Stochastic Filtering in a Probabilistic Action Model

Hannaneh Hajishirzi and Eyal Amir
Computer Science Department
University of Illinois at Urbana-Champaign
Urbana, IL 61801. USA
{hajishir, eyal}uiuc.edu

Abstract

Stochastic filtering is the problem of estimating the state of a dynamic system after time passes and given partial observations. It is fundamental to automatic tracking, planning, and control of real-world stochastic systems such as robots, programs, and autonomous agents. This paper presents a novel sampling-based filtering algorithm. Its expected error is smaller than sequential Monte Carlo sampling techniques given a fixed number of samples, as we prove and show empirically. It does so by sampling deterministic action sequences, and exact filtering of those sequences. These results are promising for applications in stochastic planning, natural language processing, and robot control.

1 Introduction

Controlling a complex system involves executing actions and estimating the system's state (*filtering*) given past actions and partial observations. Filtering determines a posterior distribution over the system's state at the current time step, and permits effective control, diagnosis, and evaluation of achievements. Such estimation is necessary when the system's exact initial state or the effects of its actions are uncertain (e.g., there may be some noise in the system or its actions may fail).

Unfortunately, exact filtering (e.g., (Kjaerulff 1992; Bacchus, Halpern, & Levesque 1999)) is not tractable for long sequence of actions in complex systems. This is because domain features become correlated after some steps, even if the domain has much conditional-independence structure (Dean & Kanazawa 1988). Sequential Monte Carlo methods (Doucet, de Freitas, & Gordon 2001) are popular sampling methods that try to circumvent this problem. Unfortunately, while efficient, they require many samples to yield low error in high-dimensional domains (frequently, exponential number in this dimensionality).

In this paper we present a novel sampling algorithm for filtering that takes fewer samples and yields better accuracy than sequential Monte Carlo (SMC) methods. The key to our algorithm's success is an underlying deterministic structure for the transition system, and efficient subroutines for *logical regression* (e.g., (Reiter 2001)) and *logical filtering* (Amir & Russell 2003) .

We represent actions' effects as multinomial distributions over a set of possible deterministic effects (every transition model can be represented this way). This is modeled conveniently in a propositional version of probabilistic situation calculus (Reiter 2001), extended with a graphical model prior (Pearl 1988) (Section 2).

Our method (Section 3) samples sequences of deterministic actions (called *logical particles*) that are possible executions of the given probabilistic action sequence. Then, it applies logical regression to the query, and finds a formula that represents all possible initial states given this sample sequence. Finally, we compute the posterior probability as the weighted sum of the probabilities of these formulae. As a special case, our algorithm is exact when actions are deterministic, still allowing a probabilistic graphical model prior.

This algorithm achieves superior precision with fewer samples than SMC sampling techniques (Doucet, de Freitas, & Gordon 2001). The intuition behind this improvement is that each logical particle corresponds to exponentially many state sequences (particles) generated by earlier techniques.

The algorithm is efficient computationally when logical regression of the deterministic effects is efficient (thus, whenever the representation of those deterministic effects is compact). We prove the claims formally (Section 3.3) and verify them empirically by several experiments (Section 4).

Our representation for a dynamic probabilistic model differs from the more commonly used Dynamic Bayesian Networks (DBNs) (e.g., (Murphy 2002)). DBNs represent stochastic processes in a compact way using a Bayes Net (BN) for time 0 and a graphical representation of a transition distribution between times t and $t+1$. Their structure emphasizes conditional independence among random variables. In contrast, our model applies a different structure, namely, a representation for the transition model as a distribution over deterministic actions. Both frameworks are universal and can represent each other, but they are more compact and natural in different scenarios.

Algorithms for exact filtering in discrete probabilistic domains trade efficiency of computation for precision. The main disadvantage of these algorithms is that they are not tractable for large domains. Exact algorithms introduced for DBNs and HMMs (e.g., (Kjaerulff 1992; Murphy 2002; Rabiner 1989)) are mostly suitable for their given probabilistic structure. (Bacchus, Halpern, & Levesque 1999) presents

an exact algorithm to answer a query given a sequence of actions in a dynamic probabilistic model in situation calculus. They assign probability to each world state individually (exponentially many in the number of variables). Instead, our algorithm approximates the posterior distribution, and uses a graphical model to represent the prior distribution.

Approximate filtering algorithms are common in the literature, and we recount some of those not already mentioned. Variational methods (Jordan *et al.* 1999) are in a range of deterministic approximation schemes and are based on analytical approximations to the posterior distribution; They make some assumptions about the posterior distribution, for example by assuming that it factorizes in a particular way. Therefore, They can never generate exact results. However, our algorithm does not make such assumptions and can generate the exact result with infinite number of samples. (Mateus *et al.* 2001) introduces a probabilistic situation calculus logical language to model stochastic dynamic systems. The assumption of knowing the exact initial state a priori is the key difference from the problem we are addressing here.

Probabilistic planning in partially observable domains uses stochastic filtering as a subroutine. Exact algorithms for probabilistic planning (e.g., (Majercik & Littman 1998)) do not scale to large domains. (Ng & Jordan 2000) approximates the optimal policy in POMDPs by using the underlying deterministic structure of the problem. They achieve this goal by sampling a look-ahead tree of deterministic executions of actions and sampling an initial state. In contrast, our algorithm generates deterministic sequences without sampling the initial state. (Bryce, Kambhampati, & Smith 2006) uses SMC to generate paths from the initial belief state to the goal with no observations. Using our action sampling algorithm results in a path which is closer to the optimal solution while it considers the effect of the observations.

2 Probabilistic Action Models

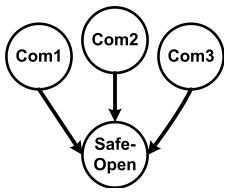
In this paper we address the problem of estimating the state of an agent given a sequence of probabilistic actions and observations in a *probabilistic action model*. We assume a prior distribution over the initial world states. Also, actions have probabilistic effects that are represented with a probability distribution over possible deterministic executions. The formal representation of a probabilistic action model is given as follows:

Definition 2.1 *A probabilistic action model is a tuple* $\langle \mathcal{X}, \mathcal{S}, P^0, \mathcal{A}, D\mathcal{A}, \mathcal{T}, P \rangle$.

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

Executing probabilistic action a has several deterministic outcomes; Each outcome is represented with a deterministic action. We specify $P(da|a, s)$ using logical cases $\psi_1 \dots \psi_k$ (mutually disjoint) for action a. When some state s satisfies ψ_i $(i \leq k)$ then $P(da|a, s) = P_i(da)$, where P_i is a probability distribution over different deterministic executions of action a corresponding to the logical case ψ_i.

Example 2.2 (safe) *Figure 1 presents a probabilistic action model for a domain in which an agent attempts to open a safe by trying several combinations.*



Action: (try-com1)
- (try-com1-succ): 0.8
  Pre: *safe-open*$\vee$*com1*
  Eff: *safe-open*
- (try-com1-fail): 0.2
  Pre: *safe-open*$\vee$*com1*
  Eff: $\neg$*safe-open*

Figure 1: *left*: BN representation for the prior distribution over states for the *safe* example. Com1 = $true$ means that combination 1 is a right one. *right*: Description of action (*try-com1*) for the logical case $\psi_1 = true$. The agent succeeds in opening the safe with probability 0.8 after executing (*try-com1*).

Figure 1, *left* presents the prior distribution P^0 over variables in the *safe* example with a Bayes net (BN). [1] We focus our presentation on probabilistic systems whose random variables are Boolean because they simplify our development. The representation can be generalized for discrete variables by encoding those variables in Boolean variables. We use the standard notation that Capital letters indicate variables and the corresponding script letters indicate particular values of those variables.

Figure 1, *right* shows the probabilistic action *(try-com1)* and its possible outcomes as deterministic actions *(try-com1-succ)* and *(try-com1-fail)*. Each deterministic action is described with a set of preconditions and effects. The preconditions and effects are represented with logical formulae over state variables. Executing an action only changes values of variables included in that action's effect (a.k.a. the Frame Assumption (McCarthy & Hayes 1969)). For instance, after executing action *(try-com1-succe)"* values of variables *"Com1", "Com2"*, and *"Com3"* do not change.

In partially observable domains, we update our knowledge as a result of executing an action and collecting observations in the resulting state. In a probabilistic action model, transition distribution P represents a distribution over possible outcomes of a probabilistic action. We do not introduce observations in the transition system. The observa-

[1] A Bayes net (Pearl 1988) is a directed acyclic graph whose nodes represent variables and arcs represent causal or probabilistic dependencies between variables. It encodes a joint probability distribution over world states using the conditional independence relationships.

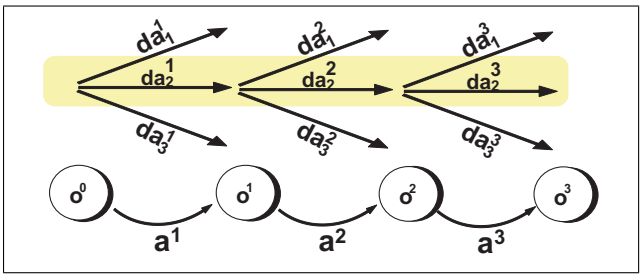


Figure 2: Sampling the logical particle $\langle da_2^1, da_2^2, da_2^3 \rangle$ given probabilistic sequence $\langle a^1, a^2, a^3 \rangle$. Each deterministic action da^t is sampled from distribution $P(da^t|a^t, da^{1:t-1}, o^{0:t-1})$.

tions 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 logical formula over state variables (e.g., "*safe-open*$\wedge$*com1*" shows an observation received at time t). When o^t is observed at time t, the logical formula o^t is true about the state of the world at time t. Note that throughout the paper superscripts for variables represent time.

3 Sampling Action Sequences

In this section we present our sampling algorithm for answering a query at time T in a probabilistic action model. The algorithm approximates the posterior probability of the query by sampling possible deterministic executions of the model. Then, it continues in a way that resembles the exact marginalization over those deterministic executions. The following equation shows the exact computation for the posterior probability of query φ^T given probabilistic action sequence $a^{1:T}$ and observations $o^{0:T}$ as $P(\varphi^T|a^{1:T}, o^{0:T})$.

$$P(\varphi^T|a^{1:T}, o^{0:T}) = \sum_i P(\varphi^T|\overrightarrow{DA_i}, o^{0:T}) P(\overrightarrow{DA_i}|a^{1:T}, o^{0:T-1}) \quad (1)$$

where, $\overrightarrow{DA_i}$ is one possible execution of the probabilistic sequence $a^{1:T}$, and does not depend on o^T.

The main (and first) step of our approximate algorithm is generating N samples (called *logical particles*) from all the possible executions of the given probabilistic sequence. The algorithm (described in Section 3.2 and illustrated in Figure 2) generates logical particle $\overrightarrow{DA_i}$ given the sequence of probabilistic actions $a^{1:T}$ and observations $o^{0:T-1}$ from the probability distribution $P(\overrightarrow{DA_i}|a^{1:T}, o^{0:T-1})$. The algorithm builds the logical particle incrementally by sampling each deterministic action in the sequence given the current probabilistic action, the previous deterministic actions in the sequence, and observations.

The next step of the algorithm computes the probability of query φ^T given the logical particle $\overrightarrow{DA_i}$ and observations $o^{0:T}$ as $P(\varphi^T|\overrightarrow{DA_i}, o^{0:T})$(described in Section 3.1).

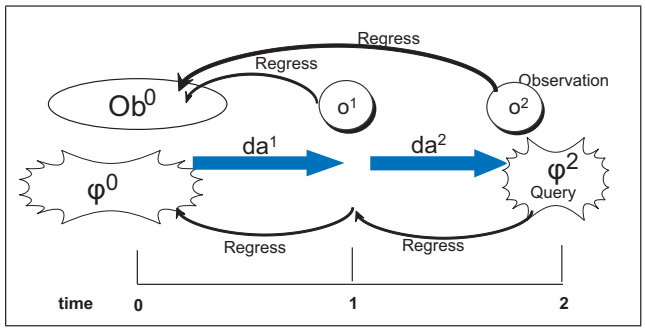


Figure 3: Regressing query formula φ^2 and observations $o^{0:2}$ to the initial time step given the logical particle $\langle da^1, da^2 \rangle$ (generated samples). $\varphi^0 = \text{Regress}(\varphi^2, da^{1:2})$ and $Ob^0 = \text{Regress}(o^{0:2}, da^{1:2})$.

Finally, the algorithm uses generated samples instead of the enumeration of $\overrightarrow{DA_i}$ in Equation (1), and computes the approximation for the posterior probability of query φ^T given the probabilistic sequence $a^{1:T}$ and observations $o^{0:T}$ as $\tilde{P}_N(\varphi^T|a^{1:T}, o^{0:T})$ by using the Monte Carlo Integration (Doucet, de Freitas, & Gordon 2001):

$$\tilde{P}_N(\varphi^T|a^{1:T}, o^{0:T}) = \frac{1}{N} \sum_i P(\varphi^T|\overrightarrow{DA_i}, o^{0:T}) \quad (2)$$

Details of each step of our Sample aCtion Approximate Inference Algorithm (SCAI, Figure 4) are explained next. We present the step of computing $P(\varphi^T|\overrightarrow{DA_i}, o^{0:T})$ first because it is used as a subroutine in the sampling step.

3.1 Computing $P(\varphi^t|\overrightarrow{DA}, o^{0:t})$

In this section we present Procedure LP-Posterior (Figure 4) that computes the probability of the query $P(\varphi^t|\overrightarrow{DA}, o^{0:t})$ given the logical particle $\overrightarrow{DA}$ and observations $o^{0:t}$. Its first step applies a logical *regression* subroutine (detailed below and illustrated in Figure 3) to the query, and as output returns a logical formula at time 0. The algorithm also regresses the observations, and also returns a logical formula at time 0. This can be done since the actions are deterministic. The algorithm's second step computes the prior probability of the regression of the query conditioned on the observations regressed by the logical particle; Recall that logical particle is a sampled sequence of deterministic actions.

Regressing a Formula Procedure **Regress** takes propositional formula φ^t and logical particle $\overrightarrow{DA}$, and returns as output another propositional formula φ^0. φ^0 represents the set of possible initial states, given that the final state satisfies φ^t, and the logical particle $\overrightarrow{DA}$ occurs. Thus, every state that satisfies φ^0 leads to a state satisfying φ^t after $\overrightarrow{DA}$ occurs. Regression of each observation o^t is defined similar to regression of formula φ^t since observations are also represented with logical formulae.

For deterministic action da^t and propositional formula φ^t, the *regression* of φ^t through da^t is a propositional formula

PROCEDURE SCAI($\varphi^T, a^{1:T}, o^{0:T}$)
Input: Probabilistic sequence $a^{1:T}$, Observations $o^{0:T}$, and Query φ^T
Output: $P(\varphi^T|a^{1:T}, o^{0:T})$
1. $Det = \{\overrightarrow{DA}_1, \ldots, \overrightarrow{DA}_N\} \leftarrow$ **Sample-Actions**$(a^{1:T}, o^{0:T-1})$
2. **for** each $\overrightarrow{DA}_i \in Det$, **LP-Posterior**$(\varphi^T, \overrightarrow{DA}_i, o^{0:T})$
3. **Return** $P(\varphi^T|a^{1:T}, o^{0:T})$ from Equation (6).

PROCEDURE Sample-Actions($a^{1:T}$, $o^{0:T-1}$)
Input: Probabilistic sequence $a^{1:T}$, and Observations $o^{0:T-1}$
Output: N logical particles $\overrightarrow{DA}_{1:N}$
1. **for** time $t = 1 : T$
   (a) **for** all ψ_{i,a^t}, **LP-Posterior**$(\psi^t_{i,a}, da^{1:t}, o^{0:t-1})$
   (b) Compute $P(da^t|a^t, da^{1:t-1}, o^{0:t-1})$ using Equation (5).
   (c) **for** i = 1 to N
      i. $da^t_i \leftarrow$ a sample from $P(da^t_i|a^t, da^{1:t-1}, o^{0:t-1})$
2. **return** $\overrightarrow{DA}_{1:N} \leftarrow \langle da^1 \ldots da^T \rangle_{1:N}$

PROCEDURE LP-Posterior(φ^t, $\overrightarrow{DA}$, $o^{0:t}$)
Input: Logical particle $\overrightarrow{DA}$, and Observations $o^{0:t}$
Output: $P(\varphi^t|\overrightarrow{DA}, o^{0:t})$
1. φ^0 = **Regress**$(\varphi^t, \overrightarrow{DA})$
2. Ob^0 = **Regress**$(o^{0:t}, \overrightarrow{DA})$
3. Compute **LP-Prior**$(\varphi^0|Ob^0)$.
return $P(\varphi^t|\overrightarrow{DA}, o^{0:t})$ as in Lemma 3.1.

PROCEDURE LP-Prior(φ^0, P^0)
Input: Formula φ^0, Graphical model prior P^0
Output: $P(\varphi^0)$
1. $\bigwedge_i C_i \leftarrow$ ConvertToCNF(φ^0)
2. Define indicator functions as Equation (3)
3. **return** $P(\varphi^0) \leftarrow$ Formula (4)

Figure 4: SCAI: Sample aCtion Approximate Inference algorithm for computing $P(\varphi^t|a^{1:t}, o^{0:t})$. **Boldface font** is used to denote subroutines.

φ^{t-1} such that state s^{t-1} satisfies φ^{t-1} iff the result of transition function $\mathcal{T}(s^{t-1}, da^t)$ satisfies φ^t. Computing the regression of φ^t through logical particle $\overrightarrow{DA}$ is done recursively:

$$Regress(\varphi^t, \langle da^1, ..., da^t \rangle) = Regress(Regress(\varphi^t, da^t), \langle da^1, ..., da^{t-1} \rangle).$$

Figure 5(a) shows regressing query φ^2 =*safe-open* through logical particle $DA = \langle$*try-com1-succ, try-com2-succ*$\rangle$. Similarly, the regression of observations $o^{0:t}$ is defined recursively:

$$Regress(o^{0:t}, \langle da^1, ..., da^t \rangle) = Regress(o^t, \langle da^1, ..., da^t \rangle) \wedge Regress(o^{0:t-1}, \langle da^1, ..., da^{t-1} \rangle).$$

Figure 5(a) shows regression of observations $o^{0:2} = \langle$*null, null,*$\neg$*com2*$\rangle$ through logical particle $\overrightarrow{DA} = \langle$*try-com1-succ, try-com2-succ*$\rangle$.

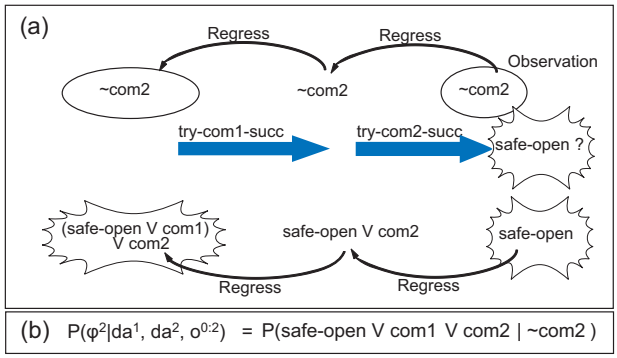


Figure 5: (a) Regressing query φ^2 =*safe-open* and observations $o^{0:2} = \langle$*null, null,*$\neg$*com2*$\rangle$ through logical particle $\overrightarrow{DA} = \langle$*try-com1-succ, try-com2-succ*$\rangle$ to φ^0 and O^0, respectively. (b) Computing $P(\varphi^2 =$ *safe-open*$|\overrightarrow{DA}, o^{0:2})$ using Lemma 3.1.

Algorithms for regression with deterministic actions (e.g., (Reiter 2001; Shahaf & Amir 2007)) work as follows: They maintain a logical formula for each variable x^t, and update it every time step, s.t. that formula is true if and only if x^t currently holds. They apply axiomatic descriptions (*successor-state axioms*) of the form

$$x^t \iff Precond^{t-1}(x, da^t)$$

for any action da^t's effect on any variable x, where $Precond^{t-1}(x, da^t)$ is a propositional formula over variables at time $t-1$. Simple techniques for regressing logical formula φ^t (e.g., (Reiter 2001)) replace every variable x^t in φ^t with $Precond^{t-1}(x, da^t)$. In our experiments (Section 4) we apply the algorithm of (Shahaf & Amir 2007) that takes linear time and representation space for t steps of regression.

We now summarize and show how to compute the posterior distribution $P(\varphi^t|\overrightarrow{DA}, o^{0:t})$ by applying regression. The algorithm first computes φ^0 and Ob^0 by regressing the query φ^t and observations $o^{0:t}$ through the logical particle $\overrightarrow{DA}$. It then computes $P(\varphi^t|\overrightarrow{DA}, o^{0:t})$ which is equal to $P(\varphi^0|Ob^0)$ as shown by the following lemma.

Lemma 3.1 *Let φ^t be a query and $o^{0:t}$ be observations. If $\varphi^0 = Regress(\varphi^t, \overrightarrow{DA})$ and $Ob^0 = Regress(o^{0:t}, \overrightarrow{DA})$, then*

$$P(\varphi^t|\overrightarrow{DA}, o^{0:t}) = P(\varphi^0|Ob^0).$$

PROOF We first use the Bayes rule and compute $P(\varphi^t|\overrightarrow{DA}, o^{0:t})$ as follows:

$$P(\varphi^t|\overrightarrow{DA}, o^{0:t}) = \frac{P(\varphi^t, o^{0:t}|\overrightarrow{DA})}{P(o^{0:t}|\overrightarrow{DA})}.$$

We then compute $P(\varphi^t, o^{0:t}|\overrightarrow{DA})$ by marginalizing over all the possible world states in the state space $\mathcal{S}$,

$P(\varphi^t, o^{0:t}|\overrightarrow{DA}) = \sum_{s\in\mathcal{S}} P(\varphi^t, o^{0:t}|s, \overrightarrow{DA})P(s|\overrightarrow{DA})$. For every world state $s \in \mathcal{S}$, $P(\varphi^t, o^{0:t}|s, \overrightarrow{DA}) = 1$ iff $s \models \varphi^0 \wedge Ob^0$ o.w. it is equal to 0. The reason is that executing sequence of deterministic actions $\overrightarrow{DA}$ in state s results in state s' that models φ^t and is consistent with observations $o^{0:t}$ iff $s \models \varphi^0 \wedge Ob^0$. Also, note that probability of s does not depend on the logical particle $\overrightarrow{DA}$. Hence,

$$P(\varphi^t, o^{0:t}|\overrightarrow{DA}) = \sum_{s\models\varphi^0\wedge Ob^0} P(s) = P(\varphi^0, Ob^0).$$

Same computations exist for $P(o^{0:t}|\overrightarrow{DA})$. Therefore,

$$\frac{P(\varphi^t, o^{0:t}|\overrightarrow{DA})}{P(o^{0:t}|\overrightarrow{DA})} = \frac{P(\varphi^0, Ob^0)}{P(Ob^0)} = P(\varphi^0|Ob^0). \quad \blacksquare$$

Figure 5(b) shows computing $P(\varphi^t|\overrightarrow{DA}, o^{0:t})$ using Lemma 3.1. Next section shows how to compute $P(\varphi^0|Ob^0)$ using the prior P^0. Note that φ^0 and Ob^0 are propositional formulae over state variables at time 0.

Computing Probability of the Initial Formula The general method for computing $P(\varphi^0)$ is summing over prior probabilities of all the states that satisfy φ^0. φ^0 can be any formula over state variables at time 0. There are many algorithms that include some heuristics for enumerating the world states satisfying the formula (e.g., (Bacchus, Dalmao, & Pitassi 2003)). Here, we describe an algorithm in Procedure LP-Prior (Figure 4) to compute $P(\varphi^0)$ given a graphical model prior P^0. The algorithm is efficient if the *underlying graph* of the Conjunctive Normal Form (CNF) [2] form of φ^0 has low *treewidth* (see (Robertson & Seymour ; Amir 2001)).

This algorithm first converts φ^0 to CNF, $\varphi^0 = \bigwedge_i C_i$, where each $C_i = \bigvee_j x_j$ is a clause over the state variables at time 0 or their negations. Then, it defines an indicator function, $\delta_{C_i}(x_{C_i})$, over all possible realizations of the variables x_{C_i} in the clause C_i. This way, we skip summing over probabilities of world states that do not satisfy φ^0.

$$\delta_{C_i}(x_{C_i}) = \begin{cases} 1 & x_{C_i} \models C_i \\ 0 & otherwise \end{cases} \quad (3)$$

Then, $P(\varphi^0)$ in the initial graphical model with potential functions $\Phi_j(x_{Cl_j})$ is:

$$P(\varphi^0) \propto \sum_{x\in\mathcal{X}} \prod_{x_{Cl_j}, x_{C_i}} \Phi_j(x_{Cl_j})\delta(x_{C_i}) \quad (4)$$

where, Cl_j is a clique in the graphical model, and the normalization factor is $\sum_{x\in\mathcal{X}} \prod_{x_{Cl_j}} \Phi_j(x_{Cl})$. We use the variable elimination algorithm (e.g., (Jordan 2006)) to compute the marginals in Formula (4) and in the normalization factor.

[2] The underlying graph of logical formula φ (in CNF) is a graph $G(V, E)$, where each vertex $v_i \in V$ corresponds to variable x_i, and each edge $e \in E$ between two vertices $v_i, v_j \in V$ indicates that the corresponding variables x_i, x_j appear in the same clause in φ.

3.2 Sampling Logical Particles

In this section we describe Procedure Sample-Actions (Figure 4) that generates N independent and identically distributed (i.i.d.) random samples (called *logical particles*) given a sequence of probabilistic actions and observations. Each logical particle is a possible execution of the given probabilistic sequence. The algorithm builds a logical particle incrementally by sampling each deterministic action in the sequence given the current probabilistic action, previous deterministic actions in the logical particle and observations. The details of the algorithm comes in the following.

The goal of the algorithm is to generate logical particle $\overrightarrow{DA} = \langle da^1, \ldots, da^T\rangle$ given probabilistic sequence $a^{1:T}$ and observations $o^{0:T-1}$. The algorithm generates samples from distribution $P(\overrightarrow{DA}|a^{1:T}, o^{0:T-1})$ (probability of executing $\overrightarrow{DA}$ as an outcome of sequence $a^{1:T}$); Recall that $\overrightarrow{DA}$ does not depend on the observation received at time T (last observation does not affect executing logical particle $\overrightarrow{DA}$).

$$P(\overrightarrow{DA}|\overrightarrow{A}, o^{0:T-1}) = P(da^1|a^1, o^0)\prod_t P(da^t|a^t, da^{1:t-1}, o^{0:t-1})$$

The above definition for $P(\overrightarrow{DA}|a^{1:T}, o^{0:T-1})$ allows iterative sampling of deterministic actions. Thus at each step t, the algorithm samples a deterministic action as an execution of the given probabilistic action from distribution $P(da^t|a^t, da^{1:t-1}, o^{0:t-1})$ which can be evaluated given prior P^0 and transition distribution $P(da^t|a^t, s^{t-1})(= P_i(da^t)$for $s^{t-1} \models \psi^{t-1}_{i,a^t})$:

$$\begin{aligned} &P(da^t|a^t, da^{1:t-1}, o^{0:t-1}) \\ &= \sum_{s^{t-1}} P(da^t|a^t, s^{t-1}) \cdot P(s^{t-1}|da^{1:t-1}, o^{0:t-1}) \\ &= \sum_i P_i(da^t) \cdot P(\psi^{t-1}_{i,a^t}|da^{1:t-1}, o^{0:t-1}). \end{aligned} \quad (5)$$

$P(da^t|a^t, s^{t-1})$ is derived directly from the transition distribution (Section 2). ψ^{t-1}_{i,a^t} is a logical formula over variables at time $t-1$, so we use Procedure LP-Posterior (Figure 4) to compute $P(\psi^{t-1}_{i,a^t}|da^{1:t-1}, o^{0:t-1})$. We first regress ψ^{t-1}_{i,a^t} and $o^{0:t-1}$ to time 0 with deterministic sequence $da^{1:t-1}$. We then use Procedure LP-Prior (Figure 4), and compute $P(\psi^0_{i,a^t})$ given the prior P^0. Therefore, at each step the algorithm samples N deterministic action $\{da^t_1, \ldots, da^t_N\}$ from $P(da^t|a^t, da^{1:t-1}, o^{0:t-1})$.

Figure 6 shows the process of sampling $\langle$*try-com1-succ, try-com2-succ*$\rangle$ given probabilistic sequence $\langle$*try-com1, try-com2*$\rangle$. Deterministic action da^1 = *try-com1-succ* is sampled with probability $P(da^1|a^1, o^0) = 0.8$, and da^2 = *try-com2-succ* is sampled with probability $P(da^2|a^1, da^1, o^{0:1}) = 0.8P($*safe-open* $\vee \neg$*com1*$)$. The probabilities are computed by applying logical case $\psi_1 = true$ in Equation (5).

As a special case, we assume that transition distribution is in the form of $P(da^t|a^t, s^{t-1}) = P(da^t|a^t)$, i.e. the probabilistic choice of a deterministic action depends only on the

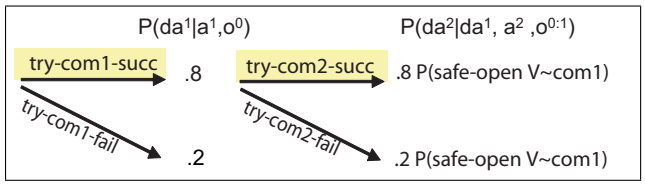


Figure 6: Sampling the logical particle ⟨*try-scom1-succ, try-com2-succ*⟩ given probabilistic sequence ⟨*try-com1, try-com2*⟩. $P(da^1 = \textit{try-com1-succ}|a^1, o^0) = 0.8$ and $P(da^2 = \textit{try-com2-succ}|a^2, da^1, o^{0:1}) = 0.8P(\textit{safe-open} \vee \neg \textit{com1})$.

probabilistic action and the current state of the agent. As a consequence, $P(da^t|a^t, da^{1:t-1}, o^{0:t-1}) = P(da^t|a^t)$. In this case the algorithm samples each deterministic action da^t independently from distribution $P(da^t|a^t)$ which is evaluated directly from transition distribution. Logical particle $\overrightarrow{DA} = \langle da^1, \ldots, da^T \rangle$ is an i.i.d. sample from distribution $P(\overrightarrow{DA}|a^{1:T})$ since $P(\overrightarrow{DA}|a^{1:T}) = \prod_t P(da^t|a^t)$.

3.3 Correctness, Accuracy and Complexity

Correctness The following theorem shows that output of our sampling algorithm, $\tilde{P}_N(\varphi^T|a^{1:T}, o^{0:T})$ converges to the exact posterior distribution $P(\varphi^T|a^{1:T}, o^{0:T})$.

Theorem 3.2 *Let φ^T be the query, $a^{1:T}$ be the given probabilistic sequence, $\overrightarrow{DA}_i$ be the logical particles, and $o^{0:T}$ be the observations. If $\varphi_i^0 = \textit{Regress}(\varphi^T, \overrightarrow{DA}_i)$ and $Ob_i^0 = \textit{Regress}(o^{0:T}, \overrightarrow{DA}_i)$, then*

$$\tilde{P}_N(\varphi^T|a^{1:T}, o^{0:T}) = \frac{1}{N} \sum_i P(\varphi_i^0|Ob_i^0) \qquad (6)$$

and

$$\tilde{P}_N(\varphi^T|a^{1:T}, o^{0:T}) \rightarrow_{N\rightarrow\infty} P(\varphi^T|a^{1:T}, o^{0:T}) \qquad (7)$$

PROOF The exact value for $P(\varphi^T|a^{1:T}, o^{0:T})$ is derived as Equation (1). Procedure Sample-Actions (Figure 4) generates samples (logical particles) from distribution $P(\overrightarrow{DA}|a^{1:T}, o^{0:T-1})$. Lemma 3.1 shows that $P(\varphi^T|\overrightarrow{DA}_i, o^{0:T}) = P(\varphi_i^0|Ob_i^0)$. Therefore, by using Monte Carlo Integration Equation (6) holds.

Moreover, because the variance of $P(\varphi^T|\overrightarrow{DA}_i, o^{0:T})$ is less than ∞, from the law of large numbers convergence, Formula (7), holds. ■

Accuracy We define a *metric* called *expected KL-distance* to evaluate the accuracy of our sampling algorithm SCAI (Figure 4). The expected KL-distance is defined as the expected value of all the KL-distances [3]between the exact distribution P and the approximation $\tilde{P}$ derived by SCAI, i.e. Expected-$\text{KL}_{\text{SCAI}} = \sum_{S_i} Pr_{\text{SCAI}}(S_i) KL(P, \tilde{P}_{S_i})$, where S_i is a set of N logical particles, and $Pr_{\text{SCAI}}(S_i)$ is the likelihood that SCAI generates set S_i. We then compare the accuracy of SCAI with the accuracy sequential Monte Carlo sampling algorithms (SMC) (Doucet, de Freitas, & Gordon 2001) based on the above metric.

SMC computes the posterior probability of a query by sampling sequences of states (called *particles*). It samples the initial states from the prior distribution $P^0(s)$; It then samples the next states in the sequence from the distribution $P(s'|s, a)$. In a probabilistic action model we compute $P(s'|s, a)$ from the transition function $\mathcal{T}$ and the transition distribution P in the model, i.e. $P(s'|s, a) = \sum_{\mathcal{T}(s,da_i)=s'} P(da_i|a, s)$. The following theorem shows that SCAI has higher accuracy than SMC for a fixed number of samples. The intuition is that each logical particle generated by SCAI covers many particles generated by SMC.

Theorem 3.3 *Let SCAI and SMC use N samples to approximate posterior distribution $P(\varphi^T|a^{1:T}, o^{0:T})$. Then, for a fixed N, Expected-$KL_{SCAI} \leq$ Expected-KL_{SMC}.*

PROOF We define a mapping f which maps each set of logical particles of SCAI to sets of particles of SMC. The mapping f is defined such that it covers all the possible sets of particles of SMC, and $f(z_i) \cap f(z_j) = \emptyset$ for two separate sets of logical particles $z_i \neq z_j$, and $Pr_{SCAI}(z) = Pr_{SMC}(f(z))$. We then prove that $\forall y \in f(z), KL(P, \tilde{P}_1^z) \leq KL(P, \tilde{P}_2^y)$ where $\tilde{P}_1$ and $\tilde{P}_2$ are approximations returned by SCAI and SMC, respectively. We build the mapping f as follows:

We map z (a set of N logical particles) to a set of particles y in $f(z)$. To do that, we map each logical particle $\langle da^1, da^2, \ldots \rangle \in z$ to some particles in the form of $(s^0, s^1, s^2, \ldots) \in f(z)$, where s^0 can be any of the world states at time 0, and the rest of states s^{t+1} are derived from the transition function $\mathcal{T}(s^t, da^t) = s^{t+1}$. This way we cover all the possible sets of particles of SMC. If two different deterministic actions da_i and da_j mapped to same transition (s, s'). Then, we change the mapping by assigning two different names s'_i and s'_j to state s', whereas $P(s'_i|s, a) = P(da_i|a, s)$ and $P(s'_j|s, a) = P(da_j|a, s)$. We then map da_i to (s, s'_i) and da_j to (s, s'_j). Therefore, $f(z_i)$ and $f(z_j)$ do not have intersection for two separate sets of logical particles z_i and z_j. Also, for every set of logical particles z, $Pr_{SCAI}(z) = \sum_{y \in f(z)} Pr_{SMC}(y)$.

We prove that $KL(P, \tilde{P}_1^z) \leq KL(P, \tilde{P}_2^y)$ by induction on the length of the sequence, t . Induction basis: SCAI returns the exact value for a sequence with length $t = 0$,i.e. $KL(P, \tilde{P}_1^z) = 0$. Induction step: In SCAI $\tilde{P}_1(s^{t+1}) = \sum_{s^t} \tilde{P}_1(s^t)\tilde{P}_1(da^{t+1}|a^{t+1}, s^t)$, and in SMC $\tilde{P}_2(s^{t+1}) = \sum_{s^t} \tilde{P}_2(s^t)\tilde{P}_2(s^{t+1}|a^t, s^t)$. Each transition (s^t, s^{t+1}) is covered by deterministic action da^{t+1}. Also, by induction assumption we know that $KL(P, \tilde{P}_1^z(s^t)) \leq KL(P, \tilde{P}_2^y(s^t))$. Therefore, $KL(P, \tilde{P}_1^z(s^{t+1})) \leq KL(P, \tilde{P}_2^y(s^{t+1}))$.

[3] $KL(P, \tilde{P}) = \sum_x P(x) log(P(x)/\tilde{P}(x))$. $KL(P, \tilde{P}) = 0$if $P = \tilde{P}$

Hence,

$$KL(P, \tilde{P}_1^z) Pr_{SCAI}(z) \leq \sum_{y \in f(z)} KL(P, \tilde{P}_2^y) Pr_{SMC}(y). \quad \blacksquare$$

Complexity Running time of our algorithm SCAI (Figure 4) is $O(N \cdot T \cdot (T_{\text{Regress}} + T_{\text{LP-Prior}}))$, where N is the number of samples, and T is the length of the given sequence of probabilistic actions.

Tractability of SCAI results from tractability of the underlying algorithms for Regress and LP-Prior. SCAI is exact when the given action sequence is deterministic. An exact algorithm also exists to compute the full joint posterior distribution given a sequence of deterministic 1:1 actions (i.e., 1:1 mappings between world states). This algorithm uses logical filtering (Amir & Russell 2003) as a subroutine. The algorithm is efficient if its underlying subroutine of logical filtering is efficient. An example of 1:1 actions is *flipping* a light switch, but turning on the light is not a 1:1 action.

4 Empirical Results

We implemented our algorithm SCAI (Figure 4) for the case that transition distribution is in form of $P(da|a, s) = P(da|a)$. Our algorithm takes advantage of a different structure than that available in DBNs. Therefore, we focused on planning-type structures, and tested our implementation in planning domains: *Safe, Homeowner, Depots, and Ferry* taken from domains in International Planning Competition at AIPS-98 and AIPS-02. [4] These domains are deterministic, so we modified them to include a probability distribution over the outcomes of actions. For example, for action *(try-com1)* in the *safe* domain we considered two possible executions *(try-com1-succ)* and *(try-com1-fail)* as in Figure 1.

We compared the accuracy of our SCAI with SMC algorithm by approximating the expected KL-distance as follows: We built the transition model $P(s'|s, a)$ for SMC from the transition distribution over the actions $P(da|a, s)$ (as explained in Section 3.3). We then built a DBN over the state variables, and ran the junction tree algorithm for DBNs (Murphy 2002) to compute the exact posterior probability of the query. We then ran SCAI and SMC for a fixed number of samples, approximate the distribution and compute the KL-distance between their approximation and the exact posterior. We iterated this process for at least 50 times, and find an average over these derived KL-distances.

We ran SCAI and SMC for the *safe* domain with different number of variables (8, 9, and 10) and different random sequences with lengths (10, 25, and 50) for the query *safe-open*. For longer sequences and more variables we did not have the exact posterior to compare with since the implementation for the exact algorithm crashes (runs out of memory). As Figure 7 shows, with the increase in the number of variables and the sequence lengths the expected KL-distance for SCAI remains lower than SMC for a fixed number of samples.

[4] Also available from:
ftp://ftp.cs.yale.edu/pub/mcdermott/domains/
http://planning.cis.strath.ac.uk/competition/domains.html

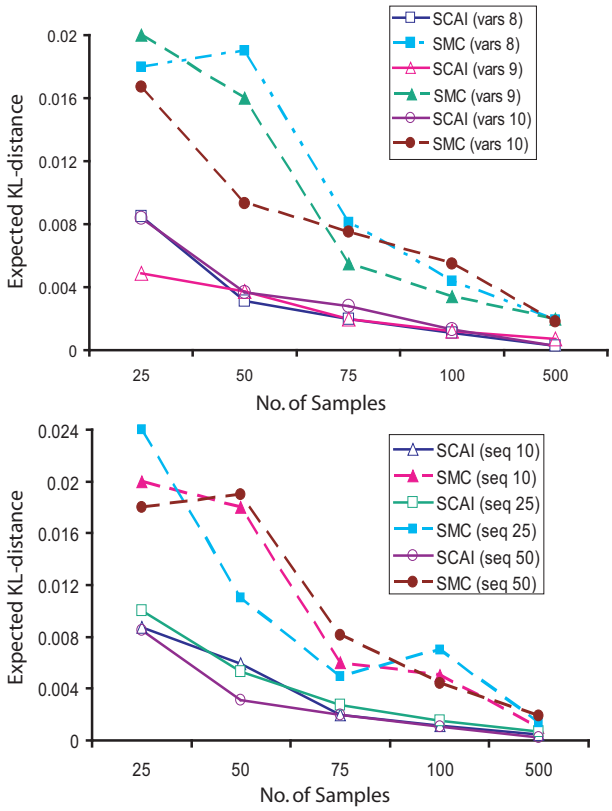


Figure 7: Expected KL-distance of SCAI and SMC with the exact distribution vs. number of samples for the *safe* example (*top*) For a sequence with length 50 with 8, 9, and 10 variables. (*bottom*) For 8 variables for random sequences with lengths 10, 25, and 50.

We report the experiments on the other domains (*Depots*, *Homeowner*, and *Ferry*) in Figure 8. For all the experiments except one the expected KL-distance of SCAI is 2 or 3 times less than that of SMC. The expected KL-distances for SCAI and SMC are almost the same for the *Homeowner* domain with 4 variables and a sequence with length 100. The reason is that the posterior distribution converges to the stationary distribution after 100 transitions for this small number of variables. But, in larger domains our SCAI returns more accurate results.

5 Conclusion and Future Work

In this paper we presented a sampling algorithm to compute the posterior probability of a query at time t given a sequence of probabilistic actions and observations. We proved and showed that for a fixed number of samples, it achieves higher accuracy compared to SMC sampling techniques.

One criticism of our algorithm is that for long sequences probability of the query given logical particles can be 0. A future work is adding a resampling step to overcome the

Number of Samples		50	100	500
Depots: seq(50)	SCAI	0.007	0.004	0.0006
	SMC	0.017	0.007	0.0010
Depots: seq(100)	SCAI	0.010	0.003	0.0006
	SMC	0.014	0.005	0.0010
Homeowner: seq(10)	SCAI	0.011	0.001	0.0005
	SMC	0.069	0.004	0.0008
Homeowner: seq(100)	SCAI	0.010	0.004	0.0008
	SMC	0.011	0.005	0.0010
Ferry: seq(10)	SCAI	0.004	0.001	0.0005
	SMC	0.01	0.003	0.0009

Figure 8: Expected KL-distance derived for our SCAI and SMC in domains *Depots* (9 variables), *Homeowner*(4 variables), and *Ferry* (6 variables) with different sequence lengths.

problem of increasing variance. Intuitively, our algorithm needs fewer resampling steps than SMC. It can be proved by a method similar to the proof of Theorem 3.3, knowing that each logical particle covers many samples in SMC.

There are several directions that we can continue this work. One direction is to use this algorithm for an approximate conformant probabilistic planning problem. We sample the logical particles as paths to the goal, regress the query with them, and find an approximation for the best plan. The other direction is finding a more efficient exact algorithm for computing the probability of logical formulae at time 0, or use an approximation method. The other direction can be extending the algorithm to continuous domains (real value random variables). The generalization can be done by discretizing the real value variables or by combining with RBPF (Rao-Blackwellised Particle Filtering) (Doucet *et al.* 2000).

Acknowledgements We would like to thank Leslie P. Kaelbling and the anonymous reviewers for their helpful comments. This work was supported by Army CERL grants DACA420-1-D-004-0014 and W9132T-06-P-0068 and Defense Advanced Research Projects Agency (DARPA) grant HR0011-05-1-0040.

References

Amir, E., and Russell, S. 2003. Logical filtering. In *Proc. Eighteenth International Joint Conference on Artificial Intelligence (IJCAI '03)*, 75–82. Morgan Kaufmann.

Amir, E. 2001. Efficient approximation for triangulation of minimum treewidth. In *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI '01)*, 7–15. Morgan Kaufmann.

Bacchus, F.; Dalmao, S.; and Pitassi, T. 2003. Algorithms and complexity results for SAT and Bayesian inference. In *Proc. 44st IEEE Symp. on Foundations of Computer Science (FOCS'03)*, 340–351.

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.

Bryce, D.; Kambhampati, S.; and Smith, D. 2006. Sequential monte carlo for probabilistic planning reachability heuristics. In *Proceedings of the 16th Int'l Conf. on Automated Planning and Scheduling (ICAPS'06)*.

Dean, T., and Kanazawa, K. 1988. Probabilistic temporal reasoning. In *Proc. National Conference on Artificial Intelligence (AAAI '88)*, 524–528. AAAI Press.

Doucet, A.; de Freitas, N.; Murphy, K.; and Russell, S. 2000. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the conference on Uncertainty in Artificial Intelligence (UAI)*.

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

Jordan, M. I.; Ghahramani, Z.; Jaakkola, T.; and Saul, L. K. 1999. An introduction to variational methods for graphical models. *Machine Learning* 37(2):183–233.

Jordan, M. 2006. *Introduction to probabilistic graphical models*. Forthcoming.

Kjaerulff, U. 1992. A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, 121–129.

Majercik, S., and Littman, M. 1998. Maxplan: A new approach to probabilistic planning. In *Proceedings of the 5th Int'l Conf. on AI Planning and Scheduling (AIPS'98)*.

Mateus, P.; Pacheco, A.; Pinto, J.; Sernadas, A.; and Sernadas, C. 2001. Probabilistic situation calculus. *Annals of Mathematics and Artificial Intelligence*.

McCarthy, J., and Hayes, P. J. 1969. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. Edinburgh University Press. 463–502.

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

Ng, A. Y., and Jordan, M. 2000. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proc. Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI '00)*, 406–415. Morgan Kaufmann.

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

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2):257–285.

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

Robertson, N., and Seymour, P. D. Graph minors XVI, wagner's conjecture. To appear.

Shahaf, D., and Amir, E. 2007. Logical circuit filtering. In *Proc. Nineteenth International Joint Conference on Artificial Intelligence (IJCAI '07)*.