Logical Particle Filtering

Luke S. Zettlemoyer, Hanna M. Pasula, and Leslie Pack Kaelbling

MIT CSAIL
{lsz,pasula,lpk}@csail.mit.edu

Abstract. In this paper, we consider the problem of filtering in relational hidden Markov models. We present a compact representation for such models and an associated logical particle filtering algorithm. Each particle contains a logical formula that describes a set of states. The algorithm updates the formulae as new observations are received. Since a single particle tracks many states, this filter can be more accurate than a traditional particle filter in high dimensional state spaces, as we demonstrate in experiments.

Consider an agent operating in a complex environment, made up of an unknown, possibly infinite, number of objects. The agent can take actions and make observations of the state of the world, and it knows a probabilistic model of how the state changes over time as a result of its actions and of how the observations are generated from the states. How can it efficiently estimate the underlying state of the environment? Filtering is the problem of predicting a distribution over the underlying environment state given a history of the agent's actions and observations. This problem is pervasive in AI: a dialogue system has to estimate the belief state of the user; an office-assistant must track the states and relationships among people, meetings, and projects; a household robot has to track the locations of furniture, the state of the dishes in the dishwasher, and the desires of the humans in its house.

At their root, these problems are controlled hidden Markov models (HMMs) or POMDPs. The standard techniques for filtering in such models require enumeration of the individual states of the environment. This quickly becomes intractable, and is impossible in infinite worlds. Particle filtering methods [1] make approximations by representing a small set of likely states. They can be executed online with constant computation per time step and can be used to track arbitrary state spaces, but reliable estimates in large domains require a very large number of particles. This problem can often be ameliorated by using Rao-Blackwellization, in which the filtering distribution is decomposed into two factors, one that is sampled and one that is computed exactly. Rao-Blackwellization has been effectively applied in both propositional [2] and relational [3] state representations. In both cases, however, a finite universe of objects must be known in advance.

Quantified logical expressions are a powerful method for using short descriptions to name large (possibly infinite) sets of states. When the underlying model of uncertainty in the domain is nondeterministic rather than probabilistic

(that is, there is a set of environment states that are consistent with the action/observation stream at any point in time), logical expressions can be used as the basis of algorithms [4] that track effectively in large state spaces. However, it is not obvious how to use them in probabilistic tracking problems. Recent work explored MCMC sampling over relational structures [5], lifted probabilistic inference [6], and exact inference in relational HMMs [7] but we do not know of relational sampling techniques for probabilistic filtering.

In this paper, we combine the ability of particle filters to focus only on the most likely underlying states with the ability of logical expressions to tractably name large sets of complex states. We develop an online, logical particle filtering algorithm that maintains a set of quantified logical formula or *hypotheses*, each of which potentially describes a large or infinite set of states. The hypotheses are built up incrementally, making discriminations only as needed to track the results of the agent's actions and incorporate the information from its observations. Aspects of the environment about which the agent gets no information are not explicitly represented. As a result, we get a robust and efficient filter whose complexity is driven by the information content of the agent's actions and observations.

As a running example, we will consider an idealized robot that needs to map an environment with topological structure, such as a sewer or street grid, which is made up of a set of interconnected locations. Fig. 1(a) shows six possible worlds, where solid lines indicate walls, dotted lines represent halls, and the circle marks the robot's location. At time zero, the robot does not know the structure or size of its environment, but it does have a prior distribution over how many locations are possible and how they might be connected. It gets local observations indicating whether there are locations next to it, but these observations can be noisy: the robot might fail to see some openings. It can also execute actions by trying, sometimes unsuccessfully, to move to neighboring locations. Although this problem seems simple, it presents a challenging filtering task because there are infinitely many possible mazes.

1   Problem and Representation

We will now define a filtering problem based on a logical representation of states, actions, transitions, and observations. This representation is inspired by a similar one previously used for probabilistic rule learning in the fully observable setting [8]. Let $\mathbf{s}$ be the (possibly countably infinite) set of possible states, and $\mathbf{o}$ be the finite set of possible observations. Then let $s_i \in \mathbf{s}$ and $o_i \in \mathbf{o}$ be a specific state and a specific observation at time i, $s_{0:t}$ be a sequence $\{s_0, \ldots, s_t\}$ of $t+1$ states, and $o_{1:t}$ be a sequence $\{o_1, \ldots, o_t\}$ of t observations. The filtering problem is to compute the distribution $p(s_t|o_{1:t})$ for a sequence of time steps $t = 1 \ldots T$. The dynamics of the world are

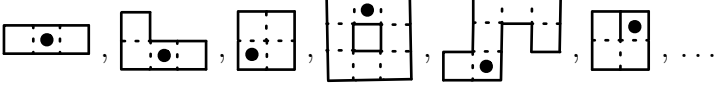$$p(s_{0:t}, o_{1:t}) = p(s_0) \prod_{i=1}^{t} p(s_i|s_{i-1})p(o_i|s_i) \ \ .$$

Fig. 1. Maze world representation: states, transition rules, observation rules, and hypotheses.

where $p(s_{0:t}, o_{1:t})$ is a joint distribution over state and observation sequences.[1]

1.1 States, Observations, and Actions

A state is represented by a conjunctive formula with constants denoting objects in the world and proposition and function symbols representing the objects' properties and relations.[2] For example, Fig. 1(b) shows a maze world state with two locations, where the robot is at l_0, and there is a location to the left, l_1. Each state is a full description of the world where each constant names a unique object. Observations are constructed using a set of n propositions: each observation can include truth values for some subset of the propositions. For example, in the maze world, the propositions are *hall-left*, *hall-right*, *hall-up* and *hall-down*. An observation might be {¬*hall-right*, *hall-left*}. Finally, we assume a finite, unparameterized, set of actions **a**. In the maze world, the actions are *go-left*, *go-right*, *go-up*, and *go-down*.

---

[1] In this paper, we actually use the model $p(s_{0:t}, o_{1:t}|a_{1:t}) = p(s_0) \prod_{i=1}^{t} p(s_i|s_{i-1}, a_i) p(o_i|s_i, a_i)$ for fixed action sequences $a_{1:t}$. To simplify notation, we drop $a_{1:t}$ when it is not directly relevant to the discussion.

[2] Our algorithm never explicitly represents fully ramified states; instead it uses logical formulae describing state sets.

1.2   Transition Rules

The transition distribution $p(s_t|s_{t-1}, a_t)$ is defined using a set of rules. Each rule has a set of applicability conditions and a distribution over a set of changes to the previous world state. Fig. 1(c) shows two transition rules. The first rule models the situation where the robot is at a location with an accessible location to the left of it, and attempts to move left. There is a high probability that it will move to the new location, but it might stay where it was. The second rule models the case where there is no accessible location to the left. Here the robot will never move. In a full transition model for the maze world, similar rules are defined for each direction the robot can move.

Each rule has three parts that determine when it is applicable: an action, a set of *deictic references* defining variables, and a context that encodes a set of preconditions. Both of these rules in Fig. 1(c) model the *go-left* action. The first has two deictic references and the second has one. Each reference is defined by a single functional term that specifies how to bind the variable in a state s_{t-1}. For example, the deictic reference $X : at = X$, specifies that X binds to the value of the *at* function, or fails to bind if this value is *null*. Given a particular state s_{t-1}, we can determine whether a rule *applies* by computing a binding θ that finds objects for all the variables and then tests whether the preconditions hold for this binding. For example, in the state in Fig. 1(b), the first rule would have the binding $\theta = \{X/l_0, Y/l_1\}$ and the second rule would have the binding $\theta = \{X/l_0\}$. The first rule would apply, since it has no preconditions in the context, while the second one would not because $left(l_0) = l_1$ instead of *null*.

Multiple rules can apply to a single state, as long as they do not conflict. To discover conflicts, we take each of the applicable rules and construct the set of ground terms it can affect by applying the binding to all the terms in the rule's outcomes. If any term is found that can be affected by more than one rule, all the relevant rules are marked as conflicting and ignored.

Given a set of non-conflicting applicable rules, the distribution over outcomes, described on the right of the $\rightarrow$, defines what changes may happen from s_{t-1} to s_t. Each outcome describes a set of conditions that might be true at time t. For example, the first outcome of the first rule in Fig. 1(c) specifies that $at = Y$ will be true for the current value of Y in θ, with probability 0.9. To compute $p(s_t|s_{t-1}, a_t)$ we must sum all of the ways s_t can be constructed from s_{t-1} given the applicable rules. Because we assume a small number of rules with a small number of outcomes, this computation is tractable.

1.3   Observation Rules

The observation distribution $p(o_t|s_t, a_t)$ is also defined using a set of rules. The preconditions for an observation rule work like those of a transition rule, but the effects are described in the observation language. To generate an observation, we assume that nothing is observed by default and only add the propositions in the outcomes sampled from the applicable rules. To compute the probability of an observation, we sum all ways of generating it from the rules, as we did with the

transition rules. For example, the rule in Fig. 1(d) models the situation where the robot is at a location that has an accessible location to the left of it. With high probability the robot will receive an observation *hall-left*, but sometimes it will not see the hall. This rule uses the special $*$ symbol to indicate that it applies for any action. The full observation model includes additional rules for the other directions.

1.4   Hypotheses

Hypotheses are logical formulae h_t that describe sets of states. Each h_t is a first-order sentence consisting of existentially quantified variables that are constrained to not equal each other, and a conjunction of literals formed with these variables. For example, the hypothesis in Fig. 1(e) represents all of the (potentially infinitely many) states where the robot is at a location with a wall above it. The hypothesis in Fig. 1(f) describes states where there is a horizontal hall of length at least two and the robot is in the right location.

Representing sets of world states this way allows us to compute the transition probabilities $p(h_t|h_{t-1}, a_t)$ and observation probabilities $p(o_t|h_t, a_t)$ efficiently for a wide range of hypotheses. For example, the hypothesis in Fig. 1(f) contains enough information to determine the effects of the *go-left* action. No matter which $s_t \in h_t$ we consider, the robot will always either stay at the location V_1 describes, or move to the location named by V_2 with known probability. The update would be more difficult if the robot instead executed *go-right*. In order to predict this action's effects, we would need to know whether there is a location to the right. This location might exist in some, but not all states $s_t \in h_t$.

More formally, we say that a rule r has *uniform application* to a hypothesis h if it applies either to all $s \in h$, or to none of them. This can be checked by directly computing application to the hypothesis, like we did for states above. If any of the necessary information about the deictic referents or precondition truth values is missing, r applies to some states in h and not others. For example, in Fig. 1(c), the first *go-left* rule has uniform application to the hypothesis in Fig. 1(f) under the binding $\theta = \{X/V_1, Y/V_2\}$, while a similar rule for the *go-right* action does not because we cannot find a binding that satisfies the preconditions, or guarantee that one does not exist. Given a rule set R, if each rule $r \in R$ has uniform application to h, we say that h is *specific enough* for R. When a hypothesis is specific enough, transition and observation probabilities can be computed by collecting applicable rules and summing over outcomes, just as for a specific s_t.

1.5   The Prior

For standard particle filters, the prior $p(s_0)$ is only used to sample initial states s_0. However, for the logical particle filter, we will need to be able to compute $p(h) = \sum_{s_0 \in h} p(s_0)$ for a wide range of possible hypotheses. If we assume a

relatively simple prior, this can be done efficiently.[3] For example, the prior for the maze world first selects the size of a grid of locations according to a distribution and then, for each pair of adjacent locations, makes an independent probabilistic decision as to whether they are connected by a hall. Finally, the robot is placed in the center location. Thus, we can compute the probability of any particular initial state by summing the probabilities of the grid sizes that are big enough to have generated the world times the probabilities of the existence of the world's halls. Since hypotheses are just partial state descriptions, their probabilities are calculated analogously. In both cases, because the probability of a hall is independent of the size of the maze, if we constrain the distribution over maze sizes to be integrable in closed form, the computations will be efficient with time that does not depend on the number of possible states, which is essential for infinite state spaces.

2   Probabilistic Logical Filtering

Before we describe the logical particle filter, we will present an algorithm that can compute $p(s_{0:t}|o_{1:t})$ exactly.[4] The algorithm works by constructing a partition $H_{0:t}$. Each $h_{0:t} \in H_{0:t}$ is a sequence of t hypotheses, where each hypothesis h_t describes a set of states. Thus, $p(h_{0:t}) = \sum_{s_{0:t} \in h_{0:t}} p(s_{0:t})$, and $p(s_{0:t}|h_{0:t}) = p(s_{0:t})/p(h_{0:t})$ if $s_{0:t} \in h_{0:t}$ and 0 otherwise. These definitions allow us to compute

$$p(s_{0:t}|o_{1:t}) = \sum_{h_{0:t} \in H_{0:t}} p(s_{0:t}|h_{0:t})p(h_{0:t}|o_{1:t}) \tag{1}$$

where $p(s_{0:t}|h_{0:t}, o_{1:t})$ is simplified to $p(s_{0:t}|h_{0:t})$ because the hypotheses are constructed so that the $s_{0:t}$ are conditionally independent of $o_{1:t}$ given $h_{0:t}$, as we will see shortly. We will also ensure that, for each $h_{0:t}$, every $s_{0:t} \in h_{0:t}$ will have the same transition and observation probabilities. This property will enable us to compute $p(s_{0:t}|h_{0:t})$ and $p(h_{0:t}|o_{1:t})$ efficiently. As a running example, we use a further simplified maze world with only one direction *left*, one observation *hall-left*, and one action *go-left*. The robot's task is to estimate the length of the corridor and its current location.

   The partition $H_{0:t}$ is defined recursively by, at each time step, taking each $h_{0:t-1} \in H_{0:t-1}$ and using it to construct an H_t. Each $h_t \in H_t$ is added to the corresponding $h_{0:t-1}$ to create an $h_{0:t}$. We always start with an initial hypothesis set, H_0, that contains the single hypothesis $h_0 = true$. The process for computing a new H_t given an h_{t-1} has three steps. It makes use of the *Specialize* procedure, described later in this section, which takes a hypothesis h and a rule set R and partitions h into a set of new hypotheses which are each specific enough for

---

[3] Exploring more complex priors is an important area for future work. One possible approach would be to use general purpose first-order probabilistic models and inference algorithms (see, for example, BLOG [9].)

[4] We focus on computing $p(s_{0:t}|o_{1:t})$ instead of the filtering distribution $p(s_t|o_{1:t})$ because it is easier to describe sampling techniques for this problem. The same sampling techniques can also estimate the marginal $p(s_t|o_{1:t})$.

R. This procedure is used repeatedly to ensure we can compute transition and observation probabilities for hypotheses as they are constructed.

Step 1: Transition Specialize First, we specialize each h_{t-1}, by splitting it into a set of mutually exclusive hypotheses $^{\bullet}H_{t-1}$ so that each new hypothesis is specific enough for the relevant transition rules. For example, with the *go-left* action, h_0 would become

$$^{\bullet}h_0^1 : \exists x.at = x \wedge left(x) = null$$
$$^{\bullet}h_0^2 : \exists x, y.x \neq y \wedge at = x \wedge left(x) = y$$

where $^{\bullet}h_0^2$ describes the set of states with an accessible location to the left of the current location and $^{\bullet}h_0^1$ the ones without such a location. We can compute the probability of each of these hypotheses using the prior, for example $p(^{\bullet}h_0^2)$ is the sum of the probabilities of the initial states that have a location to the left.

Step 2: Transition Now that we have specialized, we can compute all of the possible transitions efficiently. To do this, we create a new set of hypotheses $^{\circ}H_t$ for the subsequent time t that model all the possible action effects in all of the $^{\bullet}h_{t-1}$. In our example, $^{\circ}H_1$ is:

$$^{\circ}h_1^1 : \exists x.at = x \wedge left(x) = null$$
$$^{\circ}h_1^2 : \exists x, y.x \neq y \wedge at = x \wedge left(x) = y$$
$$^{\circ}h_1^3 : \exists x, y.x \neq y \wedge at = y \wedge left(x) = y$$

where $^{\circ}h_1^1$ models the fact that the *go-left* action would not have changed any aspects of the states described by $^{\bullet}h_0^1$, because there is no location to move to. $^{\circ}h_1^3$ represents the effects of successfully moving from the states in $^{\bullet}h_0^2$, while $^{\circ}h_1^2$ describes failure from those same states. Notice that, after the transition, we still have a partition of the possible state sequences, because the effects of the action are mutually exclusive. For example, in this case, either the robot moves, or it does not. We can also compute the joint probability of the hypothesis sequences by multiplying the transition likelihoods by the prior probabilities from the last step. For example, $p(^{\circ}h_1^3, ^{\bullet}h_0^2) = p(^{\bullet}h_0^2)p(^{\circ}h_1^3|^{\bullet}h_0^2) = p(^{\bullet}h_0^2) \cdot 0.9$, where 0.9 is the probability that the robot successfully moves.

Step 3: Observation Specialize The final step, which yields H_t, is another specialization which ensures that the hypotheses are specific enough for the observation rules. In our example, given the observation $o_1 = \neg hall\text{-}left$, we would create the following H_1:

$$h_1^1 : \exists x.at = x \wedge left(x) = null$$
$$h_1^2 : \exists x, y.x \neq y \wedge at = x \wedge left(x) = y$$
$$h_1^3 : \exists x, y.x \neq y \wedge at = y \wedge left(x) = y \wedge left(y) = null$$
$$h_1^4 : \exists x, y, z.x \neq y \neq z \wedge at = y \wedge left(x) = y \wedge left(y) = z$$

where we split $^\circ h_1^3$ into h_1^3 and h_1^4 so that each hypothesis specifies whether there is a location to the left. We can also now compute the joint probability of the new hypothesis and the observation. For example, $p(h_1^4, o_1, {}^\bullet h_0^2)$ is $p(o_1|h_1^4)p(h_1^4|^\circ h_1^3)p(^\circ h_1^3, {}^\bullet h_0^2)$ where $p(^\circ h_1^3, {}^\bullet h_0^2)$ is defined above, $p(o_1|h_1^4)$ is the observation probability (in this case 0.1) and $p(h_1^4|^\circ h_1^3)$ is the specialization probability, which will be described shortly.

As the robot acts in the world, we repeatedly perform the same three-step process for each time step.

2.1   The distributions

As we build $H_{0:t}$, we compute $p(s_{0:t}|h_{0:t})$ and $p(h_{0:t}|o_{1:t})$ as follows. The distribution

$$p(h_{0:t}|o_{1:t}) = \prod_t p(h_t|o_t, h_{t-1})p(h_0)$$

is built incrementally by computing $p(h_t|o_t, h_{t-1})$ for each h_t. We do this by first computing $p(h_t, o_t|h_{t-1})$ and then the conditional $p(h_t|o_t, h_{t-1})$ by dividing by $p(o_t|h_{t-1}) = \sum_{h_t} p(h_t, o_t|h_{t-1})$. The distribution $p(h_t, o_t|h_{t-1})$ can be computed directly by multiplying together the probabilities of the steps used to construct h_t, as we saw in the example above for the specific case of $p(h_1^4, o_1|^\bullet h_0^2)p(^\bullet h_0^2)$.

We also need to compute $p(s_{0:t}|h_{0:t})$. For $s_{0:t} \in h_{0:t}$, $p(s_{0:t}|h_{0:t}) =$

$$\frac{p(s_0)\prod_t p(s_t|s_{t-1})}{\sum_{s'_{0:t}\in h_{0:t}} p(s'_0)\prod_t p(s'_t|s'_{t-1})} = \frac{p(s_0)}{\sum_{s'_{0:t}\in h_{0:t}} p(s'_0)}$$

where the products cancel because all of the $s'_{0:t}$ described by $h_{0:t}$, including $s_{0:t}$, must have the same transition probabilities. Now, the summation $\sum_{s'_{0:t}\in h_{0:t}} p(s'_0)$ is a single computation with complexity independent of the time step t.

This same trick can be used for the specialization probabilities from before. Whenever we want to replace a hypothesis h_t with a more specific hypothesis h'_t, we must compute the specialization probability $p(h'_{0:t}|h_{0:t})$. We can write out the products, like we did above, and the transition probabilities will cancel yielding

$$p(h'_{0:t}|h_{0:t}) = \frac{\sum_{s_{0:t}\in h'_{0:t}} p(s_0)}{\sum_{s_{0:t}\in h_{0:t}} p(s_0)} \quad ,$$

which is two computations with the prior.

2.2   Specialization

Fig. 2 shows the *Specialize* algorithm which splits a hypothesis into new hypotheses that are specific enough for a rule set. There are two reasons why *Specialize* might split a hypothesis. First, we might not be able to prove that a deictic reference binds. Second, we might not be able to prove whether the preconditions hold. In both of these cases, the algorithm creates a set of new, mutually exclusive hypotheses, each of which has a different referent for the unknown deictic

---

Specialize(**Inputs:** Rule set R, Hypothesis h)
  Initialize $H = \{h\}$
  For each $r \in R$ and
      $h = \exists v_0, \ldots, v_n. \;\; v_0 \neq \ldots \neq v_n \wedge p_1 \wedge \ldots \wedge p_n \in H$
    For each deictic reference $v_i : d = v_i \in r$
      If v_i does not have a unique reference in h
        Remove h from H
        Add each $\{h \wedge d = v_0, \ldots, h \wedge d = v_n\}$ to H
        Add $h \wedge d = null$ to H
        Add $\exists v_0, \ldots, v_n, v_{n+1}. \;\; v_0 \neq \ldots \neq v_n \neq v_{n+1}$
                  $\wedge p_1 \wedge \ldots \wedge p_n \wedge d = v_{n+1}$ to H
    For each context literal $p \in r$
      If truth of p is undefined in h
        Remove h from H
        Add $h \wedge p$ and $h \wedge \neg p$ to H
  Return H

Fig. 2. The *Specialize* algorithm.

reference or a different truth value for the unknown precondition literal. In ${}^\bullet h_0^1$ from step 1 in our example above, the variable x was added as a referent for the deictic reference that names the location of the robot in the *go-left* transition rule and $left(x) = null$ was added to satisfy a precondition.

2.3   Discussion

If the size of the partition $H_{0:t}$ is small, this exact algorithm is efficient: the sum in Eq. 1 involves a small number of terms, and each term can be computed efficiently. The size of $H_{0:t}$ can never exceed the number of possible state sequences, since it partitions this space. Its rate of growth is determined by the structure of the transition and observation rules. Rules with local structure, those that have only a few deictic references and simple preconditions, will ensure that $H_{0:t}$ grows slowly. With enough specialization, $H_{0:t}$ can, in general, reach its full size. However, since we only specialize for the given actions and the observations that are actually encountered, this will not always happen. For example, if the robot only explores part of the maze, it never needs to represent or reason about the locations it has not visited.

3   The Logical Particle Filter

In complex domains, the partition $H_{0:t}$ will grow so large that computing the sum in Eq. 1 will become unmanageable. However, many of the $h_{0:t} \in H_{0:t}$ will have low probabilities and we can obtain a good approximation by sampling from the distribution over the hypotheses. In this section, we recast filtering as computing an expectation

$$I(f) = E_{p(s_{0:t}|o_{1:t})}[f(s_{0:t})] \;\; ,$$

and approximate this expectation by sampling.[5]

A straightforward sampling technique is to draw samples $s_{0:t}^{(i)}$, $i = 1 \ldots n$, from $p(s_{0:t}|o_{1:t})$ and approximate $I(f)$ as

$$\hat{I}_n(f) = (1/n) \sum_{i=1}^{n} f(s_{0:t}) \ .$$

However, we want to sample hypothesis sequences $h_{0:t}$, not state sequences. Recall that $p(s_{0:t}|o_{1:t}) = p(s_{0:t}, h_{0:t}|o_{1:t})$ when $s_{0:t} \in h_{0:t}$, and so

$$E_{p(s_{0:t}|o_{1:t})}[f(s_{0:t})] = E_{p(s_{0:t}, h_{0:t}|o_{1:t})}[f(s_{0:t})] \ .$$

Now, because of the conditional independence, $E_{p(s_{0:t}, h_{0:t}|o_{1:t})}[f(s_{0:t})]$ can be decomposed into $E_{p(h_{0:t}|o_{1:t})}[E_{p(s_{0:t}|h_{0:t})}[f(s_{0:t})]]$. We sample values for the outer expectation, computing the inner one exactly for each sample.

When we sample $h_{0:t}$, we are performing a type of Rao-Blackwellization, which is a variance–reduction technique that can be applied to sampling schemes that compute the expectation of a function of more than one random variable. The desired expectation is rewritten as

$$E_{p(x_1, x_2)}[f(x_1, x_2)] = E_{p(x_2)}[E_{p(x_1|x_2)}[f(x_1, x_2)]]$$

and the outer expectation is approximated by sampling. The key to applying it to logical hypothesis sequences is to define a random variable over the sets in each partition.[6]

The final step for defining the logical particle filter (LPF) is to show that this sampling can be done online. We do this by applying particle filter (PF) techniques; see [2] for a general discussion of Rao-Blackwellized particle filters (RBPFs). Fig. 3 shows the LPF algorithm, which performs online importance sampling from a proposal distribution $q(h_{0:t}|o_{1:t})$ that factors into

$$q(h_{0:t}|o_{1:t}) = q(h_0) \prod_t q(h_t|h_{t-1}, o_t) \ .$$

The LPF uses $p(h_t|h_{t-1}, o_t)$ for this proposal distribution because it minimizes the variance of the importance weights conditioned on h_{t-1} and o_t [1]. The associated importance weights are then $p(o_t|h_{t-1})$. We can use this combination because, as we saw before, $p(h_t|h_{t-1}, o_t)$ can be computed efficiently by enumeration; similar reasoning applies to the importance weights.

---

[5] This generalizes the standard filtering problem: setting $f(x) = \delta_{s_{0:t}}(x)$, the Dirac-delta function at $s_{0:t}$, yields $E_{p(s'_{0:t}|o_{1:t})}[\delta_{s_{0:t}}(s'_{0:t})] = p(s_{0:t}|o_{1:t})$.

[6] A more traditional way of applying Rao-Blackwellization is to factor each state so that $s = (a, b)$. Then $p(a_{0:t}, b_{0:t}|o_{1:t})$ is represented compactly, often with a dynamic Bayes net, allowing $E_{p(a_{0:t}|b_{0:t}^{(i)}, o_{1:t})}[f(a_{0:t}, b_{0:t}^{(i)})]$ to be computed efficiently. This can be seen as a partition $H_{0:t}$ with sets $h_{0:t} = \{s_{0:t}|b_{0:t} = b\}$ for each possible b.

Initialization: For $i = 1 \ldots n$, set $h_0^{(i)} = true$.

Filtering: For $t = 1 \ldots T$:

1. For $i = 1 \ldots n$, draw:

$$h_t^{(i)} \sim q(h_t | h_{t-1}^{(i)}, o_t)$$

   and set

$$h_{0:t}^{(i)} = (h_{0:t-1}^{(i)}, h_t^{(i)})$$

2. Calculate the importance weights:

$$w(h_{0:t}^{(i)}) = \frac{p(o_t | h_t^{(i)}) p(h_t^{(i)} | h_{t-1}^{(i)})}{q(h_t^{(i)} | h_{t-1}^{(i)}, o_t)}$$

3. Normalize the importance weights:

$$\widetilde{w}(h_{0:t}^{(i)}) = \frac{w(h_{0:t}^{(i)})}{\sum_{i=1}^{n} w(h_{0:t}^{(i)})}$$

4. Resample new particles $h_{0:t}^{(i)}$ according to the distribution defined by the weights $\widetilde{w}(h_{0:t}^{(i)})$.

Fig. 3. The logical particle filter.

After sampling, the LPF estimate of $I(f)$ is

$$\widehat{I_n^{rbpf}}(f) = \sum_{i=1}^{n} E_{p(s_{0:t} | h_{0:t}^{(i)})} [f(s_{0:t})] \ .$$

This expectation may be hard to compute in general, but consider $f(x) = \delta_{s_{0:t}}(x)$, the Dirac-delta function at $s_{0:t}$. In this case,

$$E_{p(s'_{0:t} | h_{0:t}^{(i)})} [\delta_{s_{0:t}}(s'_{0:t})] = p(s_{0:t} | h_{0:t}^{(i)}) \ ,$$

which we saw how to compute in Sec. 2.1. There are other interesting choices for $f(x)$ that could be computed directly from the sampled $h_{0:t}^{(i)}$. For example, we could define a function that returns the total distance the robot has traveled in the maze. Exploring the full range of possibilities is an important area for future work.

3.1 Discussion

Since the time complexity of extending the hypotheses $h_{0:t}$ at each time step t does not depend on t, the LPF is an online sampling method.

While sampling from the $q(h_t | h_{t-1}, o_t)$ proposal distribution is a form of Rao-Blackwellization, it also has another added benefit. Since we are building h_t online without ever specifying full states, we are, in effect, sampling aspects

of the initial state s_0, only as observations o_t are received for which they are relevant. For example, consider sampling one of the h_1^i that we saw in step 3 in Sec. 2 when the observation $o_1 = \neg hall\text{-}left$ is received. The LPF would be much more likely to sample h_1^1 or h_1^3 than the others because they do not have a hall to the left. However, at this stage of sampling in a traditional particle filter, the entire state of the maze would already be determined (sampled from the prior) and there would be no chance to use the observation in the proposal distribution. Since this same effect is seen at every time step, the LPF provides a type of infinite look-ahead sampling of the prior that can significantly improve performance.

Crisan and Doucet [10] present an analysis of RBPF algorithms. Although the estimate is biased, it converges to $I(f)$ as $n \to \infty$. Also, the sample variance is never larger than that of a traditional PF. Because the LPF is an RBPF, these results apply directly. The amount of variance reduction depends on the size of $H_{0:t}$. Even if $H_{0:t}$ grows large, it is often small for early sampling steps, significantly reducing the variance.

4   Evaluation and Discussion

As an initial experiment, we compared the LPF to a PF in the maze world, with four directions, described earlier. Since it is difficult to evaluate the performance of approximate inference algorithms in domains where the true answer can not be computed, we limited the size of the maze to be no larger than a three by three grid of locations and computed $p(s_{0:t}|o_{1:t}, a_{1:t})$ exactly by enumeration. Fig. 4 shows the average variational distance (absolute difference) of each filter's estimates (computed by setting $f(x) = \delta_{s_{0:t}}(x)$) of this probability as a function of the amount of computation time used. Each data point is an average over ten runs on state and observation sequences sampled from the joint model $p(s_{0:t}, o_{1:t}|a_{1:t})$ for randomly chosen sequences of ten actions and is labeled with the number of particles used by the filter. As we would expect, the LPF has lower error and less variance. Although our unoptimized implementation of the LPF runs, on average, 10 times slower per particle, the improvement in performance significantly outweighs this extra cost.

Although it is not possible to do an exact comparison in larger domains, we know that the LPF will never produce a worse estimate than a PF, because it reduces to a traditional PF algorithm in the worst case when each hypothesis describes a single state. There are also a wide range of domains where the LPF can compute reasonable estimates but the PF will perform poorly. For example, consider any domain where the prior $p(s_0)$ assigns a very low probability to most of the states. With high probability the true state sequence $s_{0:t}$ will contain one of these states, but it will be extremely unlikely for the PF to sample it. Depending on the transition distribution, if the true initial state is not sampled, the filter may never recover. For example, in the maze world, if none of the particles contain the true maze structure, the PF will never converge on the proper structure. In contrast, the LPF, with its delayed sampling of the prior, will have
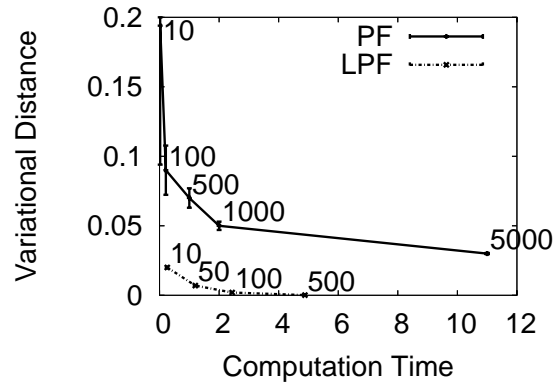
Fig. 4. The variational distance between the estimated and true probabilities as a function of computation time. Each data point is labeled with the number of particles used.

a good chance of sampling the proper structure, by extending its hypotheses to incorporate new observations about the maze as they are received. In particular, for the case of deterministic transition and observation distributions, the LPF will perform optimally. In the maze world, the hypotheses will simply record the maze as it is observed and provably compute the correct estimate with a single particle.

Our biggest focus for future work will be to explore different methods for constructing partitions. In particular, we might not want to always specialize at each time step. If we instead knew that some of the aspects of the world in our hypotheses were no longer needed for our tracking application, we could marginalize them out and generalize the partition, preventing it from growing to contain all possible state sequences.

References

1. Doucet, A., de Freitas, N., Gordon, N.: Sequential Monte Carlo Methods in Practice. (2001)
2. Doucet, A., de Freitas, N., Murphy, K., Russell, S.: Rao-Blackwellized particle filtering for dynamic Bayesian networks. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence. (2000)
3. Sanghai, S., Domingos, P., Weld, D.: Relational dynamic Bayesian networks. Journal of Artificial Intelligence Research **24** (2005)
4. Shirazi, A., Amir, E.: First-order logical filtering. In: Proceeding of the International Joint Conferences on Artificial Intelligence. (2005)
5. Milch, B., Russell, S.: General-purpose MCMC inference over relational structures. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence. (2006)

6. de Salvo Braz, R., Amir, E., Roth, D.: MPE and partial inversion in lifted probabilistic variable elimination. In: Proceedings of the National Conference on Artificial Intelligence. (2006)
7. Kersting, K., Raedt, L.D., Raiko, T.: Logical hidden Markov models. Journal of Artificial Intelligence Research **25** (2006)
8. Zettlemoyer, L., Pasula, H., Kaelbling, L.: Learning planning rules in stochastic worlds. In: Proceedings of the National Conference on Artificial Intelligence. (2005)
9. Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D.L., Kolobov, A.: BLOG: Probabilistic models with unknown objects. In: Proceeding of the International Joint Conferences on Artificial Intelligence. (2005)
10. Crisan, D., Doucet, A.: Convergence of generalized particle filters. TR 381, Cambridge University (2000)