Approximate Inference for Planning in Stochastic Relational Worlds

Tobias Lang LANG@CS.TU-BERLIN.DE
Marc Toussaint MTOUSSAI@CS.TU-BERLIN.DE
TU Berlin, Machine Learning Group, Franklinstrasse 28/29, 10587 Berlin, Germany

Abstract

Relational world models that can be learned from experience in stochastic domains have received significant attention recently. However, efficient planning using these models remains a major issue. We propose to convert learned noisy probabilistic relational rules into a structured dynamic Bayesian network representation. Predicting the effects of action sequences using approximate inference allows for planning in complex worlds. We evaluate the effectiveness of our approach for online planning in a 3D simulated blocksworld with an articulated manipulator and realistic physics. Empirical results show that our method can solve problems where existing methods fail.

1. Introduction

Building systems that act in complex environments is a central goal of Artificial Intelligence. Such systems need to be able to reason about their world in order to derive plans of actions and achieve their goals. Acting in a complex environment requires knowledge representations which can account for indeterministic action effects and cope with noise. Furthermore, they have to generalize to unencountered situations and objects of similar types.

The field of statistical relational learning investigates the combination of relational representations with probabilistic frameworks and Machine Learning techniques (Getoor & Taskar, 2007). Due to the general nature of the proposed representations, however, it is often unclear how to learn and represent complex action dynamics in an efficient way. Action decision making is often cast in a reinforcement learning (RL) framework. Relational RL investigates the usage of compact relational representations for state and action spaces (van Otterlo, 2009). Most work has been on model-free approaches which compute value functions and policies directly from experiences resulting in reactive behaviors for fixed goals (also called habit-based decision making). By contrast, model-based approaches use the experiences to learn world dynamics and reward models. This enables goal-directed or purposive decision making which allows an agent with changing goals to plan for the goal at hand by internal simulation (Botvinick & An, 2009).

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009. Copyright 2009 by the author(s)/owner(s).

Pasula et al. (2007) have recently introduced an appealing action model representation based on noisy indeterministic deictic (NID) rules which offer several advantages: *(i)* a relational representation; *(ii)* indeterministic action outcomes in order to account for stochastic domains; *(iii)* deictic references for actions to reduce action space; *(iv)* noise outcomes to avoid explicit modeling of rare and overly complex outcomes; and, *(v)* the existence of an effective learning algorithm. They showed that NID rules are able to capture the complex dynamics in a realistically simulated noisy version of the blocks world based on a three-dimensional rigid-body dynamics simulator (ODE). A severe limitation, however, is that there exists no efficient method so far to plan with NID rules.

Promising solution techniques for known Markov decision processes (MDPs) over relational domains have been proposed: for example, Kersting et al. (2004) present an exact value iteration algorithm and Sanner and Boutilier (2007) propose approximate solution techniques for factored first-order MDPs based on linear value approximation. Both require complete models, however, which NID rules do not provide due to their noise outcomes, and it is not clear how to make them deal with deictic references in the spirit of NID rules. Croonenborghs et al. (2007) learn partial world models online in form of sets of relational probability trees for individual state properties which they exploit immediately by means of Q-learning with look-ahead trees. They do not model deictic references explicitly so that their action space is much larger which heavily affects planning time. We will see later that planning with look-ahead trees is inefficient.

In this paper, we introduce the PRADA algorithm (probabilistic relational action-sampling in DBNs planning algorithm), a model-based RL approach for planning based on NID rules and probabilistic inference, making three contributions: *(i)* Following the idea of framing planning as a probabilistic inference problem (Toussaint & Storkey, 2006), we convert NID rules into a dynamic Bayesian networks (DBNs) representation. *(ii)* We derive an approximate inference method to cope with the state complexity of a time-slice of the resulting network. This enables us to efficiently predict the effects of action sequences. While Sanghai et al. (2005) propose particle filtering algorithms for approximate inference in DBNs over relational domains, we can exploit different structural assumption and develop an approximate inference technique based on the factored frontier algorithm (Murphy & Weiss, 2001). *(iii)* To enable efficient sampling-based planning, we propose a sampling distribution for actions which takes predicted state distributions into account. We evaluate our approach in a simulated blocksworld with realistic physics, similarly as in Pasula et al., but with an articulated humanoid manipulating the blocks, showing the effectiveness of PRADA for fast online planning.

The remainder of this paper is organized as follows. The next section presents the theoretical background of our work, namely NID rules and the planning algorithm which was used with NID rules so far. We introduce our DBN representation for NID rules in Sec. 3 and our approximate inference method in Sec. 4. In Sec. 5, we present our planning procedure. Then, we show our empirical results before we conclude.

2. Background

2.1. State and action representation

We use a relational representation to describe world states and rules. In a concrete instantiation (situation) we assume there is a finite set of objects $\mathcal{O}$ present and we can form logical formulae ψ of predicates (relations) and functions over these objects. Generally, a formula may use logical variables which represent any object, independent of the concrete instantiation of objects $\mathcal{O}$. We will speak of *grounding* a formula ψ if we apply a substitution σ that maps all of the variables appearing in ψ to objects in $\mathcal{O}$. We have a finite set $\mathcal{P}$ of predicates and a finite set $\mathcal{F}$ of functions. The state of the world is fully described by all ground predicates and functions. Actions are represented by positive predicates $\mathcal{A}$. NID rules incorporate knowledge about state transition dynamics and are described in the following.

Table 1. Example NID rule for a realistic blocksworld, which models to try to grab a block X. The block Y is implicitly defined as the one below X (deictic referencing). X ends up in the robot's hand with high probability, but might also fall on the table. With a small probability something unpredictable happens.

$$
\begin{array}{ll}
\mathit{grab}(X): & \mathit{on}(X,Y),\ \mathit{block}(Y),\ \mathit{table}(Z) \\
& \rightarrow \left\{ \begin{array}{lll} 0.7 & : & \mathit{inhand}(X),\ \neg \mathit{on}(X,Y) \\ 0.2 & : & \mathit{on}(X,Z),\ \neg \mathit{on}(X,Y) \\ 0.1 & : & \text{noise} \end{array} \right.
\end{array}
$$

2.2. Noisy indeterministic deictic (NID) rules

We want to learn a relational model of a stochastic world and use it for planning. Pasula et al. (2007) proposed NID rules for representing such a model and an algorithm to learn them from data. Table 1 shows an exemplary NID rule for the realistic blocksworld. A NID rule r is given as

$$
a_r(\mathcal{X}):\ \Phi_r(\mathcal{X}) \quad \rightarrow \quad \left\{ \begin{array}{lll} p_{r,1} & : & \Omega_{r,1}(\mathcal{X}) \\ & \vdots & \\ p_{r,m_r} & : & \Omega_{r,m_r}(\mathcal{X}) \\ p_{r,0} & : & \Omega_{r,0} \end{array} \right. \tag{1}
$$

where $\mathcal{X}$ is a set of logic variables in the rule (which represent a (sub-)set of abstract objects). In the rules which define our world models all formula arguments are logic variables. The rule r consists of preconditions, namely that action a_r is applied on $\mathcal{X}$ and that the state context Φ_r is fulfilled, and $m_r + 1$ different outcomes with associated probabilities $p_{r,i} > 0$, $\sum_{i=0} p_{r,i} = 1$. Each outcome $\Omega_{r,i}(\mathcal{X})$ describes which predicates and functions change when the rule is applied. The context $\Phi_r(\mathcal{X})$ and outcomes $\Omega_{r,i}(\mathcal{X})$ are conjunctions of literals constructed from the predicates in $\mathcal{P}$ as well as equality statements comparing functions from $\mathcal{F}$ to constant values. The so-called *noise outcome* $\Omega_{r,0}$ subsumes all possible action outcomes which are not explicitly specified by one of the other $\Omega_{r,i}$. The arguments of the action $a(\mathcal{X}_a)$ may be a true subset $\mathcal{X}_a \subset \mathcal{X}$ of the variables $\mathcal{X}$ of the rule. The remaining variables are called deictic references $DR = \mathcal{X} \setminus \mathcal{X}_a$ and denote objects relative to the agent or action being performed.

As above, let σ denote a substitution that maps variables to constant objects, $\sigma : \mathcal{X} \rightarrow \mathcal{O}$. Applying σ to an abstract rule $r(\mathcal{X})$ yields a *ground rule* $r(\sigma(\mathcal{X}))$. We say a ground rule r *covers* a state s and a ground action a if $s \models \Phi_r$ and $a = a_r$. Let Γ be a set of ground NID rules. We define $\Gamma(a) := \{r \mid r \in \Gamma, a_r = a\}$ to be the set of rules that provide predictions for action a. If r is the only rule in $\Gamma(a)$ to cover a and state s, we call it the *unique covering rule* for a in s. For more technical details, we refer the reader to Pasula et al.. If a pair (a, s) has a unique covering rule r, we calculate $P(s' \,|\, a, s)$ using

$$P(s'|r, s) = \sum_{i=1}^{m^r} p_{r,i} P(s'|\Omega_{r,i}, s) + p_{r,0} P(s'|\Omega_{r,0}, s), \quad (2)$$

where, for $i > 0$, $P(s' \,|\, \Omega_{r,i}, s)$ is a deterministic distribution that is one for the unique state constructed from s taking the changes of $\Omega_{r,i}$ into account. The distribution given the noise outcome, $P(s' \,|\, \Omega_{r,0}, s)$, is unknown and needs to be estimated, e.g. by assigning low probability to many successor states.

If a pair (a, s) does *not* have a unique covering rule r (e.g. two rules cover (a, s) providing conflicting predictions), one can predict the effects of a by means of a noisy default rule r_ν which explains all effects as noise: $P(s'|r_\nu, s) = P(s' \,|\, \Omega_{r_\nu,0}, s)$. This is not very meaningful and thus disadvantageous for planning. For this reason, the concept of unique covering rules is crucial in planning with NID rules.

NID rules can be learned from experience triples (s, a, s') by means of a batch algorithm that trades off the likelihood of these triples with the complexity of the learned rule-set.

2.3. Sparse sampling trees for MDP planning

To plan with NID rules, one can treat the domain described by the relational logic vocabulary as a Markov decision process (MDP). NID rules encapsulate the transition probabilities in a compact way exploiting the relational structure, as described in Eq. (2). As the MDP model created from NID rules is incomplete due to the noise outcome and exact solution methods are hard to come by in relational domains, Pasula et al. use the sparse sampling tree (SST) algorithm (Kearns et al., 2002) for MDP planning. Given a planning horizon d and a branching factor b, SST builds a look-ahead tree of states starting with the current state. In each tree node (representing a state), *(i)* SST takes all possible actions into account, and *(ii)* for each action it takes b samples from the successor state distribution using a transition model (in our case the NID rules). Values are computed for each node of the tree using the Bellman equation, and finally the action with the highest value is chosen. SST is independent of the number of states, but exponential in the time horizon taken into account.

When sampling the noise outcome while planning with SST, Pasula et al. approximate the obtained value by assuming to stay in the same state and discounting the estimated value. (We refer to this adaptation when we speak of SST planning in the remainder of the paper.) If an action does not have a unique covering rule and is modelled by the noisy default rule r_ν, it is thus always better to perform a *doNothing* action instead. Hence, in SST planning one can discard all actions for a given state which do not have unique covering rules.

While SST is near-optimal, in practice it is only feasible for very small b and d. Let the number of actions be a. Then the number of nodes at time horizon d is $(ba)^d$. (This number can be reduced if the same outcome of a rule is sampled multiple times.) While smaller choices of b lead to faster planning, they result in a significant accuracy loss in realistic domains. As Kearns et al. note SST is only useful if no special structure that permits compact representation is available. Therefore, we now introduce an alternative planning approach that exploits the structure of NID rules.

3. Graphical models for NID rules

Decision theoretic problems where agents need to choose appropriate actions can be represented by means of Markov chains and DBNs. In the following, we discuss how to convert NID rules to DBNs which the PRADA algorithm will use to plan by probabilistic inference as described in Sec. 5. We denote random variables by upper case letters (e.g. S), their values by the corresponding lower case letters (e.g., $s \in dom(S)$), variable vectors by bold upper case letters (e.g. $\mathbf{S} = (S_1, S_2, S_3)$) and value vectors by bold lower case letters (e.g. $\mathbf{s} = (s_1, s_2, s_3)$). We also use column notation, e.g. $\mathbf{s}^{2:4} = (s_2, s_3, s_4)$.

A naive way to convert NID rules to DBNs is shown in Fig. 1(a). States are represented by a vector $\mathbf{S} = (S_1, \ldots, S_N)$ where for each ground predicate in $\mathcal{P}$ there is a binary S_i and for each ground function in $\mathcal{F}$ there is an S_i with range according to the represented function. (In this paper, we restrict ourselves to functions which range over a finite subset of the integers.) Actions are represented by an integer variable A which indicates the action out of a vector of ground action predicates in $\mathcal{A}$. The reward gained in a state is represented by U and may depend only on a subset of the state variables. It is possible to express arbitrary reward expectations $P(U \,|\, \mathbf{S})$ with binary U (Dayan & Hinton, 1997). Using NID rules to define the transition dynamics leads to very complex dependencies as one needs to account for the uniqueness of covering rules. This may result in conditional probability functions comprising huge subsets of $\mathbf{S}$ which makes this representation unfeasible for planning.

Therefore, we exploit the structure of NID rules to model a state transition with the graphical model shown in Fig. 1(b) representing the joint distribution

$$\begin{aligned} &P(u', \mathbf{s}', o, r, \boldsymbol{\phi} \,|\, a, \mathbf{s}) \qquad (3) \\ &\quad = P(u' \,|\, \mathbf{s}')\, P(\mathbf{s}' \,|\, o, r, \mathbf{s})\, P(o \,|\, r)\, P(r \,|\, a, \boldsymbol{\phi})\, P(\boldsymbol{\phi} \,|\, \mathbf{s}), \end{aligned}$$

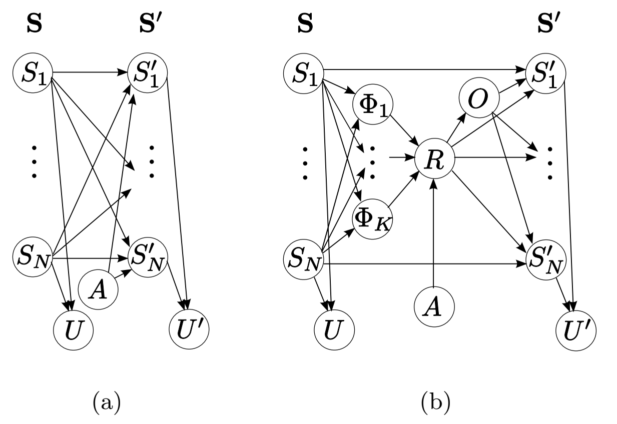

Figure 1. (a) Naive DBN; (b) DBN exploiting NID factorization

which we will explain in detail in the following. Assume we are given a set of fully abstract NID rules. We compute all groundings of these rules w.r.t. the objects of the domain and get the set Γ of K different ground NID rules. In addition to $\mathbf{S}$, $\mathbf{S}'$, A, U and U' as above, we use a binary random variable Φ_i for each rule to model the event that its context holds, which is the case if all required literals hold:

$$P(\boldsymbol{\phi}\,|\,\mathbf{s}) = \prod_{i=1}^{K} P(\phi_i|\mathbf{s}_{\pi(\Phi_i)}) = \prod_{i=1}^{K} I\left(\bigwedge_{j\in\pi(\Phi_i)} S_j = s_{r_i,j}\right). \quad (4)$$

We use $\bigwedge_i \rho_i$ to express a logical conjunction $\rho_1 \wedge \cdots \wedge \rho_n$. The function $\pi(\Phi)$ yields the set of indices of the state variables in $\mathbf{s}$, on which Φ depends. $\mathbf{s}_{r_i}$ denotes the configuration of the state variables corresponding to the literals in the context of r_i. We use an integer-valued variable R ranging over $K+1$ possible values to identify the rule which predicts the effects of the action. This is the unique covering rule for the current state-action pair, i.e., the only rule modeling action a whose context holds:

$$P(R=r|a,\boldsymbol{\phi}) = I\left(r\in\Gamma(a) \wedge \Phi_r = 1 \wedge \bigwedge_{r'\in\Gamma(a)\setminus\{r\}} \Phi_{r'} = 0\right). \quad (5)$$

If no unique covering rule exists, we predict no changes as indicated by the special value $R=0$ (assuming not to execute the action, similarly as SST does):

$$P(R=0\,|\,a,\boldsymbol{\phi}) = \bigwedge_{r\in\Gamma(a)} \neg I\left(\Phi_r = 1 \wedge \bigwedge_{r'\in\Gamma(a)\setminus\{r\}} \Phi_{r'} = 0\right) \quad (6)$$

The integer-valued variable O represents the outcome of the action as predicted by the rule. It ranges over M possible values where M is the maximum number of outcomes all rules in Γ have. (To ensure a sound semantics, we introduce empty dummy outcomes with zero-probability for those rules whose number of outcomes is less than M.) The probability of an outcome is defined as in the corresponding rule:

$$P(O=o\,|\,r) = p_{r,o}\,. \quad (7)$$

We define the probability of the successor state as

$$P(\mathbf{s}'\,|\,o,r,\mathbf{s}) = \prod_i P(s'_i\,|\,o,r,s_i)\,, \quad (8)$$

which is one for the unique state that is constructed from $\mathbf{s}$ taking the changes according to $\Omega_{r,o}$ into account. The probability of the reward is given by

$$P(U'=1\,|\,\mathbf{s}') = I\left(\bigwedge_{j\in\pi(U')} S'_j = \tau_j\right)\,. \quad (9)$$

The function $\pi(U')$ yields the set of indices of the state variables in $\mathbf{s}'$, on which U' depends. The configuration of these variables that corresponds to our planning goal is denoted by $\boldsymbol{\tau}$. We renounce on specifying a prior $P(\mathbf{s}^0)$, since the initial state $\mathbf{s}^0$ will always be given. Our choice for the distribution $P(a)$ used for sampling actions will be described in Sec. 5.

Exact inference is intractable in our graphical model. When constructing a junction tree, we will get cliques that comprise whole Markov slices (all variables representing the state at a certain time-step). Also, approximate inference by means of loopy belief propagation is unfeasible due to the deterministic dependencies in small cycles which inhibit convergence. Therefore, we propose a different approximate inference scheme which we present next.

4. Approximate inference

We follow the idea of the factored frontier (FF) algorithm (Murphy & Weiss, 2001) and approximate the belief with a product of marginals:

$$P(\mathbf{s}^t\,|\,\mathbf{a}^{0:t-1}) \;\approx\; \prod_i P(s_i^t\,|\,\mathbf{a}^{0:t-1})\,. \quad (10)$$

We define

$$\alpha(s_i^t) \;:=\; P(s_i^t\,|\,\mathbf{a}^{0:t-1}) \;\text{ and } \quad (11)$$

$$\alpha(\mathbf{s}^t) \;:=\; P(\mathbf{s}^t\,|\,\mathbf{a}^{0:t-1}) \approx \prod_{i=1}^{N} \alpha(s_i^t) \quad (12)$$

and derive the following FF filter for the model in Fig. 1(b). We are interested in inferring the state distribution at time $t+1$ given an action sequence $\mathbf{a}^{0:t}$:

$$\alpha(s_i^{t+1}) = P(s_i^{t+1}\,|\,\mathbf{a}^{0:t}) \quad (13)$$

$$= \sum_{r^t} P(s_i^{t+1}\,|\,r^t,\mathbf{a}^{0:t-1})\;P(r^t\,|\,\mathbf{a}^{0:t})\,. \quad (14)$$

We compute the first term in (14) as

$$P(s_i^{t+1} \,|\, r^t, \mathbf{a}^{0:t-1}) = \sum_{s_i^t} P(s_i^{t+1} \,|\, r^t, s_i^t)\; P(s_i^t \,|\, r^t, \mathbf{a}^{0:t-1}) \approx \sum_{s_i^t} P(s_i^{t+1} \,|\, r^t, s_i^t)\; \alpha(s_i^t)\;. \quad (15)$$

This approximation assumes that s_i^t is conditionally independent of r^t. To improve on this approximation one could examine whether s_i^t is part of the context of r^t or whether s_i^{t+1} is specified in its outcomes: if this is the case, we can infer the state of s_i^t from knowing r^t. However, we found our approximation be sufficient. We calculate the successor state distribution as

$$P(s_i^{t+1} \,|\, r^t, s_i^t) = \sum_o P(s_i^{t+1} \,|\, o, r^t, s_i^t)\; P(o \,|\, r^t)\;, \quad (16)$$

which shows us how to update the belief over S_i^{t+1} if we predict with rule r^t. For the computation of the second term in (14), we start with

$$\begin{aligned} P(R^t\!=\!r \,|\, \mathbf{a}^{0:t}) &= \sum_{\boldsymbol{\phi}^t} P(R^t\!=\!r \,|\, \boldsymbol{\phi}^t, \mathbf{a}^{0:t})\; P(\boldsymbol{\phi}^t \,|\, \mathbf{a}^{0:t}) \\ &= I(r \!\in\! \Gamma(a^t))\; P\left(\Phi_r^t\!=\!1, \bigwedge_{r' \in \Gamma(a^t)\setminus\{r\}} \Phi_{r'}^t\!=\!0 \,|\, \mathbf{a}^{0:t-1}\right) \\ &= I(r \!\in\! \Gamma(a^t))\;\; P(\Phi_r^t\!=\!1 \,|\, \mathbf{a}^{0:t-1}) \\ &\quad \cdot P\left(\bigwedge_{r' \in \Gamma(a^t)\setminus\{r\}} \Phi_{r'}^t\!=\!0 \,|\, \Phi_r^t\!=\!1, \mathbf{a}^{0:t-1}\right)\;. \end{aligned} \quad (17)$$

To simplify the summation over $\boldsymbol{\phi}^t$, we used that r models the state transition if and only if it uniquely covers $(a^t, \mathbf{s}^t)$. We calculate the second term in (17):

$$\begin{aligned} P(\Phi_r^t\!=\!1 \,|\, \mathbf{a}^{0:t-1}) &= \sum_{\mathbf{s}^t} P(\Phi_r^t\!=\!1 \,|\, \mathbf{s}^t)\; \alpha(\mathbf{s}^t) \\ &\approx \sum_{\mathbf{s}^t} P(\Phi_r^t\!=\!1 \,|\, \mathbf{s}^t)\; \prod_j \alpha(s_j^t) \quad (18) \\ &= \prod_{j \in \pi(\Phi_r^t)} \alpha(S_j^t\!=\!s_{r,j}) \quad . \quad (19) \end{aligned}$$

The approximation in (18) is the FF assumption. In (19), $\mathbf{s}_r$ denotes the configuration of the state variables according to the context of r like in (4). Thus, the terms $\alpha(S_j^t = s_{r,j})$ correspond to the probabilities of the literals in r's context. We calculate the third term in (17) as

$$P\left(\bigwedge_{r' \in \Gamma(a^t)\setminus\{r\}} \Phi_{r'}^t\!=\!0 \,|\, \Phi_r^t\!=\!1, \mathbf{a}^{0:t-1}\right) \approx \prod_{r' \in \Gamma(a^t)\setminus\{r\}} P(\Phi_{r'}^t\!=\!0 \,|\, \Phi_r^t\!=\!1, \mathbf{a}^{0:t-1}) \quad (20)$$

with

$$\begin{aligned} &P(\Phi_{r'}^t\!=\!0 \,|\, \Phi_r^t\!=\!1, \mathbf{a}^{0:t-1}) \quad (21) \\ &= \sum_{\mathbf{s}^t} P(\Phi_{r'}^t\!=\!0 \,|\, \mathbf{s}^t)\; P(\mathbf{s}^t \,|\, \Phi_r^t\!=\!1, \mathbf{a}^{0:t-1}) \\ &\approx \begin{cases} 1.0 & \text{if} \quad \Phi_r \wedge \Phi_{r'} \rightarrow \bot \\ 1.0 - \prod_{\substack{i \in \pi(\Phi_{r'}^t), \\ i \notin \pi(\Phi_r^t)}} \alpha(S_i^t\!=\!s_{r',i}) & \text{otherwise} \end{cases}, \end{aligned}$$

where the if-condition expresses a logical contradiction of the contexts of r and r'. Finally, we compute the reward probability straightforwardly as

$$\begin{aligned} P(U^t\!=\!1 \,|\, \mathbf{a}^{0:t-1}) &= \sum_{\mathbf{s}_t} P(U^t\!=\!1 \,|\, \mathbf{s}_t) P(\mathbf{s}^t \,|\, \mathbf{a}^{0:t-1}, \mathbf{s}^0) \\ &\approx \prod_{i \in \pi(U^t)} \alpha(S_i^t\!=\!\tau_i)\;, \end{aligned} \quad (22)$$

where $\boldsymbol{\tau}$ denotes the configuration of state variables corresponding to the planning goal as in (9).

The computational costs of propagating the effects of an action are quadratic in the number of rules for this action and linear in the maximum numbers of context literals and manipulated state properties of those rules.

Our inference framework requires an approximation for the distribution $P(s' \,|\, \Omega_{r,0}, s)$ (cf. Eq. (2)) to cope with the noise outcome of NID rules. From the training data used to learn rules, we estimate which predicates and functions change value over time as follows: let $\mathbf{S}_c \subset \mathbf{S}$ contain the corresponding variables. We estimate for each rule r the average number N^r of changed state properties when the noise outcome applies. We approximate the probability that $S_i \in \mathbf{S}_c$ changes in r's noise outcome by $\frac{N^r}{|\,S^C\,|}$. In case of change, all changed values of S_i have equal probability.

5. Action-sampling planning

PRADA plans in stochastic relational domains by predicting the effects of action sequences using the graphical model in Fig. 1(b) and the approximate inference method described in the last section. We sample sequences of actions $\mathbf{a}^{0:T-1}$ of length T. For $0 < t \leq T$, we compute the posteriors over states $P(s^t \,|\, \mathbf{a}^{0:t-1}, \mathbf{s}^0)$ and rewards $P(u^t \,|\, \mathbf{a}^{0:t-1}, \mathbf{s}^0)$ (in the sense of filtering or state monitoring) and calculate the value of an action sequence with a discount factor $0 < \gamma < 1$ as

$$Q(\mathbf{a}^{0:T-1}, \mathbf{s}^0) := \sum_{t=1}^{T} \gamma^t P(U^t\!=\!1 \,|\, \mathbf{a}^{0:t-1}, \mathbf{s}^0)\;. \quad (23)$$

We choose the first action of the best sequence $\mathbf{a}^* = \operatorname{argmax}_{\mathbf{a}^{0:T-1}} Q(\mathbf{a}^{0:T-1}, s^0)$, if its value exceeds a certain threshold ζ (e.g., $\zeta = 0$). Otherwise, we continue sampling until either an action is found or sampling is given up. We aim for a strategy to sample good action sequences with high probability. We propose to choose with equal probability among the actions that have a unique covering rule for the current state and thereby avoid the use of the noisy default rule. For the action at time t, PRADA samples from the distribution

$$P^t_{sample}(a) \propto \sum_{r \in \Gamma(a)} P\left(\phi^t_r = 1, \bigwedge_{r' \in \Gamma(a) \setminus \{r\}} \phi^t_{r'} = 0 \,|\, \mathbf{a}^{0:t-1}\right), \quad (24)$$

taking the current state distribution into account. Thereby, the probability to sample an action sequence $\mathbf{a}$ predicting the state sequence $s_0, \ldots, s_T$ depends on the likelihood of the state sequence given $\mathbf{a}$: the more likely the required outcomes are, the more likely the next actions will be sampled. PRADA does not miss actions which SST explores:

Theorem 1: The set of action sequences PRADA samples with non-zero probability is a super-set of the one of SST.

A proof of this theorem can be found on `http://cs.tu-berlin.de/~lang/prada/`. Besides the difference in handling the noise outcome, PRADA and SST follow opposite approaches: PRADA samples actions and calculates the transitions approximately, while SST considers all actions (and is thus exact in its action search) and samples transitions. This implies an important difference – both algorithms are faced with the problem that the search space of action sequences is exponential in the planning horizon. To calculate the value of a *given* action sequence, however, SST still needs exponential time due to its outcome sampling. In contrast, PRADA propagates the state transitions forward and is thus linear in the horizon.

5.1. An extension: Adaptive PRADA

We can exploit the fact that PRADA returns a sequence of actions to increase planning accuracy – in contrast to SST where the actions taken at $t > 0$ depend on the sampled outcomes. Adaptive PRADA (A-PRADA) examines the best found action sequence of PRADA and decides by simulation whether some action can be deleted such that the expected reward is increased – e.g., by deleting actions that do not have significant effects on achieving our goal. We refer the reader for more technical details to `http://cs.tu-berlin.de/~lang/prada/`.

6. Results

6.1. Setup

We test SST and (A-)PRADA in an extended simulated blocks world where a robot manipulates blocks scattered on a table (Fig. 2). We use a 3D rigid-body

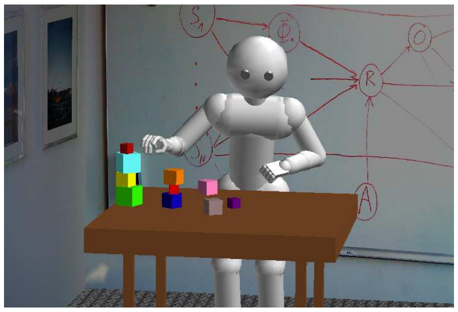

Figure 2. A simulated robot plays with cubes of different sizes scattered on a table. Blocks that have fallen off the table cannot be manipulated anymore.

dynamics simulator (ODE) that enables a realistic behavior of the blocks. For instance, piles of blocks may topple over or blocks may even fall off table (in which case they become out of reach for the robot). The robot can grab blocks and put them on top of other blocks or on the table. Its actions are affected by noise so that resulting block piles are not straight-lined. We assume full observability of triples (s, a, s') that specify how the world changed when an action was executed in a certain state. We represent the data with predicates $on(X, Y)$, $block(X)$, $table(X)$, $out(X)$, $inhand(X)$, $upright(X)$ and function $size(X)$ for state descriptions and $puton(X)$, $grab(X)$ and $doNothing()$ for actions. If there are o objects and f different object sizes, the action space contains $2o+1$ actions while the state space is huge with $f^o 2^{o^2+6o}$ different states (not excluding states one would classify as "impossible" given some intuition about real world physics).

6.2. Experiments

We use the rule learning algorithm of Pasula et al. (2007) with the same parameter settings to learn three different sets of fully abstract NID rules from independent training sets of 500 experience triples each. Training data to learn rules are generated in a world of six cubic blocks of three different sizes by performing random actions with a slight bias to build high towers. The resulting rule-sets contain 12, 9 and 13 rules respectively. We perform three experiments with planning goals of increasing difficulty. If a planning algorithm does not find a suitable action in a given situation, we restart the planning procedure: SST builds a new tree and (A-)PRADA takes new action-sequence samples. If after 10 planning runs for a given situation a suitable action still is not found, the trial fails. In each experiment, we test the planners in worlds with varying numbers of blocks of three different sizes. Thus, we transfer the knowledge gained in the training world to different, but similar worlds by using abstract NID rules. In each experiment, we create five start situations with different blocks for each blocks number.

Table 2. Average height problem. Obj. denotes the number of objects (blocks and table) and *Reward* the discounted total reward. The reward for performing no actions is 8.03.

Obj.	Planner	Reward	Trial time (s)
6+1	SST (b=3)	10.04±1.23	69.02±14.62
6+1	SST (b=4)	9.83±1.20	179.99±41.71
6+1	PRADA	**11.01±1.41**	**9.31±0.34**
6+1	A-PRADA	10.64±1.59	10.64±1.59
8+1	SST (b=3)	9.66±0.88	307.62±74.77
8+1	SST (b=4)	9.87±0.95	796.38±153.16
8+1	PRADA	**10.29±1.14**	**30.81±2.12**
8+1	A-PRADA	9.86±1.20	32.94±2.32
10+1	SST (b=3)	9.72±0.69	1213.76± 511.20
10+1	SST (b=4)	9.71±0.80	3061.37±1112.13
10+1	PRADA	9.69±0.68	**77.24±6.74**
10+1	A-PRADA	**9.96±0.77**	84.47±7.07

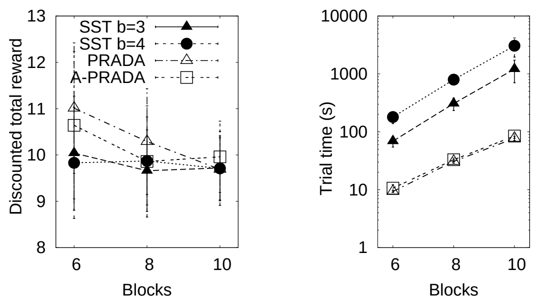

Figure 3. Average height problem

Per rule-set and start situation, we perform three independent runs with different random seeds. Thus, all our reported results are averages over 45 trials (3 rulesets, 5 start situations, 3 runs). All experiments are run on a 2 Ghz. dual-core PC.

6.2.1. Average height

First, we repeat the experiment of Pasula et al. which uses the average height of blocks as the reward function. This constitutes an easy planning problem as many different actions may increase the reward (block identities do not matter) and a small planning horizon d is sufficient. We set SST to $d = 4$ and branching factor $b = 3$ and $b = 4$ (the latter was Pasula's choice) and (A-)PRADA to horizon $d = 6$. Initial states do not contain already stacked blocks. In all our experiments we found that as long as d is not too small, its exact choice does not have significant effects on (A-)PRADA's planning quality. As Table 2 and Fig. 3 show, PRADA and A-PRADA achieve rewards comparable to SST, while their running-time is significantly smaller by an order of magnitude (although affording a longer planning horizon).

6.2.2. Tower of specific blocks

We investigate a slightly more difficult problem: the goal is to stack three *specific* blocks. Start situations are chosen such that the goal can be achieved by means of four actions. We set horizon $d = 4$ optimal for SST – although in principle d, which heavily affects planning

Table 3. Tower of three specific blocks problem. Suc. denotes the success rate, *Actions* the number of executed actions in case of success and *Act. time* the planning time for single actions – in contrast to the trial time in Fig. 4.

Obj.	Planner	Suc.	Actions	Act. time (s)
6+1	SST (b=1)	0.36	6.38±2.31	**0.26±0.06**
6+1	SST (b=2)	0.60	5.96±1.37	1.47±0.31
6+1	SST (b=3)	0.78	5.54±2.13	5.80±1.18
6+1	SST (b=4)	0.80	5.11±1.35	14.91±2.78
6+1	PRADA	0.73	5.58±2.15	0.55±0.08
6+1	A-PRADA	**0.84**	**4.82±1.09**	0.76±0.14
8+1	SST (b=1)	0.29	5.54±1.98	1.45±0.61
8+1	SST (b=2)	0.47	5.86±1.49	6.47±1.48
8+1	SST (b=3)	0.64	5.14±1.41	24.36±4.78
8+1	SST (b=4)	0.69	**4.81±1.08**	65.73±15.14
8+1	PRADA	0.73	6.73±3.47	**1.41±0.32**
8+1	A-PRADA	**0.84**	5.63±1.63	2.12±0.58
10+1	SST (b=1)	0.36	7.62±3.38	5.74±3.87
10+1	SST (b=2)	0.62	6.14±1.92	29.21±15.74
10+1	SST (b=3)	0.67	5.60±1.69	115.83±67.45
10+1	SST (b=4)	**0.84**	**5.42±1.27**	293.93±18.13
10+1	PRADA	0.76	7.53±3.55	**4.21±1.94**
10+1	A-PRADA	0.78	5.77±1.83	5.28±2.09

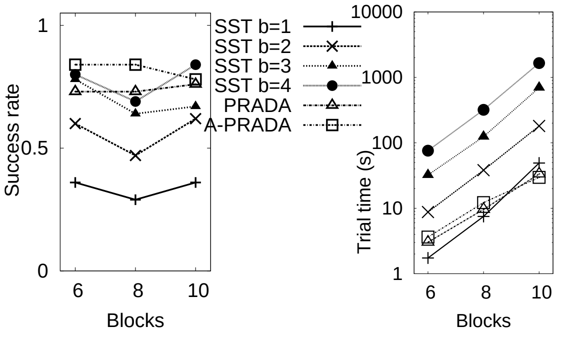

Figure 4. Tower of three specific blocks problem

time, cannot be known a-priori. By contrast, we set $d = 16$ for (A-)PRADA (using the heuristic to set d to twice the average number of blocks which we found to perform well in similar experiments not reported here). A trial is limited by a maximum number of 20 actions. If the goal is not achieved then or if one of the three blocks to be stacked falls off the table, it fails. Table 3 and Fig. 4 show that SST is either extremely slow (for large b) or its performance is bad (for small b). PRADA planning times scale well, as shown in worlds of 8 and 10 blocks, while maintaining a high planning quality. PRADA planning quality is significantly better than that for SST with small b and comparable to SST with large b – while the latter is more than 10-20 times slower. A-PRADA strongly improves planning quality of PRADA and is significantly better than SST with large $b = 3$, for it avoids senseless actions which may cause blocks to fall off the table.

6.2.3. Reverse tower

To explore its limits, we want PRADA to reverse a tower of b blocks which requires at least $2b$ actions (each block needs to be moved at least once). Apart from the long planning horizon, this is difficult due to the noise in the simulated world: towers can become unstable and topple over with blocks falling of the table. To decrease this noise slightly to obtain more reliable results, we forbid the robot to grab objects that are not clear (i.e., below other objects). We set a limit of 50 actions on each trial. Table 4 presents our results. We cannot get SST to solve this problem even for five blocks (with optimal $d = 10$). Although the space of possible actions is reduced due to the mentioned restriction, SST has enormous runtimes. With $b=1$, SST doesn't find suitable actions (no leaves with the goal state) in several starting situations even when building ten trees (taking about two hours) – the increased planning horizon leads to a high probability of sampling at least one bad outcome for a required action. For $b \geq 2$, a single tree traversal takes more than a day. In contrast, PRADA and A-PRADA often succeed in satisfactory times (with planning horizons $d = 20$ for 5 blocks and $d = 30$ for 6 and 7 blocks). When they fail this is mainly due to blocks falling of the table and not because actions cannot be found anymore. A-PRADA can reverse a tower of 7 blocks with a success rate $> \frac{1}{2}$ and trial time about $5\frac{1}{2}$ minutes.

7. Conclusions and Outlook

We have introduced an efficient planning method for the probabilistic relational rules proposed by Pasula et al. (2007) based on approximate inference in DBNs. This enables us to learn the dynamics of a complex stochastic world and quickly derive appropriate actions for varying goals. Results in a 3D simulated blocksworld with an articulated manipulator and realistic physics show that our method outperforms existing approaches. Our approach relies on working in the full ground representation. To apply it in domains with very many objects, it needs to be combined with methods that reduce the state and action space complexity in relational domains, see e.g. Gardiol and Kaelbling (2007). Learning rule-sets online and exploiting them immediately by our planning method is another direction of future research as well as extensions that manipulate sampled action sequences.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. This work was supported by the German Research Foundation (DFG), Emmy Noether fellowship TO 409/1-3.

References

Botvinick, M. M., & An, J. (2009). Goal-directed decision making in prefrontal cortex: a computational framework. *Advances in Neural Information Processing Systems (NIPS)* (pp. 169–176).

Croonenborghs, T., Ramon, J., Blockeel, H., & Bruynooghe, M. (2007). Online learning and exploiting relational models in reinforcement learning. *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)* (pp. 726–731).

Dayan, P., & Hinton, G. E. (1997). Using expectation-maximization for reinforcement learning. *Neural Computation*, *9*, 271–278.

Gardiol, N. H., & Kaelbling, L. P. (2007). Action-space partitioning for planning. *Proc. of the Nat. Conf. on Artificial Intelligence (AAAI)* (pp. 980–986).

Getoor, L., & Taskar, B. (Eds.). (2007). *Introduction to statistical relational learning*. MIT Press.

Kearns, M. J., Mansour, Y., & Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, *49*, 193–208.

Kersting, K., van Otterlo, M., & de Raedt, L. (2004). Bellman goes relational. *Proc. of the Int. Conf. on Machine Learning (ICML)* (pp. 465–472).

Murphy, K. P., & Weiss, Y. (2001). The factored frontier algorithm for approximate inference in DBNs. *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)* (pp. 378–385).

Pasula, H. M., Zettlemoyer, L. S., & Kaelbling, L. P. (2007). Learning symbolic models of stochastic domains. *Artificial Intelligence Research*, *29*, 309–352.

Sanghai, S., Domingos, P., & Weld, D. (2005). Relational dynamic Bayesian networks. *Artificial Intelligence Research*, *24*, 759–797.

Sanner, S., & Boutilier, C. (2007). Approximate solution techniques for factored first-order MDPs. *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)* (pp. 288–295).

Toussaint, M., & Storkey, A. (2006). Probabilistic inference for solving discrete and continuous state Markov decision processes. *Proc. of the Int. Conf. on Machine Learning (ICML)* (pp. 945–952).

van Otterlo, M. (2009). *The logic of adaptive behavior*. IOS Press, Amsterdam.

Table 4. Reverse tower problem. Suc. is the success rate and *Actions* the number of used actions in case of success.

Obj.	Planner	Suc.	Trial time (s)	Actions
5+1	SST (b=1)	0.0	-	-
5+1	SST (b=2)	0.0	>1 day	-
5+1	PRADA	**0.84**	79.9±26.5	12.6±2.9
5+1	A-PRADA	0.78	**66.3±15.6**	**10.6±1.4**
6+1	PRADA	0.42	**184.9±51.9**	14.6±2.5
6+1	A-PRADA	**0.49**	190.4±49.8	**12.8±1.7**
7+1	PRADA	0.47	415.9±186.3	18.1±5.1
7+1	A-PRADA	**0.56**	**331.6±118.3**	**14.8±1.8**