Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2010)

Reasoning about Deterministic Actions with Probabilistic Prior and Application to Stochastic Filtering

Hannaneh Hajishirzi and Eyal Amir
Department of Computer Science
University of Illinois at Urbana-Champaign
{hajishir, eyal}@illinois.edu

Abstract

We present a novel algorithm and a new understanding of reasoning about a sequence of deterministic actions with a probabilistic prior. When the initial state of a dynamic system is unknown, a probability distribution can be still specified over the initial states. Estimating the posterior distribution over states (*filtering*) after some deterministic actions occurred is a problem relevant to AI planning, natural language processing (NLP), and robotics among others. Current approaches to filtering deterministic actions are not tractable even if the distribution over the initial system state is represented compactly. The reason is that state variables become correlated after a few steps. The main innovation in this paper is a method for sidestepping this problem by redefining state variables dynamically at each time step such that the posterior for time t is represented in a factored form. This update is done using a progression algorithm as a subroutine, and our algorithm's tractability follows when that subroutine is tractable. Our results are for general deterministic actions and in particular, our algorithm is tractable for one-to-one and STRIPS actions. We apply our reasoning algorithm about deterministic actions to reasoning about sequences of probabilistic actions and improve the efficiency of the current probabilistic reasoning approaches. We demonstrate the efficiency of the new algorithm empirically over AI-Planning data sets.

Introduction

Many applications in AI involve stochastic dynamic systems and answering queries about them. Examples of such applications are Natural Language Processing (NLP), robotics, AI planning, autonomous agents, speech recognition, and commonsense query answering. Inference in a stochastic dynamic system is the problem of estimating the systems' state given a sequence of actions and partial observations. This type of inference is called *filtering* in stochastic dynamic systems (it is usually called *progression* when no probabilities are involved).

When applied to real-world problems, many of these applications have very large state spaces, uncertain initial states, and uncertain effects of actions. Therefore, it is crucial to choose a representation that is compact and models the stochasticity of the domain, a representation that also enables efficient inference.

In recent years, there is a growing interest in using action-centered languages for representing dynamic systems in AI planning, NLP, and robotics. Planning Domain Definition Language (PDDL)(McDermott 2000) expresses semantics of actions by describing their preconditions and effects and represents the dynamics of the system. Situation calculus (Reiter 2001) represents changing scenarios with a series of first-order logical formulas. PDDL and Situation calculus are traditionally without probabilities. To model stochasticity, they are augmented with *probabilistic choice* actions which choose deterministic executions and a probability distribution over initial states.

Current approaches to probabilistic filtering with PDDL or situation calculus (e.g., (Reiter 2001; Bacchus, Halpern, and Levesque 1999; Hajishirzi and Amir 2008)) use traditional filtering methods as subroutines for reasoning about sequences of deterministic actions with probabilistic priors. These traditional methods are inefficient or imprecise for deterministic sequences. Some approaches marginalize over all possible initial states (exponential in the number of variables) to compute the posterior probability of a query. In others (e.g., Dynamic Bayesian Networks (DBNs) (Dean and Kanazawa 1988)) all the state variables become fully correlated after a few steps even if they are independent at time 0, resulting in a posterior representation of size exponential in the number of variables. Others use logical regression and repeat $t-1$ regressions for every new added action, so are inefficient for long sequences of actions.

The main contribution of this paper is an understanding of conditional-independence structure preservation over time in systems with deterministic actions and stochastic priors over initial states. Our new understanding leads to a new exact algorithm for reasoning about sequences of deterministic actions with a probabilistic prior over the initial states. The algorithm is tractable for 1:1 and STRIPS actions, following results of (Amir and Russell 2003).

We use a propositional version of probabilistic situation calculus that is extended with a graphical model prior for representing dynamic systems. In particular, the initial knowledge is represented with a prior distribution over state variables (in a Bayesian Network (BN) (Pearl 1988) format) and transitions are modeled naturally as stochastic choices among deterministic actions. Our algorithm uses a deterministic progression subroutine and represents the posterior

at time t with a BN whose structure and conditional probabilities are identical to those of the BN at time 0, but whose nodes have a new meaning.

Specifically, every node in the BN representation of posterior at time t corresponds to a propositional logical formula that represents a set of world states. For example, when a binary node X_i (time 0) takes value 1, then it represents the set of world states that satisfy $X_i = 1$. At time t, a binary node Φ_i^t would be a logical formula over $x_1, ..., x_n$ at time t. When this binary node takes value 1, then it represents the set of world states that satisfy $\Phi_i^t = 1$. The BN comprised of such nodes at time t represents the posterior distribution over states at time t.

Finally, we apply our exact filtering algorithm to reason about sequences of probabilistic actions. Our theoretical and empirical results show that our new algorithm improves the efficiency of the sampling algorithm (Hajishirzi and Amir 2008) for filtering with probabilistic actions. The improvement is due to the fact that we remove the expensive subroutine of regressing to time zero at every time step and just use progression.

Related Work

(Reiter 2001) and (Bacchus, Halpern, and Levesque 1999) present exact algorithms to answer a query given a sequence of actions, and observations in a dynamic system represented in a probabilistic situation calculus form. Both algorithms marginalize over all the possible initial states and all the possible deterministic sequences to compute the probability of a world state at time t. Both algorithms assign probability to every world state individually, while our method uses a BN to compactly represent the prior distribution.

First order MDPs (Boutilier, Reiter, and Price 2001) use probabilistic situation calculus to represent the dynamics of the system. They introduce a dynamic programming approach for solving MDPs by describing the optimal value function and policies in a logical format. Their approach uses a logical regression subroutine which results in a combinatorial explosion even for simple deterministic actions.

(Domshlak and Hoffmann 2006; Domshlak and Hoffmann 2007) use a framework similar to ours for representing dynamic systems. They represent the belief state at each time step with a weighted logical formula and answer queries about variables at time step t using weighted model counting, similar to our approach. Unlike our method, they do not compute a compact representation of the belief at each time step.

A DBN compactly represents a dynamic system using a BN for time 0 and a graphical representation of a transition model between times t and $t + 1$. DBNs focus on conditional independence assumption whereas our representation focuses on decomposition of actions into deterministic actions. Traditional methods for exact inference algorithms in DBNs (Murphy 2002) are not tractable because all the state variables become correlated after a few steps even for deterministic transitions. (Pfeffer 2001) presents an exact tractable inference algorithm for a class of DBNs with no observations. This method assumes that the DBN is decomposed into separable subsystems. In contrast, our exact inference is applicable to deterministic inseparable DBNs.

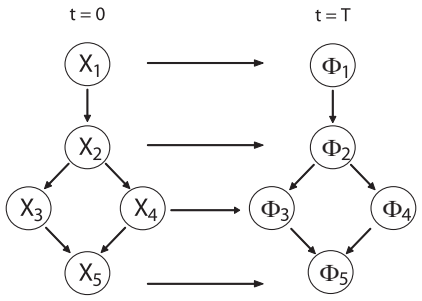

Figure 1: (*left*) *BN* at time 0 with state variables $X_1 \dots X_n$, (*right*) New *BN* constructed at time T with new BN bases $\Phi_1 \dots \Phi_n$. BN^0 and BN^T have identical structure.

Definitions and Background

In this section, we first define the semantics and language of our model. We later describe *logical filtering* (Amir and Russell 2003) that we use as a subroutine in our novel filtering algorithms.

Probabilistic Action Model

In this section, we define Probabilistic Action Model (PAM) for representing dynamic systems. This is modeled conveniently in a propositional version of probabilistic situation calculus (Reiter 2001), extended with a graphical model prior in the form of a BN. More formally,

Definition 1. A Probabilistic Action Model (PAM) is a tuple $\langle X, S, P^0, A, DA, T, PA \rangle$.

- X is a finite set of state variables.
- S is the set of world states $s = \langle x_1 \dots x_{|X|} \rangle$, where each x_i is a truth assignment to state variable $X_i \in X$, for every $1 \leq i \leq |X|$.
- P^0 is a prior probability distribution over the world states at time 0 (represented with a BN).
- A, DA are finite sets of probabilistic and deterministic action names, respectively.
- $T : S \times DA \rightarrow S$ is a transition function for deterministic actions.
- $PA : DA \times A \times S \rightarrow [0, 1]$ is a probability distribution over possible deterministic executions of probabilistic actions in a given world state.

Every state variable in this representation is considered as a logical proposition (takes either *true* or *false*), and it represents a set of world states. Every full joint assignment to state variables represents a world state. A logical formula over state variables represents a set of world states.

Definition 2 (Action specification). Every deterministic action is represented with effect rules of the form "a **causes** F **if** G" describe the preconditions G and effects F of the action. The preconditions and effects are represented compactly with logical formulas (propositional combinations of variable names).

Notice that the effect rules for a deterministic action should not contradict with each other i.e., all the rules that are activated in a world state s should result in a unique state s'. We allow for disjunctions in the effects as long as the action is still deterministic (e.g., the effect cannot include a disjunct that has not appeared in the precondition of the action).

A variable is affected by the action a in state s if there is a rule in the form of "a **causes** F **if** G" where G is *true* in s and the variable is included in the action's effect F. If a variable has not been affected by the action, then its value does not change after performing the action (a.k.a Frame Assumption). We define the deterministic transition corresponding to each deterministic action more formally as follows:

Definition 3. [Deterministic transition function] Let da be a deterministic action specified with a set of rules in the form of "a **causes** F **if** G". Let $F(s, da)$ be the set of all the effects that are activated in world state s. Let $I(s, da)$ be the variables that are not affected by action da and let *Vars*(s) be the variables in the state s. Then, for every world state $s \in S$, $T(s, da) = s'$ is the corresponding deterministic transition function iff

- $(I(s, da) \cap$ *Vars*$(s)) = (I(s, da) \cap$ *Vars*$(s'))$ and $F(s, da)$ is *true* in s'.
- The transition is deterministic i.e., if $T(s, da) = s'_1$ and $T(s, da) = s'_2$ then $s'_1 = s'_2$.

Therefore, there is a transition function assigned to every deterministic action.

Each probabilistic action is represented as a probability distribution over its deterministic actions. More formally, the probability distribution for probabilistic actions is defined as follows:

Definition 4. [Probability distribution for probabilistic actions] Let $\psi_1 \dots \psi_k$ be propositional formulas (called *partitions*) that divide the world states into mutually disjoint sets. Then, *PA*$(da|a, s)$ is defined as a probability distribution over possible deterministic executions da of the probabilistic action a in the state s which satisfies one of the partitions ψ_i $(i \leq k)$. More formally, when some state s satisfies partition ψ_i then *PA*$(da|a, s) =$ *PA*$_i(da)$, where *PA*$_i$ is a probability distribution over different deterministic executions da of action a corresponding to the partition ψ_i.

$$PA(da|a,s) = \begin{cases} PA_1(da) & s \models \psi_1 \\ PA_2(da) & s \models \psi_2 \\ \dots \end{cases} \quad (1)$$

The prior distribution over state variables is represented compactly in the form of a BN. A BN is a probabilistic graphical model that represents the conditional independence among the state variables $X_i \in X$ in a directed acyclic graph. Edges represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other. The conditional probability of each node given its parents is denoted by $P(X_i|$*parents*$(X_i))$.

For example, the graph in Figure 1(*left*) shows a BN for five variables $X_1 \dots X_5$. Probability of a world state $s = x_1, \dots, x_5$ in that BN is computed as $P(s) = \prod_i P(x_i|$*parents*$(X_i)) = P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_2)P(x_5|x_3, x_4)$.[1] The reason that we choose a BN to represent the prior distribution is that BNs represent probability distributions in a compact way assuming that there is a fairly good amount of conditional independence among the state variables.

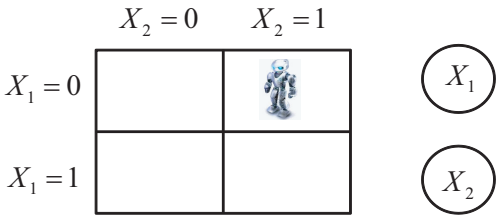

Figure 2: (*left*) X_1 and X_2 are state variables that model the position of the robot in the row x_1 and column x_2 of the grid, respectively. (*right*) Bayes net that models the prior distribution over the initial position of the robot.

Example 1. There is a robot moving in a 2×2 grid (Figure 2). The position of the robot is initially unknown. We model this with a BN whose state variables are X_1 and X_2. $P^0(x_1)$ represents the probability of being at row x_1, and $P^0(x_2)$ is the probability of being at column x_2 of the board. Therefore, there are four different world states $\{s_{00}, s_{01}, s_{10}, s_{11}\}$ as being in one of the four different cells $(0, 0), (0, 1), (1, 0)$ and $(1, 1)$ of the grid. The probability of the logical formula $\varphi^0 = (x_1 x_2) \vee (\neg x_1 \neg x_2)$ is $P(\varphi^0) = P(s_{11}) + P(s_{00})$ meaning that the robot is either at cell $(1, 1)$ or $(0, 0)$.
We define two deterministic actions *move*$_s$ and *move*$_f$ in this domain with the rules "*move*$_s(x_1, x_2)$ **causes** $(\neg x_1 \wedge \neg x_2)$ **if** *true*" (meaning that the robot successfully moves diagonally) and "*move*$_f(x_1, x_2)$ **causes** $(x_1 \wedge x_2)$ **if** *true*" (robot stays at the current cell).
We define a probabilistic action *move* in this domain as a possible execution between *move*$_s$ and *move*$_f$. For example, for action $a =$ *move*, *PA*$(da|a, s) = 0.2$ for $da =$ *move*$_f$ and any state s.

In this setting, the inference problem is defined as computing the probability of a query given a sequence of actions and observations. A query is either a world state or a propositional formula.

In what follows, we first go over *logical filtering*(Amir and Russell 2003) for compact filtering with deterministic actions and no probabilistic prior. We later show how we use logical filtering as a subroutine in our algorithms for filtering with deterministic actions and a probabilistic prior.

Logical Filtering

In this section, we review the logical filtering algorithm (Amir and Russell 2003) that is used as a subroutine in our filtering algorithms (introduced in the next sections). Logical filtering is a specific class of filtering methods in which

[1] Note that we use UPPERCASE letters to represent state variables and lower case letters to represent their values.

belief states, actions and observations are all represented compactly using logical formulae.

Definition 5 (Logical Filtering Semantics). Let σ be a set of states. The filtering of σ with a sequence of deterministic actions and observations $(a_1, o_1, \ldots, a_t, o_t)$ is defined as follows:

1. $\mathit{LogFilter}[da](\sigma) = \sigma$
2. $\mathit{LogFilter}[da](\sigma) = \{s' \mid T(s, da) = s',\ s \in \sigma\}$ where T is the transition function for deterministic actions (Definition 1).
3. $\mathit{LogFilter}[o](\sigma) = \{s \in \sigma \mid o \text{ is } \textit{true} \text{ in } s\}$ presents a set of world states that is possible for the agent to be in after receiving an observation o.
4. $\mathit{LogFilter}(da_i, o_i, \ldots, da_t, o_t)[\sigma] = \mathit{LogFilter}[da_{i+1}, o_{i+1}, \ldots, da_t, o_t](\mathit{LogFilter}[o_i](\mathit{LogFilter}[da_i](\sigma)))$.

When there is no confusion, we omit action names from the arguments of *LogFilter* and just implicitly mention the action or the action sequence.

Example 2. In the moving robot scenario in Example 1, we define the following rule for a new action move_d, "$\mathit{move}_d(x_1, x_2)$ **causes** $(\neg x_1 \wedge \neg x_2)$ **if** x_2". Therefore, $\mathit{LogFilter}[\mathit{move}_d](\sigma) = \{s | \neg x_2 \text{ is true in } s\}$ meaning that the agent will be at column 0 $(\neg x_2)$ after performing action move_d. The reason is that if the agent is initially at column 1 (x_2) then the agent would move to column 0 and therefore $(\neg x_2)$, and if the agent is at column 0 $(\neg x_2)$ then the agent stays at the same state (thus still satisfying $\neg x_2$). Call this resulting belief state σ'. Now, if an observation $o = x_1$ is received (i.e., the agent is at row 1), then $Filter[o](\sigma')$ is exactly the set of worlds that satisfy $x_1 \wedge \neg x_2$.

In what follows, we first introduce algorithms for the case that all the actions in the sequence are deterministic (Section) and then show how we take advantage of the deterministic filtering algorithm to answer queries for the case that the input sequence is in an arbitrary form (Section).

Filtering with Deterministic Actions and a Probabilistic Prior

In this section, we introduce our deterministic filtering algorithms for answering queries with a deterministic action sequence and a BN prior. The inference algorithms share a common idea of dynamic construction of a BN at each time step with an identical topology to the initial BN. The new BN is constructed in terms of new nodes that are expressions over state variables at that time (Figure 1). In essence, our algorithms consist of two major steps: (1) deterministic filtering to time t and deriving new nodes' expressions based on the state variables at time t (2) computing marginals in the constructed BN with the same structure of the BN at time zero.

We use logical filtering as a subroutine for computing the expressions of the nodes of the new BNs. Note that we can replace logical filtering with any other progression algorithms for a sequence of deterministic actions (e.g., (Shahaf and Amir 2007; Levesque 1998; R.E.Bryant 1992)). In this paper, we introduce our results based on the semantics of PAM from Definition 1. Notice that, we can use any different language as long as the filtering can be done.

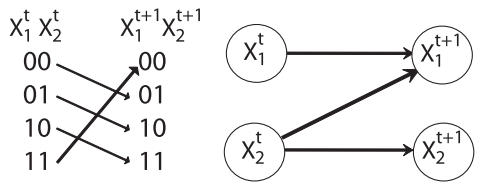

Figure 3: (*left*)1:1 transitions for Example 1. (*right*) The corresponding DBN representation.

In what follows, we describe our algorithms for computing the posterior probability of a world state at each time step given a sequence of actions and a BN prior. We start the section with limiting the actions to be deterministic 1:1 to elaborate ideas of our algorithm. We later relax the assumption of 1:1 action and introduce our algorithm for general deterministic actions. Furthermore, we analyze the correctness and the running time of our algorithms.

1:1 Deterministic Actions

We first restrict the deterministic actions to 1:1 transitions between consecutive world states. The action da is 1:1 if (1) da is deterministic; (2) every world states s_1^t uniquely maps to exactly one world state at time $t+1$, i.e., if $T(s_1^t, da) = T(s_2^t, da)$ then $s_1^t = s_2^t$. For example, flipping a light switch is a 1:1 action while turning on the light is not.

Example 3. For the robot of Example 1, we define 1:1 action move_o as $\mathit{move}_o(x_1, x_2)$ **causes** $(\neg x_1 \wedge \neg x_2)$ **if** x_2 and $\mathit{move}_o(x_1, x_2)$ **causes** $(x_1 \wedge \neg x_2)$ **if** $\neg x_2$. Figure 3(*left*) shows the 1:1 transitions of the robot. Figure 3(*right*) shows the corresponding DBN representation.

Notice that the variables in the DBN represented in Figure 3 become correlated after two steps. Therefore, even for simple 1:1 actions all the variables become correlated after few time steps. This makes the inference algorithm for DBNs intractable when there are many state variables. In what follows, we show a property of 1:1 actions and introduce a tractable filtering algorithm for answering queries given a sequence of 1:1 actions and a BN prior.

The property (proved in Lemma 1) of a 1:1 deterministic action da is that the probability distribution over world states does not change after filtering with action da i.e., $P(s^{t+1}) = P(s^t)$ where $s^{t+1} = \mathit{LogFilter}(s^t)$. This property also holds for a set of world states σ_i and σ_j and their filtering over time: $P(\sigma_i^t) = P(\sigma_i^{t+1})$ and $P(\sigma_i^{t+1} | \sigma_j^{t+1}) = P(\sigma_i^t | \sigma_j^t)$ where $\sigma_i^{t+1} = \mathit{LogFilter}(\sigma_i^t)$ and $\sigma_j^{t+1} = \mathit{LogFilter}(\sigma_j^t)$.

We provide an exact inference algorithm *InfBasis1-1* (Figure 4) for filtering with 1:1 actions given a BN prior. The algorithm dynamically constructs a new BN at each time step that has a similar structure to the BN at time 0 but with new nodes, called *BN bases*. That is, the nodes of the BN at time 0 represent the state variables X_i^0, and the nodes of BN at time t represent the new BN bases Φ_i^t. We define BN bases $\Phi_1^t, ..., \Phi_n^t$ for 1:1 actions as propositional formulas over state variables $X_1^t, .., X_n^t$ whereas every full joint

PROCEDURE *InfBasis1-1*(*PAM*, $da^{1:t}$)
Input: *PAM*($BN^0(X_1^0 \dots X_n^0)$, 1:1 action sequence $da^{1:t}$)
Output: $P(x_1^t \dots x_n^t)$
1. for all i:
   $\Phi_i^t \leftarrow$ *LogFilter*(X_i^0) and $\overline{\Phi_i^t} \leftarrow$ *LogFilter*$(\neg X_i^0)$
2. $BN^t \leftarrow BN^0$ with new nodes Φ_i^t (Theorem 1)
3. for all i:
   $\varphi_i^t \leftarrow \Phi_i^t(x_1^t \dots x_n^t)$ and $\overline{\varphi_i^t} \leftarrow \overline{\Phi_i^t}(x_1^t \dots x_n^t)$
4. Return *Joint* $\leftarrow P(\varphi_1^t \dots \varphi_n^t)$ in $BN^t(\Phi_1^t \dots \Phi_n^t)$

Figure 4: *InfBasis1-1* algorithm for computing joint distribution for 1:1 deterministic sequence $da^{1:t}$ and prior BN^0.

assignment to the state variables $X_1^t, .., X_n^t$ leads to exactly one full joint assignment to $\Phi_1^t, ..., \Phi_n^t$ and vice versa.

The BN bases are derived from applying *logical filtering* with the input 1:1 action sequence over the state variables at time 0 i.e., $\Phi_i^t =$ *LogFilter*(X_i^0) and $\overline{\Phi_i^t} =$ *LogFilter*$(\neg X_i^0)$. The prior and conditional probabilities over the new BN bases are identical to the corresponding distributions at time 0 (demonstrated in Figure 1 and proved in Theorem 1).

Example 4. We compute the new BN bases of Example 3 (Figure 3).

$$\Phi_1^1 = \textit{LogFilter}(X_1^0) = (x_1^1 x_2^1) \vee (\neg x_1^1 \neg x_2^1)$$
$$\overline{\Phi_1^1} = \textit{LogFilter}(\neg X_1^0) = (x_1^1 \neg x_2^1) \vee (\neg x_1^1 x_2^1)$$
$$\Phi_2^1 = \textit{LogFilter}(X_2^0) = \neg x_2^1$$
$$\overline{\Phi_2^1} = \textit{LogFilter}(\neg X_2^0) = x_2^1$$

From Theorem 1 we have: $P(\Phi_1^1 = 1) = P^0(X_1^0 = 1)$.

In summary, our algorithm for filtering with 1:1 actions, *InfBasis1-1*, first updates the BN at each time step by applying logical filtering over the state variables and constructs the new BN bases Φ_i and $\overline{\Phi_i}$. Then, *InfBasis1-1* computes φ_i^t and $\overline{\varphi_i^t}$ as the truth values of the Φ_i^t and $\overline{\Phi_i^t}$, respectively. Notice that both Φ_i^t and $\overline{\Phi_i^t}$ are logical formulae over state variables at time t. Following the definition of BN bases, our algorithm computes the probability of a full joint assignment to BN bases to answer a query about the full joint assignment to state variables. This means that *InfBasis1-1* calls a static inference algorithm at time t and computes the full joint distribution over the new bases at the newly constructed BN at time t which has the same topology as BN^0.

Lemma 1. Let s^t and s^{t+1} be the world states at times t and $t+1$, respectively. If da is a 1:1 action then, $P(s^{t+1}) = P(s^t)$ where $s^{t+1} =$ *LogFilter*$[da](s^t)$.

Proof. We use the Bayes rule to compute $P(s^{t+1})$: $P(s^{t+1}) = \sum_{s'} P(s^{t+1}|s')P(s')$ where s' represents a state at time t. Since da is a 1-1 action: $P(s^{t+1}|s') = 1$ when $s' = s^t$ otherwise it is equal to 0. Hence, $P(s^{t+1}) = P(s^t)$. □

Theorem 1. If $\Phi_i^t =$ *LogFilter*(X_i^0) is the filtering of X_i^0 with 1:1 transitions to time step t then
1. $P(\varphi_i^t) = P^0(x_i^0)$ and $P(\overline{\varphi_i^t}) = P^0(\neg x_i^0)$
2. $P(\varphi_i^t|$*parents*$(\Phi_i^t)) = P^0(x_i^0|$*parents*$(X_i^0))$

Proof. We use the property of a single 1:1 action (that the probability of a world state does not change by performing a 1:1 action) and generalize it to a sequence of 1:1 actions. From Lemma 1 we know that if $s^{t+1} =$ *LogFilter*(s^t) with a 1:1 action then $P(s^{t+1}) = P(s^t)$.
If $\Phi_i^t =$ *LogFilter*(X_i^0) with a sequence of 1:1 actions then we want to prove part 1 of the theorem that $P(\varphi_i^t) = P^0(x_i^0)$ as follows:

$$\begin{aligned} P^0(X_i^0 = x_i^0) &= \sum_{s^0 \models X_i^0} P(s^0) \\ &= \sum_{s^t \models \Phi_i^t} P(s^t) = P(\Phi_i^t = \varphi_i^t) \end{aligned}$$

because the actions are 1:1.
The second part of the theorem is proved from part (1) using the Bayes rule. □

The following theorem shows that the posterior probability of a world state at time t $\langle X_i^t \rangle_{i \le n}$ is computed using the prior and conditional probabilities among the new BN bases $\langle \Phi_i^t \rangle_{i \le n}$ in the derived BN at time t.

Theorem 2. Let $\Phi_i^t =$ *LogFilter*(X_i^0) be the filtering of the state variable X_i^0 with 1:1 actions and let s^t be a world state at time t. If φ_i^t is the evaluation of Φ_i^t in a given s^t then,

$$P(s^t = x_1^t, \dots, x_n^t) = \prod_i P(\varphi_i^t|\textit{parents}(\Phi_i^t)) \qquad (2)$$

Proof. The proof intuition is based on the property of 1:1 actions (Lemma 1) and also the definition of the BN bases. First notice that: $P(s^t = x_1^t \dots x_t^n) = P(s^t = \varphi_1^t \dots \varphi_n^t)$ where φ_i^t is uniquely derived by substituting values of x_i^t in Φ_i^t. Moreover, φ_i^t represents the truth value of the filtering of X_i^0 with the given sequence of 1:1 actions i.e.,

$$\varphi_i^t = \text{ truth value of } \textit{LogFilter}(X_i^0)$$

Therefore,

$$\begin{aligned} P(s^t = x_1^t \dots x_n^t) &= P(s^t = \varphi_1^t \dots \varphi_n^t) \\ &= P^0(s^0 = x_1^0 \dots x_n^0) = \prod_{i=1}^n P^0(x_i^0|\textit{parents}(X_i^0)) \\ &= \prod_{i=1}^n P(\textit{LogFilter}(x_i^0)|\textit{parents}(\textit{LogFilter}(X_i^0))) \\ &= \prod_{i=1}^n P(\varphi_i^t|\textit{parents}(\Phi_i^t)) \end{aligned}$$

□

Example 5. If in Figure 3(*right*) we add the assumption that X_1^0 depends on X_2^0 at time 0. Therefore, probability of a joint distribution at time 0 is computed as: $P^0(s^0 = 10) = P^0(X_1^0 = 1|X_2^0 = 0)P^0(X_2^0 = 0)$. We use algorithm *InfBasis1-1* to compute the full joint distribution $P(s^1 = 11)$ at time 1 given that the action described in

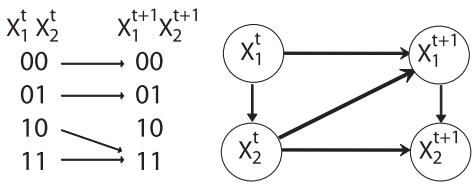

Figure 5: (*left*) Deterministic transitions for Example 6. (*right*) The corresponding DBN representation.

example 2 is executed. We use the formulas derived in Example 4 for the BN bases Φ_is.

$$\begin{aligned}
&P(s^1 = 11)\\
&\quad = P(\Phi_1^1 = \Phi_1^1(s^1) | \Phi_2^1 = \Phi_2^1(s^1)) P(\Phi_2^1 = \Phi_2^1(s^1))\\
&\quad = P(\Phi_1^1 = 1 | \Phi_2^1 = 0) P(\Phi_2^1 = 0)\\
&\quad = P^0(X_1^0 = 1 | X_2^0 = 0) P^0(X_2^0 = 0) = P^0(s^0 = 10)
\end{aligned}$$

The above computations show that $P(s^1) = P(s^0)$ where $s^1 = \mathit{LogFilter}(s^0)$ for $s^1 = 11$ and $s^0 = 10$.

Our filtering algorithm is tractable if the logical filtering subroutine is tractable ((Amir and Russell 2003) presents some tractable cases). Notice that computing probability of a state at BN at time t is $O(n)$ since BN^t has the same structure as BN^0.

Corollary 1 (Running time). Let R_{LF} be the running time of one-step logical filtering over n state variables with 1:1 actions. The running time for computing joint distribution after T-step of 1:1 actions is $O(T \cdot R_{LF} + n)$.

For 1:1 actions that just change at most two literals the running time of logical filtering per step is $O(n^2)$. Therefore, the running time of our filtering algorithm with T-step 1:1 action sequence and the given prior is $O(T \cdot n^2)$. Notice that, for the corresponding DBN (or HMM) for the same sequence the inference at time t is $O(T \cdot \exp(n))$.

General (not 1:1) Deterministic Actions

In this section deterministic actions are no longer 1:1 and can be any general deterministic transitions. Here, we also use the technique of updating the BN using BN bases. Like the previous section, every node at the updated BN is the logical filtering of the state variables at time 0. However, the inference algorithm at time t is not as simple as *InfBasis1-1* (Figure 4). In what follows, we describe *InfBasisDet* (Figure 6) for general deterministic actions. We start the description of the algorithm with an example.

Example 6. We relax the assumption that the transitions are 1:1 for the moving robot of Example 1. Figure 5 shows the new transitions and the corresponding DBN. We perform logical filtering and compute the logical formulas for the new BN bases.

$$\begin{aligned}
\Phi_1^1 &= \mathit{LogFilter}(X_1^0 = 1) = x_1^1 x_2^1 \qquad (3)\\
\overline{\Phi_1^1} &= \mathit{LogFilter}(X_1^0 = 0) = \neg x_1^1\\
\Phi_2^1 &= \mathit{LogFilter}(X_2^0 = 1) = x_2^1\\
\overline{\Phi_2^1} &= \mathit{LogFilter}(X_2^0 = 0) = (\neg x_1^1 \neg x_2^1) \vee (x_1^1 x_2^1)
\end{aligned}$$

PROCEDURE *InfBasisDet*(*PAM*, $da^{1:t}$)
Input: *PAM*($\mathit{BN}^0(X_1^0 \ldots X_n^0)$, deterministic sequence $da^{1:t}$)
Output: $P(x_1^t \ldots x_n^t)$
1. for all i:
   $\Phi_i^t \leftarrow \mathit{LogFilter}(X_i^0)$ and $\overline{\Phi_i^t} \leftarrow \mathit{LogFilter}(\neg X_i^0)$
2. $\mathit{BN}^t \leftarrow \mathit{BN}^0$ with new nodes Φ_i^t (Theorem 1)
3. for all i:
   $\varphi_i^t \leftarrow \Phi_i^t(x_1^t \ldots x_n^t)$ and $\overline{\varphi_i^t} \leftarrow \overline{\Phi_i^t}(x_1^t \ldots x_n^t)$
4. Return $\sum_{\varphi_j^t : \Phi_j^t = \overline{\Phi_j^t} = 1} P(\varphi_1^t \ldots \varphi_n^t)$ in $\mathit{BN}^t(\Phi_1^t \ldots \Phi_n^t)$

Figure 6: *InfBasisDet* algorithm for computing joint distribution given a prior BN^0 and a deterministic sequence $da^{1:t}$.

Like the previous section we build a BN at time t whose structure is the same as the BN at time 0 but with new nodes Φ_1^1 and Φ_2^1. Notice that, $\overline{\Phi_i^1} \neq \neg\Phi_i^1$ because for the given deterministic actions, two different states at time step 0 are mapped to one state at time t.

The procedure for computing the posterior distribution of a state at time t is different from the case of 1:1 transitions. For example, for computing the full joint distribution of $P(s^1 = 11)$ we first replace $x_1^1 = 1$ and $x_2^1 = 1$ in Equation 3 and derive that: $\varphi_1^1 = 1$ and $\overline{\varphi_1^1} = 0$, $\varphi_2^1 = 1$, but $\overline{\varphi_2^1} = 1$. Then, to compute the probability of being at state $s = 11$ we sum over the probabilities of combinations of φ_1 and φ_2 that are true, i.e., $P(s^1 = 11) = P(\varphi_1^1 \varphi_2^1) + P(\varphi_1^1 \overline{\varphi_2^1})$. Therefore, we compute the above marginal to report the posterior probability of state s^1. The reason that justifies this computation is that states s_{11} and s_{10} are both mapped to $s^1 = 11$ (Figure 5) and $P(s^1 = 11) = P^0(x_1^0 x_2^0) + P^0(x_1^0 \neg x_2^0)$. Note that $P(\varphi_1^1 \varphi_2^1) = P^0(x_1^0 x_2^0)$ and $P(\varphi_1^1 \neg\varphi_2^1) = P^0(x_1^0 \neg x_2^0)$ because the transition is deterministic.

We extend the ideas described in the previous example to introduce Algorithm *InfBasisDet* (Figure 6) for computing the posterior probability of a world state given a sequence of deterministic actions and a BN prior. Similar to our treatment of 1:1 actions, logical bases $\Phi_1^t, \ldots, \Phi_n^t$ are logical formulas over state variables $X_1^t, \ldots, X_n^t$. Unlike our analysis of 1:1 actions, in the present case a full joint assignment to $X_1^t, \ldots, X_n^t$ can be mapped to more than one full joint assignment of $\Phi_1^t, \ldots, \Phi_n^t$. The reason is that both φ_i^t and $\overline{\varphi_i^t}$ can be true after filtering with deterministic actions.

Procedure *InfBasisDet* is similar to Procedure *InfBasis1-1* except the last step for computing the joint distribution at time t. Like before, every logical basis Φ_i^t is derived from applying logical filtering to state variables at time step 0. But, we replace the last step of computing the full joint distribution at time t with a more expensive procedure of computing a marginal distribution at time t. This is required because given the truth values of state variables at time t both Φ_i^t and $\overline{\Phi_i^t}$ can be true.

Theorem 3. Let $\Phi_i^t = \mathit{LogFilter}(X_i^0)$ and $\overline{\Phi_i^t} =$

LogFilter($\neg X_i^0$) with the input deterministic actions. If s^t is a world state at time t then,

$$P(s^t = x_1^t, \ldots, x_n^t) = \sum_{\Phi_j^t} \prod_i P(\varphi_i^t | \textit{parents}(\Phi_i^t)) \quad (4)$$

where Φ_j^t is a BN basis where both Φ_j^t and $\overline{\Phi_j^t}$ are evaluated to *true* by substituting $x_1^t, \ldots, x_n^t$ in them.

Proof. The intuition of the proof is that we map the marginalization at time t to a marginalization the over state variables at time step 0 because probability of a state at time t is actually the summation over probabilities of states at time step 0. Using this mapping, we show how we can write the closed form formula for computing the formula directly at time t.

Assume there are states s_j at time 0 that map to s^t through the deterministic transitions. Therefore,

$$\begin{aligned} P(s^t = x_1^t \ldots x_n^t) &= \sum_j P^0(s_j^0) \quad (5) \\ &= \sum_{X_j^0} \prod_i P^0(x_i^0 | \textit{parents}(X_i^0)) \end{aligned}$$

Notice that to compute the summation over probabilities of different s_j, we marginalize over the state variables X_j^0 that take both truth values, *true* and *false*. Probability of s_j at time 0 is computed in the Bayes net at time 0.

Since actions are deterministic, $P^0(s_j^0 = x_1^0 \ldots x_n^0) = P(s^t = \textit{logFilter}(X_1^0) \ldots \textit{logFilter}(X_n^0))$ where *logFilter* represents the truth value of the *LogFilter* operation. Notice that the truth value of filtering of X_i^0 is either φ_i^t or $\overline{\varphi_i^t}$ based on the truth value of X_i^0. Therefore, we can write Equation 5 as a marginalization over logical bases. This means that X_j^0 is replaced with the corresponding Φ_j^t where both Φ_j^t and $\overline{\Phi_j^t}$ are evaluated to *true*. Hence,

$$\sum_{X_j^0} \prod_i P^0(x_i^0 | \textit{parents}(X_i^0)) = \sum_{\Phi_j^t} \prod_i P(\varphi_i^t | \textit{parents}(\Phi_i^t)).$$

□

Running time of our filtering algorithm depends on (1) running time of logical filtering over state variables (2) computing marginals in the BN at time t (similar complexity to computing marginals in the initial BN since BN^t has the same structure as BN^0).

Corollary 2 (Running time). Let R_{LF} be the running time of one-step logical filtering over n state variables with deterministic actions. If the initial BN has treewidth *tw*, then running time for computing joint distribution after T steps of deterministic actions is $O(T \cdot R_{LF} + n \cdot \exp(tw))$.

Theorem 4.12 in (Amir and Russell 2003) presents that the running time of logical filtering with a STRIPS action da over a k-CNF formula with s clauses per step is $O(s \cdot k)$ and the size of the representation remains compact. Moreover, computing marginals is tractable when the initial BN has a low treewidth. In conclusion, the running time of our deterministic filtering algorithm for STRIPS actions is $O(T \cdot n \cdot s \cdot k + n \cdot \exp(tw))$. Note that in DBNs this inferences is $O(T \cdot \exp(n))$ where n is usually much larger than the treewidth. For example, in Figure 1(*left*) the treewidth of the graph is 2, while the number of variables is 5.

What we discussed above was about computing the posterior probability of a world state at time step t given the deterministic actions and the prior distribution. We compute the probability of a logical formula δ^t over state variables at time t by applying model counting (e.g., (Gomes, Sabharwal, and Selman 2006; Chavira and Darwiche 2008)). We enumerate all the world states (full joint assignments) that satisfy δ^t at time t and then sum over the probabilities of those full joint distributions. The exact model counting algorithms are not tractable, but the good news is that we can apply importance sampling and approximate $P(\delta^t)$.

Filtering with Observations

In our settings, observations $\langle o^0, \ldots, o^T \rangle$ are given asynchronously in time without prediction of what we will observe (thus, this is different from HMMs (Rabiner 1989) where a sensor model is given). Each observation o^t is represented with a propositional formula over state variables. When o^t is observed at time t, the formula o^t is true about the state of the world at time t. We assume that a deterministic execution da^t of the probabilistic action a^t in the sequence does not depend on the future observations.

Our algorithm collects all the observations from time 0 to time t in an expression represented with *Obs*. Our algorithm uses logical filtering and updates the expression *Obs* whenever a new observation o^t is received.Notice that our algorithm does not reconstruct the BN bases based on the received observations. It collects all the observation in an expression and enforces the effects of the observations in the final computation of the joint distribution.

Our inference algorithm uses the following equation to compute the full joint distribution given observations *Obs*:

$$P(x_1^t \ldots x_n^t | o^{1:t}) = \frac{1}{Z} \Theta(X_{Obs}^t) \prod_j P(\varphi_j^t | \textit{parents}(\Phi_j^t))$$

where Θ is a potential function over state variables: $\Theta(X_{Obs}^t) = 1$ if X_{Obs}^t satisfies *Obs* and $\Theta(X_{Obs}^t) = 0$ otherwise. The normalization factor Z for the above distribution is $P(\textit{Obs})$. The reason is that: $P(s^t | o^{1:t}) = P(s^t, o^{1:t}) / P(o^{1:t})$, where *Obs* represents $o^{1:t}$. Notice that *Obs* is actually a logical formula at time t and $P(\textit{Obs})$ is derived by applying model counting to compute the probability of the logical formula *Obs*.

Applying Deterministic Filtering Algorithm for Probabilistic Transitions

In this section we apply our deterministic filtering algorithm to answer queries given a sequence of probabilistic actions. We improve the sampling technique, Logical Particle Filtering (LPF), (Hajishirzi and Amir 2008) by removing the regression step. This way, we improve the efficiency of the LPF algorithm while we maintain the accuracy for a fixed

PROCEDURE *LPF*(*PAM*, δ^T, $a^{1:T}$, $o^{0:T}$)
Input: probabilistic sequence $a^{1:T}$, observations $o^{0:T}$, and query δ^T
Output: $\tilde{P}(\delta^T|a^{1:T}, o^{0:T})$
1. $(\vec{DA}_{1:N}, curF_{1:N}) \leftarrow SampleAlg(a^{1:T}, o^{0:T})$
2. for each $\vec{DA}_i$ $computeP(\delta^T, \vec{DA}_i, curF_i)$
3. **return** $\tilde{P}(\delta^T|a^{1:T}, o^{0:T}) = \sum_i w_i P(\delta^T|\vec{DA}_i, o^{0:T})$.

Figure 7: LPF: approximating $P(\delta^t|a^{1:t}, o^{0:t})$ for the probabilistic sequence $a^{1:t}$ and observations $o^{1:t}$

number of samples. We first review the logical particle filtering technique and then show how our deterministic filtering algorithm fits into this probabilistic filtering algorithm.

Logical particle filtering algorithm (Figure 7) answers a query $P(\delta^T|a^{1:T}, o^{0:T})$ given a sequence of probabilistic actions $a^{1:T} = \langle a^1, \ldots, a^T \rangle$ and observations $o^{0:T} = \langle o^0, \ldots, o^T \rangle$ in a PAM. The algorithm approximates this probability by generating samples among possible deterministic executions of the given probabilistic action sequence. Then, it places those samples instead of the enumeration of deterministic executions and marginalizes over those samples. The following equation shows the exact computation.

$$P(\delta^T|a^{1:T}, o^{0:T}) = \sum_i P(\delta^T|\vec{DA}_i, o^{0:T}) P(\vec{DA}_i|a^{1:T}, o^{0:T}) \quad (6)$$

where $\vec{DA}_i$ is a possible execution of the sequence $a^{1:T}$.

The first step of the LPF algorithm, *SampleAlg*, generates a weighted particle $\vec{DA}_i$ with weight w_i given the sequence $a^{1:T}$ and the observations $o^{0:T}$ from the probability distribution $P(\vec{DA}_i|a^{1:T}, o^{0:T})$.

Second step of LPF computes $P(\delta^T|\vec{DA}_i, o^{0:T})$ for each sample $\vec{DA}_i$ by calling *computeP*. It first updates the current state of the system by logical progression into *curF* and then computes the posterior probability conditioned on the *curF*. To compute the probability of a formula at time step t (Hajishirzi and Amir 2008) regresses the query back to time 0 and derives formula δ^0 where the prior distribution over state variables exists. The regressed formula δ^0 represents the set of possible initial states, given that the final state satisfies δ^t, and the logical particle $\vec{DA}$ occurs. Thus, every state that satisfies δ^0 leads to a state satisfying δ^t after $\vec{DA}$ occurs.

Finally, the algorithm uses the generated samples in place of $\vec{DA}_i$ in Equation (6) and approximates posterior probability of the query φ^T given the sequence $a^{1:T}$ and the observations $o^{0:T}$ by using the Monte Carlo integration (Doucet, de Freitas, and Gordon 2001): $\tilde{P}_N(\delta^T|a^{1:T}, o^{0:T}) = \sum_i w_i P(\delta^T|\vec{DA}_i, o^{0:T})$.

To sample each logical particle, *SampleAlg* uses subroutine *computeP* and computes the posterior probability of logical formulas ψ_i^t (partitions in Definition 4) at each time step. Recall that *computeP* regresses each ψ_i^t back to time 0 at every time step to compute $P(\psi^t)$. Therefore, sampling each logical particle takes $O(T^2 \cdot R_{Reg})$ where R_{Reg} is the running time of regressing a logical formula one step. We apply

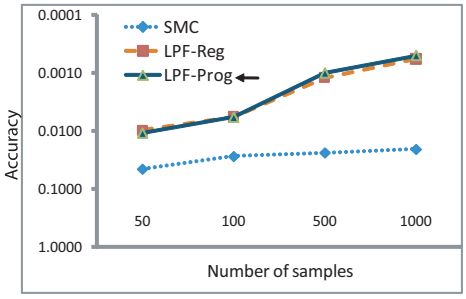

Figure 8: Sampling accuracy: Expected KL-divergence in **reverse logarithmic scale** of *LPF-Prog* (our algorithm), *LPF-Reg*(regression at every step) and SMC with the exact distribution vs. number of samples for *depots* domain.

our deterministic filtering algorithm (presented in Section) and update *computeP* procedure to compute the probability of logical formulas just with applying progression. Therefore, the running time with just progression is $O(T \cdot R_{LF})$ where R_{LF} is the running time of one step filtering a logical formula.

Empirical Results

We implemented our deterministic filtering algorithm and used it as a subroutine in probabilistic filtering algorithm (LPF). Like (Hajishirzi and Amir 2008) we examined our algorithm on a planning type domain (*depots*) taken from International Planning Competition at AIPS-98 and AIPS-02 (McDermott 2000). In (Hajishirzi and Amir 2008) the *depots* domain is augmented with a prior distribution and a probability distribution for actions. Here, we use similar action descriptions and probability distributions.

We first compared the accuracy of LPF with new deterministic filtering subroutine (LPF-Prog) with LPF with regression steps (LPF-Reg) in (Hajishirzi and Amir 2008) and sequential Monte Carlo sampling (SMC) in (Doucet, de Freitas, and Gordon 2001). We ran each algorithm 50 times for a fixed number of samples and computed the Kullback-Leibler(KL)-divergence between each approximation and the exact posterior as a measure for accuracy. We calculated the average over these derived KL-distances to approximate the expected KL-divergence. Figure 8 shows that the average KL-divergence does not grow when we remove the regression steps and replace them with just progression algorithms. Moreover, the accuracy of both techniques are much higher than the SMC algorithm for a fixed number of samples.

We next compared the running time of LPF with progression (LPF-Prog) with LPF with regression (LPF-Reg). As Figure 9 shows the running time of LPF-Prog is much better than LPF-Reg. The reason is that we removed the regression back to time 0 at every time step. Moreover, the actions are probabilistic STRIPS and therefore progression is tractable (polynomial in terms of number of variables).

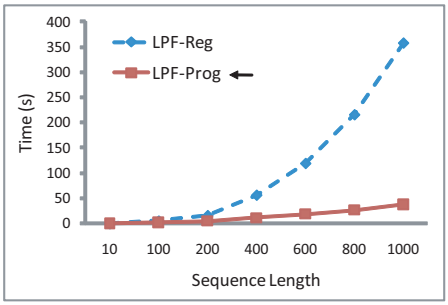

Figure 9: Running time of our algorithm *LPF-Prog* (just progression) vs. *LPF-Reg* (regression at every step) versus the sequence length in the *depots* domain.

Conclusions and Future Work

In this paper, we presented an exact filtering algorithm for answering queries given a sequence of deterministic actions and a probabilistic prior. Our results are for general deterministic actions, but the algorithm is tractable for 1:1 and STRIPS actions. In fact, tractability of our algorithm results from the tractability of the logical filtering subroutine. The algorithm dynamically applies logical filtering to variables of the initial BN and constructs a new BN (with new bases) whose structure is similar to the structure of the initial BN. This way, the query answering is reduced to answering the query in the constructed BN at time step t.

We also embedded our deterministic filtering algorithm in a filtering algorithm with a sequence of probabilistic actions and observations. This way, we improved the running time of the previous general filtering algorithm by removing the need for regressing back to time 0 at every time step.

In future, we plan to apply this representation and the inference algorithm to narrative understanding in NLP. Also, we are looking for more cases (in addition to STRIPS and 1:1 actions) that our algorithm is tractable. Moreover, in the general sampling algorithm, we would like to use the fact that the sampled deterministic action sequences has occurred. This way, we can prune our deterministic filtering algorithm and incorporate the knowledge about each action while we proceed.

Acknowledgements We would like to thank the anonymous reviewers for their helpful comments. This work was supported by DARPA SRI 27-001253 (PLATO project) and NSF CAREER 05-46663 grants.

References

Amir, E., and Russell, S. 2003. Logical filtering. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 75–82. Morgan Kaufmann.

Bacchus, F.; Halpern, J. Y.; and Levesque, H. J. 1999. Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence* 111(1–2):171–208.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first order MDPs. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 690–700.

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172(6-7).

Dean, T., and Kanazawa, K. 1988. Probabilistic temporal reasoning. In *Proc. Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, 524–528.

Domshlak, C., and Hoffmann, J. 2006. Fast probabilistic planning through weighted model counting. In *International Conference on Automated Planning and Scheduling*, 243–252.

Domshlak, C., and Hoffmann, J. 2007. Probabilistic planning via heuristic forward search and weighted model counting. *Artificial Intelligence* 30(565-620).

Doucet, A.; de Freitas, N.; and Gordon, N. 2001. *Sequential Monte Carlo Methods in Practice*. Springer, 1st edition.

Gomes, C.; Sabharwal, A.; and Selman, B. 2006. Model counting: A new strategy for obtaining good bounds. In *Proc. Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, 54–61.

Hajishirzi, H., and Amir, E. 2008. Sampling first order logical particles. In *Proc. Conference on Uncertainty in Artificial Intelligence (UAI)*.

Levesque, H. 1998. A completeness result for reasoning with incomplete first order knowledge bases. In *Proc. International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

McDermott, D. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2).

Murphy, K. 2002. *Dynamic Bayesian Networks: Representation, Inference and Learning*. Ph.D. Dissertation, Berkeley.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann.

Pfeffer, A. 2001. Sufficiency, separability and temporal probabilistic models. In *Proc. UAI '01*, 421–428. MK.

Rabiner, L. R. 1989. A tutorial on HMM and selected applications in speech recognition. *IEEE* 77(2).

R.E.Bryant. 1992. Symbolic boolean manipulation with ordered BDDs. *ACM* 24(3).

Reiter, R. 2001. *Knowledge In Action: Logical Foundations for Describing and Implementing Dynamical Systems*. MIT Press.

Shahaf, D., and Amir, E. 2007. Logical circuit filtering. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2611–2618.