From Non-Deterministic to Probabilistic Planning with the help of Statistical Relational Learning

Ingo Thon[1] and **Bernd Gutmann**[1] and **Martijn van Otterlo**[1] and **Niels Landwehr**[2] and **Luc De Raedt**[1]
[1]Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
firstname.lastname@cs.kuleuven.be
[2]Department of Computer Science, University of Potsdam, August-Bebel-Str. 89, 14482 Potsdam, Germany
lastname@cs.uni-potsdam.de

Abstract

Using machine learning techniques for planning is getting increasingly more important in recent years. Various aspects of action models can be induced from data and then exploited for planning. For probabilistic planning, natural candidates are learning of action effects and their probabilities. For expressive formalisms such as PPDDL, this is a difficult problem since they can introduce easily a hidden data problem; the fact that multiple action outcomes may have generated the same experienced state transitions in the data. Furthermore the action effects might be factored such that this problem requires solving a constraint satisfaction problem within an expectation maximization scheme. In this paper we outline how to utilize recent techniques from the field of statistical relational learning for this problem. More specifically, we show how techniques developed for the CPT-L model of relational probabilistic sequences can be applied to the problem of learning probabilities in a PPDDL model. A CPT-L model concisely specify a Markov chain over arbitrary numbers of objects in the domain and simultaneous applications of multiple actions. The use of efficient BDD-style representations allows for fast and efficient learning in such domains. Even efficient online learning is possible as we will show in this paper. We relate to other learning approaches for similar domains and highlight the opportunities for incorporating our approach into architectures that can plan, execute the plan, and learn from the outcomes, in an online and incremental fashion.

Introduction

Planning is one of the oldest problems in artificial intelligence, and yet it is still a very active field of research. In part, this is due to efficient and effective algorithms becoming available such that the range of problems tackled can be widened significantly over time. A very basic definition of planning is: **given** a set of *action definitions* (or, *operators*), an *initial state* of the world, and a set of desired *goal states*, **find** a sequence of instantiated actions such that applying this sequence starting in the initial state will reach the goal state. In this basic setting, many algorithms have been developed that can find satisfying or even optimal plans. Also the theoretical aspects of planning are well-studied.

Going beyond the basic definition of planning can be done in numerous ways. Two important directions are using more complex problem models, and using machine learning to learn aspects of the modeled domain. For the first, among others, this includes the use of sensing, conditional plans, non-deterministic or probabilistic outcomes, simultaneous actions, time and numerical quantities, and much more. In this paper we focus on *probabilistic* settings, in which there is significant uncertainty about the effects of actions. Problems similar to the *probabilistic* setting occur in *nondeterministic planning*, yet in the former the uncertainty can be numerically quantified using probability distributions over action outcomes. Concerning the second direction, various aspects of the domain could be learned, such as necessary preconditions for actions, action effects, probabilities for these effects, domain-specific heuristics and control knowledge (Zimmerman and Kambhampati 2003; Fernández, Jiménez, and de la Rosa 2009). A special case of the learning problem has been recently proposed as the *model-lite planning paradigm*(Kambhampati 2007). Even though defining a complete description of the domain is often hard, sketching the most important aspects might be easy in most cases. The idea of *model-light planning* is to plan with the current-best model. While acting and gaining experience it is possible to update the model and therefore better plans can be derived. But relational machine learning techniques for probabilistic and dynamic settings, where the world changes over time as typical in planning domains, are scarce. In fact, the CPT-L (**C**ausal **P**robabilistic **T**ime **L**ogic) models (Thon, Landwehr, and Raedt 2008) are among the first to approach the dynamic setting in statistical relational learning (SRL). CPT-L makes use of highly efficient *binary decision diagrams* (BDDs) to assign probabilities to logical structures. The intention of the present paper is to bridge between the planning and the SRL communities, by adapting our algorithms to the standardized language PPDDL for probabilistic planning (Younes et al. 2005), and by showing that SRL and planning have much to offer to each other. In this work we focus on the parameter estimation and not on the structure learning part, i.e. we assume that a non-deterministic PPDDL action model is given, and we want to learn the probabilities for the action effects. This is on the one hand because it seems to be especially hard for humans to quantify probabilities, extending the algorithm on the other hand such that also the structure is learned, can be achieved by using wrapper approach as standard in *Inductive Logic Programming* or by generating action definitions with the help of rule learner.

Contributions and Outline The main contribution is an efficient, principled way to learn the probabilities for PPDDL action models. We show how SRL techniques can be exploited for planning domains, and how online learning can be done. Furthermore we show how the use of so-called Shared BDDs allows efficient Online Learning. We first discuss related work, after which we describe our formalisms and algorithms, based on efficient BDD representations, in detail. We show how typical tasks such as *sampling*, *inference*, *parameter estimation* and *prediction* can be handled in terms of the BDDs and we conclude with directions for further research.

Related Work

Learning planning operators has been studied intensively. Specifically the problem of learning (the probabilities of) the effects of actions, in both deterministic and probabilistic settings, and furthermore some in the context of integrated architectures for planning and learning and relational reinforcement learning (van Otterlo 2009). Reinforcement learning (model-free) can be seen as the simplest and classic approach to *model-lite planning*. The *model-lite planning* paradigm itself has been proposed recently so there is only little specific research. One exception is planning with the help of probabilistic logics namely Markov Logic(Yoon and Kambhampati 2007). Problog has been proposed in this work as well but has not been further investigated. The Models learned with help of the techniques proposed in this papers can be used for planning with the help of Problog[1]. The use of Problog/CPT-L/PPDDL compared to Markov Logic seems to be more appealing as it is closer to the traditional planing languages.

World models – or models of the effects of actions – are useful things an agent can learn, because these embody *knowledge* about the environment that can be exploited in various ways, for example in a probabilistic planner, or even in *first-order* planning, e.g. (Kersting, van Otterlo, and De Raedt 2004). Learning them has been studied for several decades, with the THOTH system (Vere 1977) as one of the first. The more recent OBSERVER system (Wang 1995) can learn STRIPS operators in an *incremental* way.

For the probabilistic case, fewer approaches exist. The best known example is the approach by Pasula, Zettlemoyer, and Kaelbling (2007) which learns both the structure and the probabilities of actions in a STRIPS-like form. Learning consists of a three-step greedy search approach that finds rule sets, consisting of action outcomes and probability distributions over these outcomes. Two interesting features of the approach are the *noise outcomes* that correspond to situations for which the exact, structural outcome is hard to model (e.g. the exact effects of knocking over a tower of blocks on each block individually), and the invention of new features in the learning process. Earlier work by (Gardiol 2003) who used standard ILP learning on state transitions, exemplified some of these difficulties of probabilistic domains. Where Pasula, Zettlemoyer, and Kaelbling's approach is a batch technique, the algorithm developed by Safaei and Ghassem-Sani (2007) can learn probabilistic actions in an incremental way, similar to what Wang's (1995) OBSERVER system can do in deterministic domains. Whereas for these approaches calculating the maximum likelihood estimates (MLE) for the probabilities is simple due to the underlying assumptions, Jiménez and Cussens (2006) are first in using a more principled technique for a similar problem, by exploiting PRISM (Sato and Kameya 1997) for the MLEs. In addition, it approaches the structure learning problem too. The approach in this paper differs from the previous approaches in that it is based on a more general dynamic SRL setting, it uses efficient BDD algorithms and we employ it directly in the PPDDL formalism.

A more general related setting is that of *integrated architectures* that combine *learning*, *planning*, and *action execution*, of which PELA (Jiménez, Fernández, and Borrajo 2008) is a prominent example. PELA uses off-the-shelf techniques for each of these aspects. But whereas PELA is based on classification learning with the relational decision tree learner TILDE, we employ a general sequence learning approach. A related approach based on kernel perceptrons was developed in a broader context of *robotics* and *vision*, where high-level planners have to be combined with low-level (robotic) vision systems (Petrick et al. 2008). The approach by Safaei and Ghassem-Sani (2007) implements the planning step through real-time dynamic programming, making the connection with *model-based reinforcement learning* (MBRL) in a relational setting. MBRL approaches such as DYNA (Sutton 1991) learn an action model while acting in an environment, and use that model to improve action execution and planning. For the relational case only few examples exist.However, trading off acting and exploration can provide good opportunities for generating useful learning examples for the induction of operator descriptions. Many algorithms exists for acting and learning to act in relational worlds (see (van Otterlo 2009) for a thorough description), yet the exploitation of learned probabilistic models is not well developed yet. One positive exception by Lang and Toussaint (2009) uses learned rules from Pasula, Zettlemoyer, and Kaelbling's approach in an *approximate planning* algorithm which show good results for a realistic Blocks world. In the current paper we focus on one aspect of an integrated architecture, namely the learning of a probabilistic transition model. However, we also show that all related aspects such as efficient planning, inference and structural learning are within reach.

Preliminaries

Let us first introduce some terminology. A logical *atom* is an expression of the form $p(t_1, ..., t_n)$ where p/n is a *predicate symbol* and the t_i are *terms*. Terms are built up from constants, variables, and functor symbols. Constants are denoted in lower case (such as a), variables in upper case (such as X), and functors by f/k where k is the arity of functor f. The set of all atoms is called a *language* $\mathcal{L}$. *Ground* expres-

[1]The traditional learning setting for Problog is to learn from success probabilities of queries, not required in the fully observable setting.

PPDDL description of the "bomb and the toilet" example

```
(define (domain bomb-and-toilet)
        (:requirements :conditional-effects :probabilistic-effects)
        (:predicates (bomb-in-package ?pkg) (toilet-clogged) (bomb-defused))
        (:action dunk-package :parameters (?pkg)
                 :effect (and (when (bomb-in-package ?pkg) (bomb-defused))
                              (probabilistic 0.05 (toilet-clogged)))))
```

CPT-L description of the action

$$\texttt{rule1(Pkg)} : \texttt{bomb-defused} :: 1.0 \longleftarrow \texttt{action(dunk-package(Pkg))}, \texttt{bomb-in-package(Pkg)}.$$
$$\texttt{rule2(Pkg)} : \texttt{toilet-clogged} :: 0.05 \vee \texttt{true} :: 0.95 \longleftarrow \texttt{action(dunk-package(Pkg))}.$$

Figure 1: The "Bomb and Toilet" example in PPDDL and the corresponding CPT-L description.

sions do not contain variables. Ground atoms will be called *facts*. A substitution θ is a mapping from variables to terms, and $b\theta$ is the atom obtained from b by replacing variables with terms according to θ. As an example, consider the substitution $\theta = \{X/a\}$ that replaces variable X with a, as in $b\theta = p(a)$ for $b = p(X)$. Complex world states can now be described in terms of *interpretations*. An interpretation I is a set of ground facts $\{a_1, ..., a_N\}$. These ground facts can represent objects in the current world state, their properties, and any relationship between objects as seen in Figure 2.

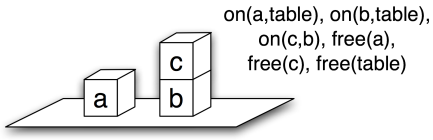


Figure 2: Blocks-world example as logical representation.

PPDDL

The **P**lanning **D**omain **D**efinition **L**anguage (PDDL) is widely-used language to specify planning problems in a formal way. It was developed for the International Planning Competition, and is based on the functional programming language LISP. It offers a well-standardized notation which allows an objective comparison of planning systems and reusing rules for different domains.

Probabilistic PDDL (PPDDL) is an extension which allows actions having probabilistic effects of the form $(\texttt{probabilistic}\ p_1\ e_1\ \ldots p_n\ e_n)$. As we want to learn the dynamics underlying the environment in which the agent operates, additional rewards do not get a special treatment. The action schema we use, subsumes PPDDL and additionally allows the parallel application of multiple actions. However, for this paper we restrict ourself to plans with non-parallel execution of actions. Following (Rintanen 2003) we define actions as follows.

Definition 1 (PPDDL action)**.** *An action $a = \langle \phi_a, e \rangle$ consists of a precondition ϕ_a, and an effect e. The action a is applicable in state s iff $s \models \phi_a$. The effect e can be:*

- $\top$ *is the empty effect*
- p *and* $\neg p$ *are effects if* p *is a state variable*
- $x \leftarrow f$ *is an effect if* x *is a real valued state variable and* f *is a real valued function.*
- $e_1 \wedge \ldots \wedge e_n$ *are effects if the* e_i*'s are effects.*
- $c \rhd e$ *is an effect if* e *is an effect and* c *is a formula*
- $p_1\ e_1 | \ldots | p_n\ e_n$ *is an effect if* e_i *are effects and* $p_i > 0$ *and* $\sum_i p_i = 1$

Like in PPDDL we require that the action model is consistent. A domain is *consistent* if there are no two effects which can be applied the same time and make a state variable true and false respectively. For example $b \wedge \neg b$ is inconsistent whereas $c \rhd b \wedge \neg c \rhd \neg b$ is consistent.

Furthermore we assume that all actions are given in conditional normal form (CNF). That is, no conditional is nested into a probabilistic effect. For example the action

$$((0.8\ move(a,b) \wedge on(a,c)) \rhd (on(a,b) \wedge \neg on(a,c)) \mid 0.1\ \top \mid 0.1\ (move(a,b) \wedge on(a,c)) \rhd (on(a,table) \wedge \neg on(a,c)))$$

is not in conditional normal form, whereas

$$(move(a,b) \wedge on(a,c)) \rhd (\ 0.8\ (on(a,b) \wedge \neg on(a,c)) \mid 0.1\ \top \mid 0.1\ (on(a,table) \wedge \neg on(a,c)))$$

is in conditional normal form.

There are basically two ways to learn PPDDL probabilities by the means of CPT-L: (1) we can adopt the algorithms of CPT-L, or (2) we can map the representation onto a slightly extended CPT-L representation while preserving the probabilistic semantics of each parameters and apply the algorithm there. For simplicity we decided for the second approach.

CPT-L

CPT-L for **C**ausal **P**robabilistic **T**ime-**L**ogic (Thon, Landwehr, and Raedt 2008) is a language for (first-order) Markov processes, where the possible successor states are described in a causal and constructive manner. It assumes, that there exists for any sequence of interpretations an underlying generative process that constructs the next interpretation from the current one.

Definition 2. *A* **CPT-theory**[2] *is a set of rules r of the form*

$$r = \underbrace{h_1 : p_1 \vee \ldots \vee h_n : p_n}_{head(r)} \leftarrow \underbrace{b_1, \ldots, b_m}_{body(r)}$$

where the h_i are atoms, $p_i \in [0,1]$ are probabilities s.t. $\sum_{i=1}^{n} p_i = 1$, and the b_l are literals (i.e., atoms or their negation).

The intuition behind such a rule r is as follows. Whenever $b_1\theta, ..., b_m\theta$ holds for a substitution θ in the current state I_t it is applicable. For each applicable rule exactly one head element $h_i\theta$ is sampled from $head(r)$ and $h_i\theta$ is added to the next state I_{t+1}. This sampling process is governed by the probabilities p_i attached to the head.

Each ground rule applicable in I_t will cause one of its grounded head elements to be selected, and the resulting atoms to become true in I_{t+1}.

A CPT-Theory consists of a set of rules describing conditional, stochastic effects. Given an action for every state the semantics of Strips or PPDDL as stochastic process and CPT-L are closely related, but differ in two points. Firstly, CPT-L does not assume the frame axiom but requires an explicit encoding. Secondly, CPT-L does not handle negative effects. Consider the following CPT-theory for the *blocks world* domain:

$$r_1 = on(X,Y) : 1.0 \longleftarrow on(X,Y), \neg move(X,Z)$$
$$r_2 = (on(A,B) : 0.8) \vee (on(A,C) : 0.1) \vee (on(A,table) : 0.1) \longleftarrow move(A,B), \neg on(X,A), \neg on(Y,B), on(A,C).$$

encoding a probabilistic blocks world domain. In CPT-L actions do not play a special role and are assumed to be part of the background knowledge for every step e.g. a policy encoded as logic program. The first rule encodes the frame axiom. The last states how a block A is transferred from C to block B. Note that the lack of the implicit frame axioms and negative effects requires to *"block"* the explicit frame axioms r_1 in case of an action executed involving the corresponding atom (but allows simpler structure learning as it removes the burden of having to check for inconsistencies). Negation is handled as *negation as failure* like in Prolog. As the current state is completely fixed when the next state is constructed this is trivial. To illustrate this consider how the transition $\{on(a,b), on(b,table)\} \rightarrow \{on(a,table), on(b,table)\}$ when executing $move(a,table)$ from Fig. 3 can be constructed. The applicable rules are $\{r_1\{X \mapsto b, Y \mapsto table\}, r_2\{A \mapsto a, B \mapsto table\}\}$.

Inference and Parameter Estimation in CPT-L

The standard questions for probabilistic sequences models are given a CPT-theory $\mathcal{T}$

- **Sampling**: given initial interpretation I_0, how to sample a sequences of interpretations $I_1, ..., I_T$
- **Inference**: given a sequence of interpretations $I_0, ..., I_T$, what is $P(I_0, ..., I_T \mid \mathcal{T})$?
- **Parameter Estimation**: given a set of sequences of interpretations, what are the maximum-likelihood parameters of $\mathcal{T}$?
- **Prediction**: given $I_0, ..., I_t$ a sequence of interpretations, and F a first-order formula. What is the probability that F holds at time $t+d$, $P(I_{t+d} \models F \mid \mathcal{T}, I_0, ..., I_t)$?

Efficient algorithms for this problems were given in (Thon, Landwehr, and Raedt 2008). Additionally an algorithm for hidden state inference based on sampling has been developed (Thon 2009) which can presumably be incorporated along the same line.

From PPDDL to CPT-L

As we want to adapt the algorithms from CPT-L for learning in PPDDL, we first translate a set of PPDDL actions into a set of rules in an extended CPT-L syntax. If there are no nested probabilistic effects this transformation features a one to one mapping of the probabilistic parameters. If this condition is violated the nested probabilistic effects have to be expanded. The transformation back can be achieved by solving the corresponding linear equation system. Three extension of CPT-L are required for this transformation:

1. The implicit encoding of the frame axiom,
2. negative effects,
3. conjuncts as (=sets of) effects[3].

This allows us to transform all actions which have their effect in normal form $a = \langle \phi_a, (c_1 \rhd e_1 \wedge \ldots \wedge c_i \rhd e_1) \rangle$ into a corresponding set of rules as follows:

$$\begin{aligned}(h_{11} \wedge \ldots \wedge h_{1,k}) : p_1 \mid \ldots \mid (h_{m,l} \wedge \ldots \wedge h_{m,l}) : p_m &\longleftarrow a, \phi_a, c_1 \\ e_2 &\longleftarrow a, \phi_a, c_2 \\ &\vdots \\ e_i &\longleftarrow a, \phi_a, c_n\end{aligned}$$

Note: multiple rules might originate from the same action.

Example 1. *The blocks world example compiles to*

$$(on(A,B) \wedge \neg on(A,C) : 0.8) \vee (\top : 0.1) \vee (on(A,table) \wedge \neg on(A,C) : 0.1) \longleftarrow \underbrace{move(A,B)}_{action}, \underbrace{free(A), free(B), on(A,C)}_{precondition}.$$

Handling numeric effects and axioms can easily be achieved using logic programming. Given an action a, a domain model defines a distribution over possible successor states, $P(I_{t+1} \mid I_t, a)$, in the following way. Let $\mathbf{R}_t = \{r_1, ..., r_k\}$ denote the set of all ground rules applicable in the current state I_t. For each ground rule applicable in I_t one head element will affect the transformation from I_t into I_{t+1}. More formally, a *selection* σ is a

[2] The term CPT-Theory refers to a model described in CPT-L.

[3] A version of CPT-L supporting conjunction in the heads - i.e. multiple effects caused by one rule in parallel - is currently under review and should not be seen an original contribution. The contribution is the support for negated facts and the support of the frame axiom.

mapping from rules r_i to indexes j_i denoting that head element $h_{ij_i} \in head(r_i)$ is selected. In the stochastic process to be defined, I_{t+1} is a possible successor for the state I_t if and only if there is a selection σ such that $I_{t+1} = (I_t \setminus \{h^-_{1\sigma(1)}, ..., h^-_{k\sigma(k)})\}) \cup \{h^+_{1\sigma(1)}, ..., h^+_{k\sigma(k)})\}$., where the $h^+_{i\sigma^i}$ corresponds to the positive and $h^-_{j\sigma^j}$ to the negative effects of the corresponding head elements. We say that σ *yields* I_{t+1} from I_t, denoted $I_t \xrightarrow{\sigma} I_{t+1}$, and define

$$P(I_{t+1}|I_t) := \sum_{\sigma: I_t \xrightarrow{\sigma} I_{t+1}} P(\sigma) = \sum_{\sigma: I_t \xrightarrow{\sigma} I_{t+1}} \Big(\prod_{(r_i, j_i) \in \sigma} p_{j_i} \Big) \quad (1)$$

where p_{j_i} is the probability associated with head element h_{ij_i} in r_i. As for other Markov processes, we can define the probability of a sequence $I_1, ..., I_T$ given an initial state I_0 by

$$P(I_1, ..., I_T) := \prod_{t=0}^{T} P(I_{t+1} \mid I_t). \quad (2)$$

This defines a distribution over all possible sequences of interpretations of length T.

Inference and Parameter Estimation

We are interested in answering the following inference questions, given model $\mathcal{M}$:

- **Sampling**: how to sample sequences of states $I_1, ..., I_T$, given a plan $\mathcal{P}$, and initial state I_0?
- **Inference**: given a trajectory that is a sequence of states $I_1, ..., I_T$, together with the actions executed $\mathcal{P}$, what is $P(I_1, ..., I_T \mid \mathcal{P}, \mathcal{M})$?
- **Parameter Estimation**: given a set of trajectories, what are the maximum-likelihood parameters of $\mathcal{M}$?
- **Prediction**: given a trajectory till $I_0, \ldots, I_T$ T, a plan starting at T $\mathcal{P}$, and F a first-order formula that constitutes a certain property of interest. What is the probability that F holds at time $T + d$, $P(I_{T+d} \models F \mid \mathcal{M}, \mathcal{P}, I_0, ..., I_T)$?

Sampling from a model $\mathcal{M}$ given an initial state I_0 and a plan $\mathcal{P}$, is straightforward due to the causal semantics employed in CPT-L (as well as in PPDDL). For $t \geq 0$, I_{t+1} can be constructed from I_t by finding all groundings $r\theta$ of rules $r \in \mathcal{M}$, and sampling for each $r\theta$ a head element. The successor state I_{t+1} can then directly be constructed by removing and adding the facts to I_t according to the sampled heads. Algorithmic solutions for solving the inference, parameter estimation, will be presented in the rest of this section.

Inference

Because of the Markov assumption, cf. Equation (2), the crucial task for solving the inference problem is to compute $P(I_{t+1} \mid I_t)$ for given I_{t+1} and I_t. According to Equation (1), this involves summing the probabilities of all selections yielding I_{t+1} from I_t. However, the number of possible selections σ is exponential in the number of ground rules $|\mathbf{R}_t|$ applicable in I_t, so a naive generate-and-test approach is infeasible.

Instead, we presented for CPT-L an efficient approach for computing $P(I_{t+1} \mid I_t)$ without explicitly enumerating all selections yielding I_{t+1} (Thon, Landwehr, and Raedt 2008). This algorithm is closely related to the inference technique discussed in (De Raedt, Kimmig, and Toivonen 2007) and which we will now further adapt toward PPDDL. The problem is first converted to a DNF formula over boolean variables such that, satisfying assignments correspond to selections yielding I_{t+1}. The formula is then compactly represented as a BDD, and $P(I_{t+1} \mid I_t)$ efficiently computed from the BDD using dynamic programming.

Although finding satisfying assignments for DNF formulas is a hard problem in general, the key advantage of this approach is that existing, highly optimized BDD software packages can be used. The conversion of a given inference problem to a DNF formula f is realized as follows:

1. Initialize $f := true$
2. Let $\mathbf{R}_t$ denote the set of applicable ground rules in I_t. Rules $r \in \mathbf{R}_t$ are of the form $r = c_1 : p_1 \vee ... \vee c_n : p_n \longleftarrow b_1, ..., b_m$, where c_i are conjunctions of effects
3. For all rules $r = c_1 : p_1 \vee ... \vee c_n : p_n \longleftarrow b_1, ..., b_m$ in $\mathbf{R}_t$ do:
   (a) $f := f \wedge (r.c_1 \vee ... \vee r.c_n)$, where $r.c_i$ denotes a new (propositional) Boolean variable whose unique name is the concatenation of the name of the rule r with the head element c_i.
   (b) $f := f \wedge (\neg r.c_i \vee \neg r.c_j)$ for all $i \neq j$
4. For all facts $l \in I_{t+1} \setminus I_t$
   (a) Initialize $g := false$
   (b) for all $r \in \mathbf{R}_t$ and $c_i : p_i \in head(r)$ such that l is one of the positive atoms in the conjunction c_i do $g := g \vee r.c_i$
   (c) $f := f \wedge g$
5. For all facts $l \in I_t \setminus I_{t+1}$
   (a) Initialize $g := false$
   (b) for all $r \in \mathbf{R}_t$ and $c_i : p_i \in head(r)$ such that l is one of the negative atoms in the conjunction c_i do $g := g \vee r.c_i$
   (c) $f := f \wedge g$
6. For all variables $r.c$ appearing in f such that one of the atomic effects contradicts a transition from I_t to I_{t+1} do $f = f \wedge \neg r.c$

A Boolean variable $r.c$ in f states that the probabilistic choice element c was selected in rule r. A selection σ thus corresponds to an assignment of truth values to the variables $r.c$, in which exactly one probabilistic choice $r.c$ is true for every rule r. The construction of f ensures that all satisfying assignments for the formula f correspond to selections yielding I_{t+1}, and vice versa. Specifically, Step 3 of the algorithm assures that selections are obtained (that is, exactly one element is selected per probabilistic choice), Step 4 and Step 5 assures that the selection generates the changes from I_t to I_{t+1}. Step 6 assures that no facts get changed that do

not change from I_t to I_{t+1}. Thus, we have a one-to-one correspondence between satisfying assignments for the formula f and selections yielding I_{t+1}.

Example 2. *The following formula f is obtained for the transition $\{on(a,b), on(b,table)\} \rightarrow \{on(b,table), on(a,table)\}$ and the set of rules given in Example 1 when executing $move(a,table)$*

$$\underbrace{(r.c_1 \vee r.c_2 \vee r.c_3)}_{\textit{Step 3.a}} \wedge \underbrace{(r.c_1 \vee r.c_2)}_{\textit{Step 4}} \wedge \underbrace{(r.c_1 \vee r.c_2)}_{\textit{Step 5}}$$
$$\underbrace{(\neg r.c_1 \vee \neg r.c_2) \wedge (\neg r.c_2 \vee \neg r.c_3) \wedge (\neg r.c_1 \vee \neg r.c_3)}_{\textit{Step 3.b}}$$

where $c_1 = on(a,table) \wedge \neg on(a,b)$ is the first , $c_2 = on(a,table) \wedge \neg on(a,b)$ is the second, $c_3 = \top$ are the head elements of rules r_3. The parts of the formula are annotated with the steps in the construction algorithm that generated them.

Afterward a *reduced ordered binary decision diagram* (BDD) (Bryant 1986) is constructed which represents the formula f. Let $x_1, ..., x_n$ denote an ordered set of boolean variables (such as the $r.h$ contained in f). A BDD is a rooted, directed acyclic graph, in which nodes are annotated with variables and have out-degree 2, indicating that the variable is either true (solid) or false (dashed edge). Furthermore, there are two terminal nodes labeled with 0 and 1 respectively. The graph compactly represents a boolean function f over variables $x_1, ..., x_n$: given an instantiation of the x_i, we follow a path from the root to either 1 or 0 (indicating f is true or false). Furthermore, the graph must be reduced, that is, it must not be possible to merge or remove nodes without altering the represented function (cf. (Bryant 1986) for details). Figure 3, left, shows an example BDD.

From the BDD graph, $P(I_{t+1} \mid I_t)$ can be computed in linear time using dynamic programming. This is realized by a straightforward modification of the algorithm for inference in ProbLog theories (De Raedt, Kimmig, and Toivonen 2007). The algorithm exploits that paths in the BDD from the root node to the 1-terminal correspond to satisfying assignments for f, and thus selections yielding I_{t+1}. By sweeping through the BDD from top to bottom contributions from all such selections are summed up (Equation (1)) without explicitly enumerating all paths. The efficiency of this method crucially depends on the size of the BDD graph, which in turn depends strongly on the chosen variable ordering $x_1, ..., x_n$. Unfortunately, computing an optimal variable ordering is NP-hard. However, existing implementations of BDD packages contain sophisticated heuristics to find a good ordering for a given function in polynomial time.

Parameter Estimation

Assume the structure of the domain is given, that is, a set $\mathcal{M} = \{r_1, ..., r_k\}$ of rules of the form

$$r = (h_{11} \wedge \ldots \wedge h_{k1} : p_1) \vee \ldots \vee (h_{1n} \wedge \ldots \wedge h_k : p_n) \leftarrow b_1, \ldots, b_m,$$

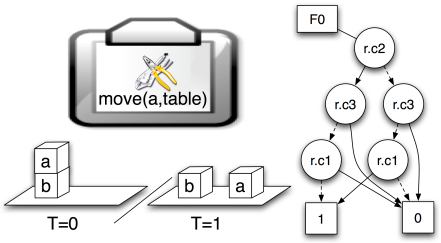


Figure 3: If executing $move(a,table)$ leads to the outcome depicted the algorithm constructs the BDD representing the formula above. Note the two paths in the BDD from root to the 1 leaf. They result from ambiguity between dropping the block during movement and moving the block to the table.

where $\pi = \{p_j\}_j$ are the unknown parameters to be estimated from a set of training sequences D. A standard approach is to find max-likelihood parameters $\pi^* = \arg\max_\pi P(D \mid \pi)$. To estimate the probability p_{ij}, we essentially need to know the number of times head element h_{ij} has been selected in an application of the rule r_i in the training data, which will be denoted by κ_{ij}. However, the quantity κ_{ij} is not directly observable. To see why this is so, first consider a single transition $I_t \rightarrow I_{t+1}$ in one training sequence. We know the set of rules $\mathbf{R}_t$ applied in the transition; however, there are in general many possible selections σ of rule head elements yielding I_{t+1}. The information which selection was used, that is, which rule has generated which fact in I_{t+1}, is hidden. The derivation of an efficient EM algorithm to estimate the ML parameters, while taken this hidden information into account can be found in (Thon, Landwehr, and Raedt 2008).

Akin to the inference problem, this calculation can be carried out in linear time given the BDD structure. This is realized by a dynamic programming algorithm similar to the forward-backward algorithm in hidden Markov models (Rabiner 1989) that sweeps through the BDD structure twice to accumulate the sufficient statistics κ_{ij}. Details of the algorithm are straightforward and omitted for lack of space. Note that the presented Expectation-Maximization algorithm, by taking the special structure of our model into account, is significantly more efficient than general-purpose parameter learning techniques employed for example in CP-logic.

Prediction

Assume we are given sequence $I_0, ..., I_t$ of observations, a model $\mathcal{M}$, a plan $\mathcal{P}$, and a property of interest F (represented as a first-order formula), and we would like to compute:

$$P(I_{t+d} \models F \mid I_0, ..., I_t, \mathcal{P}, \mathcal{M})$$

For instance, a robot needs to know the probability that a certain world state is reached at time $t + d$ given its current world model and observation history. Note, that the representation as a first-order formula allows one to express richer world conditions than queries on (sets of) atoms, as they are typically supported in statistical relational learning systems.

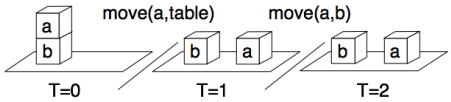
Powerful statistical relational learning systems are in principle able to compute this quantity exactly by "unfolding" the world model into a large dynamic graphical model. However, this is computationally expensive as it requires one to marginalize out all (unobserved) intermediate world states $I_{t+1}, ..., I_{t+d-1}$. In contrast, inference draws its efficiency from the full observability assumption.

As an alternative approach, we propose a straightforward sample-based approximation. Due to the Markov property we can start at I_t, after eventually re-learning the probabilities from the observation sequence. Given I_t, independent samples can be obtained from the conditional distribution $P(I_{t+1}, ..., I_{t+d} \mid I_t, \mathcal{P}, \mathcal{M})$ by simply sampling according to $\mathcal{M}$ from the initial state I_t. Ignoring $I_{t+1}, ..., I_{t+d-1}$ and checking F in I_{t+d} yields independent samples of the boolean event $I_{t+d} \models F$ from the distribution $P(I_{t+d} \models F \mid I_t, \mathcal{P}, \mathcal{M})$. The proportion of positive samples of this variable will thus quickly approach the true probability $P(I_{t+d} \models F \mid I_t, \mathcal{P}, \mathcal{M})$ (Thon, Landwehr, and Raedt 2008).

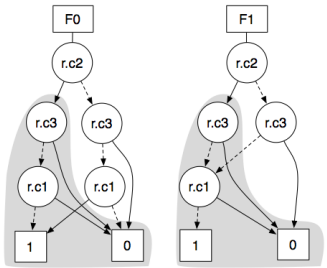
Online Learning with Shared BDD

As argued before, we want to estimate the probabilities in an online setting. While acting in the world the agent constantly updates its model. Using the current learning technique, that would require to either (a) store all the training examples and rebuild the entire BDD for each learning step or (b) store the constructed BDDs. Even though our method is efficient the first approach will be prohibitively slow as the runtime is exponential ($O(T!)$) in the sequence length T (when updating after every/a constant number of steps). If implemented using separate BDDs, the second approach requires to store a large number of BDDs in the memory. This can be circumvented as the following example shows.

Example 3. *Consider the following trajectory:*

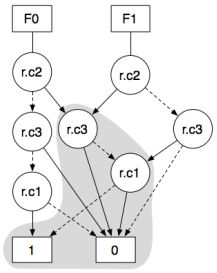


For this sequence together with model given in the previous example, the BDDs for the two transitions look as follows,



Even without understanding the meaning of the BDD, one can spot that both BDDs contain the similar sub-BDD having the two nodes $r.c3$, $r.c1$.

Exploiting this, we can save a lot of space by using so-called shared BDDs (SBDD). They are an extension of BDDs such that multiple functions are represented, while isomorphic subtrees from multiple BDDs are merged. In the previous example, this gives a graph with a total number of seven instead of nine nodes.



But this can be improved even further: The successful application of the actions $move(a, b)$ and $move(b, d)$ both lead to structural identical BDDs except for the naming of the variables. By exploiting this and re-parameterizing the actions we can increase the sharing in a way such that, independent of the number of blocks in the world, runtime is linear in the number of transitions and memory usage is constant in total.

Conclusions And Future Work

We have presented a method to learn PPDDL probabilities by means of CPT-L, a recently introduced efficient SRL framework for sequential statistical relational learning. We believe that – in general – the combination of techniques from both (dynamic) SRL settings and (probabilistic) planning can develop into a fruitful marriage of two fields that have a lot in common; learning, planning and acting in dynamic, probabilistic, and richly structured worlds. At the start of the paper, we have discussed several techniques that move into that direction but it is clear that there are many areas waiting to be explored.

CPT-L has been applied in the context of Massively Multi-player Online Games and has been proven to be efficient in these settings. The corresponding Dynamic Bayesian Network would consists out of thousands of nodes in cases where learning took seconds when learning with CPT-L. This indicates that CPT-L it is capable of handling reasonably large domains.

We are currently setting up experiments to test our approach in the PPDDL domain and to evaluate the various inference and learning tasks on planning problems. Our longer term research goes into the direction of developing a system able to exploit the uncertainty about the learned knowledge. The goal is to detect unexpected action outcomes. In general, such a system should able to cope with and recognize *concept drift*, i.e. when the dynamics of the world changes over time. In this setting, it might be possible to re-learn (parts of) the model. Other natural future directions are (approximate) planning using the learned rules, and model-based reinforcement learning.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. Bernd Gutmann is supported by the Research Foundation-Flanders (FWO-Vlaanderen). This work is supported by the GOA project 2008/08 Probabilistic Logic Learning.

References

[Bryant 1986] Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers* 35(8):677–691.

[De Raedt, Kimmig, and Toivonen 2007] De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2462–2467.

[Fernández, Jiménez, and de la Rosa 2009] Fernández, S.; Jiménez, S.; and de la Rosa, T. 2009. Improving automated planning with machine learning. In Soria, E.; Martin, J.; Magdalena, R.; Martinez, M.; and Serrano, A., eds., *Handbook of Research on Machine Learning Applications and Trends*. Information Science Reference.

[Gardiol 2003] Gardiol, N. H. 2003. Applying probabilistic rules to relational worlds. Master's thesis, Department of Electrical Engineering and Computer Science, MIT.

[Jiménez and Cussens 2006] Jiménez, S., and Cussens, J. 2006. Combining ilp and parameter estimation to plan robustly in probabilistic domains. In *Proceedings of the International Conference on Inductive Logic Programming (ILP)*.

[Jiménez, Fernández, and Borrajo 2008] Jiménez, S.; Fernández, F.; and Borrajo, D. 2008. The PELA architecture: Integrating planning and learning to improve execution. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1294–1299.

[Kambhampati 2007] Kambhampati, S. 2007. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *Proceedings of the National Conference of Artificial Intelligence*, volume 22, 1601. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

[Kersting, van Otterlo, and De Raedt 2004] Kersting, K.; van Otterlo, M.; and De Raedt, L. 2004. Bellman goes relational. In *Proceedings of the International Conference on Machine Learning (ICML)*.

[Lang and Toussaint 2009] Lang, T., and Toussaint, M. 2009. Approximate inference for planning in stochastic relational worlds. In *Proceedings of the International Conference on Machine Learning (ICML)*.

[Pasula, Zettlemoyer, and Kaelbling 2007] Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29:309–352.

[Petrick et al. 2008] Petrick, R.; Kraft, D.; Mourão, K.; Pugeault, N.; Krüger, N.; and Steedman, M. 2008. Representation and integration: Combining robot control, high-level planning, and action learning. In *Proceedings of the Sixth International Cognitive Robotics Workshop (CogRob 2008) at ECAI 2008*, 32–41.

[Rabiner 1989] Rabiner, L. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2):257–286.

[Rintanen 2003] Rintanen, J. 2003. Expressive equivalence of formalisms for planning with sensing. In Giunchiglia, E.; Muscettola, N.; and Nau, D. S., eds., *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, 185–194. AAAI.

[Safaei and Ghassem-Sani 2007] Safaei, J., and Ghassem-Sani, G. 2007. Incremental learning of planning operators in stochastic domains. In *Proceedings of the International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, 644–655.

[Sato and Kameya 1997] Sato, T., and Kameya, Y. 1997. Prism: A symbolic-statistical modeling language. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1330–1339. Morgan Kaufmann.

[Sutton 1991] Sutton, R. S. 1991. DYNA, an integrated architecture for learning, planning and reacting. In *Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures*, 151–155.

[Thon, Landwehr, and Raedt 2008] Thon, I.; Landwehr, N.; and Raedt, L. D. 2008. A Simple Model for Sequences of Relational State Descriptions. In *Proceedings of the 19th European Conference on Machine Learning*, 506–521.

[Thon 2009] Thon, I. 2009. Don't Fear Optimality: Sampling for Probabilistic-Logic Sequence Models. In *Proceedings of the International Conference on Inductive Logic Programming (ILP-2009)*.

[van Otterlo 2009] van Otterlo, M. 2009. *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for Adaptive Sequential Decision Making under Uncertainty in First-Order and Relational Domains*. Amsterdam, The Netherlands: IOS Press.

[Vere 1977] Vere, S. A. 1977. Induction of relational productions in the presence of background information. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 349–355.

[Wang 1995] Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of the International Conference on Machine Learning (ICML)*, 549–557.

[Yoon and Kambhampati 2007] Yoon, S., and Kambhampati, S. 2007. Towards Model-lite Planning: A Proposal For Learning & Planning with Incomplete Domain Models. In *International Conference on Automated Planning and Scheduling, Providence, Rhode Island, USA*.

[Younes et al. 2005] Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research (JAIR)* 24:851–887.

[Zimmerman and Kambhampati 2003] Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine* 24:73–96.