so. The plan is larger than an online plan but has the merit that it is generalized to deal with alternative sensing outcomes. Indeed plan existence for conditional planning is 2-EXP-complete (Rintanen 2004; Baral, Kreinovich, and Trejo 2000). More importantly, because offline planners are able to search and deliberate, they have the capacity to avoid deadends, and also to support the generation of optimized high quality plans. Some early conditional planners were based on partial order planning (e.g., CNLP (Peot and Smith 1992), Cassandra (Pryor and Collins 1996)) and Graphplan (e.g., SGP (Weld, Anderson, and Smith 1998)). MBP (Bertoli et al. 2001) and BBSP (Rintanen 2004) are more recent BDD-based model checking planners. Planners based on heuristic search include Contingent-FF (Hoffmann and Brafman 2005), POND (Bryce, Kambhampati, and Smith 2006), and most recently CLG which has an offline variant (Albore, Palacios, and Geffner 2009). Finally, the conditional plan we consider is similar to the "execution structure" of Kuter et al.'s "conditionalized plan" (2007), but differs in construction and generality of what the plan represents. Experimental results reported in the literature appear to indicate that CLG represents the state of the art in terms of scalability of offline conditional planning.

In this paper we present PO-PRP, a conditional planner. PO-PRP plans achieve the goal for all consistent sequences of observations for which a solution exists. The key to our planner's success is its reliance on state-of-the-art techniques for fully observable non-deterministic (FOND) planning. In particular, we use an existing compilation by Bonet and Geffner (Bonet and Geffner 2011) (henceforth BG) for converting a PPOS problem to a FOND planning problem.

*This paper also appears in the Proceedings of the Twenty-Eighth Conference on Artificial Intelligence (AAAI-14).

All actions are treated as deterministic with the exception of sensing actions, which are encoded as non-deterministic actions. We then modify a state-of-the-art FOND planner, PRP (Muise, McIlraith, and Beck 2012), to compute strong cyclic plans in the form of policies, which we roll-out into plans represented as DAGs. We address a number of critical challenges, leading to a conditional planner that is able to scale beyond previous techniques for offline planning and that is able to compute solutions that are orders of magnitude smaller than state-of-the-art CLG in some domains.

2 Contingent Planning via FOND Planning

In this section, we formulate the PPOS problem, and then review the translation of PPOS problems to FOND ones.

Syntax and Interpretation. Following BG, we specify the problem in a STRIPS-like language, where actions can additionally have *conditional effects*. Formally, a PPOS domain is a tuple $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{F}$ is the set of fluent atoms, $\mathcal{A}$ is the set of actions, $\mathcal{O}$ is the set of observations, $\mathcal{I}$ is a set of clauses over $\mathcal{F}$ that determines the initial state, and $\mathcal{G}$ is a conjunction of atoms over $\mathcal{F}$ determining the goal condition. The specification is interpreted over a set of (world) *states*, which are essentially truth valuations to the atoms in $\mathcal{F}$. We say a literal l *holds* in a state s iff s assigns l to be true. This is extended for connectives in an obvious way. In particular, since $\mathcal{I}$ is a set of clauses, $\mathcal{I}$ would hold in a number of states; the *belief state* b is the set of all (world) states where $\mathcal{I}$ holds. By extension, we say a formula α holds in b iff α holds in every state $s \in b$.

For $a \in \mathcal{A}$, we let $\textsc{Pre}(a)$ be a conjunction of atoms to denote its preconditions, and $\textsc{Eff}(a)$ be a set of pairs $\langle c, l \rangle$ to capture its conditional effects. Observations $o \in \mathcal{O}$ are of the form $\langle c, l \rangle$ with the understanding that when c is true, o informs the agent about the truth of l. In practice, this is achieved by treating observations o as a separate category of actions which have c as the precondition, and l as the effect.[1] We say an action a is *applicable* in s iff $\textsc{Pre}(a)$ holds in s. Analogously, we say a is *applicable* in b iff a is applicable in every $s \in b$. On performing a in b, a *successor* belief state b' is defined by performing a in each $s \in b$. On performing an observation $o = \langle c, l \rangle$ in b, the successor belief state b' is the *maximal* set of states in b agreeing on l. By extension, a sequence $a_0 \cdot a_1 \cdots a_k$, possibly involving observations, is applicable in b if a_0 is applicable in b, resulting in a successor belief state b_1, and inductively, a_i is applicable in b_i, ultimately resulting in b_k. A belief state b' is said to be *reachable* from b if there is some sequence of actions and observations that when applied to b results in b'.

Solutions. Generally with PPOS problems, a solution is rarely a simple sequence of actions, because the agent may need to perform a number of sensing actions to determine aspects of the world based on which further actions can be taken. Therefore, a solution to a PPOS problem is a *policy* Π that is best viewed as a branching structure, usually called a *conditional plan*, induced by the outcomes of sensing actions (Geffner and Bonet 2013). Equivalently, one may view Π as a partial function from belief states to actions (Geffner and Bonet 2013). Such a function would then advise the subsequent action to be taken based on the actual observation. Finally, we say that Π *solves* the PPOS problem $\mathcal{P}$ iff the executions advised by Π are applicable in the belief state b for $\mathcal{P}$, and they result in belief states b^* where the goal condition $\mathcal{G}$ holds.

[1] Our sensor model differs slightly from BG in that they assume sensing actions are triggered as soon as the preconditions hold. Our planner, on the other hand, chooses to apply sensing actions when needed, just like ordinary physical actions.

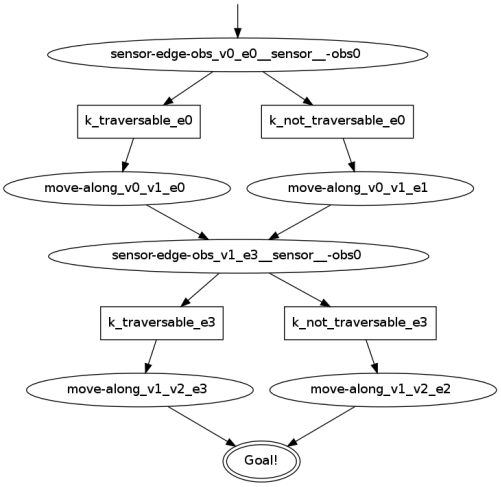


Figure 1: Example solution to the CTP. Circles represent the action to take or observation to be made while boxes contain edge labels indicating the sensing outcome that has occurred.

While conditional plans are usually *trees*, in this work we reduce the redundancy by representing many belief states as a single node in a DAG. Crucially, a node may correspond to a partial belief state, capturing many configurations of the agent's belief. As an example PPOS problem, consider the Canadian Traveller's Problem (CTP) where an agent must navigate a city with roads that may or may not be traversable (due to large amounts of snow fall). Sensing the status of a road can only be done when the agent is adjacent to it. If we consider the class of maps that have a pair of roads between every two adjacent locations in a chain, exactly one of which is traversable, then an obvious strategy presents itself: sense one of the two roads that leads to the next location; if it is traversable then take it; otherwise take the other road. Naive solutions to the problem are exponential in size (every new location has a new choice of two roads), but in this work we strive to generate plans such as in Figure 1.

Compiling Away Uncertainty. Computing successor belief states b', often called *belief tracking*, is non-trivial. Note, for example, that applying actions and observations over all world states (as needed for obtaining b') is clearly exponential in $|\mathcal{F}|$. In recent work, belief tracking is shown to be tractable against the *contingent width* of a PPOS problem, which is claimed to typically be small for most real-world domains (Albore, Palacios, and Geffner 2009). Thus, it follows that for many interesting domains, belief track-

ing is *quadratic* in $|\mathcal{F}|$. Nonetheless, even a quadratic factor is computationally challenging in large domains. A second key result is that a PPOS problem can be translated into a FOND one (Albore, Palacios, and Geffner 2009). This supports the exploitation of state-of-the-art heuristic search techniques for classical planning. Of course, the quadratic belief tracking problem has a natural analogue in the translation, and so computational issues persists.

Interestingly, BG show that when we further restrict ourselves to the so-called *simple* PPOS problems, belief tracking is *linear* in $|\mathcal{F}|$. Informally, simple problems are motivated by the observation that many domains can be characterized by *state invariants*, and actions typically do not depend on fluents whose values are unknown. In precise terms, invariant clauses are clauses that hold in every world state (Helmert 2009), and often represent multivalued functional fluents. The second assumption in simple problems is that the body of conditional effects cannot mention *hidden* literals, that is, literals l such that $\mathcal{I} \not\models l$ and $\mathcal{I} \not\models \neg l$. Hidden literals, however, may appear in the preconditions of actions and may also appear in the effects of actions. Following BG,

Definition 1. (Simple PPOS Problems.) A PPOS problem $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is *simple* if the non-unary clauses in $\mathcal{I}$ are invariant, and no hidden fluent appears in the body of a conditional effect.

Most significantly, features of simple problems include:

- **Effective Characterization:** Suppose $\mathcal{I}^*$ are the non-unary clauses in $\mathcal{I}$ and b is the belief state for $\mathcal{I}$ in $\mathcal{P}$. If b^* is reachable from b and the literals in S are known to hold in b^*, then b^* is completely characterized by $\mathcal{I}^* \cup S$.
- **Monotonicity:** Suppose b' is reachable from b and l is known in b'. If b'' is reachable from b', then l is also known in b''.

From PPOS to FOND. Adapting the work of BG, we now present a translation from simple PPOS problems to FOND ones, where the non-deterministic actions are from the sensor model. The main idea is to replace every literal l with fluent atoms Kl and $K\neg l$, which denotes knowing that l is true vs. false.

Definition 2. Suppose $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is a simple PPOS problem. We define $K'(\mathcal{P}) = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ as a fully-observable non-deterministic planning problem where

1. $\mathcal{F}' = \bigcup_{l \in \mathcal{F}} \{Kl, K\neg l\}$;
2. $\mathcal{A}'$ is the set of actions $\mathcal{A}'_{\mathcal{A}} \cup \mathcal{A}'_{\mathcal{O}} \cup \mathcal{A}'_{\mathcal{V}}$, where,
   (a) $\mathcal{A}'_{\mathcal{A}}$: for every $a \in \mathcal{A}$, there is an action $a' \in \mathcal{A}'_{\mathcal{A}}$ such that if $l \in \textsc{Pre}(a)$ then $Kl \in \textsc{Pre}(a')$, and if $\langle c, l \rangle \in \textsc{Eff}(a)$ then $\{\langle Kc, Kl \rangle, \langle \neg K\neg c, \neg K\neg l \rangle\} \subseteq \textsc{Eff}(a')$;
   (b) $\mathcal{A}'_{\mathcal{O}}$: for $o = \langle c, l \rangle \in \mathcal{O}$, there is an action $a' \in \mathcal{A}'_{\mathcal{O}}$ such that $\textsc{Pre}(a') = Kc \wedge \neg Kl \wedge \neg K\neg l$ with two possible non-deterministic effects, $\textsc{Eff}_1(a') = \{\langle \top, Kl \rangle\}$ and $\textsc{Eff}_2(a') = \{\langle \top, K\neg l \rangle\}$;
   (c) $\mathcal{A}'_{\mathcal{V}}$: for every $(c \supset l) \in \mathcal{I}^*$, there is an action $a' \in \mathcal{A}'_{\mathcal{V}}$ such that $\textsc{Pre}(a') = Kc$ and $\textsc{Eff}(a') = \{\langle \top, Kl \rangle\}$;
3. $\mathcal{I}' = \{Kl \mid l \in \mathcal{I}\}$; and
4. $\mathcal{G}' = \{Kl \mid l \in \mathcal{G}\}$;

We use the notation Kc and $\neg K\neg c$ when $c = l_1 \wedge \ldots \wedge l_k$ to mean $(Kl_1 \wedge \ldots \wedge Kl_k)$ and $(\neg K\neg l_1 \wedge \ldots \wedge \neg K\neg l_k)$ respectively. The key differences between Definition 2 and BG are that (1) we use $\neg Kl \wedge \neg K\neg l$ as part of the precondition for a sensing action and (2) we assume that the agent is free to choose when a sensing action occurs. The former helps avoid inconsistent beliefs during search, and the latter is a restriction that can be removed easily if so desired.

The intuition is this: since unary clauses in $\mathcal{I}$ are clearly known, we index K to these literals in $\mathcal{I}'$; $\mathcal{A}'_{\mathcal{V}}$, then, represents the invariants in $\mathcal{I}$. $\mathcal{A}'_{\mathcal{A}}$ determines when the agent would know the effect of an action. Sensing actions are the only non-deterministic actions in the translated FOND problem, and these indicate that at execution time, the agent would either come to know that the literal is true or would come to know that the literal is false. In practice, on performing such a non-deterministic action, we will be tracking two possible successors for a world state corresponding to each outcome. Finally, the goal state is one where every literal in the original goal $\mathcal{G}$ is known to be true.

Analogous to the notion of a policy Π at the level of belief, the solution to a FOND problem is a policy (i.e. a partial function) Π' that maps (world) states to actions. Applicability and goal reachability are defined for the FOND problem also in an analogous manner. Finally, the translation is justified by means of a *soundness* and *completeness* result by BG: a policy Π' obtained for $K'(\mathcal{P})$ can be converted to a policy Π for $\mathcal{P}$, and vice versa.

3 Approach

The modified BG translation described in Section 2 provides a FOND encoding of our PPOS planning problem, $K'(\mathcal{P})$, in which sensing actions are encoded as non-deterministic actions. Our planner, PO-PRP, leverages state-of-the-art FOND planner PRP (Muise, McIlraith, and Beck 2012), extended with conditional effects (Muise, McIlraith, and Belle 2014) and is modified to address three critical challenges for dealing with PPOS problems. FOND planners are generally predicated on an assumption of *fairness* – if an action is executed infinitely many times, every non-deterministic outcome will occur infinitely often (Cimatti et al. 2003). A key challenge was addressing the fundamental violation of fairness by sensing actions. A second challenge was to suitably handle the computation of indirect effects of actions (ramifications) resulting from the state invariants. A third critical element was leveraging a technique used in PRP, strong cyclic detection, to produce compact conditional plans.

3.1 From PRP to PO-PRP

We review and elaborate upon key elements of PRP that are essential to PO-PRP, with particular focus on one aspect of the planner referred to as *strong cyclic detection* (SCD). Our interest in the SCD procedure of PRP stems from the fact that we can leverage it for computing a compact conditional plan given the policy PO-PRP generates (cf. Section 3.4).

Problem Specification. PO-PRP takes as input a simple PPOS problem $\mathcal{P}$ and outputs a conditional plan. The first

step of PO-PRP is to apply the augmented BG translation described in Section 2 to $\mathcal{P}$, generating the associated FOND problem $K'(\mathcal{P})$. From this point, the problem specification is identical to that of PRP, which can be found in (Muise, McIlraith, and Beck 2012). The details are not necessary to the results that follow. The translated planning problem $K'(\mathcal{P})$ is input as a PDDL file to PO-PRP, which uses the SAS$^+$ representation (Helmert 2009). A significant property of PRP (likewise PO-PRP), is that states are represented as *partial states* – a subset of literals that is interpreted as a conjunctive formula, compactly representing a family of states. We exploit this compact representation to generate small-sized conditional plans (cf. Section 3.4).

General Approach. Cimatti et al. (2003) identify three types of plans for FOND problems: *weak*, *strong*, and *strong cyclic*. Intuitively, a *weak plan* corresponds to an "optimistic plan" that reaches the goal under at least one possible set of action outcomes of the actions in the plan. A *strong plan* corresponds to a "safe plan" and is a closed policy that achieves the goal in a finite number of steps while never visiting the same state twice. Often, however, weak plans are not acceptable and strong plans do not exist. As a viable alternative, a *strong cyclic plan* is a closed policy with the property that every reachable state will eventually reach the goal via the policy under an assumption of fairness.

PRP creates a strong cyclic plan in the form of a policy that maps states to actions. The representation of this policy, however, is non-standard. Rather than storing an explicit mapping of complete states to actions, PRP creates a set of *condition-action pairs* P, each of the form $\langle p, a\rangle \in P$ with p a partial state. In order to return a unique action for a given state s, there is a total ordering over the pairs that allows us to use P and simply return the action in the "most preferred" pair $\langle p, a\rangle$ such that $s \models p$ (ordered, for example wrt distance from goal). We will use $P(s)$ to designate the pair that is most preferred, and we say that P *handles* state s if P returns some pair. PRP's core algorithm is as follows:

1. Let $Open = \{s_0\}$ and $Cls = \emptyset$;
2. Select and move a state s from $Open$ to Cls such that,
   (i) If $P(s) = \bot$, compute a classical plan for s and augment the policy with the result
   (ii) If $P(s) = \langle p, a\rangle$ and $a \in \mathcal{A}_{\mathcal{O}}$, add to $Open$ the states $\{Prog(s, a, \text{Eff}_1(a)), Prog(s, a, \text{Eff}_2(a))\} \setminus Cls$;[2]
   (iii) If $P(s) = \langle p, a\rangle$ and $a \notin \mathcal{A}_{\mathcal{O}}$, add to $Open$ the state $Prog(s, a, \text{Eff}(a))$ if it is not in Cls;
   (iv) If $P(s) = \bot$, process s as a *deadend*;
3. If $Open$ is empty, return P. Else, repeat from step 2;

Note that step 2(i) is essentially computing a weak plan by way of solving a classical planning problem. In Section 3.3, we describe more precisely how this search procedure is modified to incorporate indirect effects of actions (ramifications) found in the translated PPOS problems.

[2] $Prog(s, a, e)$ is the *progression* of s wrt a's effect e and is defined as usual for planning with conditional effects (Reiter 2001).

The PRP planner has a number of components (described in (Muise et al., 2012)) that are key to the success of PO-PRP, including (1) deadend detection, generalization, and avoidance, (2) stopping conditions for weak plans, and (3) conditions for terminating the simulation of action effects (replacing steps 2(ii) and 2(iii)). It is this final phase that we discuss below.

Strong Cyclic Detection. PRP has the facility to "mark" certain pairs in the policy P to indicate that if $\langle p, a\rangle$ is marked, then P is a strong cyclic plan for every state s where $P(s) = \langle p, a\rangle$. This feature allows PRP to forgo steps 2(ii) and 2(iii) if the pair is marked – instead of expanding the state with every possible outcome, the state s is considered handled completely and the process continues. While reproducing the full strong cyclic detection (SCD) procedure is beyond the scope of this paper, later we do rely on one aspect of SCD for the export of conditional plans.

PRP will only allow $\langle p, a\rangle$ to be marked if P is guaranteed to return a marked pair $P(s') = \langle p', a'\rangle$ for every state s' that could be reached by the pair $\langle p, a\rangle$. In other words, we have the following property (following Definition 7 and Theorem 3 of (Muise, McIlraith, and Beck 2012)):

Proposition 1. The SCD procedure of PRP ensures that a pair $\langle p, a\rangle$ of policy P is marked only if for every state s such that $P(s) = \langle p, a\rangle$ and every non-deterministic effect Eff of action a, the pair $P(Prog(s, a, \text{Eff}(a))) = \langle p', a'\rangle$ is marked or p' is a goal.

3.2 Violating Fairness with Sensing Actions

Following Cimatti et al. (2003), solutions to non-deterministic planning problems typically rely on an assumption of fairness with respect to non-deterministic actions: if an agent executes a non-deterministic action infinitely many times, every one of its outcomes will occur infinitely often. Unfortunately, in our current PPOS problems, the only non-deterministic actions are sensing actions and they are decidedly *unfair*. The outcome of a sensing action will reflect the state of the world, which is unchanging unless changed by an action. Fortunately, since we assume that the state of the world only changes as the result of the actions of the plan (i.e., there are no exogenous actions) then once a fluent has been sensed, the planner should not be compelled to execute the sensing action again because the value of the fluent is known.

This is realized by our problem encoding, $K'(\mathcal{P})$, together with the assumption of monotonicity of knowledge. Recall that a precondition of any sense action is that the outcome is currently not known and the effect is that one of the literal or its negation is known, violating the precondition for the sense action to be executed again. This together with monotonicity ensures that a particular ground sensing action can only ever be executed at most once during online execution. As a result, the space of solutions to the FOND planning problem, and subsequently the space of conditional plans, will *never contain a cycle with a sensing action*. This key property allows us to preserve the properties of our FOND planner, despite the inherent unfairness of sensing actions.

3.3 Computing Ramifications

When state invariants or state constraints are combined with an action theory, they result in indirect effects of actions – further consequences of an action that result from enforcing the truth of the state invariants. Understanding what effects should be generated and how is an instance of the *ramification problem*, a well-studied problem in Knowledge Representation (KR). A variety of solutions have been proposed for dealing with this issue, including compilation of these indirect effects or ramifications into further direct effects of actions, representing indirect effects as actions that are triggered by states, representing indirect effects as further axioms or derived predicates, etc. (e.g., (Pinto 1999; McIlraith and Scherl 2000; Strass and Thielscher 2013)).

Our BG inspired encoding, $K'(\mathcal{P})$, captures these indirect effects as a distinguished set of so-called invariant actions, $\mathcal{A}'_{\mathcal{V}}$. Intuitively, following the execution of a regular action, these invariant actions should be executed as bookkeeping actions to compute all indirect effects of the action. Unfortunately, if left unconstrained, this can lead to extensive wasted search effort in parts of the state space that will never be reached during execution, as well as the discovery of meaningless deadends that PO-PRP will try to avoid.

We elected to continue to compute the effects of invariants via the application of actions so that the heuristics PO-PRP uses to compute a weak plan would inform the computation of ramifications. Further, PRP's/PO-PRP's use of regression to compute condition-action pairs would become prohibitively complex were we instead to compile the ramifications into extra effects of actions. Inspired by KR research on the ramification problem, we modified the search procedure of PO-PRP to compute the indirect effects of actions by applying invariant actions at every node in the search until quiescence. That is, until no more invariant actions are applicable. To avoid state explosion, we enforce an arbitrary ordering over the application of invariant actions. Given the existing restrictions for simple domains, enforcing the order does not alter the state that the planner reaches once there are no more applicable invariant actions. This is because there is a unique interpretation in these simple domains, and because knowledge is assumed to accumulate monotonically.

3.4 Exporting a Conditional Plan

Like PRP, PO-PRP produces a policy in the form of condition-action pairs, which, in principle, could be used if the belief of the agent is maintained in its compiled form. Since this cannot be guaranteed, we convert the policy into a form more traditionally used for PPOS solutions: a conditional plan. The policy quite often will be far smaller than the conditional plan it implicitly represents, but there is merit to producing one: it helps in verifying PO-PRP's final solution, and it provides a plan in a form suitable for execution by an agent that does not maintain its belief in a compiled form (e.g., a simple controller).

While conditional plans are typically tree structures, PO-PRP conditional plans are constructed more compactly as DAGs. Nodes in the DAG correspond to either actions drawn from $\mathcal{A}'_{\mathcal{A}}$, or decision points – sensing actions drawn from $\mathcal{A}'_{\mathcal{O}}$ whose outgoing edges denote the possible observations. There are three additional distinguished node types: a single source node denoting the first action of the plan, deadend nodes where no plan exists, and goal nodes.

To convert PO-PRP's policy P to a conditional plan, we use an approach that essentially simulates P on every possible action outcome. Whenever we see a *repeated* state, we need not continue expanding the conditional plan. Intuitively, the procedure is as follows:

1. Let $Open = \{s_0\}$, $Cls = \emptyset$, and $\langle N, E\rangle = \langle\{s_0\}, \emptyset\rangle$;
2. Select and move a state s from $Open$ to Cls such that,
   (i) Let $P(s) = \langle p, a\rangle$ and if $a \in \mathcal{A}'_{\mathcal{O}}$ let $Succ = \{Prog(s, a, \textsc{Eff}_1(a)), Prog(s, a, \textsc{Eff}_2(a))\}$ (otherwise let $Succ = \{Prog(s, a, \textsc{Eff}(a))\}$);
   (ii) Add $Succ \setminus Cls$ to N and $Open$;
   (iii) Add (s, s') to E for every s' in $Succ$;
3. If $Open$ is empty, return $\langle N, E\rangle$. Else repeat from 2.

We further modify the DAG $\langle N, E\rangle$ so that every state $s \in N$ is labelled with the action a from the pair $P(s) = \langle p, a\rangle$, and every exit edge from a state labelled with an action $a \in \mathcal{A}'_{\mathcal{O}}$ is labelled with the appropriate observation outcome. Additionally, we reduce the graph by merging any state labelled with an invariant action into its successor.

The similarity to PRP's general approach (described above) is not coincidental: both approaches operate by enumerating the reachable states of P. To avoid a full state explosion of reachable states we can appeal to PO-PRP's SCD procedure and create a more compact conditional plan. The key is to replace every state $Prog(s, a, \textsc{Eff}_i(a))$ in $Succ$ on line 2(i) with p whenever $P(Prog(s, a, \textsc{Eff}_i(a))) = \langle p, a\rangle$ is marked as strong cyclic. This means that the closed list Cls will contain both complete and partial states of the FOND encoding, $K'(\mathcal{P})$. The partial states p contain only the relevant information for a strong cyclic solution to exist with P, and this allows us to reuse parts of the conditional plan for all states s such that $P(s) = \langle p, a\rangle$. As an example, consider the CTP problem from earlier. The states computed on the modified line 2(i) would be partial states that retain the belief about future roads, but forgo knowledge about roads observed in the past. In this way, the agent can use the same conditional plan regardless of how it has travelled so far.

Theorem 1. Given a simple PPOS problem $\mathcal{P}$, the conditional plan produced by PO-PRP is sound.

This follows from Proposition 1 and the soundness and completeness of the BG translation from $\mathcal{P}$ to $K'(\mathcal{P})$; the soundness of the partial policy produced by PO-PRP with respect to the class of FOND problems represented by $K'(\mathcal{P})$; and by the correctness of the transformation of the partial policy with respect to $K'(\mathcal{P})$ into a compact conditional plan with respect to the original PPOS problem.

Theorem 2. Given a simple PPOS problem $\mathcal{P}$, PO-PRP will return a conditional plan if one exists.

This similarly follows from the soundness and completeness of the BG translation of $\mathcal{P}$ into a FOND problem, together with the property, inherited from PRP, that if a strong cyclic solution exists for $K'(\mathcal{P})$, then PO-PRP will compute it and export it as a conditional plan with respect to $\mathcal{P}$.

Problem	Time (seconds)			Size (actions + sensing)			PO-PRP Policy Size		
	CLG	PO-PRP$^-$	PO-PRP	CLG	PO-PRP$^-$	PO-PRP	all	strong	used
cballs-4-1	0.20	0.03	**0.02**	343	365	**261**	150	24	125
cballs-4-2	19.86	**0.6**	0.67	22354	18643	**13887**	812	46	507
cballs-4-3	1693.02	**87.03**	171.28	1247512	899442	**671988**	3753	81	1689
cballs-10-1	211.66	1.63	**1.57**	4829	5096	**4170**	1139	20	815
cballs-10-2	T	M	M	T	M	M	-	-	-
ctp-ch-1	**0.00**	**0.00**	**0.00**	5	5	**4**	10	10	9
ctp-ch-5	0.02	0.01	**0.00**	125	125	**16**	52	52	37
ctp-ch-10	2.2	0.08	**0.02**	4093	4093	**31**	131	127	72
ctp-ch-15	133.24	2.79	**0.07**	131069	131069	**46**	233	227	107
ctp-ch-20	T	M	**0.22**	T	M	**61**	361	352	142
doors-5	0.12	**0.00**	0.01	169	174	**82**	55	18	55
doors-7	3.50	**0.04**	**0.04**	2492	2603	**1295**	123	16	114
doors-9	187.60	**1.07**	**1.07**	50961	53942	**28442**	194	16	182
doors-11	T	M	M	T	M	M	-	-	-
wumpus-5	0.44	**0.14**	0.16	854	441	**233**	706	160	331
wumpus-7	9.28	**1.14**	1.54	7423	1428	**770**	2974	241	992
wumpus-10	1379.62	**7.56**	11.17	362615	4693	**2669**	8122	281	2482
wumpus-15	T	**51.06**	86.16	T	25775	**15628**	21342	304	8241
wumpus-20	T	M	M	T	M	M	-	-	-

Table 1: Comparing compilation time and conditional plan size for CLG and PO-PRP. Also listed are statistics on the size of policy PO-PRP generates. Bold represents the best performance, while T and M represent time limit or memory exceeded.

4 Evaluation

We compared PO-PRP with the state of the art in offline conditional planning, CLG (Albore, Palacios, and Geffner 2009), to assess both the efficiency of the solving process and succinctness of the generated plans. We measure the efficiency in terms of the time it takes to compute a complete solution, and the succinctness in terms of the generated conditional plan. All experiments were conducted on a Linux desktop with a 3.4GHz processor, with time / memory limits of 1hr / 2GB respectively. The times listed here do not include parsing, but this portion of the solving process never exceeded 3 seconds in the problems tested.[3]

We consider four domains that fall under the category of simple domains with partial observability and sensing actions: Coloured Balls (cballs), Canadian Traveller's Problem (ctp-ch), Doors (doors), and Wumpus World (wumpus). The cballs domain involves navigating a known maze to sense for coloured balls and placing them in the correct bin (there is uncertainty in the ball locations and the colour of the balls). The ctp-ch is the variant of the Canadian Traveller's Problem introduced earlier in the paper where the map consists of a chain of locations connected with a pair of roads (one of which is safe to travel on). The doors domain requires the agent to find the unknown position of a door in a long wall before moving on to another wall. Finally, the classic wumpus domain requires the agent to navigate through a maze of wumpus monsters and pits (which can be sensed in a neighbouring location) in order to retrieve a bag of gold. Aside from ctp-ch, we retrieved all problems from the benchmark set that is included with the online contingent planner of BG, *K*-Planner (Bonet and Geffner 2011).[4] All problems, example plans, and PO-PRP source is available online at,

`http://www.haz.ca/research/poprp/`

When measuring the size of the conditional plan, we additionally considered the size of the conditional plan that is produced when we disable the SCD procedure. In such cases, the conditional plan is almost always a tree, as every belief state reachable by the policy is distinct. We use PO-PRP$^-$ to designate the use of PO-PRP with SCD disabled when exporting the conditional plan.

To further assess the succinctness of the partial policy that PO-PRP generates, we include the size of the policy prior to computing the conditional plan. Recall that the policy PO-PRP produces is a mapping of partial belief states to an action – it thus has the potential to be far more compact than a traditional mapping from belief states to actions. Additionally, we list the number of pairs in the policy that were marked as strong cyclic as well as the number of pairs that were used during the construction of the conditional plan.

Table 1 shows the results for a selection of problems from the four domains considered here. The sizes reported for CLG differ from the original published results, as we include the number of branching points (i.e., sensing actions) in our

[3]The conversion to SAS$^+$ rarely created multi-valued variables; so we restricted the invariant synthesis of the parser to 3 seconds.

[4]`https://code.google.com/p/cp2fsc-and-replanner/`

calculation. Invariant actions are suppressed from the conditional plan for both CLG and PO-PRP, and they do not appear in the totals. For the 'Policy Size' column, **all** refers to the full policy size (i.e., the number of condition-action pairs), **strong** refers to the number of pairs marked strong cyclic, and **used** refers to those pairs that were used in generating the conditional plan with PO-PRP. The **used** column is almost identical for PO-PRP$^-$, and is thus omitted.

We found that PO-PRP consistently outperformed CLG in both time to compute a solution and in the size of the final conditional plan. For some domains the size is comparable (e.g., doors and cballs), but the runtime shows a dramatic improvement. This is due in large part to the generalized policy that PRP produces internally, which is used to enumerate the reachable belief states. CLG, in contrast, must continually recompute a weak plan for every new belief state.

The improvement in conditional plan size for PO-PRP over CLG is promising, and it is even more striking when we consider the representation size of the policy. The policy is not smaller than the conditional plan in every case (see for example ctp-ch and wumpus domains), but the policy can be smaller than the conditional plan produced by orders of magnitude. The conditional plan is much more succinct when many of the pairs in the policy are marked as strong cyclic because of the approach we use to compile the conditional plan (cf. Section 3.4). This is most evident in the ctp-ch domain where PO-PRP is able to compute the optimal conditional plan very quickly.

We conclude by noting the potential for further improvement. The SCD procedure of PRP, and subsequently PO-PRP, is merely a sufficient condition. With stronger techniques to mark more of the policy as strong cyclic, we expect the conditional plans produced to be more compact. Additionally, the difference between the **all** and **used** columns gives an indication of the amount of redundant or unnecessary information in the policy. This suggests room for improvement in the final policy that PO-PRP produces.

5 Concluding Remarks

Planning under partially observability is a computationally challenging problem applicable to a number of AI endeavours. We focused on solving a compelling class of PPOS problems where the initial state specification includes a set of state constraints and where uncertainty about the state monotonically decreases. We demonstrated how PPOS problems can be solved offline by exploiting and extending a modern FOND planner. In contrast to the commonly held belief that offline planning for PPOS is impractical, PO-PRP can produce conditional plans several orders of magnitude faster and smaller than the best planners. In our view, offline planners come with significant advantages, such as deadend detection, which is critical in certain settings. Our contribution includes novel techniques for solving PPOS problems and producing a compact plan; it opens the door to a new variety of offline planning techniques.

There are many avenues for future work, including generalizing the framework to effectively handle a larger class of problems, and characterizing problems that can be given succinct policy representations (*e.g.* the ctp-ch domain).

Acknowledgements

We gratefully acknowledge funding from the Ontario Ministry of Innovation and the Natural Sciences and Engineering Research Council of Canada. We also would like to thank the anonymous reviewers for their insightful feedback, and Blai Bonet for making the source of *K*-Planner both publicly available and straightforward to use / extend.

References

Albore, A., and Geffner, H. 2009. Acting in partially observable environments when achievement of the goal cannot be guaranteed. In *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*.

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *21st International Joint Conference on Artificial Intelligence (IJCAI)*, 1623–1628.

Baral, C.; Kreinovich, V.; and Trejo, R. 2000. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence* 122(1-2):241–267.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *17th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2001, 473–478.

Bonet, B., and Geffner, H. 2011. Planning under partial observability by classical replanning: Theory and experiments. In *22nd International Joint Conference on Artificial Intelligence (IJCAI)*, 1936–1941.

Brafman, R. I., and Shani, G. 2012. Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research* 45:565–600.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research* 26:35–99.

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.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan and Claypool Publishers.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173(5-6):503–535.

Hoffmann, J., and Brafman, R. I. 2005. Contingent planning via heuristic forward search witn implicit belief states. In *15th International Conference on Automated Planning and Scheduling (ICAPS)*, 71–80.

Kuter, U.; Nau, D.; Reisner, E.; and Goldman, R. 2007. Conditionalization: Adapting forward-chaining planners to partially observable environments. In *ICAPS 2007—workshop on planning and execution for real-world systems*.

McIlraith, S., and Scherl, R. B. 2000. What sensing tells us: Towards a formal theory of testing for dynamical systems. In

17th National Conference on Artificial Intelligence(AAAI), 483–490.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-Deterministic Planning by Exploiting State Relevance. In *22nd International Conference on Automated Planning and Scheduling (ICAPS)*, The 22nd International Conference on Automated Planning and Scheduling.

Muise, C.; McIlraith, S.; and Belle, V. 2014. Non-deterministic planning with conditional effects. In *24th International Conference on Automated Planning and Scheduling*.

Peot, M., and Smith, D. 1992. Conditional nonlinear planning. In *International Conference on AI Planning and Scheduling (AIPS)*, 189–197.

Pinto, J. 1999. Compiling ramification constraints into effect axioms. *Computational Intelligence* 15:280–307.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research* 4:287–339.

Reiter, R. 2001. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. The MIT Press.

Rintanen, J. 2004. Complexity of planning with partial observability. In *14th International Conference on Automated Planning and Scheduling (ICAPS)*, 345–354.

Strass, H., and Thielscher, M. 2013. A general first-order solution to the ramification problem with cycles. *Journal of Applied Logic* 11(3):289–308.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty and sensing actions. In *15th National Conference on Artificial intelligence (AAAI)*, 897–904.

Diagnostic Problem Solving via Planning with Ontic and Epistemic Goals (Abridged Report)*

Jorge A. Baier
Depto. de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

Brent Mombourquette
Department of Computer Science
University of Toronto
Toronto, Canada

Sheila A. McIlraith
Department of Computer Science
University of Toronto
Toronto, Canada

Abstract

Diagnostic problem solving involves a myriad of reasoning tasks associated with the determination of diagnoses, the generation and execution of tests to discriminate diagnoses, and the determination and execution of actions to alleviate symptoms and/or their root causes. Fundamental to diagnostic problem solving is the need to reason about action and change. In this work we explore these myriad of reasoning tasks through the lens of AI automated planning. We characterize a diversity of reasoning tasks associated with diagnostic problem solving, prove properties of these characterizations, and define correspondences with established automated planning tasks and existing state-of-the-art planning systems. In doing so, we characterize a class of planning tasks with epistemic and ontic goals which we show can be compiled into non-epistemic planning, allowing state-of-the-art planners to compute plans for such tasks. Furthermore, we explore the effectiveness of using the conditional planner Contingent-FF with a number of diagnostic planning tasks.

1 Introduction

Automated diagnosis seeks to determine what is wrong with a system, prompted by some observations of egregious behaviour. As our world becomes increasingly instrumented with sensors – our street corners, our homes, our cars, and even our bodies – and as the infrastructure that controls our power, communication, and transportation systems grows in complexity, we must rely on computers to monitor the state of these systems and to oversee their operation. Unfortunately, these systems can and do malfunction, resulting in diagnostic problems of such enormous complexity that they confound human reasoning.

Diagnostic Problem Solving (DPS) refers to the myriad of reasoning tasks associated with the diagnosis, testing, and repair of a system. In this work we advocate for a *purposeful view of diagnostic problem solving*. While a naïve approach to DPS suggests that we generate candidate diagnoses, identify a unique diagnosis through testing, and then treat or repair, we instead observe that identifying candidate diagnoses may be unnecessary or perhaps only necessary to the extent that it informs an appropriate course of action to be selected – a course of action that may result in the realization of further tests to discriminate diagnoses, or to alleviate the symptoms or potential root causes, potentially without actually identifying a unique root cause.

*Full paper appears in the Proceedings of KR2014.

Example 1 *Consider a run-of-the-mill flashlight that is not emitting light. A common response is to turn the flashlight on and off a few times. If it's still malfunctioning, the most likely hypothesis is that the batteries are dead, but it could also be the case that the bulb is burned out, or that there is a loose connection somewhere. That's three candidate diagnoses, and there could be more. A typical course of action would be to open up the flashlight, take out the batteries, put in new ones, re-assemble the flashlight and turn it on again. If the flashlight emits light, you'll likely be happy, recycle the batteries and consider yourself "done." Your purpose was* not *to diagnose the flashlight, but rather to get it working again. A more careful examination of what went on shows that the course of action you took, together with the happy outcome that the flashlight is now "working," served to eliminate the hypothesis that the bulb was burned out, and it tightened the connection, effectively* repairing *the connection regardless of whether it was faulty or not. This changed the space of hypotheses under consideration. What is equally interesting is that this sequence of actions neither confirmed nor refuted the hypothesis that the batteries were dead. The cause of the faulty behaviour could have been the result of a loose connection which got fixed in the process of changing the batteries. Those batteries you recycled could still be OK! What's also noteworthy is that if after executing the procedure the flashlight had not emitted light, you would still be left with the hypotheses that the original batteries were dead or that the bulb was broken, but you would also have the further (granted, unlikely) hypothesis that the new batteries were also dead.*

The above seemingly simple DPS scenario illustrates the need for reasoning about action and change as well as reasoning about knowledge, and in particular what an agent comes to know about aspects of the world (symptoms and diagnoses in this case) based upon the execution of both world-altering and sensing actions. It is also suggestive of the somewhat subordinate role the actual candidate diagnoses may play in the resolution of a system failure scenario.

In this paper, we explore this purposeful view of diagnostic problem solving through the lens of AI automated planning. Our motivation for doing so is pragmatic. We are interested in characterizing the fundamental knowledge representation and reasoning tasks that underlie DPS in its many guises, but we wish to do so in a manner that supports establishing its correspondence with the state of the art in AI automated planning theory and practice. There have

been tremendous advances in AI automated planning systems over the last decade and a number of highly optimized planning systems exists. Further there have been significant recent advances in non-classical planning such as conformant or contingent planning that support planning with incomplete information and/or with sensing. A major contribution of this paper is to show that many purposeful DPS tasks that have been heretofore unsolvable can be realized in varying measure through recent advances in state-of-the-art AI planning systems. Our characterization not only allows us to solve certain problems now, but it provides the insight and understanding that will support the realization of these and isomorphic tasks as computational machinery in planning improves in the coming years.

Characterizing diagnostic problem solving tasks in terms of planning builds upon a large body of research that includes research on topics as varied as reasoning about knowledge and action (e.g., Scherl and Levesque 2003; Petrick and Bacchus 2002), planning with epistemic goals (e.g., Herzig, Lang, and Marquis 2003), and conformant (e.g., Palacios and Geffner 2009), contingent (e.g., To, Pontelli, and Son 2011), and classical (e.g., Helmert 2006) planning. We discuss this related research later in the paper.

In Section 2 we introduce the mathematical formalisms upon which our work rests. In Section 3 we introduce the notion of a diagnosis that we use here, and contrast it to other forms of diagnosis discussed in the literature. With a definition of diagnosis in hand, in Section 4 we present the notion of a diagnostic plan, establish its correspondence to known planning paradigms, and establish properties of various forms of diagnostic plans. In Section 5 we introduce a notion of epistemic diagnostic planning – planning to determine a particular diagnosis, or to discriminate diagnoses. We define compelling classes of epistemic goals and then show that it is possible to use state-of-the-art non-epistemic planners to plan for such epistemic goals, by providing a sound and complete translation of such epistemic tasks to conditional planning. In Section 6 we discuss the realization of our newly established diagnostic planning tasks via existing planning systems. We conclude with some reflections on our work, its relationship to other work and future prospects.

2 Preliminaries

In this section we introduce a planning language that will allow us to define various kinds of planning tasks which we then will show can model interesting DPS tasks. In particular, we define a common language for deterministic (classical), conformant, and conditional planning.[1] The planning language we present below builds on the ADL planning language (Pednault 1989), and considers extensions for uncertainty about the initial state and conditional plans that have been presented in a similar way by other researchers (e.g., Palacios and Geffner 2009; To, Pontelli, and Son 2011).

2.1 Dynamical Systems

Dynamical systems can be formally described in many ways. In this paper we model them as transition systems, which we represent with a standard planning language. As such, transitions occur as the result of actions described in terms of preconditions and effects, and the domain is finite (i.e., there is a finite number of system configurations). Formally, a dynamical system is a tuple $\Sigma = (F, A, \Omega, I)$, where F is a finite set of fluent symbols, A is a set of deterministic actions, Ω is a set of sensing actions, and I is a set of clauses over F that defines a set of possible initial states. If $p \in F$, then p and $\neg p$ are *fluent literals*. If ℓ is a literal, we denote its complement by $\overline{\ell}$; thus, $\overline{p} = \neg p$ and $\overline{\neg p} = p$. Every action $a \in A$ is defined by a precondition $prec(a)$, which is a conjunction of fluent literals, and $\mathit{eff}(a)$, a set of conditional effects of the form $C \rightarrow L$, where C is a conjunction of fluent literals and L is a fluent literal. We sometimes write the unconditional effect $\rightarrow L$ as simply L, and use $true$ to denote an empty precondition. Each sensing action, on the other hand, is defined by its precondition $prec(a)$, which is a conjunction of fluent literals, and $obs(a)$, which is the fluent literal that is observed by the sensing action.

A *system state* s is a set of fluent symbols, which intuitively defines all that is true in a particular state of the dynamical system. For a system state s, we define $M_s : F \rightarrow \{true, false\}$ as the truth assignment that assigns the truth value $true$ to p if $p \in s$, and assigns $false$ to p otherwise. We say a state s is *consistent* with a set of clauses $\mathcal{C}$, if $M_s \models c$, for every $c \in \mathcal{C}$.

We denote by S_0 the set of planning states consistent with the clauses of the initial state I. We say a dynamical system has a *complete initial state* iff $|S_0| = 1$; i.e., I has only one model. Σ has an *incomplete initial state* iff $|S_0| > 1$. We formalize the notion of conditional plans through *action trees* as follows.

Definition 1 (Action Tree) *Given a system* $\Sigma = (F, A, \Omega, I)$*, an action tree* T *is:*

- ϵ *(the empty tree); or*
- aT'*, where* $a \in A$*, and* T' *is an action tree; or*
- $a(T', T'')$*, where* $a \in \Omega$ *and* T' *and* T'' *are action trees.*

We denote by $\mathcal{T}_\Sigma$ the set of action trees in Σ. Furthermore, we say that an action a is *executable* in a state s if $M_s \models prec(a)$. If $a \in A$ is executable in a state s, we define its successor state as $\delta(a, s) = (s \setminus Del) \cup Add$, where Add contains a fluent f iff $C \rightarrow f$ is an effect of a and $M_s \models C$. On the other hand Del contains a fluent f iff $C \rightarrow \neg f$ is an effect of a, and $M_s \models C$.

Now we define how to compute the set of states that result from executing an action tree. To that end, we define the relation $\vdash_\Sigma$: $\mathcal{T}_\Sigma \times 2^F$ such that $(T, S) \vdash_\Sigma (T', S')$ intuitively means that if the agent is in any of the states in S, then performing one step of T results in being in some state in S', with action tree T' remaining. Formally,

- $(aT, S) \vdash_\Sigma (T, S')$ if a is executable in every state in S, and $S' = \{\delta(a, s) \mid s \in S\}$.

[1] A paradigm related to conditional planning is *contingent planning*. Although originally understood to define the same class of problems as conditional planning, current research in contingent planning frequently sees it as an incremental process in which planning is intertwined with execution (see e.g., Brafman and Shani 2012). We thus stick here to conditional planning to emphasize that we are looking for a conditional plan in an offline manner.

- $(a(T', T), S) \vdash_\Sigma (T, S')$ if a is executable in every state in S and $S' = \{s \in S \mid M_s \models obs(a)\}$ or $S' = \{s \in S \mid M_s \not\models obs(a)\}$.

We use $\vdash$ instead of $\vdash_\Sigma$ when the transition system is obvious from the context. We denote by $\vdash^*$ the reflexive and transitive closure of $\vdash$. This allows us to define what states result from executing an action tree in the initial state.

Definition 2 (Resulting State) *Given a transition system Σ, state s is a resulting state from executing action tree T in Σ iff $(T, S_0) \vdash^* (\epsilon, S')$ and $s \in S'$.*

2.2 Classical, Conformant, and Conditional Planning

Below we define classes of planning tasks that have been studied extensively by the planning community. Deterministic (classical) planning is the most standard form of planning in which there is a unique initial state, and hence knowledge about the state of the world is always complete. In conformant planning, on the other hand, there is uncertainty but no observability, and actions that change the state of the world are deterministic; hence solutions to these tasks are sequences of actions. Finally, in conditional planning,[2] there is uncertainty about the initial state and the agent may observe the world through sensing actions. The resulting plan in this case is typically an action tree.

Definition 3 (Classes of Planning Tasks) *Given a set of literals G, and a system $\Sigma = (F, A, \Omega, I)$ we define the following classes of planning tasks:*

- *(Σ, G) is a* deterministic *or* classical *planning task if I defines a complete initial state and $\Omega = \emptyset$.*
- *(Σ, G) is a* conformant *planning task if I does not define a complete initial state and $\Omega = \emptyset$.*
- *(Σ, G) is a* conditional *planning task if I does not define a complete initial state and $\Omega \neq \emptyset$.*

Definition 4 (Plan) *Action tree T is a plan for planning task (Σ, G) iff for every state s_f that may result from the execution of T in Σ it holds that $M_{s_f} \models G$.*

When T is a plan for a deterministic task, then we say T is a *deterministic plan*. Analogously, we use the terms *classical plan*, *conformant plan*, and *conditional plan*.

3 Characterizing Diagnoses

Automated diagnosis has long been a problem of interest to the AI community. Well-publicized early work focused on expert systems approaches as exploited in the medical diagnosis expert systems MYCIN. In the mid-1980's AI researchers turned their attention to model-based diagnosis. Early work in this area included Geneserth's DART system (Genesereth 1984), as well as GDE, the General Diagnosis Engine by de Kleer and Williams (de Kleer and Williams 1987) among others. In 1987 Reiter published his seminal paper formalizing the notion of *consistency-based diagnosis* and *minimal diagnosis*, which were predicated on a so-called first principles model of the normative behaviour of a system (Reiter 1987). This characterization defined a diagnosis to be a minimal set of components that must be designated as *abnormal* in order for observations of system behaviour to be logically consistent with the model (axiomatization) of the system. The exploitation of fault models and other aspects of system behaviour and function necessitated a more stringent characterization of diagnosis in terms of *abduction* (e.g., (Console and Torasso 2006), (Poole 1994)) in which faults or other explanations were posited in order to *entail* or otherwise explain observations. Such abductive diagnoses were originally conceived to work with fault models, rather than normative models, in order to entail faulty behaviour. de Kleer, Mackworth, and Reiter published a follow-on to Reiter's 1987 paper in 1992 that combined aspects of abductive and consistency based diagnosis into kernel diagnoses (de Kleer, Mackworth, and Reiter 1992). All of these characterizations of diagnosis related to static systems, described using a triple $(SD, COMPS, OBS)$ – the system description, a finite set of components to be diagnosed, and an observation. There was no notion of system dynamics – just a single state.

The systems we are concerned with in this paper include not only these static descriptions but also a rich theory of action that supports diagnostic planning in its many guises. We begin with a definition of such a diagnostic system.

- SD, the system description, a set of propositional sentences;
- $COMPS$, the components, a finite set of constants;
- OBS, the observation, a conjunction of ground literals:
- $\Sigma = (F, A, \Omega, I)$, a dynamical system that describes actions relevant to diagnostic problem solving tasks associated with the system described by SD.

We illustrate these with a simple example.

Example. Consider a flashlight, comprised of a battery and a switch. If the switch is on and both the battery and switch are operating normally, then light will be emitted. This fact can be described by the following logical formula, which is included in the system's description SD:

$$on \wedge \neg AB(battery) \wedge \neg AB(switch) \supset light.$$

The set of system components is simply defined by $COMPS = \{battery, switch\}$.

Now we assume the flashlight can be operated by a human user. The available actions are: turn on the switch, change the battery, and fix the switch. In addition, we have the ability to observe whether or not there is light in the room. The dynamics of these actions is described using a transition system Σ, where $F = \{on, AB(switch), AB(battery), light\}$, $A = \{turn\text{-}on, change\text{-}battery, fix\text{-}switch\}$, and $\Omega = \{sense\text{-}light\}$. The effects of the actions are described using our planning language as shown by Table 1. §

In general, the action theory described by Σ supports reasoning about actions to test, repair, or eradicate egregious

[2] Early literature in conditional planning (e.g. Pryor and Collins 1996) assumed complete observability of the world. However, current state-of-the-art solvers consider sensing actions as the only mechanism to observe the world (e.g. To, Pontelli, and Son 2011).

a	$prec(a)$	Effect/Observation
$turn\text{-}on$	$\neg on$	on
$change\text{-}battery$	$true$	$\neg AB(battery)$
$fix\text{-}switch$	$true$	$\neg AB(switch)$
$sense\text{-}light$	$true$	$light$

Table 1: Preconditions and direct effects of the actions in our flashlight example.

a	New Effect
$turn\text{-}on$	$\neg AB(battery) \wedge \neg AB(switch) \rightarrow light$
$change\text{-}battery$	$on \wedge \neg AB(switch) \rightarrow light$
$fix\text{-}switch$	$on \wedge \neg AB(battery) \rightarrow light$

Table 2: Additional effects of actions in our example theory.

behaviour in a system. On the other hand, the system description describes complex interactions between the components of the system.

Our notion of *diagnostic system*, which we define formally below, intuitively integrates the system description SD with the dynamics of the world described by Σ. In doing so, we need to address the so-called *ramification problem* – the problem of characterizing the indirect effects of actions. To see why this is, in our example, observe that performing the action $turn\text{-}on$ may have the indirect effect of $light$, under the condition $\neg AB(battery) \wedge \neg AB(switch)$.

The ramification problem is a well-studied problem in the KR community (e.g., Lin 1995; McCain and Turner 1995; Thielscher 1995; Sandewall 1996; Pinto 1999; Strass and Thielscher 2013) and has also been previously studied in the specific context of DPS (e.g., McIlraith 2000; McIlraith and Scherl 2000). For the purposes of this paper, we adopt an existing solution to the ramification problem that compiles the indirect effects of actions into additional direct effects of actions proposed both by Pinto (1999) and in variation most recently by Strass and Thielscher (2013). The solution is predicated on augmentation of the constraints, SD, that additionally captures the causal relationship between fluents. These causal relationships have historically been captured via an explicit causal relation in the constraints (e.g., Lin 1995; McCain and Turner 1995) or by the augmentation of SD with a causal graph structure that ranges over the literals in SD (e.g., McIlraith 2000).

Example (continued). In our flashlight example, we consider the following ramification constraint, which is extracted directly from SD.

$$on \wedge \neg AB(battery) \wedge \neg AB(switch) \textbf{ causes } light.$$

From there, we use Pinto's algorithm (1999) to compute additional effects for the actions in our theory. Some of the resulting effects are shown in Table 2. §

Now we provide a formal definition for a diagnostic system.

Definition 5 (Diagnostic System) *Given SD, $COMPS$, OBS, and $\Sigma = (F, A, \Omega, I)$, as defined above, a diagnostic system Σ_{SD} is a tuple (F', A', Ω, I') where:*

- *F' contains the elements in F and in $Vars(SD)$, the propositional variables in SD, and ground fluents of the form $AB(c)$ for every $c \in COMPS$;*
- *A' contain the same actions in A but augmented, following Pinto (1999), with conditional effects to address the ramification problem that emerges from the integration of Σ and SD;*
- *$I' = I \cup SD \cup OBS$.*

Note that the initial state I' contains both the system's description and the observation. In our example, we could have that $I = \{on\}$ and that $OBS = \{\neg light\}$. By including SD in I' we enforce that states consistent with I' have at least one abnormal component. Note that an alternative means of characterizing a diagnostic system is to treat ramifications as further actions that are triggered by direct effects.

We now formally define the notion of diagnosis, which we borrow from de Kleer, Mackworth, and Reiter (1992)'s, in which we posit a minimal subset of components that must be behaving abnormally in order to account for the observation in the initial state of our dynamical system. To facilitate explication, we appeal to a characterization of diagnosis in terms of abnormal components; however the work in this paper is applicable to a diversity of definitions of diagnoses or hypotheses and need not rely on the use of distinguished components. The generalization is straightforward. Recall that I', of Σ_{SD}, includes our observation, OBS of (potentially egregious) behaviour.

Definition 6 (Diagnosis) *Given a diagnostic system Σ_{SD}, $\Delta \subseteq COMPS$ is a diagnosis iff*

$$I' \cup \bigcup_{c \in \Delta} AB(c) \cup \bigcup_{c' \in COMPS \setminus \Delta} \neg AB(c')$$

is satisfiable.

Definition 7 (Minimal Diagnosis) *Δ is a minimal diagnosis of Σ_{SD} if Δ is a diagnosis and no other proper subset Δ' of Δ is a diagnosis.*

In previous work, we examined diagnosis of dynamical systems with respect to a narrative of observations that evolved over a period of time. We characterized the notion of an *explanatory diagnosis* in the situation calculus (Sohrabi, Baier, and McIlraith 2010; McIlraith 1998) and relatedly the notion of an *explanation* in a planning-inspired propositional language (Sohrabi, Baier, and McIlraith 2011). These definitions of diagnosis and explanation conjecture a set of zero or more assumptions together with a sequence of actions to account for the observations. These definitions of diagnosis and explanation are not at odds with what is proposed here. Indeed, a diagnosis, Δ would hold in the final state of an explanatory diagnosis, with OBS as the final observation of the narrative of observations used to construct the explanatory diagnosis. The generation of explanatory diagnosis looks back in time to conjecture what happened based on past observations. Observations going forward are integrated with the diagnostic plans.

4 Diagnostic Planning

While automated diagnosis remains a well-studied area of research, we argue for a purposeful view of diagnosis. In

particular, rather than generating candidate diagnoses and performing tests to identify a unique diagnosis, after which a course of action (amelioration/treatment) is determined, we argue that the determination of a unique diagnosis is generally not an end in itself and that a pragmatic view of DPS should focus on acting.

One argument in support of this viewpoint is that in many settings there are a limited number of courses of action that can achieve a particular goal. These actions/plans induce an equivalence class of diagnoses for which a particular course of action is relevant. For example, many families of bacterial infection require treatment with the same antibiotic and we need not completely discriminate the nature and location of the infection before treating. Similarly, if a photocopier is malfunctioning, despite the ability to perform an in-depth diagnosis, the first course of action is often to turn the machine off and on, resolving a large family of faults simultaneously without the need for time-consuming, and time-wasting in this case, differential diagnosis.

All of the diagnostic planning tasks we examine take the same general form. The dynamical model of the system, Σ_{SD}, is augmented with additional information about the initial state, the goal state and some constraints that need to be enforced throughout execution of the plan. In this section, we provide a general formulation of a diagnostic plan. We discuss the diversity of diagnostic tasks that can be achieved via the specification of different initial conditions, goals, and constraints, and the complexity of plan existence in some of these different settings. While the types of plans we are interested in here change the state of the world, we are also interested in plans that are designed to change our state of knowledge without necessarily changing (much of) the world. In the section that follows, we look at epistemic goals – planning to know whether or not a diagnosis is true, to refute a diagnosis, or to discriminate a collection of candidate diagnoses.

Definition 8 (Diagnostic Plan) *Given a diagnostic planning task $(\Sigma_{SD}, Init, \Phi, G)$ where*

- *$\Sigma_{SD} = (F, A, \Omega, I)$, is the diagnostic system;*
- *$Init$, is a set of logical formulae that provide additional information about the initial state;*
- *Φ, is a logical formula representing additional state constraints that must be enforced throughout the plan; and*
- *G, is a set of literals that prescribe the diagnostic planning goal,*

and where $Init \cup I \cup \Phi$ is satisfiable.

Action tree T is a diagnostic plan for the diagnostic planning task $((F, A, \Omega, I \cup Init), G)$ under the constraint of Φ iff T is a plan for the planning task $((F, A, \Omega, I \cup Init), G)$, and for every S such that $(T, S_0) \vdash^* (T', S)$ it holds that $M_s \models \Phi$, for every $s \in S$.

4.1 Properties of Diagnostic Plans

As evident from the definition presented, our characterization of a diagnostic plan is coupled to our previously defined classes of planning tasks. These classes differ with respect to the completeness of their initial states and whether they exploit sensing of any form. Such characteristics have implications with respect to the complexity of planning.

The following are the complexity classes of the decision problem for the different planning tasks.

Theorem 1 (Diagnostic Planning w/ Complete Info.) *Given a diagnostic planning problem $(\Sigma_{SD}, Init, \Phi, G)$, if $I \cup Init \cup \Phi$ defines a complete initial state and if $\Omega = \emptyset$ then this is a* classical *planning task and deciding whether there exists such a plan is PSPACE-complete.*

This result follows from Bylander's result on the complexity of deterministic planning (Bylander 1994).

Theorem 2 (Diagnostic Planning without Sensing) *Given a diagnostic planning problem $(\Sigma_{SD}, Init, \Phi, G)$, if $\Omega = \emptyset$ but $I \cup Init \cup \Phi$ does not define a complete initial state then this is a* conformant *planning problem and deciding whether there exists such a plan is EXPSPACE-complete.*

This follows from Haslum's result on the complexity of conformant planning (Haslum and Jonsson 1999). Finally,

Theorem 3 (Diagnostic Planning with Sensing) *Given a diagnostic planning problem $(\Sigma_{SD}, Init, \Phi, G)$, if $I \cup Init \cup \Phi$ does not define a complete initial state and $\Omega \neq \emptyset$ then this is a* conditional *planning problem and deciding whether there exists such a plan is 2-EXPTIME-complete.*

This follows from Rintanen's results on the complexity of conditional planning (Rintanen 2004).

4.2 The Many Guises of Diagnostic Planning

Our characterization of a diagnostic plan encompasses a diversity of DPS scenarios. Here we informally explore some of these varied scenarios in order to illustrate the broad utility of our characterization. Each scenario is realized by varying the $Init$ and G and can be performed with or without sensing as the scenario necessitates, and with commensurate implications regarding the type of planner.

Eradicate egregious behaviour By setting $Init = \emptyset$ and $G = \neg OBS$ we can plan to eradicate behaviour without first explicitly computing a diagnosis. If sensing is permitted, sufficient information will be garnered to select an appropriate course of action. Such information may or may not entail a unique diagnosis as illustrated in some of the previous examples. Without sensing, a conformant plan will be generated (where possible) that will work regardless of what is wrong with the system.

Fix the system, given a diagnosis Given a diagnosis Δ, $Init = \{AB(c) \mid c \in \Delta\} \cup \{\neg AB(c) \mid c \in COMPS \setminus \Delta\}$ and $G = \bigwedge_{c \in \Delta} \neg AB(c)$. When a unique diagnosis has been determined, this can be added to the initial state and a plan generated accordingly. In such a case I may transform or be transformable into a classical planning problem, greatly diminishing the computational effort involved in generating a plan. $Init$ may also be used to capture the set of candidate diagnoses. (E.g., *The car won't start because it's either out of gas or the battery is dead.*) Even in a scenario such as this one, the best plan may require no sensing since this candidate diagnosis set may dictate the same "fix" – call for road-side assistance.

Assume a diagnosis and fix it or eradicate behaviour We can also use the planner to do what-if planning. Whereas the previous scenario used $Init$ to add facts to the initial state, it is often compelling to *assume* a particular likely diagnosis and generate a plan that will work predicated on the assumption being correct. (E.g., *Assume the battery is dead and fix it.*) A subset of candidate diagnoses may similarly be assumed.

Discriminate between different diagnoses Given two diagnoses Δ_1 and Δ_2, return a plan to determine which one of Δ_1 or Δ_2 may be causing the egregious behaviour. We require a notion of *epistemic goals*, which we define below, to formalize this as a planning problem.

Our definition of diagnostic plan also supports the enforcement of (safety) constraints. There are a number of compelling uses for such constraints, particularly in cases where a diagnosis is being assumed. In such a scenario, the user may wish to prescribe avoidance of things that would be fatal should the assumptions be flawed (e.g., giving penicillin to a patient with allergies). Constraints of this form can often be compiled away so as not to add to the difficulty of planning. Finally, the user can exercise flexibility in eliminating sensing actions to generate conformant rather than contingent plans, or to enforce Φ by eliminating the actions that would result in its violation.

5 Epistemic Diagnostic Planning

In the previous section we examined tasks to achieve some state of the world. Here we wish to generate plans to achieve epistemic goals. For example, we may wish to generate a plan to *know* a particular diagnosis (*E.g., I know Ralph has meningitis.*), to discriminate between diagnoses (*E.g., I know Ralph has one and only one of meningitis, strep-A, or influenza.*), or to eliminate a diagnosis (*E.g, I've eliminated the possibility that Ralph has strep-A.*).

The notion of planning and reasoning to achieve a state of knowledge dates back to work by Moore. Scherl and Levesque (2003) later integrated Moore's approach with Reiter's solution to the frame problem (2001). More recently, epistemic planning has been discussed in the context of dynamic epistemic logic (e.g, (Herzig, Lang, and Marquis 2003; Andersen, Bolander, and Jensen 2012).

Currently, there are no competitive planning systems that implement the possible world semantics. There are a few systems that however implement the idea of *knowledge-level planning* in which the knowledge of the agent is explicitly represented as propositions of the language (e.g., using a fluent KF to represent that the agent knows F). These planners are capable of carrying out a limited, but still reasonably expressive form of reasoning about knowledge. A few systems like PKS (Petrick and Bacchus 2002) adopt the approach of *planning at the knowledge level* to implement these ideas. Planning at the knowledge level may achieve good performance in many planning benchmarks however, their ability to reason about knowledge is limited. Along the same lines, Palacios and Geffner (2009) proposed a compilation technique that maps contingent planning into knowledge-level deterministic tasks. As expected, their translation is compact, sound, and complete for only a certain class of problems (Palacios and Geffner 2009).

While many simple epistemic diagnostic planning tasks can indeed be mapped into knowledge-level planning domains we propose an alternative compilation into conditional planning with sensing actions. Our motivation is practical since this enables computing diagnoses with a variety of existing planning systems.

Our translation still takes a somewhat impoverished view of the world, choosing not to appeal to rich modal theories of knowledge and belief in favour of adopting the stance of planning at the so-called *belief level*. In this view, the state of the system captures the agent's beliefs about the world. Specifically, in the initial state, we assume that the agent knows any formula ϕ that is such that for every s consistent with I it holds that $M_s \models \phi$. Similarly, when the agent performs actions, the agent knows all formulae that hold in all states that it could reach. Formally, given a set of states S, and a formula ϕ, we say that:

$$K(\phi, S) \text{ iff } M_s \models \phi \text{ for each } s \in S,$$

where the intuitive meaning of $K(\phi, S)$ is that the agent knows ϕ when the set of states it is possibly in is S.

With this definition in hand we are ready to define our notion of planning with epistemic goals.

Definition 9 (Epistemic Plan) *Let $\Sigma = (F, A, \Omega, I)$ be a transition system and S_0 be the set of all plan states consistent with I. Then,*

- *T is a plan for $\mathsf{Know}(\phi)$ iff for every S such that $(T, S_0) \vdash^* (\epsilon, S)$ it holds that $K(\phi, S)$,*
- *T is a plan for $\mathsf{KnowWhether}(\phi)$ iff for every S such that $(T, S_0) \vdash^* (\epsilon, S)$ it holds either $K(\phi, S)$ or $K(\neg\phi, S)$, and*
- *T is a plan for $\mathsf{Discriminate}(\phi, \psi)$ iff for every S such that $(T, S_0) \vdash^* (\epsilon, S)$ it holds either $K(\phi \wedge \neg\psi, S)$ or $K(\neg\phi \wedge \psi, S)$.*

Under this definition it is simple to prove that T is a plan for ϕ if and only if T it is an epistemic plan for $\mathsf{Know}(\phi)$. This is compatible with our notion of planning at the belief level: the knowledge of the agent is captured by the set of states the agent is at. This also means that if one wants a plan for $\mathsf{Know}(\phi)$, then one can obtain such a plan by querying a regular conditional planner for the goal ϕ. However, it is not immediately straightforward how to obtain plans for the other types of epistemic goals using a conditional planner. For example, using the non-epistemic goal $\phi \vee \neg\phi$ instead of $\mathsf{KnowWhether}(\phi)$ would not work since $\phi \vee \neg\phi$ is tautological and therefore achieved by every action tree.

Interestingly, it *is* possible to treat all types of epistemic goals as ontic planning. Below we propose a simple compilation which maps our notion of planning with epistemic goals to ontic conditional planning.

5.1 From Epistemic Planning to Conditional Planning

Given an epistemic goal of the form $\mathsf{KnowWhether}(\phi)$, the main idea underlying this compilation is to add an additional fluent kw-ϕ that can be used to replace the epistemic goal $\mathsf{KnowWhether}(\phi)$.

To simplify the presentation we will assume, for now, that the goal is $\mathsf{KnowWhether}(L)$, where L is a literal, and that as usual $\Sigma = (F, A, \Omega, I)$ is the transition system. The compiled planning problem is $\Sigma' = (F', A', \Omega, I)$, and F' and A' are generated by performing the following steps:

1. Let F' be $F \cup \{kw\text{-}L\}$.
2. Add to A' all actions in A plus actions $kw\text{-}act\text{-}pos\text{-}L$ and $kw\text{-}act\text{-}neg\text{-}L$, whose preconditions are, respectively, L and $\overline{L}$. Both actions have a single effect, $kw\text{-}L$.
3. For each action a in A' that contains either $C \rightarrow L$ or $C \rightarrow \overline{L}$ as a conditional effect, for some C different from $\{true\}$, we add the unconditional effect $\neg kw\text{-}L$ to a.

Note that Step 2 generates actions that add the fact $kw\text{-}L$. These actions can only be performed if the set of states the agent believes it is in, say, S, is such that $K(L, S)$ or $K(\neg L, S)$ (i.e., the agent knows whether L is true). Step 3 handles the case in which there may be "loss of knowledge" due to a conditional effect of an action. To see this, imagine a system with a single action A which is always executable and which has conditional effect $p \rightarrow L$. Assume I is such that the set of states consistent with I is $\{\{p\}, \{\}\}$. Even though the agent knows whether L in the states of I, it is not the case anymore after performing A, since the set of resulting states is $\{\{p, L\}, \{\}\}$.

We now prove that our proposed translation is sound and complete, in the senses defined below.

Theorem 4 (Completeness) *Let $\Sigma = (F, A, \Omega, I)$ be a transition system, and let Σ' be defined as above. If T is a plan for $(\Sigma, \mathsf{KnowWhether}(L))$ then there exists a plan T' for $(\Sigma', kw\text{-}L)$ which differs from T only in that it contains actions of the form $kw\text{-}act\text{-}pos\text{-}L$ or $kw\text{-}act\text{-}neg\text{-}L$ at the end of each of T''s branches.*

Proof. We note that if there exists an action tree T that is a plan for $(\Sigma, \mathsf{KnowWhether}(L))$ then we can construct a plan T' for $(\Sigma', kw\text{-}L)$ by simply adding an additional $kw\text{-}act\text{-}pos\text{-}L$ or $kw\text{-}act\text{-}neg\text{-}L$ as the final action in each branch of the tree. Such actions are executable at that point since it holds that $K(L, S)$ or $K(\neg L, s)$, where S is the set of states reached by that branch. ■

Theorem 5 (Soundness) *Let $\Sigma = (F, A, \Omega, I)$ be a transition system, and let Σ' be defined as above. If T is a plan for $(\Sigma', kw\text{-}L)$ then, by removing all actions of the form $kw\text{-}act\text{-}pos\text{-}L$ or $kw\text{-}act\text{-}neg\text{-}L$ from T we obtain an action tree that is a plan for $(\Sigma, \mathsf{KnowWhether}(L))$.*

Proof. Let T be a plan for $kw\text{-}L$. Take any S_f that results from the execution of a branch of T; i.e., such that $(T, S_0) \vdash^* (\epsilon, S_f)$. Observe that $kw\text{-}L \in S_f$. Now let (kT', S) be the configuration visited by the execution of the branch in which kwL is added for the last time (here k is either $kw\text{-}act\text{-}pos\text{-}L$ or $kw\text{-}act\text{-}neg\text{-}L$). In other words, let S be such that $(T, S_0) \vdash^* (kT', S) \vdash (T_1, S_1) \vdash (T_2, S_2) \vdash^* (T_n, S_n)$, with $T_n = \epsilon$, $S_n = S_f$, and such that for all $i \in \{1, \ldots, n\}$, $kw\text{-}L \in S_i$. Because k is executable in S, either $K(L, S)$ or $K(\neg L, S)$ holds. Furthermore, because S and S_1 differ in at most $kw\text{-}L$, it also holds that either $K(L, S_1)$ or $K(\neg L, S_1)$ holds. Now assume that it holds that $K(L, S_i)$ or $K(\neg L, S_i)$ for some $i \geq 1$. Since $kwL \in S_{i+1}$ the action that was performed in S_i to yield S_{i+1} was either an action that does not change the truth value of L or changes it unconditionally (in other words, it is not an action modified by Step 3 of the compilation). In either case it holds either $K(L, S_{i+1})$ or $K(\neg L, S_{i+1})$. We conclude that $K(L, S_n)$ or $K(\neg L, S_n)$, which means that T achieves $\mathsf{KnowWhether}(L)$. We observe now that if we remove all occurrences of $kw\text{-}act\text{-}pos\text{-}L$ or $kw\text{-}act\text{-}neg\text{-}L$ we obtain a plan that also achieves $\mathsf{KnowWhether}(L)$. ■

Now consider the epistemic goal $\mathsf{Discriminate}(L_1, L_2)$, where L_1 and L_2 are literals. The compilation follows the same intuitions from above. We generate a new transition system $\Sigma' = (F', A', \Omega, I)$, where and F' and A' are computed by performing the following steps:

1. Let F' be $F \cup \{disc\text{-}L_1\text{-}L_2\}$.
2. Add to A' all actions in A plus actions $disc\text{-}act\text{-}1\text{-}L_1\text{-}L_2$ and $disc\text{-}act\text{-}2\text{-}L_1\text{-}L_2$, whose preconditions are $\{L_1, \overline{L_2}\}$ and $\{\overline{L_1}, L_2\}$, respectively. Both actions have a single effect, $disc\text{-}L_1\text{-}L_2$.
3. For each action a in A' that contains $C \rightarrow L_1$, $C \rightarrow \overline{L_1}$, $C \rightarrow L_2$, or $C \rightarrow \overline{L_2}$ as a conditional effect, for some C, we add the unconditional effect $\neg disc\text{-}L_1\text{-}L_2$ to a.

This translation is also sound and complete. The proofs are similar to the ones presented above.

Theorem 6 (Completeness) *Let $\Sigma = (F, A, \Omega, I)$ be a transition system, and let Σ' be defined as above. If T is a plan for $(\Sigma, \mathsf{Discriminate}(L_1, L_2)))$ then there exists a plan T' for $(\Sigma', disc\text{-}L_1\text{-}L_2)$ which differs from T only in that it contains actions of the form $disc\text{-}act\text{-}1\text{-}L_1\text{-}L_2$ or $disc\text{-}act\text{-}2\text{-}L_1\text{-}L_2$ at the end of each of T''s branches.*

Theorem 7 (Soundness) *Let $\Sigma = (F, A, \Omega, I)$ be a transition system, and let Σ' be defined as above. If T is a plan for $(\Sigma', \mathsf{Discriminate}(L_1, L_2))$ then, by removing all actions of the form $disc\text{-}act\text{-}1\text{-}L_1\text{-}L_2$ or $disc\text{-}act\text{-}2\text{-}L_1\text{-}L_2$ from T we obtain an action tree that is a plan for $(\Sigma, \mathsf{Discriminate}(L_1, L_2))$.*

Extending the Compilation to Formulae We have discussed how to compile goals of the form $\mathsf{KnowWhether}(L)$ and $\mathsf{Discriminate}(L_1, L_2)$, for the case of literals. To extend our compilation to general formulae we can use a precompilation step in which, for each formula ϕ involved in the epistemic goal, we generate the ramification constraints ϕ **causes** L_ϕ and $\neg\phi$ **causes** $\neg L_\phi$. Then we apply the compilation we described above. Soundness and completeness now follows from the theorems above together with soundness of the compilation of ramifications into effect axioms.

6 Computing Diagnostic Plans

The purpose of this paper was largely to define the theoretical underpinnings that support the exploitation of state-of-the-art AI planning systems for diagnostic problem solving. As such, our translation can be exploited by any planning system capable of handling conditional effects and sensing actions; in particular it can be used along with conditional

planners such as Contingent-FF (Hoffmann and Brafman 2005), POND (Bryce, Kambhampati, and Smith 2006), and the CNF- and DNF-based conditional planners by To, Pontelli, and Son (2011).

Our translation could also be used with the latest contingent online planning systems CLG (Albore, Palacios, and Geffner 2009), CLG+ (Albore and Geffner 2009), and the SDR planner by (Brafman and Shani 2012), even though some of these systems may not return a complete conditional plan. However, for our experiments we wished to generate offline conditional plans and we opted to use Contingent-FF.

We developed 8 diagnostic planning problem scenarios in the following domains: an extension of the light-switch-battery domain given in the Section 3 example, an agent moving between rooms fixing light bulbs to complete a circuit, and embedded computer chips. We varied the complexity of the problems and ran them with Contingent-FF using a PC with an Intel Xeon 2.66 GHz processor with 4GB RAM running Linux. The simple problems generated reasonable plans, requiring from under 0.01 seconds to 0.03 seconds to complete. The more complex problems resulted in poor quality plans that favoured conformant rather than contingent solutions. We attribute much of this poor quality to the general effectiveness of the planner and how it's underlying heuristics interact with our modeling of ramifications. We also experimented with encoding ramifications as additional actions rather than compiling them into effects. In future work, we will explore different representations for our ramification constraints across different planners.

An interesting result came from the problems created for the embedded computer chips domain, which consists of a collection of computer chips which themselves contain chips, and so forth, where all chips must be working normally for the output to be displayed. When allowing the planner to sense the status of and replace any chip, the resulting plan was always to replace the top level chips. This supports the notion of taking a purposeful view of diagnosis – it can be much faster to simply repair the issue than to determine the unique diagnosis. More generally, selecting the best course of purposeful actions can be informed by many factor including the cost of actions (sensing and world altering), time-criticality of a response, and issues of specificity or generality with respect to how a grouping of related faults should be addressed (e.g., fix them individually vs. doing a more general fix).

7 Related Work and Concluding Remarks

In addition to the literature on model-based diagnosis cited in Section 3, there is a body of previous work that relates diagnosis to theories of action in some guise. The relationship between actions and diagnoses was observed sometime ago by (e.g., Cordier and Thiébaux 1994; McIlraith 1994), while Sampath et al. (1995) were the first to present comprehensive results diagnosing discrete event systems via finite state automata. Thielscher (1997), McIlraith (1998), and subsequently Iwan (2001) and Baral, McIlraith, and Son (2000) cast the diagnosis problems in terms of an AI theory of action and change. More recently Grastien et al. (2007), Sohrabi, Baier, and McIlraith (2010), Lamperti and Zanella (2011) and Yu, Wen, and Liu (2013) have all addressed aspects of dynamical diagnosis.

Specifically in the area of diagnostic and repair planning, the most notable related work is that of Baral, McIlraith, and Son (2000) who introduced the notion of diagnostic and repair plans as conditional plans, and who have a treatment of causality and sensing. That work shares a number of intuitions with the work presented here, but without the focus on planning for epistemic goals, and the correspondence to conformant and contingent planning. Also of note is the work of Kuhn et al. (2008) who introduce the notion of *pervasive diagnosis* which produces diagnostic production plans that achieve production goals while coincidentally uncovering additional information about system health.

The notion of combining diagnosis with repair has been addressed in varying fashion. For example, Thiébaux et al. (1996), in this and later work, discuss the challenges of diagnosis and repair in the context of a power supply restoration problem, identifying the task as problem of planning with uncertainty. These works share intuitions with the approach advocated here, but do not reflect the advances in our collective understanding of the representational and computational issues associated with planning and sensing. Finally, outside the area of diagnostic problem solving there has been a variety of work looking at planning with some form of sensing. Of particular note is the work of Brenner and Nebel (2009) on MAPL, a continual planning system that interleaves planning with acting and sensing. This paradigm of planning and sensing is also one that is very amenable to diagnostic problem solving.

In this paper we have argued for and explored a purposeful view of diagnostic problem solving, examining the problem through the lens of AI automated planning. We have characterized diagnostic planning with both ontic and epistemic goals, and established properties of these diagnostic planning tasks, including both complexity results and an understanding of their relationship to classical, conformant, and conditional planning systems. Of particular note was the characterization of diagnostic epistemic goals such as Discriminate and KnowWhether and their translation into planning problems with ontic goals. The correspondence with existing planning paradigms enables diagnostic planning to leverage ongoing advances in the development of non-classical planners. We discuss the exploitation of such planners, outlining our experience addressing diagnostic problem solving with Contingent-FF. Results to date are guardedly encouraging but expose the need for further investigation of the nuances of these planners as a complement to the results of this paper.

Beyond diagnostic problem solving, the work presented is relevant to a diversity of problems that involve generating hypotheses to conjecture system state, and sensing and acting in the world to discriminate those hypotheses or to purposefully effect change in response to observed behaviour. Some such problems include active vision applications, activity recognition, and goal recognition.

Acknowledgements: We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Albore, A., and Geffner, H. 2009. Acting in partially observable environments when achievement of the goal cannot be guaranteed. In *Proc. of ICAPS Workshop on Planning and Plan Execution for Real-World Systems*.

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *IJCAI*, 1623–1628.

Andersen, M. B.; Bolander, T.; and Jensen, M. H. 2012. Conditional epistemic planning. In *JELIA*, volume 7519 of *LNCS*. Springer. 94–106.

Baral, C.; McIlraith, S.; and Son, T. 2000. Formulating diagnostic problem solving using an action language with narratives and sensing. In *KR*, 311–322.

Brafman, R. I., and Shani, G. 2012. Replanning in domains with partial information and sensing actions. *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.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning graph heuristics for belief space search. *J. Artif. Intell. Res. (JAIR)* 26:35–99.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *AIJ* 69(1-2):165–204.

Console, L., and Torasso, P. 2006. Automated diagnosis. *Intelligenza Artificiale* 3(1-2):42–48.

Cordier, M.-O., and Thiébaux, S. 1994. Event-based diagnosis of evolutive systems. In *DX*, 64–69.

de Kleer, J., and Williams, B. 1987. Diagnosing multiple faults. *AIJ* 32:97–130.

de Kleer, J.; Mackworth, A.; and Reiter, R. 1992. Characterizing diagnoses and systems. *AIJ* 56(2–3):197–222.

Genesereth, M. 1984. The use of design descriptions in automated diagnosis. *AIJ* 24:411–436.

Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *AAAI*, 305–310.

Haslum, P., and Jonsson, P. 1999. Some results on the complexity of planning with incomplete information. In *ECP*, 308–318.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Herzig, A.; Lang, J.; and Marquis, P. 2003. Action representation and partially observable planning using epistemic logic. In *IJCAI*, 1067–1072.

Hoffmann, J., and Brafman, R. I. 2005. Contingent planning via heuristic forward search witn implicit belief states. In *ICAPS*, 71–80.

Iwan, G. 2001. History-based diagnosis templates in the framework of the situation calculus. In *KR/ÖGAI*, 244–259.

Kuhn, L. D.; Price, B.; de Kleer, J.; Do, M. B.; and Zhou, R. 2008. Pervasive diagnosis: The integration of diagnostic goals into production plans. In *AAAI*, 1306–1312.

Lamperti, G., and Zanella, M. 2011. Context-sensitive diagnosis of discrete-event systems. In *IJCAI*, 969–975.

Lin, F. 1995. Embracing causality in specifying the indirect effects of actions. In *IJCAI*, 1985–1991.

McCain, N., and Turner, H. 1995. A causal theory of ramifications and qualifications. In *IJCAI*, 1978–1984.

McIlraith, S. A., and Scherl, R. B. 2000. What sensing tells us: Towards a formal theory of testing for dynamical systems. In *AAAI*, 483–490.

McIlraith, S. 1994. Towards a theory of diagnosis, testing and repair. In *DX*, 185–192.

McIlraith, S. 1998. Explanatory diagnosis: Conjecturing actions to explain observations. In *KR*, 167–179.

McIlraith, S. 2000. Integrating actions and state constraints: A closed-form solution to the ramification problem (sometimes). *AIJ* 116(1-2):87–121.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *JAIR* 35:623–675.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *KR*, 324–332.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *AIPS*, 212–222.

Pinto, J. 1999. Compiling ramification constraints into effect axioms. *Computational Intelligence* 15:280–307.

Poole, D. 1994. Representing diagnosis knowledge. *Ann. Math. Artif. Intell.* 11(1-4):33–50.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *JAIR* 4:287–339.

Reiter, R. 1987. A theory of diagnosis from first principles. *AIJ* 32(1):57–95.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, MA: MIT Press.

Rintanen, J. 2004. Complexity of planning with partial observability. In *ICAPS*, 345–354.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control* 40(9):1555–1575.

Sandewall, E. 1996. Assessments of ramification methods that use static domain constraints. In *KR*, 99–110.

Scherl, R., and Levesque, H. 2003. Knowledge, action, and the frame problem. *AIJ* 144(1–2):1–39.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. 2010. Diagnosis as planning revisited. In *KR*, 26–36.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. 2011. Preferred explanations: Theory and generation via planning. In *AAAI*.

Strass, H., and Thielscher, M. 2013. A general first-order solution to the ramification problem with cycles. *Journal of Applied Logic* 11(3):289–308.

Thiébaux, S.; Cordier, M.-O.; Jehl, O.; and Krivine, J.-P. 1996. Supply restoration in power distribution systems: A case study in integrating model-based diagnosis and repair planning. In *UAI*, 525–532.

Thielscher, M. 1995. Computing ramifications by postprocessing. In *IJCAI*, 1994–2000.

Thielscher, M. 1997. A theory of dynamic diagnosis. *Electronic Transactions on Artificial Intelligence* 1:73–104.

To, S. T.; Pontelli, E.; and Son, T. C. 2011. On the effectiveness of CNF and DNF representations in contingent planning. In *IJCAI*, 2033–2038.

Yu, Q.; Wen, X.; and Liu, Y. 2013. Multi-agent epistemic explanatory diagnosis via reasoning about actions. In *IJCAI*.

A Contingent Planning-Based POMDP Replanner

Ronen Brafman and **Alexander Gorohovski** and **Guy Shani**
Ben-Gurion University of the Negev
Be'er Sheva, Israel

Abstract

POMDPs offer one of the richer models for sequential decision making. But this expressiveness comes with a price, both theoretical and practical. In this paper we seek to address the practical difficulty of solving POMDPs by focusing on symbolic, goal-oriented POMDP models, exploiting these features within a simple, generic, and novel online replanning architecture. This architecture leverages recent improvements in contingent planning, addressing the problem of selecting the next action at each decision epoch by solving a contingent planning problem derived from the original POMDP and the current belief state. While this is work in progress, we provide some initial empirical results.

Introduction

Partially observable Markov Decision Processes (POMDP) are a popular model for sequential decision making under uncertainty with partial observability. This rich model is also quite difficult to solve, and some version of this problem are undecidable (Madani, Hanks, and Condon 2003). This has not deterred researchers in the area, and a number of approaches exist for solving POMDPs offline and online. Nevertheless, this is a very challenging problem on which scaling up is difficult, although some methods that exploit domain structure have been able to scale up nicely (Hoey and Poupart 2005).

In this paper we are concerned with solving goal-oriented symbolic POMDPs, online. In principle, any POMDP can be transformed into a goal oriented domain (Bonet and Geffner 2009), although it is not completely clear how effective this transformation is in practice. Nevertheless, domains that are naturally goal-oriented are common and important. These are domains that represent decision problems in which we seek to achieve some property of the world, and where our actions are stochastic, our information is partial, and we can perform observations to obtain useful information.

Symbolic POMDPs represent the transition and reward functions implicitly by describing the impact of actions over state variables, using an appropriate language, such as PPDDL (Younes and Littman 2004) or RDDL (Sanner 2010). This compact representation makes practical and natural the description of models with hundreds of thousands of states – models that would be impractical to describe using an explicit transition matrix. Methods that operate on these models typically scale up better than methods that operate on explicit state models, but the problem remains challenging. Finally, online planning allows the solver to focus on the true current state, rather than arbitrary states, and does not require the generation of a complete policy, which requires considerable time and space to generate. Consequently, online solvers are likely to scale up better than offline solvers.

Contingent planning is a model that is very similar to symbolic, goal oriented POMDPs. In a sense, contingent planning models are the non-deterministic (as opposed to stochastic) analogue of goal-oriented POMDPs. Developed as an extension of classical planning, they are naturally symbolic, extending planning description languages such as PDDL (Fox and Long 2003). Thus, they are goal oriented, allow for an uncertain belief state, non-deterministic actions, and observations. Thus, they only differ from our target class of POMDPs by the use of non-deterministic actions and observations, as opposed to stochastic observations.

In recent years, the area of contingent planning has seen a host of new planners that considerably improve the state of the art. These planners directly manipulate the symbolic representation of the problem in manners more sophisticated than symbolic POMDPs, using powerful heuristic functions, and can handle symbolic models that are much larger than those of current symbolic POMDP solvers. Of course, the comparison between the two models is unfair – the contingent model is qualitative and most state of the art techniques handle deterministic actions only. Nevertheless, we propose to exploit this success with the following simple replaninng approach, reminiscent of FF-Replan (Yoon, Fern, and Givan 2007), where instead of MDPs we address POMDPs, and instead of determinization into classical planning, we transform the problem into contingent planning.

Thus, the architecture we propose is very simple: Given a POMDP and an initial belief state, generate a corresponding contingent planning problem and solve it. Execute the policy generate by the contingent planner until some condition is met. Intuitively, this condition should capture the fact that the real belief state is substantially different from the belief state the contingent executor would be in. At that point, replan. The main benefit of this architecture is that it is simple and modular, yet uses a sophisticated simplifica-

tion of the POMDP model to inform local decisions. Moreover, any progress made in contingent planning, such as the ability to handle non-deterministic actions well, will immediately translate into more informed choices by the POMDP replanner.

The rest of this paper describes this architecture briefly, and provides some initial experimental results.

Background

Contingent Planning

A contingent planning model is a tuple of the form $\langle S, S_0, S_G, A, Tr, \Omega, O\rangle$, where S is a set of states, $S_0 \subseteq S$ is the set of possible initial states, also called the *initial belief state* and is often denoted b_I, $S_G \subseteq S$ is the set of goal states, A is a set of action symbols, and Tr is the transition function, such that $Tr(s,a) \subseteq S$ is the set of states that can be reached by applying a in state s, Ω is a set of observation symbols, and $O(a,s') \in \Omega$ is the observation obtained when s' is reached following the application of a.

At each point in execution, there is a set of states considered possible, called the current belief state. b_I is the initial belief state, and if b is the current belief state, a is executed, and o is the observed then the resulting belief state $\tau(b,a,o)$ is defined as:

$$\tau(b,a,o) = \{s' | s \in b \text{ and }, s' \in Tr(s,a), o \in O(a,s')\} \quad (1)$$

That is, states s' that can result from the execution of a in a state in b, such that o can be observed. We extend this notation to a sequence $\bar{a}, \bar{o}$ of actions and observations recursively as follows:

$$\tau(b, \bar{a}\cdot a, \bar{o}\cdot o) = \tau(\tau(b,\bar{a},\bar{o}), a, o) \quad (2)$$

A contingent planning problem is specified as a tuple $\langle P, \mathcal{A}, \varphi_{\mathcal{I}}, \mathcal{G}\rangle$. P is a set of propositions, $\mathcal{A}$ is the set of actions, φ_I is a propositional formula over P in prime implicate form describing the possible initial states, and $G \subset P$ is the set of goal propositions.

A state of the world s assigns a truth value to all elements of P. A *belief-state* is a set of possible states, and the initial belief state, b_I, consists of the set of states initially possible, i.e. $S_0 = b_I = \{s : s \models \varphi_I\}$. The goal is to arrive at a belief state in which all propositions in G hold, i.e., $S_G = \{s \in S$ such that $s \models g_i$ for every $g_i \in G\}$.

For the sake of this paper, we focus on contingent planning problems with deterministic actions, as this is what current state-of-the-art planners support. A deterministic action, $a \in \mathcal{A}$, is a triple: $\{$*pre*(a), *effects*(a), *obs*$(a)\}$. We shall use the more common $a(s)$ to denote $Tr(s,a)$. The action precondition, *pre*(a), is a set of literals. The action effects, *effects*(a), is a set of pairs, $(c_{a,l}, l)$, denoting conditional effects, where $c_{a,l}$ is a propositional formula and l is a literal. For notational convenience, we'll assume one condition $c_{a,l}$ exists for every action a and literal l. In practice, $c_{a,l} = \mathit{false}$ for most literals, l, i.e., l is typically not a possible conditional effect of a, and this pair can be omitted. *obs*(a) is also a set of pairs, $\{(\omega_{a,o}, o) | o \in \Omega\}$, where $\omega_{a,o}$ is a propositional formula over P and $o \in \Omega$.[1] Thus, $o = O(a,s')$ iff $s' \models \omega_{a,o}$.

Since one observation must occur following the execution of an action, $\bigvee_{o\in\Omega} \omega_{a,o} = true$ for every $a \in A$. As sensing is deterministic, the $\omega_{a,o}$ for different o's are mutually exclusive. Our implementation uses special *no-obs* observations denoting nothing-observed, but as *no-obs* can be treated like any other observation, we make no special distinction between it and "real" observations.

We restrict our attention to deterministic observations: every non-deterministic observation can be compiled away by adding an observable state variable, whose value changes non-deterministically following the action, representing the observation result.

Related Work

We are aware of two planners that use the replanning approach to solve variants of POMDPs: POND-Hindsight (Olsen 2011) and the planner by Wang and Dearden (Wang and Dearden 2011). POND-Hindsight operates by attempting to solve the goal-based belief-space MDP that corresponds to the original POMDP. The belief-state MDP is solved by using replanning techniques formulated for regular MDPs, following FF-hindsight (Yoon et al. 2008). The belief-state is tracked online using a Rao-Blackwellized particle filter.

The Wand and Dearden planner attempt to solve a "quasi-deterministic" POMDP problem. These are POMDPs in which non-observation actions are deterministic. This planner translates the quasi-deterministic problem into a series of classical planning problems. This is done by using single outcome determinization, so the non-determinism of each observation action is replaced by its most probable outcome. For every step in the generated classical plan in which an observation action is performed, the planner is called to generate another classical plan, which is combined into the global contingent plan as a branch. The above process continues until the contingent plan is completed. Unfortunately, this contingent plan cannot be executed directly because the conditions leading to each branch may not be known. In order to overcome this problem, while executing the plan, the planner uses execution monitoring to choose which branch to take next. The execution monitoring system keeps up a distribution on the belief state. Whenever the execution reaches, at the global contingent plan, some branch point that is dependent on the value of some variable x, the execution monitoring system uses a value of information calculation in order to greedily choose some observation action to discover the value of x. When there is no available observation action whose value of information (w.r.t. x) is greater than zero, the execution chooses the branch with the highest expected value.

The Architecture and Its Components

The architecture relies on the following components:

[1] Formally, Ω is specified implicitly as the set of observation symbols appearing in actions in $\mathcal{A}$.

1. Belief state maintenance module;
2. POMDP-to-contingent translator;
3. Contingent planner;
4. Replanning criterion.

The planner then operates using a simple loop:

Algorithm 1 POMDP Replanner

Input: π – POMDP model, θ – goal probability
1: $b \leftarrow b_0$ the initial belief state
2: **while** $Pr(G|b) < \theta$ **do**
3: π_C = Generate Contingent Problem(b, π).
4: $\rho = \text{Solve}(\pi_C)$
5: **repeat**
6: Apply ρ for one step
7: Update b based on current action and observation
8: **until** Replanning Criterion is true or $b \models G$

The architecture is simple and modular and leverages the improved state of modern contingent planners to solve a nontrivial simplification of the POMDP model that takes into account uncertainty about the initial state, as well as the need to make observations. It can benefit from any improvement to belief maintenance methods and to contingent planning. We now explain each of the components.

Belief State Maintenance with a Particle Filter

For the purpose of replanning, the agent must maintain some model of the current belief state. While a precise representation would be ideal, it is usually impossible to maintain because even if the initial belief state can be represented compactly, this is not longer true after a number of updates. Particle filters (Arulampalam et al. 2002) are probably the most widely used belief state tracking method for POMDPs, thanks to their simplicity and their good empirical performance. A particle filter approximates the belief state by maintaining a set of particles – essentially states. Sometimes, weights are also assigned to each particle. For every state s, $b(s)$ is approximated by the frequency of s-states in the set of particles, or if weights are assigned, by the relative weight of the set of s particles. Initially, the set of particles is obtained by sampling from the true initial belief state. Following each action, the set of particles is updated to reflect the effect of the action. There are different techniques for doing this, but essentially, the action is applied to each particle, and a possible effect is sampled. Following an observation, the set of particles is updated as well. One popular technique is to assign a weight to each particle that is proportional to the probability of the observation given the state and the action. When an unweighted particle filter is used, one then samples a new set of (unweighted) particles from the distribution that corresponds to the weighted set of particles. Our implementation is based on the SIS (sequential importance sampling) with resampling method (Arulampalam et al. 2002).

Generating a Contingent Planning Problem

The input to our problem is a description of a POMDP in PPDDL. Thus, it is not difficult to generate a contingent problem from this input in a PDDL-like format. In principle, the initial state is obtained from the initial state of the POMDP by simply ignoring probabilities – i.e., states with positive support are considered possible, or it is possible to consider only states with probability greater than some $c > 0$. Currently, we focus on initial states where the distribution over possible states is uniform. Thus, they can be described using a suitable propositional formula, which simply describes the set of possible states. Thus, the translation is immediate. The goal of the contingent planning problem is identical to that of the POMDP. As for actions and observations, our current contingent planner (MPSP (Brafman and Shani 2012)) supports deterministic actions and observations only. For actions, we use the all-outcome determinization. For observations, we use the single-outcome (most likely) determinization.

Contingent Planner

Our contingent planner is MPSP (Brafman and Shani 2012), which is itself a translation-based planner that reduces contingent planning into classical planning. Nevertheless – this reduction is very different from a direct reduction to classical planning via determinization because MPSP is, theoretically, a sound and complete *contingent* planner that can generate an entire contingent plan tree. In practice, MPSP cannot generate solutions for problems with more than a few initial possible states. Thus, we must sample a small subset of states (typically, two) from the initial belief state, and use them as the initial belief state of the contingent planner. Nevertheless, the sophisticated nature of the translation used is such that important information about uncertainty and the agent's belief state is maintained and updated, unlike a shallow projection to classical planning. MPSP uses a variant of PDDL that is suitable for contingent planning because it allows for a description of observations. Actions, however, are deterministic.

Replanning Condition

An important question is when should the contingent planner be called again for replanning. In MDP replanners, this happens when the state is different from the expected state. In the case of POMDPs, one could apply a similar criteria to the belief state – i.e., replan when your belief state is different from what the contingent planner expected. However, since the contingent planner uses a crude approximation of the initial belief state, this would trigger replanning after almost every step. An alternative is to mimic the replanning criterion for contingent planners. In most replanners, planning is triggered following an observation. This is due to the separation between sensing and actuating actions in current contingent planning benchmarks. However, most POMDP benchmarks lead to more frequent observations. Moreover, ideally, the contingent planner takes into account, at least to some extent, the effect of different observations, and a call to a contingent planner is more costly than a call to a classical planner. Thus, one needs to be more conservative.

Intuitively, one would like to perform replanning following an informative observation, or one that is highly unexpected. Our planner takes the second approach. It keeps I_p

– the set of particles that were sampled for use as the initial state for the contingent planning problem. Once, it receives a new observation o it checks all current successor particles of I_p. In case none of them is consistent with o, the planner performs replan, otherwise, it continues the classical plan execution.

The approach of applying replanning following a very informative observation may be implemented by using the weights on the particles. One can compare the weight of the particles before the observation and after the observation using some similarity score, such as Pearson correlation coefficient, or KL-divergence. If this score passes some threshold, replanning should be performed.

Empirical Evaluation

We implemented an initial version of the planner described above and tested its performance on a number of goal-oriented benchmarks. Some are known from the POMDP literature, and some are probabilistic version of contingent benchmarks. Current running times of our planners appear in Table 1. We show the average (over 30 experiments) number of actions and running time until success. The planner was implemented in $C\#$. Experiments were run on a PC with an Intel i3 processor with 2.53 Ghz, and 4.00 GB RAM running Windows 7.

We compared our planner to the winner of the 2011 planning competition: POMDPX NUS by Wu, Lee, and Hsu, which uses Monte Carlo sampling techniques. As expected, in goal-oriented domains and without a suitable heuristic function, it does not perform well, as rollouts that do not reach the goal provide no useful information. In many problems it simply run out of memory (OOM) and in the doors example we had problem generating an appropriate RDDL encoding of the problem (ENC). We are now working on a comparison with POND-Hindsight, which is the planner closest in spirit to outs, and is also the only planner we are aware of that accepts PPDDL domains.

Table 1: Initial Empirical Results

		POMDP-Replan		POMDP-NUS	
Problem	$\|Vars\|$	# actions	Time	# actions	Time
Localize 13	174	63.33	3.11	774	15.39
Sliding-doors 70	405	114.27	13.76	ENC	ENC
EBTCS70	72	2.77	0.39	OOM	OOM
RckSmpl 6-6	135	353.63	21.64	373	47.5
RckSmpl 6-8	456	466.53	32.43	527	65.24
RckSmpl 8-4	582	639.03	53.77	746	147.68
RckSmpl 8-6	682	789.17	88.85	871	183.27
RckSmpl 8-8	812	1053.77	107.72	1006	245.02
Doors5	135	101.41	8.68	ENC	ENC
Doors7	271	179.03	45.39	ENC	ENC
Doors9	455	473.59	126.47	ENC	ENC
Doors11	687	700.2	281.09	ENC	ENC
Doors13	967	950.25	404.14	ENC	ENC
Unix1	22	30.46	0.67	64	2.17
Unix2	46	164.53	4.1	OOM	OOM
Unix3	94	409.57	14.59	OOM	OOM
Unix4	190	1497.97	112.69	OOM	OOM

Summary and Future Work

We described a simple architecture for a POMDP replanning algorithm based on a reduction to contingent planning. While the effectiveness of our approach is yet unclear, the method's advantages are its simple, modular design, and its simplicity. For instance, we are now working on a version that will use a more powerful contingent planner – HCP (Maliah et al. 2014), as well as a symbolic particle filter. Similarly, should contingent planner be able to handle non-deterministic domains well, we could take advantage of this and replace the all-outcome determinization with a more sophisticated projection to contingent domains. Moreover, in the future, we hope to integrate more complex reward structures, e.g., by reduction to goal-based POMDPs (Bonet and Geffner 2009).

Acknowledgements: We would like to thank the anonymous reviewers for their helpful suggestions. The authors are supported in part by ISF grant 933/13. Brafman and Gorohovski are supported in part by the the Paul Ivanier Center for Robotics Research and Production Management, and the Lynn and William Frankel Center for Computer Science.

References

Arulampalam, M. S.; Maskell, S.; Gordon, N. J.; and Clapp, T. 2002. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing* 50(2):174–188.

Bonet, B., and Geffner, H. 2009. Solving pomdps: Rtdp-bel vs. point-based algorithms. In *IJCAI*, 1641–1646.

Brafman, R. I., and Shani, G. 2012. A multi-path compilation approach to contingent planning. In *AAAI'12*.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of AI Research* 20:61–124.

Hoey, J., and Poupart, P. 2005. Assisting persons with dementia during handwashing using a partially observable markov decision process. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1332–1338.

Madani, O.; Hanks, S.; and Condon, A. 2003. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.* 147(1-2):5–34.

Maliah, S.; Brafman, R. I.; Karpas, E.; and Shani, G. 2014. Partially observable online contingent planning using landmark heuristics. In *ICAPS'14*.

Olsen, A. 2011. Pond-hindsight: Applying hindsight optimization to partially-observable markov decision processes. Master's thesis, Utah State University.

Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description.

Wang, M., and Dearden, R. 2011. Planning with state uncertainty via contingency planning and execution monitoring. In *SARA*.

Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI*, 1010–1016.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *ICAPS*.

Younes, H., and Littman, M. 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.

A Relevance-Based Compilation Method for Conformant Probabilistic Planning

Ran Taig and **Ronen I. Brafman**
Computer Science Department
Ben Gurion University of The Negev
Beer-Sheva, Israel 84105
taig,brafman@cs.bgu.ac.il

Abstract

Conformant probabilistic planning (CPP) differs from conformant planning (CP) by two key elements: the initial belief state is probabilistic, and the conformant plan must achieve the goal with probability $\geq \theta$, for some $0 < \theta \leq 1$. Taig and Brafman observed that one can reduce CPP to CP by finding a set of initial states whose probability $\geq \theta$, for which a conformant plan exists. Previous solvers based on this idea used the underlying planner to select this set of states and to plan for them simultaneously. We suggest an alternative approach: Our planner starts with a preprocessing relevance analysis phase that determines a promising set of initial states on which to focus. It then calls an off-the-shelf conformant planner to solve the resulting problem. This approach has a number of advantages. First, we can introduce specific, efficient relevance reasoning techniques for selecting the set of initial states, rather than depend on the heuristic function used by the planner. Second, we can benefit from various optimizations used by conformant planners that are unsound when applied to the original CPP. Finally, we have the freedom to select among different existing CP solvers. Consequently, the new planner dominates previous solvers on almost all domains and scales to instances that were not solved before.

Introduction

In conformant probabilistic planning (CPP) we are given a set of actions – which like most past work, are assumed to be deterministic, a distribution over initial states, a goal condition, and a real value $0 < \theta \leq 1$. We seek a plan π such that following its execution, the goal is achieved with probability $\geq \theta$. Not many natural problems fit the frameworks of conformant planning (CP) and CPP, yet both serve as basic frameworks on which ideas for planning under uncertainty can be developed and tested. Indeed, important ideas developed in CP were later extended to the richer framework of contingent planning, including techniques for representing and reasoning about belief states (Hoffmann and Brafman 2005) and various translation schemes (Albore, Palacios, and Geffner 2009). The latter represent an important trend in research on planning under uncertainty, where simple planners are used to solve more complex problems (Yoon, Fern, and Givan 2007; Palacios and Geffner 2009; Albore, Palacios, and Geffner 2009).

This paper utilizes this reduction idea to develop an effective reduction scheme for solving CPP problems with deterministic actions, motivated by an observation by Taig and Brafman 2013 (TB): One can solve CPP by finding a set of initial states with joint probability $\geq \theta$ for which a conformant plan exists. TB used this idea to build a translation-based CPP solver that has special actions that let the planner ignore a certain state in the solution. Each such action carries a cost equal to the probability of the ignored state. A plan with cost $\leq 1 - \theta$ is a solution to the original CPP problem. TB's technique has several weaknesses. First, the choice of which states to ignore is carried out by the underlying planner's heuristic function. It is not clear that this function is well suited for this task. Second, the planner need not minimize cost, but rather, ensure that a certain cost bound is met. Few current planners or search techniques support this optimization criterion. Finally, various optimization schemes used by CP solvers are unsound under this scheme.

We suggest an alternative, simpler approach where, first, dedicated relevance-based preprocessing analysis is conducted, determining a promising set of states with probability $\geq \theta$ to plan on. Then, this state set is given to an off-the-shelf conformant planner as its initial state. Besides addressing the above shortcomings of TB's method, this method provides the flexibility of selecting the underlying conformant planner to suit the current planning domain. Our empirical evaluation shows that this approach dominates existing state-of-the-art planners on almost all problem instances.

In the next section, we provide some required background on CPP. Then, we discuss related work, followed by a formal description of our planner and its properties. We conclude with an empirical evaluation and a discussion of our results and potential future work.

Conformant Probabilistic Planning

We assume familiarity with the basic notation of classical planning domains via STRIPS with conditional effects: (V, A, I, G), corresponding to a set of *propositions*, *actions*, *initial world state*, and *goal*. A CP problem, (V, A, b_I, G), generalizes this framework, replacing the single initial state with a set of initially possible states, called the *initial belief*

state b_I. This initial state is often described by means of a formula φ_I, such that $b_I = \{w | w \models \varphi_I\}$. A plan is an action sequence $\overline{a}$ such that $\overline{a}(w_I) \supseteq G$ for every $w_I \in b_I$.

CPP extend CP by quantifying the uncertainty regarding b_I using a probability distribution b_{π_I}. In its most general form, CPP allows for stochastic actions, but we leave this to future work, and assume all actions are deterministic. CPP tasks are 5-tuples $(V, A, b_{\pi_I}, G, \theta)$, corresponding to the *set of propositions*, *set of actions*, *initial belief state*, *goals*, and *acceptable goal satisfaction probability*. As before, G is a conjunction of propositions. b_{π_I} denotes a probability distribution over the world states, where $b_{\pi_I}(w)$ is the probability that w is the true initial world state.

In many settings, achieving G with certainty is impossible. CPP introduces the parameter θ, which specifies the required *lower bound* on the probability of achieving G. A sequence of actions $\overline{a}$ is called a *plan* if the weight of the initial states from which $\overline{a}$ reaches the goal is at least θ.

Some approaches to CPP require that the plan be executable in *all* initial states, even those from which it does not reach the goal. That is, each plan prefix must be conformant (with probability 1) with respect to the preconditions of the next action. This extra requirement may make sense in domains where executing an action without satisfying its preconditions may have catastrophic consequences. However, it seems to conflict with the very explicit success criterion of CPPs specified by the parameter θ, and we adapt this requirement as the only requirement from a plan for a CPP problem in our work.

A CPP specification language must provide a way to specify the initial distribution b_{π_I}. Following previous work (Domshlak and Hoffmann 2007) we assume that b_{π_I} is specified using a Bayes net (BN) $\mathcal{N}_{b_I}$. (We assume the reader is familiar with the notion of BN's.) BNs are typically defined over a set of multi-valued variables, and there are numerous formats used in the literature for their specification, all having the same semantics. Here, we adopt the notation used by the PFF planner (Domshlak and Hoffmann 2007) which contains two parts. First, a definition of an induced set of multi-valued variables, corresponding to a set of literals, only one of which can be true at a time (e.g., literals denoting possible locations of an object). And second, $\mathcal{N}_{b_I}$, which is a BN defined over this set of multi-valued variables.

Related Work

The best current CPP solvers are *Probabilistic FF* (PFF) (Domshlak and Hoffmann 2007) and PCBP (Taig and Brafman 2013). Each works well on a subset of benchmark domains. Probabilistic-FF uses a time-stamped Bayesian Networks (BN) to describe probabilistic belief states, extending Conformant-FF's (Hoffmann and Brafman 2006) belief state encoding to model these BN. It uses SAT reasoning (following Conformant-FF) and weighted model-counting to determine whether goal probability is at least θ. In addition, it introduces approximate probabilistic reasoning into Conformant-FF's heuristic function. PFF performs well on many domains, but it is sensitive to the syntax of conditional effects (the order of literals in effect conditions), and requires complex probabilistic reasoning. To model the belief state during planning, it extends the original BN into a time stamped dynamic (and thus continually growing) BN, on which it performs probabilistic inference by compiling probabilistic queries into WMC queries. The algorithm we introduce reasons with a single, static BN that represents the initial state probability. Reasoning is performed at the preprocessing time only, in order to assess initial state restrictions. No probabilistic queries or data structures need be maintained at planning time. This leads to significant saving of time and memory, as reflected in our results.

Closely related to our work are the *CLG+* planner (Albore and Geffner 2009) and the assumption-based planning approach introduced recently (Davis-Mendelow, Baier, and McIlraith 2013). Both attempt to solve planning problems with incomplete information in which goal achievement cannot be guaranteed from all states, by making assumptions that reduce the uncertainty, and planning under these assumptions. However, neither planner uses an explicit probabilistic semantics – a core part of CPP. In *CLG+*, assumptions are made online w.r.t. information gathered during planning and sensing, while our assumptions are made by an efficient analysis preprocessing phase. (Davis-Mendelow, Baier, and McIlraith 2013) consider the idea of planning under assumptions in general, but do not describe a particular strategy for selecting them, possibly allowing a domain expert to specify them.

Another line of related work by Tran et al. (Tran et al. 2009) and Nguyen et al. (Nguyen et al. 2011) considers methods for reducing the initial belief state by a preprocessing analysis phase while ensuring that the resulting plan applies to all original states. Thus, this is a completeness preserving, sound optimization method that can be used by our underlying conformant planner.

Relevance-Based Reduction

The planner most relevant to our current work is PCBP. It is a relevance-based translation-based planner that exploits the following Lemma:

Lemma 1 (Taig & Brafman, 2013) *A CPP* $\mathcal{CP} = (V, A, b_{\pi_I}, G, \theta)$ *is solvable iff there exists a solvable CP problem* $\mathcal{C} = (V, A, b_I, G)$ *such that* $b_{\pi_I}(\{w \in b_I\}) \geq \theta$. *Moreover,* $\overline{a}$ *is a solution to* $\mathcal{CP}$ *iff it is a solution to* $\mathcal{P}$.

PCBP augments the regular set of actions with special actions — one per possible initial state. Each such action allows the planner to ignore a possible state by essentially making the goal easy to achieve from that state. The cost of each such action equals the initial probability of the corresponding state, while all other actions have zero cost. A plan for the new problem with cost $\leq 1 - \theta$ represents a solution to the original CPP because the set of states from which it fails to reach the goal has probability $\leq 1 - \theta$.

PCBP performs well on a number of domains, but almost completely fails on others (such as logistics, grid, rovers). It suffers from a number of weaknesses — some inherent, and some tied to the limitation of existing translation methods and search methods. First, the choice of which states to ignore is essentially carried out by the planner's heuristic function, because it is implemented by the action selection

mechanism. It is not clear that this function is well suited for this task. Second, the planner needs to solve a cost-bounded planning problem which few current planners or search techniques support. Third, there are various optimization that existing CP solvers carry out which are unsound in this scheme. For example, the completeness preserving pruning methods mentioned above, or T_0's relevance-analysis (Palacios and Geffner 2009) which attempts to identify clauses and propositions that can be safely ignored during planning. These techniques are unsound for CPPs. Finally, PCBP's reduction does not work if actions have cost and we seek a (truly) cost-optimal, or cost-efficient plan.

The RBPP Planner

To overcome these problems, we suggest a simpler, preprocessing approach. First, relevance-based analysis is carried out to identify those states which would be most profitable to ignore. Then, a CP problem is defined in which the initial state consists of all states that are not ignored. This is given to an off-the-shelf CP solver. This method addresses the above shortcomings of PCBP, and in addition gives us the flexibility to choose which conformant planner to use. While in this paper we do not attempt to provide an automated portfolio-based method, our empirical analysis indicates that different solvers have advantages in different domains, showing the potential for farther improvement by automatically selecting the underlying conformant planner which best suits the characteristics of the problem in hand.

Algorithm 1 RBPP (P, *conf-planner*)

$\psi_I \Leftarrow$ RESTRICT(P);
return *conf-planner*($\widetilde{P} = (V, A, \psi_I, G)$);

Algorithm 2 RESTRICT (P)

$Q \Leftarrow$ SORT-CLAUSES(P);
$\psi_I \Leftarrow \varphi_I$;
while $(b_{\pi_I}(\psi_I) \geq \theta)$ **do**
  $C \Leftarrow$ *Extract-First*(Q);
  $\psi_I \Leftarrow$ *RESTRICT-CLAUSE*(C, ψ_I, P);
end while
return ψ_I;

Algorithm 3 SORT-CLAUSES (P)

return A sorted list of all non-unit clauses in φ_I according to the following parameters order:
1. RL(C) (high to low).
2. Rad(C) (high to low).
3. PI(C) (High to low).
4. RP(C). (low to high).
5. prefer, if exist, Clauses C where: $C \cap G \neq \emptyset$.
6. If all previous parameters equal – choose arbitrarily.

The first step in our algorithm is to generate an initial belief-state formula φ_I from the BN describing the initial belief state $\mathcal{N}_{b_I}$. φ_I is constructed as follows: for every multi-value variable X whose initial value is uncertain, φ_I contains a *One-Of* clause C_X with one literal for every value of X possible (i.e., which has probability > 0). In addition, if $Pr(X = x|Parents(X) = y) = 0$ (i.e., $X = x$ is not possible when $Pa(X)$ have value y, an additional *Or* clause expressing $Pa(X) = y \rightarrow \neg(X = x)$ is added. Note that this formula can be generated by a single top down traversal of the BN, requiring linear time. The following is immediate:

Lemma 2 $\varphi_I \models w$ *iff* $b_{\pi_I}(w) > 0$.

Next, we seek to simplify φ_I by ignoring a set of states with probability at most $1 - \theta$. Thus, at this stage we operate on the structure: $P = (V, A, \varphi_I, \mathcal{N}_{b_I}, G, \theta)$ which denotes the original CPP together with the generated initial state formula. Algorithm 1 describes the high-level structure of RBPP – our *relevance-based probabilistic planner.* We denote the formula expressing the set of states valid after the simplification of φ_I as ψ_I. We also use: $b_{\pi_I}(\varphi)$ to denote $\Sigma_{\varphi \models w} b_{\pi_I}(w)$.

The main element – relevance-based analysis – is carried out by the *Restrict* procedure (Algorithm 2) which determines which initial states to ignore. This analysis is heuristic, and it is strongly motivated by the notions of conformant width and relevance developed by (Palacios and Geffner 2009) in the context of conformant planning.

First, *Restrict* sorts the clauses. Then, at each iteration it selects the first non-unit clause (i.e., a clause expressing uncertainty) in the list and restricts it by dropping some literal(s) using Algorithm 4. This results in a smaller clause satisfied by fewer states, hence ignoring those states satisfied by the dropped literal. The procedure is repeated until no further restrictions are possible given the probability bound. We note that each restriction is made only after we check (using a standard BN software package) that the probability of the removed initial states does not exceed $1 - \theta$.

Algorithm 4 RESTRICT-CLAUSE (C, ψ_I, P)

while $(|C| > 1 \wedge b_{\pi_I}(\psi_I) \geq \theta)$ **do**
  $C' \leftarrow C$;
  choose next proposition $p \in C'$ by the following order:
1. $p \notin g$.
2. $b_{\pi_I}(\psi_I \wedge \neg p) \geq \theta$.
3. Prefer not removing $p \in DRC(C)$ before other propositions.
4. $b_{\pi_I}(p)$ - The initial probability of p. (low to high).
5. If all previous parameters equal - choose arbitrarily.

  $C' \leftarrow C \setminus p$;
  $\psi_I \leftarrow \psi_I \setminus C \cup C'$;[1]
end while
return ψ_I;

Restrict maximally restricts a clause before moving on to the next clause. We have found this priority to be most useful, as there appears to be greater advantage to making a single clause much smaller than to making multiple clauses smaller. Clauses are sorted based on a prioritized list of parameters, as shown in Algorithm 3. These parameters attempt to assess the impact of a clause on the difficulty of

[1]For clarity, we use set notation, treating ψ as a set of clauses and treating clauses as a set of literals. Here, p is removed from C to obtain C', and C is replaced by C' to obtain the updated ψ_I.

solving a problem. Future work could improve this by adapting the parameters and their priority to take into account their effect on the underlying conformant planner.

Choice of Clauses Our analysis is strongly motivated by the notion of *conformant relevance* (Palacios and Geffner 2009) (PG), and many of our definitions are adaptation of their concepts to the probabilistic case. Relevance is a transitive relation between facts expressing whether uncertainty about the value of one fact affects our knowledge about the value of the other. The relation propagates only through conditional effects of actions. Our relevance analysis focuses on facts relevant to subgoals, and while it is possible to consider also facts relevant to preconditions, we found the overhead of these computations costly.

Definition 1 (Conformant Relevance (PG 2009))
1. p is relevant to p.
2. p is relevant to q if there exist an action $a \in A$ with conditional effect $C \rightarrow q$ and $p \in C$.
3. If p is relevant to r and r is relevant to q then p is relevant to q.
4. p is relevant to q if p is relevant to $\neg r$ and r is relevant to $\neg q$.

We extend this relation to relevance between an initial non-unit clause and a (goal) fact as follows:

Definition 2 (probabilistic clause relevance) *A non unit clause $\varphi \in \varphi_I$ is considered relevant to a proposition p if $\exists q \in \varphi$ s.t $q \in rel(p)$.*

PG consider φ relevant to p only if all propositions of φ are in $rel(p)$. The latter yields fewer relevant propositions but cannot be extended to the probabilistic case. Suppose, for example, that we have *One-Of(p,q)* in φ_I. Assume p is relevant to some goal fact g but q is not. Assume also that $b_{\pi_I}(p) = 0.5$ and $\Theta = 0.4$. In the conformant case, this clause may be ignored, but in CPP, because we do not need to succeed with certainty, we must be take into account the possibility to plan from all p states, even though planning from all q is infeasible.

Definition 3 (relevant clauses set) *$\forall g \in G : rel(g)$ contains all the clauses from φ_I relevant to g;*

$rel(g)$ expresses the maximal amount of initial uncertainty relevant to a specific goal fact. $Max_{g \in G}\{|rel(g)|\}$ is closely related to the notion of *conformant width* introduced by PB. They show that the complexity of their reduction-based planning algorithm grows exponentially with this parameter. This effect on the problem's complexity stems from the fact that the solver must potentially consider initial states (or more accurately, initial state-sets) that correspond to all possible assignments to $rel(g)$ variables to compute the current probability of g, in the probabilistic case, and to determine whether g is valid, in the standard conformant case. And while this value was suggested by PG in the context of their analysis of the *T-0* planner, it appears relevant to the performance of other conformant planners, such as CFF, as our empirical analysis shows.

There are two technical differences between our notion of $rel(g)$ and PG's conformant width: First, we use probabilistic clause relevance, explained earlier. Second, PG compute the closure of $rel(g)$, called $C_I(L)^*$. We omit the costly computation of $C_I(L)^*$, using $rel(g)$ instead. For the purpose of our heuristic, this provides us with a better tradeoff, and in fact, $rel(g) = C_I(L)^*$ in almost all experimented benchmarks.

Given the above theoretical and empirical justifications for relation between $rel(g)$ and problem hardness, our most important parameter for assessing clauses is $RL(C)$:

Definition 4 (clause relevance level) $RL(C) = MAX\{|rel(g)|\}_{g \in G \wedge C \in rel(g)}$.

Relevance sets can also be used to understand how many goal facts are influenced by the uncertainty expressed by a clause. Restricting clauses that affect many goal facts is likely to be better, motivating the following definition:

Definition 5 (clause radius) $Rad(C) = \#g \in G \; s.t \; C \in rel(g)$.

Example: Consider the *logistics* domain with 10 cities,10 trucks, 10 packages, and one plane. Initially, there is one package and truck per city. There are 10 sub-goals specifying the final location of each package. Actions have no pre-conditions, but only conditional effects. The initial location of trucks and the plane are unknown. Each truck can be in one of three possible locations in a city. Each truck's location uncertainty is relevant only to one sub-goal (corresponding to the package in this city). Uncertainty regarding the plane is relevant to all goals. $RL(C) = 2$ for all clauses, while the radius of the clause expressing the plane's uncertainty is 10, and that of other clauses is 1. Restricting this clause will prevent the conformant planner from repeatedly reasoning about the plane's location. It is likely to yield a simpler plan, too, as there is no need for extra flights that will ensure a known location to the plane.

Relevance analysis ignores the initial probabilistic distribution. $PI(C_X)$ takes this information into account. It is a rough and tractable estimate of the effect evidences on X (e.g restrictions on C_X's literals) might have on our knowledge regarding C_Y's literals.

Definition 6 (Probabilistic Influence) *Let $X \in \mathcal{N}_{b_I}$ be the corresponding node of a clause C_X. We define:* $RelChildren(X) = \{Y | X \in Parents(Y) \wedge \exists g \in G \; s.t \; C_Y \in rel(g)\}$
$PI(C_X) = |RelChildren(X)|$.

Potentially, restricting C_X might induce immediate restrictions on C_Y which, in turn, reduces the relevant uncertainty in the resulting conformant problem. Ideally, our definition should have been transitive, but again, to limit the cost of the restriction heuristic, we consider immediate parents only. In order to exploit this information and monitor such effects, after each restriction of a clause C_X s.t $PI(C_X) > 0$, we query $\mathcal{N}_{b_I}$ to check whether any of the literals in C_Y for each $Y \in Rel - Children(X)$ has probability 0. In that case we restrict C_Y accordingly.

Example (continued): Suppose that the plane's initial location distribution is dependent on the initial distribution on the weather in some city *city*. Other than that, the weather does not influence other problem variables and does not appear in action descriptions. The relevance analysis will not

	$\theta = 0.25$				$\theta = 0.5$				$\theta = 0.75$			
	t/l				t/l				t/l			
Task	PFF	PCBP	RBPP[T-0]	RBPP[CFF]	PFF	PCBP	RBPP[T-0]	RBPP[CFF]	PFF	PCBP	RBPP[T-0]	RBPP[CFF]
Safe-uni-70	2.3/18	**0.02/18**	0.07/18	0.1/18	5.5/35	**0.05/35**	0.1/35	1.13/35	9.6/53	6.4/70	**0.07/53**	1.44/**53**
Safe-cub-70	0.07/5	**0.03/5**	0.09/5	0.06/5	1.62/**12**	**0.04**/21	0.07/**12**	0.07/**12**	2.92/21	**0.04**/21	0.08/**12**	0.08/**12**
NC-Safe-uni-70	**1.15/5**	1.41/12	1.39/**5**	**1.13/5**	2.28/**12**	1.47/**12**	**1.39/12**	2.31/**12**	4.92/21	1.56/20	1.39/20	**1.3/20**
Cube-uni-corner-15	3.44/**26**	OOT	**0.11**/36	2.37/36	4.7/**33**	OOT	**0.07**/42	4.34/44	6.9/**38**	OOT	**0.07**/42	4.29/42
Cube-cub-corner-15	1.86/**20**	OOT	**0.06**/37	2.93/37	4.2/**28**	OOT	**0.09**/37	3.01/37	5.03/**33**	OOT	**0.1**/40	3.32/40
Cube-uni-center-15	OOT	OOT	**6.15/70**	OOT	OOT	OOT	**5.27/68**	OOT	OOT	OOT	**6.94/56**	OOT
Cube-cub-center-15	OOT	OOT	**6.02/64**	OOT	OOT	OOT	**5.72/61**	OOT	OOT	OOT	**3.87/61**	OOT
NC-cube-cub-15	5.22/**20**	OOT	**2.93**/37	10.42/37	2.94/**27**	OOT	**0.86**/37	2.63/37	5.74/**33**	OOT	**1.29**/37	2.37/37
NC-cube-uni-15	2.28/**26**	OOT	**0.08**/34	**0.08**/34	13.98/47	OOT	**3.06/34**	4.00/34	NP	NP	NP	NP
Push-Cube-uni-15	7.39/50	2.08/**42**	1.21/66	**0.06**/57	27.28/66	2.33/**45**	1.79/72	**0.09**/63	55.21/74	2.28/**46**	1.81/74	**0.11**/63
Push-Cube-cub-15	0.08/**15**	1.48/28	0.07/42	**0.05**/33	0.09/**17**	3.88/37	1.26/58	**0.05**/48	0.09/**18**	2.09/37	1.34/59	**0.09**/48
Bomb-50-50	0.01/0	0.01/0	0.01/0	0.01/0	0.1/**16**	9.02/50	**0.09**/50	6.13/50	0.2/**36**	8.4/90	**0.07**/50	6.22/50
Bomb-50-10	0.01/0	0.01/0	0.01/0	0.01/0	2.89/**22**	8.4/90	**0.04**/90	1.43/**22**	5.74/63	8.4/90	**0.04**/90	5.59/63
Bomb-50-5	0.01/0	0.01/0	0.01/0	0.01/0	1.94/**27**	8.6/95	**0.04**/95	1.83/**27**	8.02/**63**	9.14/95	**0.04**/95	7.07/67
Bomb-50-1	0.01/0	0.01/0	0.01/0	0.01/0	2.21/**31**	5.02/49	**0.03**/100	1.88/**31**	10.12/**71**	4.8/74	**0.05**/100	7.9/**71**
10-Log-2	3.73/**72**	OOT	4.51/85	**2.79**/77	3.17/79	OOT	4.97/84	**2.56/77**	**2.46/80**	OOT	3.13/**80**	5.3/87
10-Log-3	7.47/64	OOT	**5.16/57**	2.83/58	35.4/98	OOT	8.98/**77**	**3.23/77**	8.91/99	OOT	24.28/123	**4.12/91**
10-Log-4	8.35/75	OOT	8.41/**47**	**4.59/47**	34.2/81	OOT	13.29/78	**5.38/70**	12.09/**95**	OOT	36.53/111	**7.43/95**
15-Log-4	OOT	OOT	OOT	**15.02/98**	OOT	OOT	OOT	**28.31/138**	OOT	OOT	OOT	**32.5/154**
2-planes-log-3	19.81/84	OOT	14.09/**64**	**7.62/69**	57.98/97	OOT	31.24/104	**18.72/89**	**10.12**/112	OOT	229/134	13.7/**109**
2-planes-log-4	OOT	OOT	23.19/62	**12.64/57**	OOT	OOT	74.03/101	**17.91/87**	OOT	OOT	522/145	**47.52/107**
2-planes-C	OOT	OOT	OOT	**3.65/83**	OOT	OOT	OOT	**4.92/83**	OOT	OOT	OOT	**14.06/108**
grid-uni-2	**0.07/21**	OOT	4.33/75	3.15/47	**1.35/48**	OOT	OOT	3.18/53	6.11/69	OOT	OOT	**5.41/66**
grid-uni-3	16.01/**76**	OOT	**6.22**/86	41.14/86	15.8/**89**	OOT	**5.93**/96	127/103	82.24/123	OOT	**7.26/102**	132/105
grid-uni-4	28.15/**96**	OOT	**6.78**/111	134.21/115	**51.58/111**	OOT	OOT	247.58/117	**50.80/115**	OOT	OOT	721/146
grid-cub-3	OOT	OOT	18.14/40	**9.47/30**	OOT	OOT	31.14/43	**16.11/35**	OOT	OOT	**27.71/64**	39.42/76
grid-cub-4	OOT	OOT	**10.8/77**	83.84/92	OOT	OOT	**9.37/78**	114/92	OOT	OOT	**14.95/85**	200/106
Rovers-3	0.04/14	0.06/19	**0.02/12**	**0.02/12**	0.05/17	0.06/19	**0.02/12**	**0.02/12**	0.06/**18**	0.06/25	**0.02**/20	0.03/24
Rovers-7	5.72/65	4.12/54	**0.04**/47	**0.04/41**	5.48/75	4.14/54	OOT	**0.07/56**	6.55/83	10.74/88	OOT	**2.09/72**
C-Rovers-PP-7	10.54/65	OOT	OOT	**1.1/41**	15.4/75	OOT	OOT	**1.12/56**	39.1/77	OOT	OOT	**2.23/72**
C-Rovers-PPP-7-3-ID	3060/68	OOT	OOT	**3.31/47**	OOT	OOT	OOT	**11.34/71**	OOT	OOT	OOT	**127.53/96**
C-Rovers-PPP-NC-7	30.11/67	OOT	OOT	**3.12/41**	46.3/79	OOT	OOT	**4.14/57**	NP	NP	NP	NP

Table 1: Empirical results. t: time in seconds. l: plan length. Entries marked *OOT* means the search did not return after 30 minutes. 'NP'-no plan for this goal probability exists.

identify the "weather" clause C_w as having any importance. However. since $PI(C_w) = 1$, while for all other clauses $PI(C) = 0$, the algorithm might choose to restrict C_w, setting the weather in *city* to *extreme*. Given this value there is 0 probability for the plane to be in *city* with the effect of restricting the clause expressing the plane's location uncertainty – C_p. This can help us in cases where θ and the initial distribution limits our ability to directly restrict C_p but allow the restriction of C_w.

Restrictions come with a cost: the probability mass we lose. We wish to restrict as many propositions as possible, so we prefer restrictions that carry lower probability "cost".

Definition 7 (clause restriction potential) *RP(C)* = $Min\{b_{\pi_I}(p) \mid p \in C\}$. *(By $b_{\pi_I}(p)$ we mean The initial probability of p).*

$RP(C)$ improves our ability to identify, in advance, clauses that are potentially more attractive for restriction. As such, it serves as a good tie-breaker. Note that if the node corresponding to C in $\mathcal{N}_{b_I}$ is not barren, this parameter is ignored.

Clause Restriction First, we make sure no facts which can negatively affect the completeness or the quality of solution are removed. Thus, goal facts which must be achieved are not removed. In addition, sometimes, only a subset of the facts are responsible for making the clause relevant to a goal and removing them can make the goal unreachable. We define the following set:

Definition 8 (Direct Relevance causes) *DRC(C)* = $\{p \in C \mid \exists g \in G$: *p is relevant to g*$\}$.

If, for a clause C: $DRC(C) \subsetneq C$ a plan might not exist from all assignments to C. Thus, we don't remove facts from $DRC(C)$ unless no other option exists.

Finally, we prefer to remove facts whose initial probability is smaller, to leave more probabilistic mass for farther restrictions but once again, if the node corresponding to $C's$ in $\mathcal{N}_{b_I}$ is not barren, the latter parameter is ignored.

Properties

Soundness: Our algorithm is sound if the underlying CP solver is sound. This follows from Lemma 1, provided we ensure that the probability of our new initial state $\geq \theta$. Each clause restriction is equivalent to ignoring a possible value of a variable. To accept such a restriction, we compute the probability of the set of ignored states, ensuring it does not exceed $1 - \theta$. When multiple values are ignored, we use the standard probabilistic semantics to compute their aggregated weight. (E.g., if we ignore $\neg p$ and then $\neg q$, we compute the weight of $\neg p$, and add the weight of $\neg q \wedge p$).

Completeness: Our algorithm is incomplete because we do not attempt to systematically examine all possible restrictions with weight $\leq 1-\theta$. Conceptually, it is a simple matter to add an outer loop that will make the algorithm complete, (e.g., as used by some greedy algorithms to provide theoretical completeness). Practically (and theoretically) this is a potentially super-exponential algorithm and we see little value in it (unlike soundness), as it will not help us scale up any better. As our experiments demonstrate, our algorithm does scale up better than earlier method. Moreover,

as the problem is at least as hard as CP, which is *PSPACE-COMPLETE* (Haslum and Jonsson 1999) even with deterministic actions, ensuring completeness seems ill advised.

Complexity: There are three elements contributing to the complexity to our algorithm: the algorithm for selecting possible initial-state restrictions, the BN queries, and the CP solver. In theory, there are super-exponentially many possible restrictions to the initial state and the computation of each one's probability can be NP-hard. In practice, we use a heuristic low-order polynomial time algorithms to identify promising restrictions. For this we pay by sacrificing completeness. To check the validity of restrictions, we compute marginal distribution of variables in a BN, which is NP-hard (Cooper 1990). In practice, the queries we seek to compute are very simple, and are not likely to be the bottle-neck even when we go beyond current CPP benchmarks which feature very simple initial state BNs. Solving CP is a difficult problem (Haslum and Jonsson 1999), and this appears to be the more significant bottleneck in practice. This, of course, is to be expected, as we are solving a problem that is at least as difficult as CP. Thus, we have focused on making the reduction process simple and fast.

Empirical Evaluation

We implemented the algorithms and experimented with them on a variety of domains taken from *PFF* repository and the 2006 IPPC, as well as new, modified versions of these domains. Beyond our tools, we used a modification of PG's $cf2cs$ code for the relevance analysis and *NORSYS NETICA java api* program for the Bayesian net creation and reasoning. As the underlying conformant planner for RBPP we used both *T-0* and *CFF*. GC[LAMA] (Nguyen et al. 2012) is unable to handle most of the domains we tested on. The two RBPP variants were compared with state-of-the-art CPP planners: *PFF* and *PCBP*, the results are presented in Table 1. Each task was tested with three different θ value and two different initial state distributions: uniform (*uni*) and cubic (*cub*). Results show dominance of *RBPP[T-0]* on almost all benchmarks, in many cases by an order of magnitude. An interesting observation is that in most cases *RBPP[CFF]* performs better than *PFF* which is a specially designed extension of Conformant-FF's. This demonstrates the effectiveness of the compilation approach over direct probabilistic reasoning, as used by *PFF*. On some of the simplest tasks such as *safe-70 PCBP* has a small advantage due to the overhead of relevance analysis relative to solution time in this problem. In more complex problems, such as *cube-15-corner* (an agent can initially be in any of the cube's 15^3 locations and needs to navigate to a corner) our planners run-time dominance is clear, although *PFF*'s plans are shorter. When, in the same setting, the agent needs to plan to the center of the cube, *RBPP[T-0]* produces good plans in a matter of seconds, while all other planners fail. Similar performance is seen in the *push-cube* domain (Taig and Brafman 2013). Here, a player must push a ball with unknown initial position to some location on a grid. It can select *(position, direction)* pairs, and if the ball is in *position*, it moves in *direction*. Solving this problem requires many actions, and *RBPP* is the clear winner. The *m-logistics-n* problem refers to logistics with m packages and m cities of size n. Here, too, the various *RBPP* versions are faster and generate shorter plans, when we set $m = 15$ only *RBPP[CFF]* scales up. The *2-planes* variation of logistics has 2 planes, instead of 10, as well as uncertainty on their initial location. Both *rovers* and *grid* have large probabilistic width, and our relevance analysis is crucial. Results here are mixed. As the uncertainty grows (*grid-4*,*rovers-7*) both *PFF* and *RBPP[T-0]* fail to scale up. *RBPP[CFF]* is the only planner that manages to solve all tasks, typically dominating other planners.

We also experimented on domains with no conformant plan (marked *NC*). For example, we modified *cube* such that some transitions between adjacent locations are blocked so the agent cannot reach the goal location if initially located beyond the blocked path. These experiments demonstrate the ability of RBPP's preprocessing phase to select initial states from which the goal is reachable.

Another important line of experiments is on the problems marked with *C*. In these problems the initial state distribution is more complex and involves dependencies. *2-planes-C* reflects the example given in this paper. In rovers-pp, the visibility of an objective from a waypoint depends on whether or not a rock sample is located at the waypoint. The probability of visibility is much higher if the latter is not the case. Rovers-PPP extends RoversPP by introducing the need to collect data about water existence. Each soil sample has a certain probability to be wet. For communicated sample data, an additional operator tests whether the sample was wet. The probability of being wet depends on sample location. In the instance marked "ID," one of the 3 samples must be wet, increasing dependencies to a level that only *RBPP[CFF]* can handle. In the instance marked *NC*, there is no guarantee that one of the 2 samples is wet, and thus, there is no conformant plan. Here, *RBPP[CFF]* scales better than *PFF* by an order of magnitude.

Conclusion and Future work

We presented a new approach to CPP based on pre-planning analysis of the initial belief state. We use relevance analysis and heuristics that attempt to identify states that would be most beneficial to ignore in planning, while making sure that the remaining states are sufficiently likely, and a conformant plan exists for them. The empirical evaluation shows this method to be currently the strongest CPP algorithm with better coverage and scaling than previous approaches. While theoretically incomplete, in practice, this method handles more problems, and its performance clearly motivates the potential of this reduction approach.

Presently, we make no attempt to automatically select the underlying conformant solver. In future work, we intend to attempt to identify problem features that can help predict which planner will perform better, and to use these within a portfolio-based solver. In addition, we intend to extend the approach presented here to handle CPP domain with stochastic actions. A first step is to simply consider deterministic version of these actions, and to be able to verify that the solution addresses a sufficiently large probability mass.

Acknowledgments The authors were supported in part by ISF grant 933/13 and the Lynn and William Frankel Center for Computer Science.

References

Albore, A., and Geffner, H. 2009. Acting in partially observable environments when achievement of the goal cannot be guaranteed. In *ICAPS'09 Planning and Plan Execution for Real-World Systems Workshop*.

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *IJCAI*, 1623–1628.

Cooper, G. F. 1990. The computational complexity of probabilistic inference using bayesian belief networks. *Artif. Intell.* 42(2-3):393–405.

Davis-Mendelow, S.; Baier, J. A.; and McIlraith, S. A. 2013. Assumption-based planning: Generating plans and explanations under incomplete knowledge. In *AAAI*.

Domshlak, C., and Hoffmann, J. 2007. Probabilistic planning via heuristic forward search and weighted model counting. *J. Artif. Intell. Res. (JAIR)* 30:565–620.

Haslum, P., and Jonsson, P. 1999. Some results on the complexity of planning with incomplete information. In *ECP*, 308–318.

Hoffmann, J., and Brafman, R. I. 2005. Contingent planning via heuristic forward search witn implicit belief states. In *ICAPS*, 71–80.

Hoffmann, J., and Brafman, R. I. 2006. Conformant planning via heuristic forward search: A new approach. *Artif. Intell.* 170(6-7):507–541.

Nguyen, H.-K.; Tran, D.-V.; Son, T. C.; and Pontelli, E. 2011. On improving conformant planners by analyzing domain-structures. In *AAAI*.

Nguyen, H.-K.; Tran, D.-V.; Son, T. C.; and Pontelli, E. 2012. On computing conformant plans using classical planners: A generate-and-complete approach. In *ICAPS*.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *JAIR* 35:623–675.

Taig, R., and Brafman, R. I. 2013. Compiling conformant probabilistic planning problems into classical planning. In *ICAPS*.

Tran, D.-V.; Nguyen, H.-K.; Pontelli, E.; and Son, T. C. 2009. Improving performance of conformant planners: Static analysis of declarative planning domain specifications. In *PADL*, 239–253.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, 352–.

Structured Possibilistic Planning using Decision Diagrams

Nicolas Drougard and **Florent Teichteil-Königsbuch** and **Jean-Loup Farges**
{Nicolas.Drougard,Florent.Teichteil,Jean-Loup.Farges}@onera.fr
Onera – The French Aerospace Lab
2 avenue Edouard Belin
31055 Toulouse Cedex 4, France

Abstract

Possibilistic POMDPs (π-POMDPs) are well-suited to planning under uncertainty with partial observability when transition, observation and reward functions are not precisely known. In this qualitative framework, functions defining the model as well as intermediate calculations are valued in a fixed and finite possibilistic scale $\mathcal{L}$. Contrary to their probabilistic counterparts, π-POMDPs reduce to belief MDPs with *finite* states, thus π-MDP algorithms *as is* handle partially observable problems too. In this paper, we propose the first study of factored representations of π-(PO)MDPs in order to solve large planning problems under *imprecise* uncertainty. Building upon the SPUDD algorithm for solving factored MDPs, we conceived a symbolic algorithm named PPUDD for solving large factored π-(PO)MDPs. Whereas the size of SPUDD's ADD leaves may grow exponentially large since their values are real numbers aggregated through additions and multiplications, PPUDD's ones always remain in the finite scale $\mathcal{L}$ via $\min$ and $\max$ operations only. Finally, we present a sound transformation from factored mixed-observable possibilistic problems with both hidden and visible state variables to fully observable ones, on which PPUDD is run. Experiments with possibilistic and probabilistic versions of the same benchmarks show that PPUDD runs significantly faster than its probabilistic counterpart while providing high-quality policies.

Introduction

Most sequential decision making under uncertainty problems can be easily expressed in terms of Markov Decision Processes (MDPs) (Bellman 1957; Puterman 1994). Partially Observable MDPs (POMDPs) (Smallwood and Sondik 1973) have been developed to take into account situations where the system's state is not totally visible to the agent. Yet, this framework often encounters difficulties in computing optimal policies: dynamic programming algorithms like incremental pruning (Cassandra, Littman, and Zhang 1997) can only solve small POMDPs. Many approximation algorithms have been proposed to speed up computations while controlling the quality of resulting policies (Pineau, Gordon, and Thrun 2003; Smith and Simmons 2004; Hanna Kurniawati 2008). In this paper, we proceed quite differently starting with an approximated model and exactly solving it.

The π-POMDP (Possibilistic Partially Observable Markov Decision Process) framework was first introduced in (Sabbadin 1999): this possibilistic counterpart of classical (probabilistic) POMDPs is based on Possibility Theory (Dubois and Prade 1988) and more specifically on Qualitative Decision Theory (Dubois, Prade, and Sabbadin 1998; Dubois and Prade 1995). A possibility distribution, as all plausibility distributions, classically models imprecision or lack of knowledge about the uncertainty model. Using Possibility Theory instead of Probability Theory in POMDPs necessarily leads to an approximation of the initial probabilistic model. But this approach benefits from computations on *finite* belief state spaces, whereas probabilistic POMDPs tackle *infinite* ones. What is lost in precision of the uncertainty model is gained in computational complexity.

However, the belief space of a π-POMDP still exponentially grows with the number of states and only small problems can be solved in practice. Possibilistic Mixed-Observable MDPs (π-MOMDPs) proposed by (Drougard et al. 2013) take advantage of the specific structure of problems where parts of the system state are fully observable. This extension of π-MDPs and π-POMDPs is essentially the counterpart of probabilistic MOMDPs introduced by (Ong et al. 2010; Araya-Lòpez et al. 2010).

Nevertheless, existing works on π-(MO)MDPs do not totally profit from the problem's structure, i.e. visible and hidden parts of the state can be themselves factored into many state variables, which are yet flattened by current possibilistic approaches. In probabilistic settings, factored MDPs and Symbolic Dynamic Programming (SDP) (Boutilier, Dearden, and Goldszmidt 2000; Hoey et al. 1999) have been extensively studied in order to reason directly at the level of state variables rather than flattened states. The famous algorithm SPUDD (Hoey et al. 1999) solves factored probabilistic MDPs by using symbolic functional representations of value functions and policies in the form of Algebraic Decision Diagrams (ADDs) (Bahar et al. 1997), which compactly encode real-valued functions of boolean variables. ADDs are directed acyclic graphs whose nodes are instantiated state variables and leaves are the function's values. Instead of updating states individually at each iteration of

the algorithm, states are aggregated within ADDs and operations are symbolically and directly performed on ADDs.

However, SPUDD and all its variants suffer from an exponential growth of ADDs in the worst case as computations go on: expectation, amongst other operations, involves additions and multiplications of real values (probabilities and rewards), creating other values in-between, in such a way that the number of ADD leaves may eventually equal the size of the state space, which is exponential in the number of state variables. On the contrary, the qualitative possibilistic formalism operates only over a *finite* possibilistic scale $\mathcal{L}$, but not over a dense space like in Probability Theory. Moreover, only $\max$ and $\min$ operations are involved, which implies that all manipulated values remain in $\mathcal{L}$ from the beginning to the end. Therefore, our work is motivated by the simple observation that *symbolic operations with* ***possibilistic*** *MDPs would necessarily limit the size of ADDs*, since the number of their leaves would be at most equal to the cardinal of the possibilistic finite scale $\mathcal{L}$, which is generally far smaller than the number of states. As a result, operating ADDs in the possibilistic framework would behave much like manipulating Binary Decision Diagrams (BDDs) (Bryant 1992), which are more compact than ADDs.

This paper begins with an overview of possibilistic models for sequential decision-making: π-MDPs, π-POMDPs, and π-MOMDPs. As possibility distributions are in a finite scale $\mathcal{L}$, we will show that all these models reduce to π-MDPs with finite states, so that an algorithm for π-MDPs can solve all of them *as is*. This is a major difference with probabilistic sequential decision-making, where partial observable problems give rise to continuous belief MDPs, which belong to a higher complexity class. Thus, we will present our first contribution: a Symbolic Dynamic Programming algorithm for solving factored π-MDPs and manipulating symbolic possibilistic functions, named Possibilistic Programming Using Decision Diagram (PPUDD). Our second contribution is a principled reduction of any factored π-POMDP or π-MOMDP to factored (finite-state) π-MDPs, assuming a structural assumption on the structured problem. In brief, this assumption means that all post-action variables are independent given the past. Finally, we assess our algorithms with two experimental comparisons: PPUDD against SPUDD (Hoey et al. 1999) for solving possibilistic and probabilistic versions of the same benchmarks under total observability; PPUDD against SARSOP-based probabilistic solver (Hanna Kurniawati 2008; Ong et al. 2010) for solving mixed-observable problems, and a POMDP solver using ADDs, symbolic-HSVI (Sim et al. 2008).

Background

Markov Decision Processes (MDPs) are commonly used to model problems where an agent and its environment (called the *system*) are changing over time (represented by $\mathbb{N}$). As suggested by its name, a MDP is adapted to systems whose successive states ($s \in \mathcal{S}$) have a Markovian evolution which depends on decisions ($a \in \mathcal{A}$) chosen at each time step ($t \in \mathbb{N}$). Classically, this model is better known and used within the Probability Theory framework. However, possibilistic counterparts of MDPs have been studied in the literature, and are well-suited to problems whose uncertainties or rewards are imprecisely known to the decision maker, or which are too complex to be solved by probabilistic solvers. Indeed, possibilistic MDPs can be viewed as an approximated model of (probabilistic) MDPs, where probabilities and rewards are replaced by qualitative statements that lie in a finite numeric scale (as opposed to continuous ranges in the probabilistic framework), resulting in simpler computations.

Possibilistic Markov Decision Processes

The possibilistic MDP (π-MDP) model has been first formulated in (Sabbadin, Fargier, and Lang 1998) and (Sabbadin 2001). Let us define $\mathcal{L} = \{0, \frac{1}{k}, \ldots, \frac{k-1}{k}, 1\}$, the fixed possibilistic scale ($k \in \mathbb{N}^*$). A *possibility distribution* over $\mathcal{S}$ is a function $\pi : \mathcal{S} \to \mathcal{L}$ which verifies the possibilistic normalization: $\max_{s \in \mathcal{S}} \pi(s) = 1$. This distribution ranks plausibilities of events: $\pi(s) < \pi(s')$ means that s is less plausible than s'. A π-MDP process starts in an initial state $s_0 \in \mathcal{S}$. If the system is in state $s_t \in \mathcal{S}$ at time step $t \in \mathbb{N}$ and if the agent chooses action $a_t \in \mathcal{A}$, then next state $s_{t+1} \in \mathcal{S}$ is reached with *possibility degree* $\pi(s_{t+1} \mid s_t, a_t) \in \mathcal{L}$. This defines the transition function T^π as the *possibility* of reaching $s' \in \mathcal{S}$ conditioned on current state and action $(s, a) \in \mathcal{S} \times \mathcal{A}$: $T(s, a, s') = \pi(s' \mid s, a)$. Contrary to probabilistic MDPs, transitions in the possibilistic framework are labeled by **qualitative** measures of their occurrences. Finally, a preference distribution $\mu : \mathcal{S} \mapsto \mathcal{L}$ models the **qualitative** goals of the agent: $\mu(s) < \mu(s')$ means that s is less preferable than s'. The tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{L}, T^\pi, \mu \rangle$ defines a π-MDP.

A *decision rule* is a function $\delta : \mathcal{S} \to \mathcal{A}$. A policy is a sequence of decision rules: $(\delta) = (\delta_0, \delta_1, \ldots)$. Given a policy (δ) for a π-MDP process, $a = \delta_t(s)$ is the action selected by the agent in state s at time t. A *trajectory* is a sequence of states: $\tau = (s_1, s_2, \ldots)$. As the process $(s_t)_{t \geqslant 0}$ is a Markov chain, the possibility degree of the finite (n-size) trajectory $\tau = (s_1, s_2, \ldots, s_n)$ starting in s_0 and under the n-size policy $(\delta) = (\delta_0, \ldots, \delta_{n-1})$ is:

$$\Pi(\tau \mid s_0, (\delta)) = \min_{i=0}^{n-1} \pi(s_{i+1} \mid s_i, \delta_i(s_i)).$$

The preference of a trajectory is defined as the preference of the last state: if $\tau = (s_1, \ldots, s_n)$, $M(\tau) = \mu(s_n)$. The set of all n-size trajectories is denoted by $\mathcal{T}_n$. Besides, the set of all n-size policies is denoted by Δ_n and $\Delta = \cup_{n \geqslant 0} \Delta_n$ is the set of all policies. Finally, $\#\delta$ is the size of policy $(\delta) \in \Delta$. These notations are useful to define the infinite-horizon value of a policy (δ) starting in s_0:

$$u(s_0, (\delta)) = \max_{\tau \in \mathcal{T}_{\#\delta}} \min \{ \Pi(\tau \mid s_0, (\delta)), M(\tau) \}.$$

The π-MDP infinite-horizon problem consists in finding a policy in Δ which maximizes this criterion, named value function: $u^*(s) = \max_{(\delta) \in \Delta} u(s, (\delta))$. As proved in (Drougard et al. 2013), there exist an optimal policy (which is stationary) that can be found by possibilistic dynamic programming (see Algorithm 1), **if** there exists an action $\overline{a}$ such that $\pi(s' \mid s, \overline{a}) = \mathbb{1}_{s,s'} = 1$ if $s = s'$, and 0 otherwise. Action $\overline{a}$ is similar to the discount factor in probabilistic MDPs,

which allows dynamic programming to converge under infinite horizon to a unique fixed point corresponding to stationary history-dependent policies. This hypothesis is yet not a constraint in practice: in the returned optimal policy (δ^*), action $\overline{a}$ is only used for goals whose preference degree is greater than possibility degree of transition to better goals. Finally, Algorithm 1 is guaranteed to converge in at most $\#\mathcal{S}$ iterations.

Algorithm 1: π-MDP Value Iteration Algorithm

for $s \in \mathcal{S}$ **do**
  $u^*(s) \leftarrow 0_{\mathcal{L}}$; $u^c(s) \leftarrow \mu(s)$; $\delta^*(s) \leftarrow \overline{a}$;
while $u^* \neq u^c$ **do**
  $u^* \leftarrow u^c$;
  for $s \in \mathcal{S}$ **do**
    $u^c(s) \leftarrow \max_{a \in \mathcal{A}} \max_{s' \in \mathcal{S}} \min \{ \pi(s' \mid s, a), u^*(s') \}$;
    if $u^c(s) > u^*(s)$ **then**
      $\delta(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} \max_{s' \in \mathcal{S}} \min\{\pi(s' \mid s, a), u^*(s')\}$;
return (u^*, δ^*) ;

Possibilistic Mixed-Observable MDPs

The possibilistic MOMDP model (Drougard et al. 2013) further complicates the previous one, by considering problems where part of the state variables are partially observable, the other ones being totally observable as in factored π-MDPs. The agent should estimate the system's state using observations $o \in \mathcal{O}$, which it receives from the environment at each time step. These observations allow him to maintain a *belief state* in the form of a possibilistic distribution over hidden states, which is updated at each time step using possibilistic Bayes' rule. In fact, π-MOMDPs include π-MDPs as special cases when all state variables are totally observable. On the opposite side, when all state variables are partially observable, π-MOMDPs reduce to possibilistic Partially Observable MDPs (π-POMDPs), first studied by (Sabbadin 1999). Whereas we presented π-MDPs for pedagogical reasons, we will not explicitly define π-POMDPs, since they are special cases of π-MOMDPs.

In the probabilistic framework, Mixed-Observability has been set up and studied in (Ong et al. 2010) and (Araya-Lòpez et al. 2010). These works consist in formalizing situations where the set of states can be expressed as a Cartesian product of a set of visible states and a set of partially observable (hidden) ones: $\mathcal{S} = \mathcal{S}_v \times \mathcal{S}_h$. In the mixed-observable probabilistic setting, a belief state b is a probability distribution over $\mathcal{S}_h$, which is updated using Bayes' rule after each observation $o \in \mathcal{O}$ is received. The set of all probability distributions over $\mathcal{S}_h$ is denoted by $B \subsetneq [0,1]^{\mathcal{S}_h}$. Since the sequence of beliefs is a Markov process, it is well-known that partially observable problems can be transformed into fully observable ones, yielding a *belief MDP* over a continuous infinite state space B. Indeed, the value function $V : \mathcal{S}_v \times B \rightarrow \mathbb{R}$ is piecewise linear and convex for each fixed visible state in $\mathcal{S}_v$ and each finite horizon (Smallwood and Sondik 1973), which requires MOMDP solvers to deal with successive sets of α-vectors $\alpha \in \Gamma_n(s_v) \subsetneq \mathbb{R}^{\mathcal{S}_h}$ such that $V(s_v, b) = \max_{\alpha \in \Gamma_n(s_v)} \alpha \cdot b$ (Ong et al. 2010; Araya-Lòpez et al. 2010). MOMDPs are actually a recent extension of POMDPs, which consider the entire state space as partially observable even if some parts are directly visible to the agent (Cassandra, Littman, and Zhang 1997; Pineau, Gordon, and Thrun 2003; Smith and Simmons 2004). Thus, MOMDP planners are proven to run significantly faster than POMDP ones, since they reason about a reduced continuous belief space.

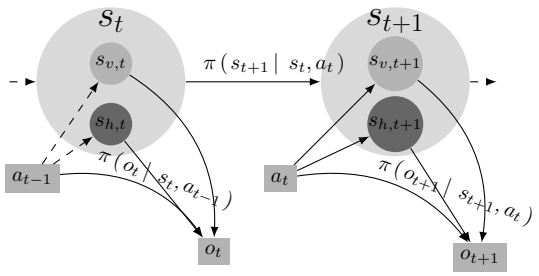


Figure 1: Dynamic influence diagram of a (π-)MOMDP

Mixed-Observable MDPs have been also studied in (Drougard et al. 2013) for the possibilistic framework. Figure 1 depicts the influence diagram of a π-MOMDP: as in the probabilistic model, the state space of a π-MOMDP is a Cartesian product of a totally visible set of states and a partially observable one. Uncertainty over transitions between states is modeled by $T^\pi(s, a, s') = \pi(s' \mid s, a) = \pi((s'_v, s'_h) \mid (s_v, s_h), a) \in \mathcal{L} = \{0, \frac{1}{k}, \ldots, 1\}$. We introduce now the new possibility distribution used to model uncertainty over observations $o \in \mathcal{O}$ which serve to estimate the current hidden state: $\forall o' \in \mathcal{O}$, $\forall s' = (s'_v, s'_h) \in \mathcal{S}$ and $\forall a \in \mathcal{A}$, $\Omega^\pi(s', a, o') = \pi(o' \mid s', a)$ is the possibility of the current observation o' conditioned on the current state s' and the previous action a. A π-MOMDP is then entirely defined by the tuple $\langle \mathcal{S} = \mathcal{S}_v \times \mathcal{S}_h, \mathcal{A}, \mathcal{L}, T^\pi, \mathcal{O}, \Omega^\pi, \mu \rangle$.

Like probabilistic MOMDPs, π-MOMDPs can be reduced to belief π-MDPs. However, contrary to the probabilistic case and thanks to the finite uncertainty measure used in Possibility Theory, state space of belief π-MDPs remains finite as explained in the next subsection. This fundamental property means that any algorithm for π-MDPs can solve *as is* any π-MOMDPs (and π-POMDPs as a special case) after reduction to a finite-state belief π-MDP.

Possibilistic belief and reduction to π-MDPs

The hidden part of the system state may initially not be entirely known: an estimation can be however available. This estimation is expressed in terms of a possibility distribution $\beta_0 : \mathcal{S} \rightarrow \mathcal{L}$ called *initial possibilistic belief* and is generally between the two following extrema: if $\forall s_h \in \mathcal{S}_h$ $\beta_0(s_h) = 1$, the initial hidden state is completely unknown ; and if $\exists \overline{s}_h \in \mathcal{S}_h$ such that $\beta_0(s_h) = \mathbb{1}_{\overline{s}_h, s_h}$, the initial hidden state is known to be $\overline{s}_h$. Let us define recursively the history $(h_t)_{t \geqslant 0}$ of a π-MOMDP: $h_0 = \{\beta_0, s_{v,0}\}$ and for each time step $t \geqslant 1$, $h_t = \{o_t, s_{v,t}, a_{t-1}, h_{t-1}\}$. The possibilistic belief at time step $t \geqslant 0$ is defined as the possibility distribution over variable $s_{h,t}$ conditioned on history h_t: $\forall s_h \in \mathcal{S}_h$, $\beta_t(s_h) = \pi(s_h \mid h_t)$.

Let β_{t+1} be the belief distribution at time $t+1$: if o' is the current observation, s'_v the current visible state and a the previous action, it comes from definitions that $\forall s'_h \in \mathcal{S}_h$, $\beta_{t+1}(s'_h) = \pi(s'_h \mid s'_v, o', a, h_t)$. This can be computed using possibilistic Bayes' rules (Dubois and Prade 1990) from the joint possibility distribution $\pi(s'_h, o', s'_v \mid a, h_t)$. Computation of this joint distribution depends on data of the π-MOMDP problem: writing arbitrarily (s'_v, s'_h) or s' the system state,

$$\begin{aligned}\pi(s'_h, o', s'_v \mid a, h_t) &= \min\{\pi(o' \mid s', a), \pi(s' \mid a, h_t)\} \\ &= \min\{\Omega^\pi(s', a, o'), \pi(s' \mid a, h_t)\}.\end{aligned}$$

Finally: $\pi(s' \mid a, h_t) = \max_{s \in \mathcal{S}} \pi(s', s \mid a, h_t)$

$$\begin{aligned}&= \max_{s \in \mathcal{S}} \min\{\pi(s' \mid s, a), \pi(s \mid h_t)\} \\ &= \max_{s_h \in \mathcal{S}_h} \min\{\pi(s'_v, s'_h \mid s_v, s_h, a), \pi(s_h \mid h_t)\} \\ &= \max_{s_h \in \mathcal{S}_h} \min\{T^\pi(s_v, s_h, a, s'_v, s'_h), \beta_t(s_h)\}\end{aligned}$$

where s_v is the previous visible state.
The joint possibility distribution can be expressed in terms of only the previous belief state β_t and visible state s_v (not all history), T^π and Ω^π: it will be from now on denoted by $\pi(s'_h, o', s'_v \mid s_v, \beta_t, a)$. The possibilistic Bayes' rule finally yields:

$$\beta_{t+1}(s'_h) = \begin{cases} 1 \text{ if } s'_h \in \underset{s_h \in \mathcal{S}_h}{\operatorname{argmax}} \pi(s'_h, o', s'_v \mid s_v, \beta_t, a) > 0 \\ \pi(s'_h, o', s'_v \mid s_v, \beta_t, a) \text{ otherwise.} \end{cases} \quad (1)$$

The set of all possibility distributions over $\mathcal{S}_h$ is denoted by B^π. **Note that B^π is finite of size $\#\mathcal{L}^{\#\mathcal{S}_h} - (\#\mathcal{L}-1)^{\#\mathcal{S}_h}$**, which is certainly exponential in the size of the hidden state space but finite. The belief update Equation 1 can be formulated as a belief update function $U : B^\pi \times \mathcal{A} \times \mathcal{S}_v^2 \times \mathcal{O} \to B^\pi$ such that $\beta_{t+1} = U(\beta_t, a, s_v, s'_v, o')$.

We see that the agent can reason on a belief π-MDP with finite state space $\mathcal{S}_v \times B^\pi$, by updating its belief over $\mathcal{S}_h$ on the basis of previous action a and state s_v, current state s'_v and observation o'. This belief update defines the transitions and preferences of the belief π-MDP, which requires to firstly introduce $K(\beta, a, s_v, s'_v, \beta') = \{o' \in \mathcal{O} \text{ such that } U(\beta, a, s_v, s'_v, o') = \beta'\}$ the set of observations which lead to β' given previous belief β, action a, state s_v and current state s'_v. Secondly, we construct the possibility distribution over observations and visible states as:

$$\pi(o', s'_v \mid s_v, \beta, a) = \max_{s'_h \in \mathcal{S}_h} \pi(s'_h, o', s'_v \mid s_v, \beta, a).$$

The new transition possibility is then

$$\pi(s'_v, \beta' \mid s_v, \beta, a) = \max_{o' \in K(\beta, a, s_v, s'_v, \beta')} \pi(o', s'_v \mid s_v, \beta, a)$$

with the convention $\max_\emptyset = 0$.

Finally, we define the preference over (s_v, β) such that this paired state is considered as good if it is necessary (according to β) that the system is in a good state:

$$\mu(s_v, \beta) = \min_{s_h \in \mathcal{S}_h} \max\{\mu(s_v, s_h), 1 - \beta(s_h)\}.$$

This reduction of any π-MOMDP to a finite-state belief π-MDP with structured states $s = (s_v, \beta)$ has been exploited in (Drougard et al. 2013). However, this previous work did not consider s_v nor β to be themselves factored into variables, meaning that it did not tackle *factored* π-MOMDPs. In the next section, we present our first contribution: the first symbolic algorithm to solve factored π-MDPs, which also handles factored π-MOMDPs by reduction to π-MDPs, and factored π-POMDPs by inclusion in π-MOMDPs.

Solving factored π-MDPs using symbolic dynamic programming

Factored MDPs (Hoey et al. 1999) have been used to efficiently solve structured sequential decision problems under probabilistic uncertainty, by symbolically reasoning on functions of states via decision diagrams rather than on individual states. In this framework, the state space $\mathcal{S}$ consists of a set of binary variables[1] $X = (X_1, \ldots, X_n)$: $\#\mathcal{S} = 2^n$. The current state is denoted by X and the next one by $X' = (X'_1, \ldots, X'_n)$. Dynamic Bayesian Networks (DBNs) (Dean and Kanazawa 1989) offer a useful graphical representation of π-MDP transitions, as depicted in Figure 2. In DBN semantics, $parents(X'_i)$ is the set of state variables on which X'_i depend. From now on, we assume that the next state's variables depend for each action only on a subset of X, *i.e.* $X'_1, \ldots, X'_n$ are independent given X. Using DBN semantics, it means that $parents(X'_i) \subset X$, which is often denoted as *uncorrelated action effects*: methods are discussed in the literature to circumvent this restrictive assumption (Boutilier 1997). In the possibilistic settings, this assumption allows us to compute the joint possibility transition as $\pi(s' \mid s, a) = \pi(X' \mid X, a) = \min_{i=1}^n \pi(X'_i \mid parents(X'_i), a)$.

Thus, a factored π-MDP can be defined with transition functions $T^{a,i}$ for each $a \in \mathcal{A}$ and each $1 \leqslant i \leqslant n$ such that $T^{a,i}(parents(X'_i), a, X'_i) = \pi(X'_i \mid parents(X'_i), a)$. Each transition function can be compactly encoded in an Algebraic Decision Diagram (ADD) (Bahar et al. 1997). An ADD, as illustrated in Figure 3a, is a directed acyclic graph which represents a real-valued function of binary state variables, whose identical sub-graphs are merged and zero-valued leaves are not memorized. Thus, ADDs are compact symbolic representations of real-valued functions of boolean variables. The possibilistic dynamic programming's update equation of Line 6 of Algorithm 1 can be rewritten in a symbolic form, so that states are now globally updated at once instead of individually ; if $u^*(s) = \max_{(\delta) \in \Delta_t} u(s, (\delta))$ is the current optimal value function (step $t > 0$), the Q-value of an action $a \in \mathcal{A}$, defined by $q^a(s) = \max_{s' \in \mathcal{S}} \min\{\pi(s' \mid s, a), u^*(s')\}$, is:

Proposition 1. *Consider the current value function $u^* : \{0,1\}^n \to \mathcal{L}$. For a given action $a \in \mathcal{A}$, let us define:*

- $q_0^a = u^*(X'_1, \cdots, X'_n)$
- $q_i^a = \max_{X'_i \in \{0,1\}} \min\{(X'_i \mid parents(X'_i), a), q_{i-1}^a\}$

[1] Non-binary state variables with m discrete values can be transformed into $\lceil \log_2 m \rceil$ binary variables.

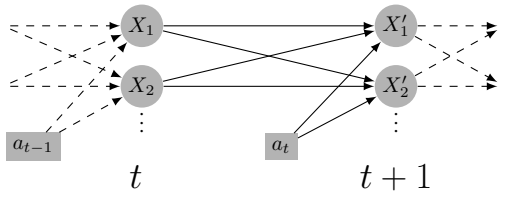


Figure 2: Dynamic Bayesian Network of a factored π-MDP

Then, the possibilistic Q-value of action a is: $q^a = q_n^a$.

Proof.

$$
\begin{aligned}
q^a &= \max_{s' \in \mathcal{S}} \min\{\pi(s'|s,a), u^*(s')\} \\
&= \max_{s' \in \mathcal{S}} \min\left\{ \min_{i=1}^{n} \pi\left(X'_i \mid \mathit{parents}(X'_i), a\right), u^*(s') \right\} \\
&= \max_{(X'_1,\cdots,X'_n)\in\{0,1\}^n} \min\left\{ \min_{i=1}^{n} \pi\left(X'_i \mid \mathit{parents}(X'_i), a\right), \right. \\
&\qquad\qquad \left. u^*(X'_1,\cdots,X'_n) \right\} \\
&= \max_{X'_n\in\{0,1\}} \min\Big\{ \pi\left(X'_n \mid \mathit{parents}(X'_n), a\right), \cdots \\
&\qquad \max_{X'_2\in\{0,1\}} \min\left\{ \pi\left(X'_2 \mid \mathit{parents}(X'_2), a\right), \right. \\
&\qquad\qquad \max_{X'_1\in\{0,1\}} \min\{ \pi\left(X'_1 \mid \mathit{parents}(X'_1), a\right), \\
&\qquad\qquad\qquad u^*(X'_1,\cdots,X'_n)\} \left. \right\} \cdots \Big\}
\end{aligned}
$$

where the last equation is due to the fact that, for any boolean variables A and B and any function $\varphi : \{0,1\} \to \mathcal{L}$ and $\psi : \{0,1\} \to \mathcal{L}$, we have:

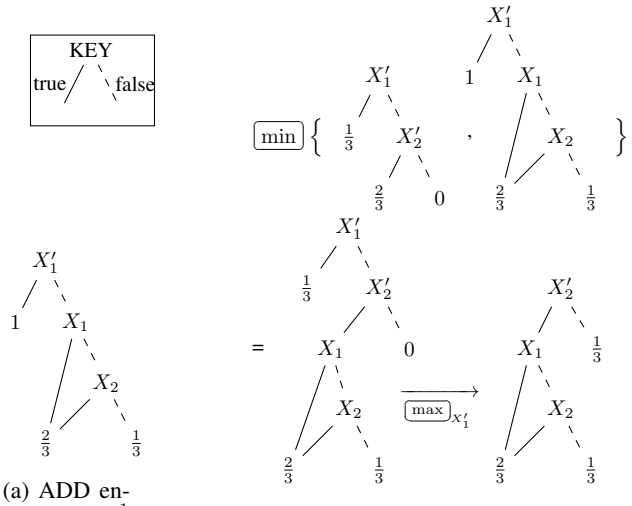
$$\max_{B\in\{0,1\}} \min\{\varphi(A), \psi(B)\} = \min\{\varphi(A), \max_{B\in\{0,1\}} \psi(B)\}$$

□

This proposition means that we can iteratively regress the Q-value of action a, represented as an ADD, over successive post-action state variables $X'_i, 1 \leqslant i \leqslant n$. In order to make explicit that we are working with symbolic functions encoded as ADDs, we will use the following notations:

- $\boxed{\min}\{f, g\}$ where f and g are 2 ADDs;
- $\boxed{\max}_{X_i} f = \boxed{\max}\{f^{X_i=0}, f^{X_i=1}\}$, which can be easily computed because ADDs are constructed on the basis of the Shannon expansion: $f = \overline{X_i} \cdot f^{X_i=0} + X_i \cdot f^{X_i=1}$ where $f^{X_i=1}$ and $f^{X_i=0}$ are sub-ADDs representing the positive and negative Shannon cofactors (see Fig. 3a).

Figure 3b illustrates the possibilistic regression of the Q-value of an action for the first state variable X_1. Contrary to factored *probabilistic* MDPs, for which additions and multiplications involved in the computation of Q-values create new values (i.e. ADD leaves) most of the time, factored π-MDPs' $\min$ and $\max$ operations do not create new values from the input ADDs. More concretely, ADDs with probabilistic operations can have up to 2^n leaves (i.e. as many values as the number of states), whereas ADDs with possibilistic operations can have up to $\#\mathcal{L} \ll 2^n$ leaves. It means



(a) ADD encoding $T^{a,1}$ of Fig. 2

(b) Symbolic regression of the current Q-value ADD with $T^{a,1}$ of Figure 3a

Figure 3: π-MDPs with Algebraic Decision Diagrams

that ADDs should be far smaller in practice under possibilistic settings, which was a motivation for this paper and is experimentally checked in a forthcoming section.

Algorithm 2 is a symbolic version of Algorithm 1, which relies on the regression scheme defined in Proposition 1. Inspired by SPUDD (Hoey et al. 1999), PPUDD means *Possibilistic Planning Using Decision Diagrams*. As for SPUDD, PPUDD needs to swap state variables to primed ones in the ADD encoding the current value function before computing the Q-value of an action a (see Line 5 of Algorithm 2). This operation is required to differentiate the next state, which is represented by primed variables, from the current one when operating ADDs. Lines 4-9 apply Proposition 1 and correspond to Line 6 of Algorithm 1.

Algorithm 2: Possibilistic Planning Using Decision Diagrams (PPUDD)

1 $u^* \leftarrow 0_{\mathcal{L}}$; $u^c \leftarrow \mu$; $\delta \leftarrow \overline{a}$;
2 **while** $u^* \neq u^c$ **do**
3 $u^* \leftarrow u^c$;
4 **for** $a \in \mathcal{A}$ **do**
5 $q^a \leftarrow$ swap each X_i variable in u^* with X'_i ;
6 **for** $1 \leqslant i \leqslant n$ **do**
7 $q^a \leftarrow \boxed{\min}\{q^a, \pi(X'_i \mid \mathit{parents}(X'_i), a)\}$;
8 $q^a \leftarrow \boxed{\max}_{X'_i} q^a$;
9 $u^c \leftarrow \boxed{\max}\{q^a, u^c\}$;
10 update δ to a where $q^a = u^c$ and $u^c > u^*$
11 **return** (u^*, δ) ;

Factored π-MOMDPs

As explained in the background section, any π-MOMDP defined over the state space $\mathcal{S}_v \times \mathcal{S}_h$ can be transformed into a finite-state belief π-MDP whose states are in $\mathcal{S}_v \times B^\pi$ with

$B^\pi \subsetneq \mathcal{L}^{\mathcal{S}_h}$. In order to symbolically solve π-MOMDPs with PPUDD, we need: (1) to factorize B^π as a product of discrete state variables; (2) to reason about independent post-action state variables (so-called uncorrelated action effects). We will prove in the following that this requires to assume the following structural assumption: *post-action state variables are independent, hidden state variables does not depend on other previous hidden state variables and each observation can only depend on action, previous visible state variables and current corresponding hidden state variable*. This assumption is depicted in Figure 4, which represents the DBN of a factored π-MOMDP $\langle \mathcal{S}_v \times \mathcal{S}_h, \mathcal{A}, \mathcal{L}, T^\pi, \mathcal{O}, \Omega^\pi, \mu \rangle$, where: $\mathcal{S}_v = \mathcal{S}_{v,1} \times \ldots \times \mathcal{S}_{v,m}$, $\mathcal{S}_h = \mathcal{S}_{h,1} \times \ldots \times \mathcal{S}_{h,l}$ and $\mathcal{O} = \mathcal{O}_1 \times \ldots \times \mathcal{O}_l$ with $\forall 1 \leqslant j \leqslant l$, $\mathcal{O}_j$ the set of observations corresponding to $\mathcal{S}_{h,j}$. We note that factored probabilistic MOMDP have not yet been studied to the best of our knowledge. We would not be surprised if factored probabilistic MOMDPs would require the same structural assumption as the one presented here, which could interestingly inspire future work under probabilistic settings.

Factorizing B^π as a product of discrete state variables. Our structural assumption allows us to first prove in Theorem 1 that the current belief state is actually the minimum of all marginal beliefs, defined as the possibility distribution over a hidden state variable.

Theorem 1. *If $s_{h,1}, \ldots, s_{h,l}$ are initially independent, then at each time step $t > 0$ the belief over hidden states can be written $\beta_t = \min_{j=1}^{l} \beta_{t,j}$ with $\forall s_j \in \mathcal{S}_{h,j}$, $\beta_{t,j}(s_j) = \pi(s_j \mid h_t)$ the belief over $\mathcal{S}_j$.*

Proof. First $s_{h,1}, \ldots, s_{h,l}$ are initially independent, then $\exists (\beta_{0,j})_{j=1}^{l}$ such that $\beta_0(s_h) = \min_{j=1}^{l} \beta_{0,j}(s_{h,j})$. Now recall that $h_{t+1} = \{o_{t+1}, s_{v,t+1}, a_t, h_t\}$. The independence between hidden variables conditioned on the history can be shown using the *d-separation* relationship (Pearl 1988) used for example in (Stefan J. Witwicki and Spaan 2013). In fact, as shown figure 4 given $1 \leqslant i < j \leqslant l$, $s'_{h,i}$ and $s'_{h,j}$ are d-separated by the evidence h_{t+1}. Thus $\pi(s'_h \mid h_{t+1}) = \min_{j=1}^{l} \pi\left(s'_{h,j} \mid h_{t+1}\right)$ *i.e.* $\beta_t(s'_h) = \min_{j=1}^{l} \beta_{t,i}(s'_{h,j})$. Note however that it would not be true if the same observation variable o would have concerned two different hidden state variables $s_{h,p}$ and $s_{h,q}$: as o is part of the history, there would be a convergent (towards o) relationship between $s_{h,p}$ and $s_{h,q}$ and the hidden state variable would have been dependent (because d-connected) conditioned on history. Moreover if hidden state variable $s'_{h,p}$ could depend on previous hidden state variable $s_{h,q}$, $s'_{h,p}$ and $s'_{h,q}$ would have been dependent conditioned on history because d-connected trough $s_{h,q}$. □

Theorem 1 allows us to rewrite the state space in the form of $\mathcal{S}_{v,1} \times \ldots \times \mathcal{S}_{v,m} \times B_{h,1} \times \cdots \times B_{h,l}$ with $B_{h,i} \subsetneq \mathcal{L}^{\mathcal{S}_{h,i}}, 0 \leqslant i \leqslant l$. Each $B_{h,i}$ state variable has a finite domain with $\#\mathcal{L}^{\#\mathcal{S}_{h,i}} - (\#\mathcal{L} - 1)^{\#\mathcal{S}_{h,i}}$ values. If all state variables are binary, $\#B_{h,i} = 2\#\mathcal{L} - 1$ for all $1 \leqslant i \leqslant l$, so that the total number of flattened states of the belief π-MDP would be $2^m(2\#\mathcal{L} - 1)^l$: contrary to probabilistic settings, **hidden state variables and visible ones have a similar impact on the solving complexity**, i.e. both singly-exponential in the number of state variables. In the general case, by noting $\kappa = \max\{\max_{1\leqslant i\leqslant m} \#S_{v,i}, \max_{1\leqslant j\leqslant l} \#S_{h,j}\}$, there are $\mathcal{O}(\kappa^m(\#\mathcal{L})^{(\kappa-1)l})$ flattened states, which is indeed exponential in the arity of state variables too.

Uncorrelated action effects of the belief π-MDP. PPUDD requires that post-action state variables are all independent given the past. In order to prove this property for belief π-MDP, we first need to demonstrate Lemma 1, which shows how marginal beliefs are actually updated. For this purpose, we recursively define the history concerning hidden variable $s_{h,j}$: $h_{0,j} = \{\beta_{0,j}\}$ and $\forall t \geqslant 0$ $h_{t+1,j} = \{o_{t+1,j}, s_{v,t}, a_t, h_{t,j}\}$. Similarly to the belief update presented in the background section concerning the transformation of π-MOMDPs to π-MDPs, we note:

$$\pi(s'_{h,j} \mid s_v, \beta_j, a) = \max_{s_{h,j}} \min\{\pi(s'_{h,j} \mid s_v, s_{h,j}, a), \beta_j(s_{h,j})\}$$

$$\pi(o'_j, s'_{h,j} \mid s_v, \beta_j, a) = \min\{\pi(o'_j \mid s'_{h,j}, s_v, a), \pi(s'_{h,j} \mid s_v, \beta_j, a)\}.$$

Lemma 1. *If the agent was at time t in visible state s_v, had the belief $\beta_{j,t}$ over hidden states $s_{h,j}$, executed action a and then get observation o'_j, update of the belief state over $\mathcal{S}_{h,j}$ is: $\beta_{t+1,j}(s'_{h,j})$*

$$= \begin{cases} 1 \text{ if } s'_{h,j} \in \underset{s'_{h,j} \in \mathcal{S}_{h,j}}{\operatorname{argmax}} \pi\left(s'_{h,j}, o'_j \mid s_v, \beta_{t,j}, a\right) > 0 \\ \pi\left(s'_{h,j}, o'_j \mid s_v, \beta_{t,j}, a\right) \text{ otherwise.} \end{cases} \tag{2}$$

Proof. First note that $s_{h,j}$ and $\{o_{m,s}\}_{s\leqslant t, m\neq j} \cup \{s_{v,t}\}$ are d-separated by the evidence $h_{t,j}$ then $s_{h,j}$ is independant of $\{o_{m,s}\}_{s\leqslant t, m\neq j} \cup \{s_{v,t}\}$ conditionned on $h_{t,j}$: $\pi(s_{h,j} \mid h_t) = \pi(s_{h,j} \mid h_{t,j})$. Finally the update can be computed in the same way that update 1 using functions of the problem: $\beta_{t,j}$, $\pi\left(s'_{h,j} \mid s_v, s_{h,j}, a\right)$ and $\pi\left(o'_j \mid s'_{h,j}, s_v, a\right)$. □

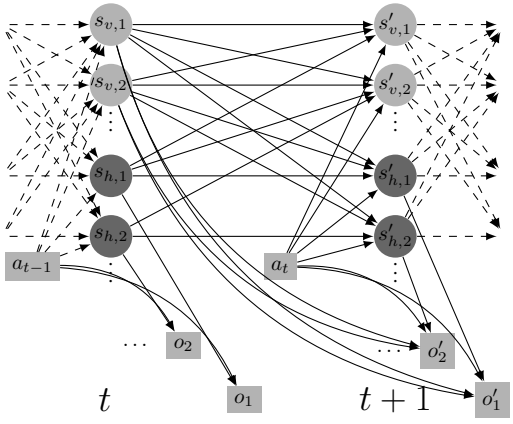


Figure 4: DBN of a factored π-MOMDP

Finally, Theorem 2 relies on Lemma 1 to exploit the independence of all post-action state variables of the belief π-MDPs given the past, which allows us to run PPUDD on it, and provide the possibilistic transition of this belief π-MDP.

Theorem 2. $\forall \beta, \beta' \in B$, $\forall s_v, s'_v \in \mathcal{S}_v$, $\forall a \in \mathcal{A}$, $\pi\left(s'_v, \beta' \mid s_v, \beta, a\right)$

$$= \min\left\{ \min_{i=1}^{k} \pi\left(s'_{v,i} \mid s_v, \beta, a\right), \min_{j=1}^{l} \pi\left(\beta'_j \mid s_v, \beta_j, a\right) \right\}$$

Proof. Let $s_{h,1}$ and $s_{h,2}$ two hidden states: using that $\pi\left(s'_h \mid s_v, s_h, a\right)$

$$= \min\left\{ \pi\left(s'_{h,1} \mid s_v, s_{h,1}, a\right), \pi\left(s'_{h,2} \mid s_v, s_{h,2}, a\right) \right\}$$

and $\beta(s_h) = \min\left\{\beta_1(s_{h,1}), \beta_2(s_{h,2})\right\}$, we get easily

$$\pi\left(s'_h \mid s_v, \beta, a\right)$$
$$= \min\left\{ \pi\left(s'_{h,1} \mid s_v, \beta_1, a\right), \pi\left(s'_{h,2} \mid s_v, \beta_2, a\right) \right\}.$$

This result and the fact that $\pi\left(o' \mid s_v, s'_h, a\right)$

$$= \min\left\{ \pi\left(o'_1 \mid s_v, s'_{h,1}, a\right), \pi\left(o'_2 \mid s_v, s'_{h,2}, a\right) \right\}$$

leads to the equality $\pi\left(o' \mid \beta, a\right)$

$$= \min\left\{ \pi\left(o'_1 \mid s_v, \beta_1, a\right), \pi\left(o'_2 \mid s_v, \beta_2, a\right) \right\}. \quad (3)$$

The previous equation shows that observations are independent given the past. Moreover, we proved in Lemma 1 that updates of each marginal belief can be performed independently on other marginal beliefs, but depends on the corresponding observation only. Thus, we conclude that marginal belief state variables are independent given the past. Finally as s'_v and o' are independant given the past, $\pi\left(s'_v, \beta' \mid s_v, \beta, a\right) = \max_{o' \in K(\beta, a, s_v, \beta')} \pi\left(s'_v, o' \mid s_v, \beta, a\right)$

$$= \max_{o' \in K(\beta, a, s_v, \beta')} \min\left\{ \pi\left(s'_v \mid s_v, \beta, a\right), \pi\left(o' \mid s_v, \beta, a\right) \right\}$$

$$= \min\left\{ \pi\left(s'_v \mid s_v, \beta, a\right), \max_{o' \in K(\beta, a, s_v, \beta')} \pi\left(o' \mid s_v, \beta, a\right) \right\}$$
$$= \min\left\{ \pi\left(s'_v \mid s_v, \beta, a\right), \pi\left(\beta' \mid s_v, \beta, a\right) \right\}$$

which conclude the proof. □

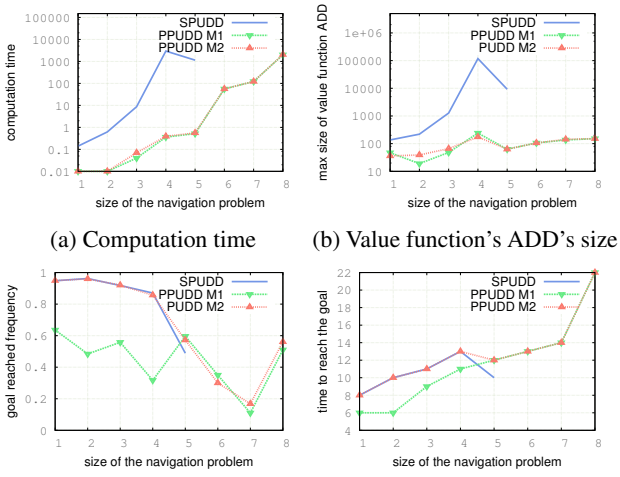
Experimental results

We should now summarize the main benefits of using factored π-MDPs over probabilistic models:

- values of ADDs are in the finite scale $\mathcal{L}$ rather than $\mathbb{R}$, so that the number of their leaves is at most $\#\mathcal{L} \ll 2^n$ (probabilistic models' ADDs can have up to 2^n leaves);
- partially observable models (factored π-POMDPs and π-MOMDPs) boil down to factored *finite*-state belief π-MDPs that can be solved by PPUDD *as is* assuming some structural assumption of the underlying actions' DBNs;
- partially-observable features of the state do not impact much PPUDD's performances than totally-observable ones *if all hidden state variables are binary*, since π-MOMDPs are then in the same complexity class as π-MDPs (in probabilistic models, partially-observable problems are always in a higher complexity class).

Of course, we have to pay a price: namely, possibilistic models are approximations of probabilistic ones (except if probabilities in the model are anyway imprecisely known). Yet, many state-of-the-art probabilistic algorithms are approximate while our PPUDD algorithm is exact. In this section, we compare our approach against probabilistic solvers in order to answer the following question: what is the efficacy/quality tradeoff achieved by reasoning about an approximate model but with an exact efficient algorithm?

The navigation domain. We first compared PPUDD against SPUDD on the nagivation domain used in planning competitions (Sanner 2011). In this domain, a robot navigates in a grid where it must reach some goal location most reliably. It can apply 5 actions which cost 1 except if we reach the goal: going north, east, south or west, and staying at the current location. When moving, it can suddenly disappear with some probability defined as a Bernoulli distribution, so that a good policy tries to reach the goal by avoiding situations where it may disappear. We approximated this probabilistic model by two possibilistic ones, M1 based on preference preservation (Dubois, Prade, and Sandri 1993), and M2 more pessimistic: the preference of reaching the goal is 1; for M1 the highest probability of each Bernoulli distribution is replaced by 1 (for possibility normalization reasons), for M2 the probability to disappear is replaced by 1, and we keep the same value for the lowest probability. Figure 5a shows SPUDD runs out of memory from the 5th problem, and PPUDD computation's time outperforms SPUDD's one by many orders of magnitude. Intuitively, this result comes from the fact that PPUDD's ADDs should be smaller because their leaves' values are in the finite scale $\mathcal{L}$ rather than $\mathbb{R}$, which is indeed demonstrated in Figure 5b. Figures 5c and 5d shows performances of the three models: unlike M1 whose performances are reduced due to possibilistic approximation, M2's performances are similar to probabilistic model's ones, as danger is modeled as necessary. We note that goal reached frequency decreases with



(a) Computation time (b) Value function's ADD's size

(c) Goal reached frequency (d) Time to reach the goal

Figure 5: PPUDD vs. SPUDD on the navigation domain

instance's size as risk of disappearance increases.

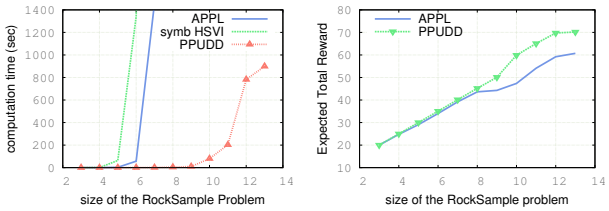
The RockSample domain. This problem is detailed in (Smith and Simmons 2004): a rover navigates in an environment modeled as a $N \times N$ grid and has to collect scientific samples of some interesting rocks. It knows the locations of the R rocks $(x_i, y_i)_{i=1}^{R}$ but not which ones are actually of interest (called "good" rocks). However, sampling a rock is expensive: the rover is fitted with a noisy long-range sensor that can be used to determine if a rock is "good" or not ("bad"). When a rock is sampled, it becomes (or stays) "bad" (no more interesting). At the end of the mission, the rover has to reach the exit location at the right side of the grid. This problem is modeled as follows:

- $\mathcal{S}_v$ consists of all the possible locations of the rover (x_r, y_r) in addition to the exit ($\#\mathcal{S}_v = N^2 + 1$),
- $\mathcal{S}_h$ consists of all the possible natures of the rocks ($\mathcal{S}_h = \mathcal{S}_{h,1} \times \ldots \times \mathcal{S}_{h,R}$ with $\forall 1 \leqslant i \leqslant R, \mathcal{S}_{h,i} = \{ good, bad \}$),
- $\mathcal{A}$ contains the (deterministic) moves in the 4 directions $(a_{north}, a_{east}, a_{south}, a_{west})$, checking rock i (a_{check_i}) $\forall 1 \leqslant i \leqslant R$ and sampling the current rock (a_{sample}),
- $\mathcal{O} = \mathcal{O}_1 \times \ldots \times \mathcal{O}_R$ where $\forall 1 \leqslant i \leqslant R$, $\mathcal{O}_i = \{ o_{good_i}, o_{bad_i} \}$ are observations concerning the i^{th} rock.

Without going into details of the model, the factorization of the observation set naturally yields to the structural assumption required for factored π-MOMDPs, since each observation variable is mapped to a hidden state variable. Note that the observation set of this domain is best known in the literature in the simpler form $\mathcal{O} = \{o_{good}, o_{bad}\}$, where the observation concerns the current rock, which is strictly equivalent to the previous factorization but does not highlight the structural assumption. The rationale behind observations is the following: the more the rover is close to the checked rock, the better it observes its nature. In the original probabilistic model, the probability of a correct observation equals $\frac{1}{2}\left(1 + e^{-c\sqrt{(x_r - x_i)^2 + (y_r - y_i)^2}}\right)$ with $c > 0$ a constant (the smaller is c, the more effective is the sensor). The rover gets a reward $+10$ for each good rock sampled, -10 for each bad rock sampled, and $+10$ when it reaches the exit.

In the possibilistic model, we approximate the observation function by using a critical distance $d > 0$ beyond which checking a rock is uninformative: $\pi(o'_i \mid s'_i, a, s_v) = 1 \; \forall o'_i \in \mathcal{O}_i$. If the rover is distant from the rock less than d, the possibility degree of erroneous observation becomes $\frac{1}{\#\mathcal{L}-1}$, or zero if it stands at the checked rock. Finally, as possibilistic semantics does not allow sum of rewards, we have to introduce an additional visible state variable $s_{v,2} \in \{1, \ldots, R\}$ which counts the number of checked rocks. The qualitative dislike of sampling is modeled with $\mu(s) = \frac{R+2-s_{v,2}}{R+2}$ if s_v is terminal and zero otherwise.

We compared PPUDD on the transformed factored belief π-MDP against a recent probabilistic MOMDP planner APPL (Ong et al. 2010), and a POMDP solver using ADDs symbolic HSVI (Sim et al. 2008). The two probabilistic planners need a precision of computation, which impacts performances: we set it to 1, and Figure 6a compares computation times of these three algorithms. We note that APPL



(a) Computation time (b) Expected total reward

Figure 6: PPUDD vs. APPL on the RockSample domain

runs out of memory with 8 rocks, symbolic HSVI with 7 rocks, and PPUDD outperforms it by many orders of magnitude. In order to assess the loss in quality of PPUDD due to possibilistic settings, we also compared the expected total rewards of both approaches. As APPL is an anytime solver we can fix a stopping computation time: we stopped APPL at the time equal to PPUDD's computation time to compare performances. For PPUDD, we simply evaluated with the probabilistic model the policy that was computed under possibilistic settings. Surprisingly, Figure 6b shows that rewards gathered are higher with PPUDD than with APPL. The reason is that APPL is in fact an approximate probabilistic planner, which shows that our approach consisting in exactly solving an approximate model can outperform algorithms that approximately solve an exact model. Moreover, exact POMDP planners are unable to scale to problems of the size of the RockSample ones. Finally, it is worth noting that probabilities of the observation model, which represent uncertainties of sensor outputs, may be difficult to precisely know in practice, in which case possibilistic models may be more physically accurate.

Conclusion

We presented PPUDD, the first algorithm to the best of our knowledge that symbolically solves factored possibilistic (MO)MDPs. In our opinion, possibilistic models are a good tradeoff between non-deterministic ones, whose uncertainties are not at all quantified yielding a very approximate models, and probabilistic ones, where uncertainties are fully specified. Moreover, π-MDPs reason about finite values in a qualitative scale $\mathcal{L}$ whereas probabilistic MDPs deal with real values, which implies larger ADDs for symbolic algorithms. Also, partially-observable problems reduce to finite-state belief π-MDPs, which allow PPUDD to solve them too, contrary to their probabilistic counterparts that yield *continuous*-state belief MDPs. Our experimental results show that using an exact algorithm (PPUDD) for an approximate model (π-MDPs) can run significantly faster than reasoning about exact models, while providing better policies than approximate algorithms (APPL) for exact models. In the future, we would like to investigate automatic translations of probabilistic models to possibilistic ones, and thoroughly study the class of problems that can be efficiently solved by our approach.

References

Araya-Lòpez, M.; Thomas, V.; Buffet, O.; and Charpillet, F. 2010. A closer look at MOMDPs. In *Proceedings of the Twenty-Second IEEE International Conference on Tools with Artificial Intelligence (ICTAI-10)*.

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1997. Algebric decision diagrams and their applications. *Form. Methods Syst. Des.* 10(2-3):171–206.

Bellman, R. 1957. A Markovian Decision Process. *Indiana Univ. Math. J.* 6:679–684.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artif. Intell.* 121(1-2):49–107.

Boutilier, C. 1997. Correlated action effects in decision theoretic regression. In *UAI*, 30–37.

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24:293–318.

Cassandra, A.; Littman, M. L.; and Zhang, N. L. 1997. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *In Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, 54–61. Morgan Kaufmann Publishers.

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Comput. Intell.* 5(3):142–150.

Drougard, N.; Teichteil-Konigsbuch, F.; Farges, J.-L.; and Dubois, D. 2013. Qualitative Possibilistic Mixed-Observable MDPs. In *Proceedings of the Twenty-Ninth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-13)*, 192–201. Corvallis, Oregon: AUAI Press.

Dubois, D., and Prade, H. 1988. *Possibility Theory: An Approach to Computerized Processing of Uncertainty (traduction revue et augmentée de "Théorie des Possibilités")*. New York: Plenum Press.

Dubois, D., and Prade, H. 1990. The logical view of conditioning and its application to possibility and evidence theories. *International Journal of Approximate Reasoning* 4(1):23 – 46.

Dubois, D., and Prade, H. 1995. Possibility theory as a basis for qualitative decision theory. In *IJCAI*, 1924–1930. Morgan Kaufmann.

Dubois, D.; Prade, H.; and Sabbadin, R. 1998. Qualitative decision theory with sugeno integrals. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, UAI'98, 121–128. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Dubois, D.; Prade, H.; and Sandri, S. 1993. On possibility/probability transformations. In *Proceedings of Fourth IFSA Conference*, 103–112. Kluwer Academic Publ.

Hanna Kurniawati, David Hsu, W. S. L. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems IV*.

Hoey, J.; St-aubin, R.; Hu, A.; and Boutilier, C. 1999. Spudd: Stochastic planning using decision diagrams. In *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 279–288. Morgan Kaufmann.

Ong, S. C. W.; Png, S. W.; Hsu, D.; and Lee, W. S. 2010. Planning under uncertainty for robotic tasks with mixed observability. *Int. J. Rob. Res.* 29(8):1053–1068.

Pearl, J. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for pomdps. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1025 – 1032.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1st edition.

Sabbadin, R.; Fargier, H.; and Lang, J. 1998. Towards qualitative approaches to multi-stage decision making. *Int. J. Approx. Reasoning* 19(3-4):441–471.

Sabbadin, R. 1999. A possibilistic model for qualitative sequential decision problems under uncertainty in partially observable environments. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, UAI'99, 567–574. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Sabbadin, R. 2001. Possibilistic markov decision processes. *Engineering Applications of Artificial Intelligence* 14(3):287 – 300. Soft Computing for Planning and Scheduling.

Sanner, S. 2011. Probabilistic track of the 2011 international planning competition. http://users.cecs.anu.edu.au/∼ssanner/IPPC_2011.

Sim, H. S.; Kim, K.-E.; Kim, J. H.; Chang, D.-S.; and Koo, M.-W. 2008. Symbolic heuristic search value iteration for factored pomdps. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI'08, 1088–1093. AAAI Press.

Smallwood, R. D., and Sondik, E. J. 1973. *The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon*, volume 21. INFORMS.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for pomdps. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, UAI '04, 520–527. Arlington, Virginia, United States: AUAI Press.

Stefan J. Witwicki, Francisco S. Melo, J. C. F., and Spaan, M. T. 2013. A flexible approach to modeling unpredictable events in MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS-2013)*. To appear.

Compiling Contingent Planning into Classical Planning: New Translations and Results

Héctor Palacios
Universitat Pompeu Fabra
Barcelona, Spain
hector.palacios@upf.edu

Alexandre Albore
ONERA and INRA
Toulouse, France
alexandre.albore@onera.fr

Hector Geffner
ICREA and Universitat Pompeu Fabra
Barcelona, Spain
hector.geffner@upf.edu

Abstract

Recently, Brafman and Shani have introduced a mapping for transforming deterministic contingent planning problems into classical ones. Their translation is interesting and elegant, but exponential in the number of possible initial states. They use it for action selection in an *on-line* contingent planner by calling a classical planner on an *approximate translation*, where the set of possible initial states is replaced by a sample of few states.

In this work, we introduce two alternative translations of contingent into classical problems, that are both polynomial, and we test them by solving contingent problems *off-line* using classical planners. While the results are not at the level of the last generation of off-line contingent planners like CNF, DNF, or CLG, they are meaningful enough and are on par with other recent contingent planners such as Contingent-FF, POND, and MBP. Moreover, the limitations in performance are not always a result of the size of the translations, but more a consequence of the limitations of current classical planning algorithms. These new translations thus enlarge the scope of problems that can be effectively solved by classical planners, while at the same time they present classical planners with new challenges.

Introduction

In the last few years, it has been shown that translations that map planning problems with incomplete information into planning problems with complete information can be computational effective and useful. This approach includes the translation of conformant problems into classical problems that underlies the conformant planner T0 (Palacios and Geffner 2009), the translation of contingent problems into fully observable non-deterministic problems used in the off-line and on-line contingent planner CLG (Albore, Palacios, and Geffner 2009), and the translation of contingent problems into classical problems used by the on-line contingent planner MSPR (Brafman and Shani 2012b). The first two translations are based on considerations of *width*, and are polynomial in the problem size. The translations used by these planners are a special case of more general complete translations that in the worst case are exponential in the number of problem variables. The translation underlying MSPR, on the other hand, is in the worst case doubly exponential in the problem size, as is exponential in the number of possible initial states. Brafman and Shani use their translation heuristically, for selecting actions in an *on-line* contingent planner by calling a classical planner on the *approximate translation* that results when the set of possible initial states is replaced by a smaller set of 4–8 samples.

In this work, we build on Brafman's and Shani's ideas to introduce two alternative translations for mapping contingent planning problems to classical problems. Unlike Brafman's and Shani's translations, however, our translations are polynomial in the number of possible initial states. Furthermore, we do not test these translations heuristically for selecting actions but for *computing full solutions to contingent problems using classical planners.* We show that, while the results from our translations are not at the level of the last generation off-line contingent planners like CNF, DNF (To, Pontelli, and Son 2011) or CLG, they are meaningful enough and are on par with other recent contingent planners such as Contingent-FF (Hoffmann and Brafman 2005), POND (Bryce, Kambhampati, and Smith 2006), and MBP (Bertoli et al. 2006). Moreover, the limitations in performance are not always a result of the size of the translations, but of limitations of current classical planning algorithms. The new translations thus enlarge the scope of problems that can be effectively solved by classical planners while at the same they present classical planners with new challenges. Furthermore, we expect these ideas to be relevant to other types of planning problems as well. For example, Brafman and Shani, along with Zilberstein, have used the ideas underlying their contingent translation to provide an approach for solving Q-Dec-POMDPs using classical planners (Brafman, Shani, and Zilberstein 2013). Q-Dec-POMDPs are collaborative multiagent planning models that are similar to Dec-POMDPs (Bernstein, Zilberstein, and Immerman 2000), except that uncertainty is represented by sets of states rather than probability distributions. In principle, our translations could be used for the same purpose but with potentially better results.

The rest of the paper is organized as follows. We consider the model and representation of (deterministic) contingent planning problems, then present each of two new translations along with their formal properties, and finally, the experimental results.

Contingent Planning

We review the model and representation of contingent planning problems.

Model

The model underlying deterministic contingent planning can be characterized as a tuple $\mathcal{S} = \langle S, S_0, S_G, A, O, f, o\rangle$ where S is a finite set of states, $S_0 \subseteq S$ is the set of possible initial states, S_G is the set of goal states, A is a set of actions with $A(s)$ denoting the actions in A that are applicable in the state s, and O is a set of observation tokens. An action a applicable in a state s changes the state to $s' = f(a, s)$ and results in the observation token $o(s', a) \in O$. Executions are sequences of action-observation pairs $a_0, o_0, a_1, o_1, \ldots$ and beliefs represent the sets of states that are possible. The initial belief state is $b_0 = S_0$, and if b is the belief before the action a is applied, the belief right after a is $b_a = \{s' \mid s' = f(a, s) \text{ and } s \in b\}$, while $b_a^o = \{s \mid s \in b_a \text{ and } o = o(s, a)\}$ is the belief after getting then the observation token o.

An execution $a_0, o_0, a_1, o_1, \ldots$ is *possible* in the model $\mathcal{S}$ if, starting from the initial belief b_0, each action a_i is applicable in the belief b_i resulting from the execution up to a_i, i.e. $a_i \in A(s)$ for all $s \in b_i$, and b_{i+1} is non-empty, where $b_{i+1} = b_a^o$ for $b = b_i$, $a = a_i$, and $o = o_i$. A *belief policy* π is a function mapping belief states into actions, while a *tree policy* π is a function mapping executions into actions. The executions $a_0, o_0, a_1, o_1, \ldots$ induced by a belief or tree policy π are the possible executions in which a_i is the action dictated by the policy given the belief b_i and the execution up to a_i respectively. A policy solves the model if all such executions reach a *goal* belief state, i.e., a belief state $b \subseteq S_G$. *Off-line methods* focus on the computation of such policies; *on-line methods* focus on the computation of the action for the current belief or execution. The difference between belief and tree policies is that the former treat executions leading to the same belief state as equivalent. Thus, while belief policies are represented by graphs, tree policies are represented by trees (Geffner and Bonet 2013).

Representation

We assume that contingent models are represented in compact form through tuples $P = \langle F, I, A_F, A_N, G\rangle$ where F stands for a set of atoms, I is a set of clauses over F representing the initial situation, and G is a set (conjunction) of literals over F representing the goals. We assume that A_F represents a set of physical actions, and A_N a set of sensing actions. Both physical and sensing actions a have a precondition $Pre(a)$ that is a conjunction of literals. Physical actions a are characterized by a set of conditional effects $a : C \rightarrow E$ on the world, where C and E are sets (conjunction) of literals, while sensing actions $a(q)$ reveal the truth value of the atom q in F.

The planning problem $P = \langle F, I, A_F, A_N, G\rangle$ defines the state model $\mathcal{S}(P) = \langle S, S_0, S_G, A, O, f, o\rangle$, where S is the set of valuations over the atoms in F, S_0 and S_G are the sets of valuations that satisfy I and G respectively, $A(s)$ is the set of actions in $A_F \cup A_N$ whose preconditions are true in s, $f(a, s)$ is the state-transition function determined by the conditional effects associated with physical actions $a \in A_F$, and $f(a, s) = s$ for sensing actions in $a \in A_N$. The set O contains the tokens $\top$ and $\bot$ such that $o(s, a)$ is $\top$ for actions a in A_F, and $o(s, a)$ is $\top$ or $\bot$ according to the truth value of atom q in s when a is the sensing action $a(q)$ in A_N.

The distinction between purely physical actions and purely sensing actions is a convenient simplification; the generalization to actions that involve both physical and sensing effects requires extra notation, but is not a computational challenge.

First Translation

The first translation of contingent problems P into classical problems $C_i(P)$ that we consider here, follows Brafman's and Shani's closely, but while they map each action a in P to $2^{|S_0|}$ actions $a(S')$, where S' is a subset of S_0, our translation is polynomial in $|S_0|$ and introduces a linear number of actions.[1]

For convenience, we will assume that the language of the classical translations $C_i(P)$ supports *axioms*. Axioms allow the definition of new, derived atoms in terms of primitive ones, called then the primitive fluents. The derived fluents can be used in action preconditions, goals, and in the body of conditional effects. Axioms are part of the classical PDDL standard and many classical planners support axioms. While it is possible to compile axioms away, there are also benefits for dealing with them directly in the computation of the heuristics and in the progression of the state (Thiébaux, Hoffmann, and Nebel 2005). In our implementation, we compile axioms away so that the translations can be fed into almost any existing classical planner supporting STRIPS, negation, and conditional effects.

The first translation $C_1(P)$ comprises fluents L/s for the literals L in P and the possible initial states $s \in b_I$ where $b_I = S_0$. The literals in P are those of the form p and $\neg p$ for $p \in F$. The literals L/s represent that L is true under the assumption that s is the true hidden initial state. The other primitive fluents in the translation $D(s, s')$ represent that the execution so far contains enough information to distinguish one hidden initial state s from another s' (Brafman and Shani 2012a). The expression C/s, when C is a conjunction of literals L, stands for the conjunction of the literals L/s.

Definition 1. *Let* $P = \langle F, I, A_F, A_N, G\rangle$ *be a* deterministic contingent problem. *The translation* $C_1(P)$ *of* P *is the* classical planning problem with axioms *such that* $C_1(P) = \langle F', I', A', G', X'\rangle$, *where*

[1] There is actually a paragraph in (Brafman and Shani 2012b) that suggests that while translations involving a number of actions linear in $|S_0|$ appears to be feasible, the authors didn't consider them because such actions would have "many conditional effects" and hence would be "challenging for current classical planners". This is a puzzling comment though, as an exponential number of actions is certainly much worse than a quadratic number of effects. For example, for a single action and $|S_0| = 30$, the contrast is between 1 billion actions with 20 conditional effects each vs. 30 actions with 900 effects.

- $F' = \{L/s \,:\, L \in P, s \in b_I\} \cup \{D(s,s') \,:\, s, s' \in b_I\}$,
- $I' = \{L/s \,:\, L \in P, s \in b_I, s \models L\}$,
- $G' = G$,
- $A' = \{a(s) \,:\, a \in A_F \cup A_N, s \in b_I\}$ *such that preconditions* L *in* $Pre(a)$ *are replaced by preconditions* XL/s *in* $Pre(a(s))$*, and effects*

  $C/s', \neg D(s,s') \rightarrow E/s'$

  for each $s' \in b_I$ *in place of the effect* $C \rightarrow E$ *for* $a \in A_F$,

  $\neg D(s,s'), \neg D(s,s''), p/s', \neg p/s'' \rightarrow D(s',s''), D(s'',s')$

  for each pair of states s', s'' *in* b_I *for* $a = a(p) \in A_N$,
- X' *is a set of axioms:*
  - *one for each* derived fluent XL/s *such that* L *is a literal precondition in* P *and* $s \in b_I$*, with definition* $\wedge_{s' \in b_I}[L/s' \vee D(s,s')]$,
  - *one for each literal* L *in* G*, with definition* $\wedge_{s \in b_I} L/s$.

In words, the primitive fluents in $C_1(P)$ represent the truth of the literals L in P conditioned on each possible hidden initial state s, and the (in)accessibility relation $D(s,s')$ among the "worlds" s and s'. Initially, these worlds are all accessible from each other and $D(s,s')$ is false for all such pairs. On the other hand, L/s is true initially if L is true in s. The goal G' of $C_1(P)$ is the same as the (conjunctive) goal G of P, and the truth of each goal literal L follows from the truth of the primitive fluent literals L/s in the translation via an axiom. For each action a in P, there is an action $a(s)$ in $C_1(P)$ for $s \in b_I$, with preconditions XL/s replacing the preconditions L of a in P. *The intuitive meaning of the derived literal* XL/s *is that* L *is known to be true in the execution that is associated with the hidden state* s. We don't use the notation KL/s from (Palacios and Geffner 2009), which means something different; namely, that L is true given s. Using the notation from modal logics, XL/s stands for $s \supset KL$, while KL/s stands for $K(s \supset L)$. Last, we write L/t rather than KL/t as in (Palacios and Geffner 2009), because when "tags" t represent the complete initial states s, it is not possible for KL/t and $K\neg L/t$ to be both false. Since if one is false, the other must be true, there is also no need for "cancellation axioms" (Palacios and Geffner 2009). The translation appears closest to the one in (Brafman and Shani 2012a), except for the use of action preconditions XL/s rather than L, which otherwise would not preserve completeness. Indeed, for an action $a(s)$ to be applicable in the classical problem $C_1(P)$, it is *not sufficient* for the literal L/s to be true for a precondition L of a in P; that is too weak. But it is *not necessary* for the derived literal L in $C_1(P)$ to be true either; that is too strong. Rather XL/s must be true, which according to its axiom will be true when L/s' is true for all the states s' that cannot be distinguished from s.

The number of actions in the translation is $O(A \cdot |b_I|)$, the number of fluents is $O(|b_I|^2)$, and the maximum number of conditional effects per (sensing) action is $O(|b_I|^2)$. In Brafman's and Shani's translation the numbers are $O(A \cdot 2^{|b_I|})$, $O(|b_I|^2)$, and $O(|b_I|)$ respectively. Thus, the number of conditional effects is reduced from quadratic to linear, but the number of actions explodes from linear to exponential.

The soundness and completeness of the translation $C_1(P)$ can be expressed in terms of the *tree policies* that solve P.

Theorem 1 (Completeness $C_1(P)$)**.** *Let* $n_0, \ldots, n_k$ *be a topological enumeration of the internal nodes in a policy tree that solves* P *where no node precedes its parent, and let* $\pi(n_i)$ *represent the action performed by the policy in the node* n_i *and* $D(n_i)$ *represent the hidden states in* b_I *that are compatible with the execution up to* n_i*. Then, any action sequence* $\pi'(n_0), \ldots, \pi'(n_k)$ *where* $\pi'(n_i) = a(s)$*,* $\pi(n_i) = a$*, and* $s \in D(n_i)$*, is a classical plan for* $C_1(P)$.

Theorem 2 (Soundness $C_1(P)$)**.** *Let* $b_0, \ldots, b_k$ *be a classical plan for* $C_1(P)$ *such that* $D_i(s,s')$ *represents the status of the* $D(s,s')$ *fluents when the action* b_i *is applied. Let* $n_0, \ldots, n_k$ *be a set of nodes such that node* n_i *is the parent of node* n_j *if A)* $i < j$*,* $b_i = a(s)$*,* $b_j = a'(s')$*,* $D_i(s,s')$ *is true, and B) condition A is not true for any* k*,* $i < k < j$*. In such a case, the edge from* n_i *to* n_j *is labeled* $\top$ *either if* a *is a physical action, or if* a *is a sensing action and* $D_j(s,s')$ *is true. Else the edge is labeled* $\bot$*. Then the policy* $\pi(n_i) = a$ *over the resulting labeled tree is a tree policy that solves* P.

There are a number of optimizations that can be accommodated in the translation. In particular, all the actions $a(s)$ for the same action a from P are equivalent in states s' where $D(s,s')$ holds, making the branching factor of the classical problem $C_1(P)$ unnecessarily large. A simple optimization in this case is to make just one of those actions applicable. This can be achieved by introducing a static ordering "$<$" among states, and by adding a derived fluent $first(s)$ as precondition of $a(s)$, such that $first(s)$ is defined by means of the axiom $\wedge_{s'<s} D(s',s)$. This means that s represents the first state in the group of all states that are indistinguishable from s. Notice that the predicate D is symmetric, and hence $D(s',s)$ is true iff $D(s,s')$ is true. This property can also be used to reduce the quadratic number of fluents and conditional effects by half.

Second Translation

The translation above converts the sequence of actions in *any* topological traversal of the policy tree into a classical plan. This however is not needed for the translation to be complete. For this, it is enough to account for the actions in *one specific traversal* of the policy tree; for example the unique depth-first search traversal where the child following a sensing actions $a(p)$ where p is true is considered first. In order to achieve this, we accommodate a *stack* in the translation where the states that predict p to be false are pushed, in order to be dealt with later. A parameter of this translation is the *stack size*. A stack size of k will suffice to obtain contingent plans with branches that accommodate up to k observations. The translation will be polynomial in this stack parameter, yet the resulting classical plans may be exponential in it. This however is not a characteristic of the translation but of the nature of full contingent plans represented by trees: their size is exponential in the maximum number of observations gathered along a branch of the tree. For this reason, the stack parameter is small in the benchmarks, as otherwise off-line contingent plans would not be able to solve them.

Of course, on-line contingent planners do not compute full solutions and do not have this limitation.

The second translation $C_2(P)$ preserves the fluents L/s from the first translation but removes the $D(s, s')$ fluents, that are quadratic in number and require a quadratic number of conditional effects. Instead, the new translation uses fluents $m(s)$ for keeping track of the set of possible initial states s that are possible given the execution so far. In addition, preconditions XL/s in the first translation are replaced by preconditions XL where the context given by the hidden state s is implicit. The stack is represented by fluents $lev(l)$ to indicate the top of the stack, $stack(s, l)$ to indicate that the hidden state s has been pushed onto the stack at level l, and static fluents $next(l, l + 1)$ that represent that one level follows the other. For simplicity, we omit these static fluents which are true for each $l \in [0, M]$ where $M + 1$ is the max stack level. For convenience, we also assume that the goal G is a single atom; if it is not the case, problems can be brought into this form in the standard way by adding a dummy goal and a final action.

Definition 2. *Let $P = \langle F, I, A_F, A_N, G\rangle$ be a* deterministic contingent problem*. Then the translation $C_2(P)$ of P for a given stack parameter $M > 0$ is the* classical planning problem with axioms $C_2(P) = \langle F', I', A', G', X'\rangle$ *where*

- $F' = \{L/s, m(s), lev(l), stack(s, l) : L \in P, s \in b_I, l \in [0, M]\}$
- $I' = \{L/s, m(s), lev(0) : L \in P, s \in b_I, s \models L\}$,
- $G' = G$,
- $A' = A_F \cup \{a(q, l), pop(l+1) : a(q) \in A_N, l \in [0, M]\}$ *such that*
  - ***Physical actions:*** *preconditions L of $a \in A_F$ replaced by XL and $\neg XG$; effects $a : C \to E$ replaced by $a : C/s, m(s) \to E/s$ for each $s \in b_I$,*
  - ***Sensing actions:*** *preconditions L of $a(q)$ in A_N become preconditions XL for $a(q, l)$ in addition to $lev(l)$, $\neg XG$, $\neg Xq$, $\neg X\neg q$; effects of $a(q, l)$ are $\neg lev(l)$, $lev(l + 1)$, and conditional effect $m(s), \neg q/s \to stack(s, l + 1), \neg m(s)$,*
  - ***Pop actions:*** *preconditions of $pop(l)$ are $lev(l)$ and XG; effects are $\neg lev(l)$, $lev(l - 1)$ and conditional effects $m(s) \to \neg m(s)$ and $stack(s, l) \to m(s), \neg stack(s, l)$ for each $s \in b_I$.*
- *X' is a set of axioms:*
  - *one for each* derived fluent *XL such that L is literal precondition or goal in P with definition $\wedge_{s\in b_I}[L/s \vee \neg m(s)]$,*
  - *one for each literal L in G with definition $\wedge_{s\in b_I} L/s$.*

The number of actions in the translation is $O(|A_F| + M \cdot |A_N|)$ where $M + 1$ is the stack size, the number of fluents is $O((|F| + M) \cdot |b_I|)$, while the maximum number of conditional effects per action is $O(|S| \cdot |F|)$. None of these numbers grows with the square number of states in b_I as in the translation $C_1(P)$, or exponentially in $|b_I|$ as in the translation of Brafman and Shani. The emulation of a stack in the translation is reminiscent of Domshlak's compilation scheme for mapping a class of fault tolerant planning problems into classical problems (Domshlak 2013).

In relation to $C_1(P)$, the new translation replaces the $D(s, s')$ literals encoding the accessibility relations among worlds by the $m(s)$ literals that represent the set of hidden initial states that are possible given the current execution. The new translation uses also a stack where the hidden states s that predict $\neg q$ after the execution of the sensing action $a(q, l)$ are stored. The translation makes also sure that q is not known either true or false by adding the literals $\neg Xq$ and $\neg X\neg q$ in the action precondition. Finally, the translation adds the $pop(l+1)$ actions: they are triggered when an execution reaches the goal in the form of the XG literal, so that the executions associated with hidden states that have been pushed onto the stack become current and can be extended to reach the goal as well.

The soundness and the completeness of the translation $C_2(P)$ can be expressed as follows. In a policy tree for P, we refer by the level $lev(n_i)$ of a node n_i to the number of "right turns" in the path from the root of the tree to the node n_i, where a right turn corresponds to the observation that an atom q is false following a sensing action $a(q)$.

Theorem 3 (Completeness $C_2(P)$)**.** *Let $n_0, \ldots, n_k$ be a DFS enumeration of* all *the nodes in a policy tree that solves P where no node precedes its parent and where children associated with positive observations come first. Let $\pi(n_i)$ represent the action performed by the policy in node n_i, let $D(n_i)$ represent the set of hidden states in b_I which are compatible with the execution up to n_i, and let $lev(n_i)$ be the level of the node n_i in the tree. If these levels are not greater than $M + 1$, then the action sequence $\pi'(n_0), \ldots, \pi'(n_{k-1})$ is a classical plan for $C_2(P)$, where $\pi'(n_i) = \pi(n_i)$ if $\pi(n_i)$ is a physical action, $\pi'(n_i) = a(q, l)$ if $\pi(n_i)$ is the sensing action $a(q)$ and $l = lev(n_i)$, and $\pi'(n_i) = pop(l)$ if n_i is leaf node of the tree and $l = lev(n_i)$.*

Theorem 4 (Soundness $C_2(P)$)**.** *Let $b_0, \ldots, b_k$ be a classical plan for $C_2(P)$ such that $m_i(s)$ represents the status of the $m(s)$ fluents when the action b_i is applied. Let $n_0, \ldots, n_k$ be a set of nodes such that node n_i is the parent of node n_j if $i < j$ and there is a state s such that both $m_i(s)$ and $m_j(s)$ are true and there is no k, $i < k < j$, such that $m_k(s)$ is true as well. In such a case, the edge from n_i to n_j is labeled $\top$ either if a is a physical action, or if a is a sensing action and $j = i+1$. Else the edge is labeled $\bot$. Then the policy $\pi(n_i)$ over the internal nodes n_i of the tree such that $\pi(n_i) = b_i$ if b_i is a physical action, and $\pi(n_i) = a(q)$ if b_i is a sensing action $a(q, l)$, represents a policy that solves P.*

Empirical evaluation

We will turn now to evaluate the translations empirically over benchmarks collected from the distributions of contingent-FF, Pond, and CLG, and DNFct. The experiments were run on a cluster of Linux boxes AMD Opteron Abu Dhabi 6378 processors at 2.4 GHz. Each experiment had a cutoff of 2h or 4GB of memory. For the translation C_2 we used an stack size $M = 6$.

Table 1 compares existing contingent planners with the classical planner Fast Downward LAMA 2011, that we will

	$C_1(P)$		$C_2(P)$		Pond		MBP		CFF		CLG		DNFct	
Problem	time	size	time	size	time	size	time	size	time	size	time	size	time	size
blocks 3	0.0	7	0.2	6	0.01	5	0.33	7	0.02	6	0.08	6	0.58	5
blocks 7	45.8	63	116.9	63	OM		TO		0.46	49	4.58	55	11.6	69
blocks 11	490	115	TO		OM		TO		TO		35.68	115/18	OM	
blocks 15	463.9	119	TO		OM		TO		TO		144.2	157/22	OM	
ebtcs 50	1805	100	388.7	100	6.0	99	TO		11.96	99	0.02	21	0.13	101
ebtcs 70	MT		2216	140	29.8	139	TO		69.66	139	24.79	209	1.49	276
ebtcs 90	MT		TO		TO		TO		255	179	69.99	269	3.19	356
ebtcs 150	MT		TO		TO		TO		MC		603.3	449	6.25	596
grid 3	92.7	26	153.7	119	105	148	TO		943	58	1180	111	1.97	313
grid 4	1809	45	TO		OM		TO		TO		1558	884	2.9	982
grid 5	1260	70	134.5	97	OM		TO		TO		657	208	12.59	133
medpks 50	2495	101	6084	101	192.5	100	TO		164.9	100	2.32	101	1.45	201
medpks 70	MT		OM		TO		TO		968.6	140	7.51	141	1.49	141
medpks 90	MT		OM		TO		TO		TO		24.35	199	2.69	199
unix 1	0.1	19	0.2	17	5.1	26	11.58	21	0	7	0.01	21	0.01	17
unix 2	12.4	72	14.3	48	1.71	48	TO		0.13	48	0.35	50	0.78	48
unix 3	1521	197	TO		OM		TO		3.84	111	4.93	113	2.02	111
unix 4	MT		OM		OM		TO		143	238	78.9	240	16.26	238
doors 3	0.0	13	0.0	13	-	-	-	-	E		0	13	0.01	17
doors 5	429.7	164	TO		-	-	-	-	E		0.4	144	0.04	146
doors 7	MT		TO		-	-	-	-	E		13.4	2153	4.28	2193
localize 3	5.7	31	0.7	24	-	-	-	-	42	53	0.02	33	0.01	19
localize 5	301	81	8.0	76	-	-	-	-	MC		1.86	112	0.57	49
localize 7	5551	171	161.2	177	-	-	-	-	MC		6.89	231	0.72	80
localize 11	TO		TO		-	-	-	-	MC		63.72	577	1.2	144

Table 1: Comparison of Contingent planners. Translations solved by classical planner LAMA. TO stands for time out, OM for out of memory, MC and E that the problem is too big or a faulty execution respectively for CFF, MT that translator went memory out, '-' means no information on performance of planner over instance.

refer simply as LAMA (Richter and Westphal 2010; Helmert 2006), ran over the two translations. As it can be seen from the table, CLG and DNFct do best, but LAMA over $C_1(P)$ and $C_2(P)$ does not trail behind POND, Contingent-FF, and MBP in terms of coverage and quality.

Table 2 shows performance of three classical planners over the translations C_1 and C_2. The planners LAMA, FF (Hoffmann and Nebel 2001), and a version of the SIW planner that uses the h_{add} heuristic (Lipovetzky and Geffner 2012).

In almost all the instances, SIW is slower in generating the nodes than the other planners, but this is compensated by a better ratio between the number of expanded nodes and the length of plans, meaning that the search is informed enough to go to the goal while expanding few nodes. LAMA has the opposite behavior, generating nodes fast but expanding many more nodes per step in the plan. The relaxed heuristic of FF is not very informed except for the Medpks domain. LAMA uses a heuristic derived from landmarks in combination with the well-known FF heuristic; the initial values of these two heuristics are reported in 'heur' column.

There is a correlation between the size of the PDDL file and the overhead that affects the scalability of the classical planners. The size of the PDDLs is directly proportional to the number of fluents, actions and conditional effects in the translation. For creating the actual PDDLs, axioms are compiled away in ramifications and actions interleaved during the execution. These dummy actions represent many of the actions appearing in the resulting classical plans. Due to the treatment of axioms, the PDDLs are actually larger than the sizes that follow from the analysis in Sections 3 and 4. As a reference, the translation K_{s0} that maps conformant into classical planning problems (Palacios and Geffner 2009) and is linear in the number of initial states, dealing with a conformant problem equivalent to Blocks7, produces a classical problem with 445/1k/13k actions/atoms/conditional-effects. In contrast C_1 produces 3k/1k/177k and C_2 produces 777/1k/184k. K_{s0} and C_2 have a similar number of actions and atomos, while C_1 has more actions, as they grow linear with the number of initial states. In contrast, both C_1 and C_2 have many more conditional effects. Table 3 shows figures concerning the translation of selected instances, in comparison with the figures that correspond to the translation used by the CLG planner.

We have also tested our translations with many other classical planners and configurations that we are not reporting in the tables. We tried for example a version of the Fast Downward planner with the causal graph and the landmark heuristics. These are heuristics that are not as sensible to the approximations made by the delete-relaxation. We also tried other variants of the width-based algorithms in the SIW and BFS(f) family reported in (Lipovetzky and Geffner 2012). And finally, we tried the SAT-based planner Mp. In all cases, we didn't obtain better results, and for Mp the results were definitely inferior, reporting solutions to 8 of the 60 instances we tried. The coverage of the FD planner with the

C_1 **Translation**		SIW			LAMA				FF		
problem	pddl	size	#exp/#act	node/sec	size	#exp/#act	heur	node/sec	size	heur	node/sec
dispose 3 1	103	213	11.77	839	174	24.32	13/6	21018	OM	6	
dispose 4 1	522	576	33.35	40	375	1310	20/6	7688	OM	6	
doors 3	11.1	39	1.18	N/A	39	141.5	10/6	58417	39	6	N/A
doors 5	2115	930	22.74	16	492	67.63	35/8	1031	OM	8	
ebtcs 30	913.4	180	4.67	6	180	32.37	35/4	24459	OM	4	
ebtcs 50	4043	300	4.56	1	300	169.5	55/4	2338	OM	4	
elog 5	1081	450	25.97	3	330	2085	27/18	15812	OM	18	
elog 7	3621	TO			TO		39/23		OM	23	
grid 2	891	111	2.22	39	27	24081	40/9	1603	27	9	N/A
grid 4	7367	TO			135	212.01	106/18	24	111	20	49
localize 3	235	90	3.66	195	93	3383	11/7	8763	72	17	2597
localize 5	3431	558	35.68	2	243	90.17	22/6	948	OM	24	
medpks 30	1184	180	2.81	2	180	2.09	92/62	27755	180	63	342
medpks 50	5542	TO			303	307	152/102	17574	303	103	16
push 3 1	562.4	177	8.19	65	165	13.87	12/7	4926	OM	8	
unix 2	3169	453	35.62	4	216	1.45	15/6	1116	OM	6	
unix 3	81058	OM			591	3.26	31/6	35	OM		
C_2 **Translation**		SIW			LAMA				FF		
problem	pddl	size	#exp/#act	node/sec	size	#exp/#act	heur	node/sec	size	heur	node/sec
dispose 4 1	450	395	4.38	89	259	16	17/7	122	OM	6	
dispose 5 1	1303	667	9.11	25	411	4.72	26/7	18	OM	6	
doors 3	34.1	29	1.69		29	28.07	3/5	11450	29	5	8600
doors 5	850	355	3.24	43	TO		25/9		335	9	683
ebtcs 30	1317	243	2.61	28	177	4.2	31/5	83		4	
ebtcs 50	5405	363	6.42	5	297	4.4	51/5	27	OM	4	
elog 5	362	301	5.99	52	287	148.4	24/21	1758		19	
elog 7	553	481	7.23	42	401	554.5	36/21	778	OM	17	
grid 3	535	407	6.72	44	257	62.77	36/12	187	89	12	732
grid 5	1367	IW>2			97	163.15	136/16	39	OM	16	
localize 3	82.4	75	4.05	527	59	16469	8/5	2472	65	10	6564
localize 7	778	791	7.35	13	417	20.16	34/5	146	OM	22	
medpks 50	10609	299	5.87	1	299	1.49	101/53	21	299	53	108
medpks 99	86763	OM			TO				593	102	9
push 3 1	208	147	2.23	163	121	6.96	9/7	994		7	
push 3 2	4342	OM			1467	10.51	162/10	7	OM	9	
unix 2	718	137	8.39	36	117	4.21	12/9	147	OM	5	
unix 3	5918	OM			TO		28/9		OM	5	

Table 2: Performance of classical planners over the translations. 'N/A' indicates that the resolution time is too small to be reported. 'pddl' is the size in KB of the input files, '#exp/#act' the ratio between expanded nodes and actions in the plan, 'heur' the initial heuristic value, 'node/sec' generated nodes per second. TO stands for time out, OM for out of memory. IW>2 means SIW failed.

indicated heuristics was 26/30 for C_1/C_2, and for FD using the landmark heuristic was 15/25. FF coverage was 11/14. SIW, as presented by the authors, coverage was 16/25.

It is interesting to note that none of the reported classical planner dominated the others, and moreover, the planners exhibit very different behaviors over the translations, with some planners solving some of the instances rather easily while others produce memory or time outs in the same instances. Also, some encoding details are important. For example, we found that copying the preconditions into conditions[2] before the translation made a big difference in the performance of the FF planner that jumped from solving only 14 instances of C_2, to solving 27.

The size of the translations does not coincide with the sizes predicted by the analysis in Sections 3 and 4. The reason is the compilation of the axioms. We compiled the axioms away because many of the planners do not handle axioms properly. This, however, doesn't seem to be a good idea, because the elimination of the axioms produces many additional fluents and actions, that may be affecting performance drastically. One of the few modern planners that provides native support for axioms is FF-X (Thiébaux, Hoffmann, and Nebel 2005), but the overhead is then considerable as the class of axioms handled by FF-X is much more expressive than what is needed by our translations. FF-X supports indeed recursive, stratified definitions, while our definitions are simple and acyclic.

One open question is related with the low performance of the SAT-based planner Mp, which solves seven instances over the two translations C_1 and C_2. A translation to be ef-

[2]Such operation is sound and helps the delete-relaxation heuristics.

Problem	P	$C_1(P)$		$C_2(P)$		CLG	CLG-S_0
	Act / Atom / CE	Time	Act / Atom / CE	Time	Act / Atom / CE	Act / Atom / CE	Act / Atom / CE
blocks7	455 / 72 / 1k	1.9	3k / 1k / 177k	5.4	777 / 1k / 184k	6k / 4k / 263k	1k / 2k / 56k
blocks11	1k / 152 / 5k	6.4	12k / 2k / 570k	31.4	2k / 2k / 488k	15k / 10k / 736k	4k / 4k / 176k
dispose-2 2	62 / 54 / 96	1.4	322 / 1k / 19k	1.1	73 / 1k / 15k	171 / 596 / 2k	716 / 1k / 24k
dispose-2 3	71 / 60 / 110	125.7	1k / 8k / 1m	7.3	102 / 7k / 108k	239 / 716 / 3k	6k / 8k / 818k
doors 3	48 / 50 / 72	0	71 / 125 / 385	0.1	91 / 155 / 1k	139 / 497 / 1k	149 / 354 / 1k
doors 5	160 / 138 / 240	15.2	1k / 3k / 188k	5.7	303 / 2k / 73k	867 / 2k / 16k	2k / 3k / 115k
ebtcs-30	61 / 40 / 91	9.2	962 / 3k / 54k	3.4	218 / 3k / 189k	2k / 3k / 108k	2k / 4k / 197k
ebtcs-50	101 / 60 / 151	70.9	2k / 8k / 251k	16.4	358 / 8k / 823k	5k / 8k / 450k	7k / 11k / 845k
elog5	93 / 56 / 159	0.6	746 / 712 / 31k	0.5	235 / 865 / 27k	246 / 602 / 2k	672 / 1k / 12k
elog7	93 / 56 / 159	1.7	1k / 1k / 100k	0.8	235 / 1k / 44k	266 / 628 / 2k	992 / 1k / 23k
grid-4	232 / 134 / 436	26.7	6k / 6k / 649k	8.3	359 / 6k / 101k	2k / 3k / 49k	6k / 8k / 308k
grid-5	196 / 122 / 364	23.2	5k / 5k / 581k	7.2	311 / 5k / 90k	2k / 2k / 41k	5k / 7k / 278k
localize 11	9 / 90 / 633	229.7	686 / 15k / 7m	8.5	36 / 25k / 200k	12k / 13k / 1m	18k / 19k / 2m
localize 13	9 / 117 / 853	672.7	929 / 28k / 18m	19.5	36 / 45k / 350k	22k / 23k / 3m	32k / 35k / 7m
medpks-050	102 / 112 / 151	89.7	2k / 14k / 259k	46.5	358 / 21k / 1m	7k / 8k / 573k	13k / 16k / 1m
medpks-150	300 / 310 / 448	-	MT	2862	1k / 181k / 41m	67k / 70k / 13m	112k / 138k / 27m
push 3 1	59 / 52 / 109	0.4	533 / 472 / 20k	0.2	111 / 616 / 17k	266 / 683 / 5k	349 / 644 / 8k
push 3 2	94 / 62 / 170	-	MT	5.4	191 / 8k / 446k	502 / 1k / 9k	9k / 12k / 1m
unix 2	253 / 38 / 491	1.7	3k / 946 / 134k	0.7	335 / 1k / 62k	649 / 951 / 21k	795 / 1k / 30k
unix 3	1k / 70 / 2k	49	28k / 4k / 3m	9.4	1k / 5k / 608k	2k / 3k / 184k	3k / 4k / 280k

Table 3: Translation data for selected instances: Act, Atom, and CE stand for the number of actions, fluents, and conditional effects. Time is the translation time in seconds. P stands for the original problem, MT means that the translator went memory out.

fective should usually take into account the kind of problems where state-of-the-art planners perform the best. Which translation should work better for Mp? The translation C_2 is indeed smaller than C_1, but imposes an ordering on the traversals of the policy trees.

The implementations of the translations could be improved in a number of ways. One of the first to be explored is the value of compiling the axioms vs. dealing with them directly in the state progression and in the computation of the heuristics. Other details in the implementation could narrow the gap in performance with the best off-line contingent planners further. More powerful classical planning algorithms would help as well, indicating that those translated problems are a challenging test bench for classical planners. The huge differences in performance that results from the use of the different planners over the translations, suggest that there is also a lot of room for improvement, and for adjusting the details of the translations, to make the most of the techniques captured by these planners.

Summary

We have introduced two translations for mapping deterministic contingent problems into classical problems that, unlike Brafman's and Shani's translation, are polynomial and can be used for solving contingent problems off-line. From the empirical results obtained so far, the approach is not yet at the level of the last generation of off-line contingent planners, but it is certainly on par with other recent contingent planners such as Contingent-FF, POND, and MBP. Moreover, the limitations in performance are not always a result of the size of the translations, so further advances in classical planning algorithms will have an impact on the class of contingent problems that can be solved in this manner. A limitation of off-line contingent planners regardless of the approach is that the size of plans is often exponential in the number of observations, a limitation that is not shared by online planners. We expect these ideas to be relevant to other types of planning problems as well, like Q-Dec-POMDPs (Brafman, Shani, and Zilberstein 2013), and other types of partially observable multiagent planning problems.

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *Proc. Int. Joint Conf. of Artificial Intelligence (IJCAI-09)*, 1623–1628.

Bernstein, D.; Zilberstein, S.; and Immerman, N. 2000. The complexity of decentralized control of Markov decision processes. In *Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence*, 32–37.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2006. Strong planning under partial observability. *Artificial Intelligence* 170(4-5):337–384.

Brafman, R., and Shani, G. 2012a. Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research* 45(1):565–600.

Brafman, R. I., and Shani, G. 2012b. A multi-path compilation approach to contingent planning. In *Proc. of AAAI*.

Brafman, R. I.; Shani, G.; and Zilberstein, S. 2013. Qualitative planning under partial observability in multi-agent domains. In *Proc. of AAAI*.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research* 26:35–99.

Domshlak, C. 2013. Fault tolerant planning: Complexity and compilation. In *Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS 13)*.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

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

Hoffmann, J., and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proc. 15th Int. Conf. on Automated Planning and Scheduling (ICAPS 2005)*, 71–80.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In *Proc. of European Conf. of Artificial Intelligence (ECAI 12)*, 540–545.

Palacios, H., and Geffner, H. 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificial Intelligence Research* 35:623–675.

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

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1–2):38–69.

To, S. T.; Pontelli, E.; and Son, T. C. 2011. On the effectiveness of CNF and DNF representations in contingent planning. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI-11)*, 2033–2038.