Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks

Kevin Murphy and Stuart Russell

1 Introduction

Particle filtering in high dimensional state-spaces can be inefficient, because a large number of samples are needed to represent the posterior. A standard technique to increase the efficiency of sampling techniques is to reduce the size of the state-space by marginalizing out some of the variables analytically; this is called Rao-Blackwellisation (Casella and Robert 1996). Combining these two techniques results in Rao-Blackwellised particle filtering (RBPF) (Doucet 1998, Doucet, de Freitas, Murphy and Russell 2000). In this chapter, we explain RBPF, discuss when it can be used, and give a detailed example of its application to the problem of map learning for a mobile robot, which has a very large ($\sim 2^{100}$) discrete state-space.

The key idea of RBPF is to partition the state-space Z_t into two subspaces, R_t and X_t, such that the distribution $P(X_t|R_{1:t}, y_{1:t})$ can be updated analytically and efficiently; the distribution $P(R_{1:t}|y_{1:t})$ is updated using particle filtering. The justification for this decomposition follows from the chain rule of probability:

$$P(X_{1:t}, R_{1:t}|y_{1:t}) = P(X_{1:t}|R_{1:t}, y_{1:t})P(R_{1:t}|y_{1:t})$$

Sampling just R_t will generally require many fewer particles (to reach some fixed accuracy threshold) than standard particle filtering, which would sample both R_t and X_t. (Please see the chapter by Doucet, de Freitas and Gordon (2000: this volume) for an introduction to standard PF.)

RBPF is very similar to standard PF, except that each particle now maintains not just a sample from $P(R_{1:t}|y_{1:t})$, which we will denote by $r^{(i)}_{1:t}$, but also a *parametric* representation of $P(X_t|r^{(i)}_{1:t}, y_{1:t})$, which we will denote by $\alpha^{(i)}_t$. (The parametric representation might be a mean vector and a covariance matrix, for instance.) The R_t samples are updated as in standard PF, and then the X_t distributions are updated using an exact filter, conditional on R_t. The overall algorithm is shown in Figure 1.

In Section 2, we discuss when this algorithm can be usefully applied, which is best described using the language of dynamic Bayesian networks. In Section 3, we discuss in detail how to compute the equations used by the algorithm. Finally, in Section 4, we discuss the application of RBPF to map learning.

Generic RBPF

1. Sequential importance sampling step

   - For $i = 1, \ldots, N$, sample
   $$\left(\widehat{r}_t^{(i)}\right) \sim q(r_t; r_{1:t-1}^{(i)}, y_{1:t})$$
   and set
   $$\left(\widehat{r}_{1:t}^{(i)}\right) \overset{\text{def}}{=} (\widehat{r}_t^{(i)}, r_{1:t-1}^{(i)})$$
   - For $i = 1, \ldots, N$, evaluate the importance weights up to a normalising constant:
   $$w_t^{(i)} \propto \frac{p\left(y_t \middle| y_{1:t-1}, r_{1:t}^{(i)}\right) p\left(r_t^{(i)} \middle| r_{1:t-1}^{(i)}, y_{1:t-1}\right)}{q\left(r_t; r_{1:t-1}^{(i)}, y_{1:t}\right)}$$
   - For $i = 1, \ldots, N$, normalise the importance weights:
   $$\widetilde{w}_t^{(i)} = w_t^{(i)} \left[\sum_{j=1}^{N} w_t^{(j)}\right]^{-1}$$

2. Selection step

   - Resample N samples from $(\widehat{r}_{1:t}^{(i)})$ according to the importance distribution $\widetilde{w}_t^{(i)}$ to obtain N random samples $(r_{1:t}^{(i)})$ approximately distributed according to $p(r_{1:t}^{(i)}|y_{1:t})$.

3. Exact step

   - Update $\alpha_t^{(i)}$ given $\alpha_{t-1}^{(i)}$, $r_t^{(i)}$, $r_{t-1}^{(i)}$, and y_t.

Figure 1. Generic Rao-Blackwellised particle filtering algorithm. If we replace references to r_t with $z_t = (x_t, r_t)$ and omit step 3, the result is a "regular" (non Rao-Blackwellised) particle filter. Note that if we are only interested in filtering, we do not to store the full trajectory $r_{1:t}^{(i)}$ in each particle, just its most recent component, $r_t^{(i)}$, since we are updating $\alpha_t^{(i)}$ online. This figure is adapted from (Doucet et al. 2000).

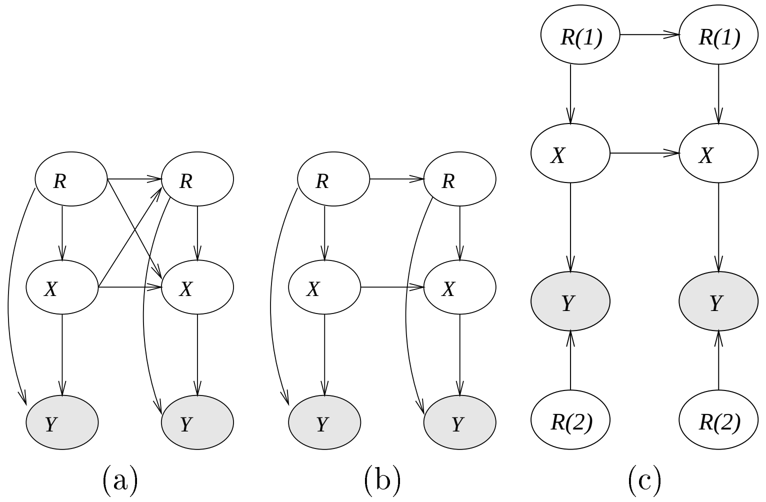


Figure 2. (a) The canonical DBN to which RBPF can be applied. The shaded Y_t nodes denote observations. (b) A simplification in which we have removed the "cross arcs." The R_t nodes are called the roots, and the X_t nodes the leaves. (c) We have partitioned the root node into two components, $R_t(1)$, which affects the dynamics of X_t, and $R_t(2)$, which affects the observation model Y_t.

2 RBPF in general

We can most easily characterize the models to which RBPF may be efficiently applied by representing them graphically as dynamic Bayesian networks (DBNs), which we now briefly explain. Bayesian networks (Pearl 1988, Cowell, Dawid, Lauritzen and Spiegelhalter 1999) are directed acyclic graphs, in which nodes represent random variables, and the lack of arcs represents conditional independencies. Dynamic Bayesian networks extend Bayesian networks to probability distributions that evolve over time. As in a state-space model, we must specify the transition model, $P(Z_t|Z_{t-1})$, the observation model, $P(Y_t|Z_t)$, and the prior, $P(Z_1)$. We use a "two slice" graph to represent the conditional independence relationships in $P(Z_t|Z_{t-1})$. In addition to the graph structure, we must specify the conditional probability distribution (CPD) of each node given its parents. For a more detailed introduction to DBNs, please see Koller and Lerner (2000: this volume).

RBPF can be applied to any DBN that can be made topologically equivalent to the model shown in Figure 2(a), clustering nodes together if necessary. However, to apply the algorithm in Figure 1, we must compute the term

$$P(R_t|r_{1:t-1}^{(i)}, y_{1:t-1}) = \sum_{x_{t-1}} P(R_t|r_{t-1}^{(i)}, x_{t-1})P(x_{t-1}|r_{1:t-1}^{(i)}, y_{1:t-1}) \qquad (2.1)$$

If X_{t-1} is a vector in $\{1, \ldots, k\}^m$, i.e., the cross-product of m discrete k-valued random variables, then this equation takes $O(k^m)$ time to compute, which is usually unacceptably high, especially since it must be computed once per particle. Likewise, if X_{t-1} is a vector in $\mathbb{R}^m$, the required integration often cannot be computed in closed form. Hence it is common to assume there is no arc from X_{t-1} to R_t, so that we can eliminate the marginalization over X_{t-1}. In other words, the equation becomes

$$P(R_t | r_{1:t-1}^{(i)}, y_{1:t-1}) = P(R_t | r_{t-1}^{(i)})$$

If we consider a single time slice, there are now no arcs entering R_t, so we refer to it as a "root." Similarly, X_t will be called a "leaf." It is also common to assume there is no arc from R_{t-1} to X_t, although this does not change the algorithm in any significant way.[1] After eliminating both of these inter-slice "cross arcs", we end up with the DBN shown in Figure 2(b).

Given the simplified model of Figure 2(b), we are left with two tasks: to sample the root nodes efficiently, and to update the leaf distributions efficiently, given that the roots have known values (and are therefore effectively disconnected from the graph). Since the roots can be sampled using a standard particle filter, many of the techniques discussed in this book are directly applicable. Hence we focus on the latter issue here.

A simple example of when the leaves can be efficiently updated is when the CPDs for X_t and Y_t are linear-Gaussian, in which case the model is called a conditionally linear-Gaussian state-space model. We can compute $P(X_t | r_{1:t}, y_{1:t})$ recursively using a Kalman filter. If R_t is discrete, the model is called a jump Markov linear system, or switching state-space model (Chen and Liu 1999, Doucet, Gordon and Krishnamurthy 1999). In this case, it is often useful to split the root node into two independent components, as in Figure 2(c). $R_t(1)$ is a parent of X_t, and can be used to model discontinuous changes in state (c.f., McGinnity and Irwin (2000: this volume)). $R_t(2)$ is a parent of Y_t, and can be used to model outliers in the observations. Another case when we might have multiple root nodes is when we are doing online parameter estimation: in the Bayesian approach, the parameters are just additional random variables (nodes in the graph) that can be sampled using particle filtering.

In the jump Markov linear system, the jumps occur "spontaneously", as dictated by the Markov transition matrix on the $R_t(1)$ node. For some models, it is useful to have the system's state, X_t, "trigger" the jump, which can be modelled by adding an arc from X_{t-1} to R_t, and making R_t's CPD a logistic or softmax function. Unfortunately, Equation 2.1 becomes hard to compute in this case, since we have a hidden continuous parent connected to a (hidden) discrete child. One possible approach would be to use the variational approximation discussed in (Murphy 1999).

[1]Specifically, it just means we only have to condition on $r_t^{(i)}$, instead of both $r_t^{(i)}$ and $r_{t-1}^{(i)}$, when updating $\alpha_t^{(i)}$ in step 3 of the algorithm in Figure 1.

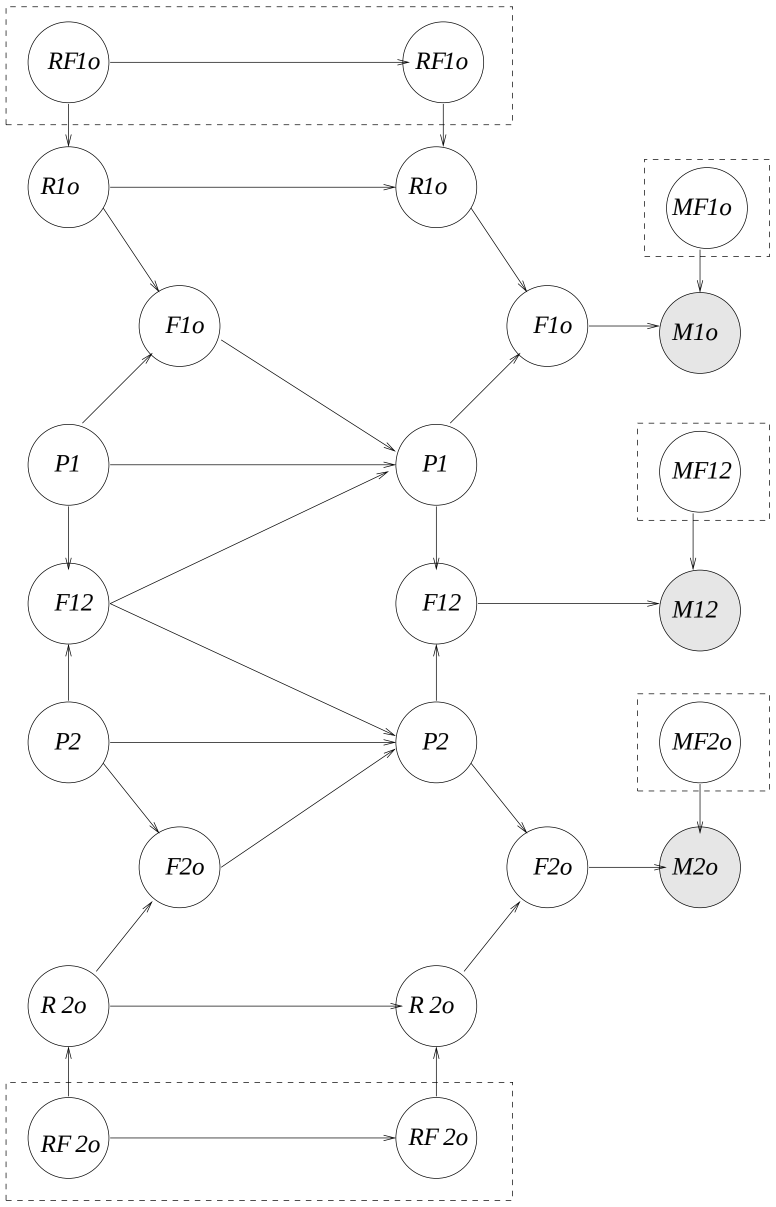


Figure 3. The two-tank DBN, adapted from Koller and Lerner (2000: this volume). We have omitted the $RF12$ and $R12$ nodes to keep the figure simple. RF standards for resistance failure, and represents a permanent change in the resistivity of a pipe. MF stands for measurement failure, and represents a temporary error in the sensor. The MF and observed (shaded) nodes are only shown for slice 2, for simplicity, since they are transient variables, i.e., they are not connected across time-slices.

A more complex DBN is shown in Figure 3. We would like to sample the discrete indicator variables (which model failures of various kinds) shown inside the dotted boxes, and apply exact inference on the remaining, continuous-valued nodes, which we can group into a single vector-valued node, X_t. Unfortunately, in this model, although the noise is Gaussian, the dynamics are non-linear, making it hard to integrate out X_t. We could apply an approximate inference technique, such as the extended Kalman filter, or the unscented filter (Julier and Uhlmann 1997), but we would no longer be doing strict Rao-Blackwellisation. In particular, these approximations may diverge. (Koller and Lerner apply particle filtering to both R_t and X_t.)

Given an arbitrary DBN with a potentially complex topology, which nodes should be considered as roots, and which as leaves? If we disallow arcs from X_{t-1} to R_t for the reasons discussed above, the answer is fairly straightforward. We define the set of nodes which are eligible to be roots, $\mathcal{R}$, to be all the nodes which either have no parents, or whose parents are also in the set $\mathcal{R}$, possibly shifted back in time. We initialize $\mathcal{R}$ to be all the nodes $X_t(i)$ which have no parents, or whose only parent is $X_{t-1}(i)$. We then add to $\mathcal{R}$ all the nodes, $X_t(i)$, whose parents are in $\mathcal{R} \cup \{X_{t-1}(i)\}$. For example, in the two-tank DBN in Figure 3, we start with $\mathcal{R}_1$ containing all the nodes in dotted boxes, and can then add $R1o$ and $R2o$ to get

$$\mathcal{R}_2 = \{R1o, R2o, RF1o, RF2o, MF1o, MF2o, MF12\}.$$

The idea is to keep growing the set $\mathcal{R}$ until the set of remaining variables, $\mathcal{X}_i \overset{\text{def}}{=} \mathcal{S} \backslash \mathcal{R}_i$, can be updated exactly and efficiently, where $\mathcal{S}$ represents all the nodes in a single slice of the DBN, and $\mathcal{R}_i$ represents the root set at iteration i of the above root-growing algorithm. In the two-tank case, $\mathcal{R}_2$ is locally maximal: there is no single node that can be added such that the desired closure property would be maintained. The next valid root set is $\mathcal{R}_3 = \mathcal{S}$, which corresponds to sampling all the variables, as in standard PF.

Now suppose the set of non-boxed variables $\mathcal{X}_1$ had *linear*-Gaussian dynamics. In this case, we could sample the discrete root nodes $\mathcal{R}_1$ and apply the Kalman filter to the continuous nodes $\mathcal{X}_1$, as we discussed before. Expanding from $\mathcal{R}_1$ to $\mathcal{R}_2$, while legal, would probably not be very helpful, since $R1o$ and $R2o$ are jointly Gaussian with $\mathcal{X}_2$ by assumption, and hence can be marginalized out efficiently.

When all the (hidden) nodes in a DBN are discrete, we can always perform exact inference in closed form, using the HMM filter or the junction tree algorithm (Smyth, Heckerman and Jordan 1997, Cowell et al. 1999). The problem is that the complexity is generally $O(k^n)$, where n is the number of hidden nodes, and k is the number of values each node can take on. (We will see an example of this in Section 4.) In this case, we should keep growing $\mathcal{R}$ until $\mathcal{X}$ becomes small enough that exact inference becomes computationally

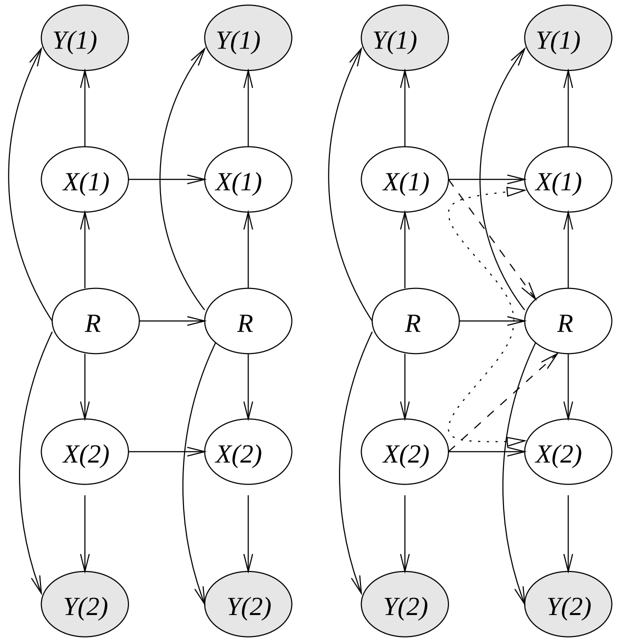


Figure 4. (a) We have partitioned the leaves and observations into two components. (b) The dashed arcs entering the root node induce correlation between the leaves, as indicated by the dotted "path of influence." The root nodes are shown shaded since they are instantiated.

tractable.[2] Of course, the larger $\mathcal{R}$, the more samples we will need, so this tradeoff must be made carefully. We hope to examine this issue in the future. In this chapter, we assume that the set $\mathcal{R}$ has been pre-specified.

In some cases, the non-root nodes might be conditionally independent given the roots, as in Figure 4(a). We will see an example of this in Section 4. The advantage of this is that we can update each leaf independently, conditioned on the root, which is exponentially more efficient than updating the leaves jointly. Note, however, that, if we had arcs from the leaves entering the root, as in Figure 4(b), we would no longer be able to update the leaves independently. This is because R_t would act like a common observed child node, inducing correlation amongst its parents, a phenomenon called "explaining away" (Pearl 1988).

[2]For example, applying the greedy root-growing algorithm to the BAT DBN (Figure 6) in the chapter by Koller and Lerner gives the following results, where we use the notation $\Delta_i \stackrel{\text{def}}{=} \mathcal{R}_i \setminus \mathcal{R}_{i-1}$ to represent the extra nodes added at the ith iteration. $\mathcal{R}_1 = \{$LeftClr, RightClr, EngStatus, BYdotDiff, SensorValid, Bclr$\}$, $\Delta_2 = \{$ FYdotDiff, BXdot, Bclose-Fast, Fclr $\}$, $\Delta_3 = \{$ FcloseSlow $\}$, $\Delta_4 = \{$ FrontBackStatus $\}$. To continue growing, we would have to consider additions of two or more parents simultaneously, etc.

3 The RBPF algorithm in detail

We now explain the RBPF algorithm, which is sketched in Figure 1, in more detail. We assume there are no arcs from X_{t-1} to R_t, and for simplicity, that there are also no arcs from R_{t-1} to X_t, so the generic structure is one that is isomorphic to Figure 2b.

As we mentioned in the introduction, the belief state, $P(X_t, R_{1:t}|y_{1:t})$, is represented by a set of N weighted particles. The marginal distribution on the root nodes is approximated as

$$P(r_{1:t}|y_{1:t}) \approx \sum_{i=1}^{N} w_t^i \delta(r_{1:t}^{(i)}, r_{1:t})$$

where w_t^i is the weight of the i'th particle, and $\delta(x, y) = 1$ if $x = y$ and is 0 otherwise. The marginal on the leaf is approximated as

$$\begin{aligned} P(X_t|y_{1:t}) &= \sum_{r_{1:t}} P(X_t|r_{1:t}, y_{1:t}) P(r_{1:t}|y_{1:t}) \\ &\approx \sum_{r_{1:t}} P(X_t|r_{1:t}, y_{1:t}) \sum_{i=1}^{N} w_t^i \delta(r_{1:t}^{(i)}, r_{1:t}) \\ &= \sum_{i=1}^{N} w_t^i P(X_t|r_{1:t}^{(i)}, y_{1:t}) \end{aligned}$$

For notational simplicity, we will assume that all nodes are discrete, so we can represent each leaf marginal as a vector in $[0, 1]^k$ which sums to 1, where k is the number of possible values of the node. In addition, since the nodes are discrete, all the CPDs can be expressed as tables:

$$\begin{aligned} \pi_R(r) &\overset{\text{def}}{=} P(R_1 = r) \\ T_R(r'; r) &\overset{\text{def}}{=} P(R_t = r|R_{t-1} = r') \\ T_X(x', r; x) &\overset{\text{def}}{=} P(X_t = x|R_t = r, X_{t-1} = x') \\ O_Y(x, r; y) &\overset{\text{def}}{=} P(Y_t = y|X_t = x, R_t = r) \end{aligned}$$

Since the leaf nodes are discrete, we can update their distributions, conditional on having sampled R_t, using the HMM filter as follows. First we compute the one step-ahead prediction:

$$\begin{aligned} \alpha_{t|t-1}^{(i)}(x) &\overset{\text{def}}{=} P(X_t = x|y_{1:t-1}, r_{1:t}^{(i)}) \\ &= \sum_{x'} P(X_t = x|X_{t-1} = x', r_t^{(i)}) P(X_{t-1} = x'|y_{1:t-1}, r_{1:t-1}^{(i)}) \\ &= \sum_{x'} T_X(x', r_t^{(i)}; x) \alpha_{t-1}^{(i)}(x') \end{aligned}$$

Then we do Bayesian updating:

$$\begin{aligned}
\alpha_t^{(i)}(x) &\overset{\text{def}}{=} P(X_t = x | y_{1:t}, r_{1:t}^{(i)}) \\
&= (1/Z_t^{(i)}) P(y_t | X_t = x, r_t^{(i)}) P(X_t = x | y_{1:t-1}, r_{1:t}^{(i)}) \\
&= (1/Z_t^{(i)}) O_Y(x, r_t^{(i)}; y_t) \alpha_{t|t-1}^{(i)}(x)
\end{aligned}$$

where the denominator is equal to the likelihood:

$$Z_t^{(i)} \overset{\text{def}}{=} P(y_t | y_{1:t-1}, r_{1:t}^{(i)}) = \sum_x O_Y(x, r_t^{(i)}; y_t) \alpha_{t|t-1}^{(i)}(x) \tag{3.1}$$

If we have L leaves conditionally independent leaves, as in Figure 4(a), we apply these equations to each leaf separately. Each such update will result in a "local likelihood" term, $Z_t(j)^{(i)}$, like the one above. The overall likelihood then becomes a product of the local likelihoods:

$$\begin{aligned}
P(y_t | y_{1:t-1}, r_{1:t}^{(i)}) &= \sum_{x_1, \ldots, x_L} \prod_{j=1}^{L} \left[P(y_t(j), X_t(j) = x_j | r_{1:t}^{(i)}, y_{1:t-1}) \right] \\
&= \prod_{j=1}^{L} \left[\sum_x P(y_t(j) | X_t(j) = x, r_{1:t}^{(i)}) P(X_t(j) = x | r_{1:t-1}^{(i)}, y_{1:t-1}) \right] \\
&= \prod_{j=1}^{L} Z_t(j)^{(i)}
\end{aligned}$$

All that remains is to specify how to do the following standard PF steps:

- Sample new values of the roots.
- Compute the weight of a particle.
- Resample the particles.

We will now explain these steps in detail. We drop the i superscript for brevity.

As discussed in the chapter by Doucet, de Freitas and Gordon (2000: this volume), in sequential importance sampling, if we sample from the proposal distribution $q(R_t; r_{1:t-1}, y_{1:t})$, we must assign the particle weight equal to the ratio between the true posterior and the proposal density:

$$w_t \propto \frac{P(y_t | y_{1:t-1}, r_{1:t}) P(R_t | r_{t-1})}{q(R_t; r_{1:t-1}, y_{1:t})}$$

The simplest case is if we sample from the prior, $q(R_t) = P(R_t | r_{t-1})$. In this case, the weight is simply the likelihood computed in Equation 3.1.

The "optimal" proposal distribution, in the sense of minimizing the variance of the importance weights (Doucet et al. 1999), is given by

$$P(R_t|r_{1:t-1}, y_{1:t}) = \frac{P(y_t|y_{1:t-1}, r_{1:t})P(R_t|r_{t-1})}{P(y_t|y_{1:t-1}, r_{1:t})}$$

where the denominator is the one step-ahead likelihood

$$P(y_t|y_{1:t-1}, r_{1:t}) = \sum_{r=1}^{|R_t|} P(y_t|y_{1:t-1}, r_{1:t-1}, R_t = r)P(R_t = r|r_{t-1})$$

This requires computing the likelihood $|R_t|$ times, which can be quite expensive. Whether the computational expense is worthwhile depends on the relative reliability of the observations and the transition prior: if the prior is weak (diffuse) and the observation likelihood is strong (sharply peaked), many particles may be proposed in a part of the state space that has low likelihood, which is wasteful. In this case, it might be worthwhile to take the most recent evidence, y_t, into account before proposing. See Pitt and Shephard (2000: this volume) for a more detailed discussion.

Finally, given a set of particles and weights, we can resample a fresh set using any of the standard methods, such as residual resampling, discussed in Doucet, de Freitas, and Gordon (2000: this volume).

4 Application: Concurrent localisation and map learning for a mobile robot

In this section, we discuss an application of RBPF to a highly simplified version of the problem of map learning for mobile robots (Murphy 2000). The application of standard (non-RB) PF to the problem of robot localisation (finding the robot's position given a known map) using real robots is discussed in Thrun et al. (2000: this volume).

Consider a robot which can move on a discrete, two-dimensional grid. The goal is to learn the color of each grid cell, which can be either black or white (say). The difficulty is that the color sensors are not perfect (they may accidently flip bits with probability p_o), nor are the motors (the robot may fail to move in the desired direction with probability p_a, due e.g., to wheel slippage). Consequently, it is easy for the robot to get lost. And when the robot is lost, it does not know what part of the map to update. (Note that, if the robot always knew its location, e.g., by using GPS, map learning would be easy; unfortunately, GPS does not work indoors, nor is it accurate enough.)

The optimal Bayesian solution to this problem is to maintain a belief state over both the location of the robot, $L_t \in \{1, \ldots, N_L\}$, and the color of each grid cell, $M_t(l) \in \{0, 1\}$, $l = 1, \ldots, N_L$, where N_L is the number of cells. For notational simplicity, in this subsection we shall assume there are only two colors; the technique easily generalizes.

We assume the color of the cells can change, to represent the fact that the environment can be dynamic. For example, in Section 4.2, we use four "colors" that represent whether a cell is unoccupied, or contains a wall, or an open door, or a closed door, and we allow doors to change between open and closed. In this case, M_t is like an occupancy grid (Moravec and Elfes 1985), which is a simple kind of *m*ap. For simplicity, we assume the colors of the cells change independently, but with an identical distribution, specified by the matrix $T_M(c;c') \overset{\text{def}}{=} P(M_t(l) = c'|M_{t-1}(l) = c)$. If this is an identity matrix, it means that the colors do not change.[3]

The observation model is that the robot sees the color of the grid cell at its current location, corrupted by noise:

$$P(y_t|m_1, \ldots, m_{N_L}, L_t = l) \overset{\text{def}}{=} B(m_l; y_t) = \begin{cases} 1 - p_o & \text{if } y_t = m_l \\ p_o & \text{if } y_t \neq m_l \end{cases}$$

where p_o is the probability that a color gets misread, and B is the 2×2 observation matrix with $1 - p_o$ on the diagonal and p_o off the diagonal. A more realistic model would capture the fact that the robot can see the color of neighboring cells as well. This is not hard to do, but for notational simplicity, we shall stick to the single-cell model for now.

Let us assume for now that the robot is in a one-dimensional grid world, so it can only move left or right, depending on the control input, U_t. The robot moves in the desired direction with probability $1 - p_a$, and otherwise stays put. In addition, it cannot move off the edges of the grid. Algebraically, this becomes

$$P(L_t = l'|L_{t-1} = l, U_t = \rightarrow) = \begin{cases} 1 - p_a & \text{if } l' = l + 1 \text{ and } 1 \leq l' < N_L \\ p_a & \text{if } l = l' \text{ and } 1 \leq l' < N_L \\ 1 & \text{if } l = l' = N_L \\ 0 & \text{otherwise} \end{cases}$$

The equation for $P(L_t = l'|L_{t-1} = l, U_t = \leftarrow)$ is analogous. In Section 4.2, we will consider the case of a robot moving in two dimensions. In that case, the motion model will be slightly more complex.

Finally, we need to specify the prior. The robot is assumed to know its initial location, so the prior $P(L_1)$ is a delta function. If the robot did not know its initial location, there would be no well-defined origin to the map; in other words, the map is relative to the initial location. Also, the robot has a uniform prior over the colors of the cells. However, it knows the size of the environment, so the state space has fixed size.

[3]If the colors do not change, and if we are satisfied with learning a maximum likelihood estimate of the map, instead of a full posterior, we can treat the $M(i)$s as fixed parameters and use EM to learn them (Thrun, Burgard and Fox 1998). However, doing this online does not work very well (Murphy 2000).

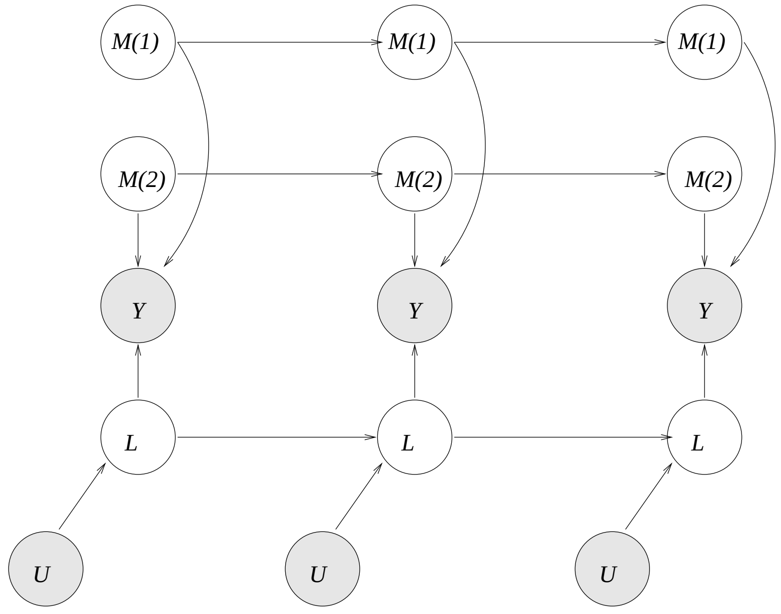


Figure 5. The DBN used in the map learning problem. $M_t(l)$ represents the color of grid cell l at time t, L_t represents the robot's location, Y_t the current observation, and U_t is the current input (control).

The DBN we are using is shown in Figure 5. This kind of topology (modulo the observed input ndoes) is called a factorial HMM (Ghahramani and Jordan 1997). Inference in these models is computationally intractable. To be precise, if there are n chains, each of which can take on k possible values, then the belief state has size $O(k^n)$: all the chains become coupled because Y_t is a common observed child. Exact inference, using the junction tree algorithm (Smyth et al. 1997, Cowell et al. 1999), takes $O(nk^{n+1})$ operations per time step. In general, if the number of values the nodes in chain j can take on is k_j, then the belief state has size $S = \prod_{j=1}^n k_j$, and exact inference takes $O(S \sum_{j=1}^n k_j)$ operations per time step. In our application, there are N_L chains with 2 possible values each, and one chain with N_L possible values, so we need $3N_L^2 2^{N_L}$ operations per time step. In Section 4, we use a 10×10 grid, so this requires $O(2^{100})$ operations per time step for exact inference.

In our case, however, the observation model has the crucial property that Y_t only depends on a single element of $\mathbf{M}_t$ once L_t is known. Note that this conditional independence property is not obvious from the structure of the graph, but is implicit in Y_t's CPD c.f., (Boutilier, Friedman, Goldszmidt and Koller 1996). The upshot is that we can rewrite the model of Figure 5 to take on the form of Figure 4(a) as follows: L_t is equivalent to the root R_t, and the map cells $M_t(j)$ are equivalent to the leaves $X_t(j)$. (We ignore the U_t nodes for simplicity.) The transition model for the root is the motion model of the

Figure 6. A one-dimensional grid world.

robot, and the transition model of the leaves is the matrix T_M, defined to be independent of L_t. Finally, the observation nodes are as follows. We define

$$P(Y_t(j) = m' | M_t(j) = m, L_t = l) = \begin{cases} B(m; m') & \text{if } j = l \\ 1/2 & \text{if } j \neq l \end{cases}$$

where $j, l \in \{1, \ldots, N_L\}$ and $m, m' \in \{0, 1\}$. This observation model only gives information about the cell at the robot's current location, $L_t = l$, as desired; all other cells (leaves) will be effectively be updated with no observations. It is now straightforward to apply RBPF to this model.

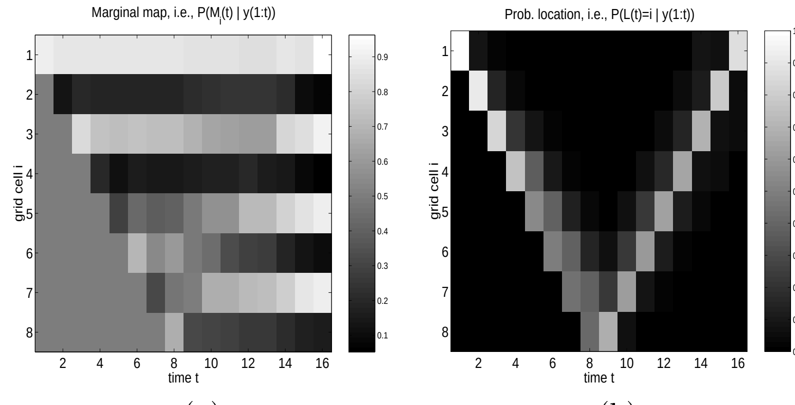
4.1 Results on a one-dimensional grid

To evaluate the effectiveness of this algorithm, we first applied it to a problem which was sufficiently small (8 cells) that we could compute the "ground truth" using exact inference. In particular, consider the one-dimensional grid shown in Figure 6. We have $N_L = 8$, so exact inference takes about 50,000 operations per time step (!). For simplicity, we will fix the control policy as follows: the robot starts at the left, moves to the end, and then returns home. Suppose there are no sensor errors, but there is a single "slippage" error at $t = 4$. We summarize this below, where U_t represents the input (control action) at time t.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L_t	1	2	3	**4**	**4**	5	6	7	8	7	6	5	4	3	2	1
Y_t	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0
U_t	-	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$

To study the effect of this sequence, we used exact inference to compute $P(L_t|y_{1:t})$ and $P(\mathbf{M}_t|y_{1:t})$: see Figure 7. At each time step, the robot thinks it is moving one step to the right, but the uncertainty gradually increases. However, as the robot returns to "familiar territory", it is able to better localize itself, and hence the map becomes "sharper", even in distant cells. Note that this effect only occurs because we are modelling the correlation between cells c.f., the stochastic map representation of (Smith, Self and Cheeseman 1988). For a more detailed interpretation of this example, see (Murphy 2000).

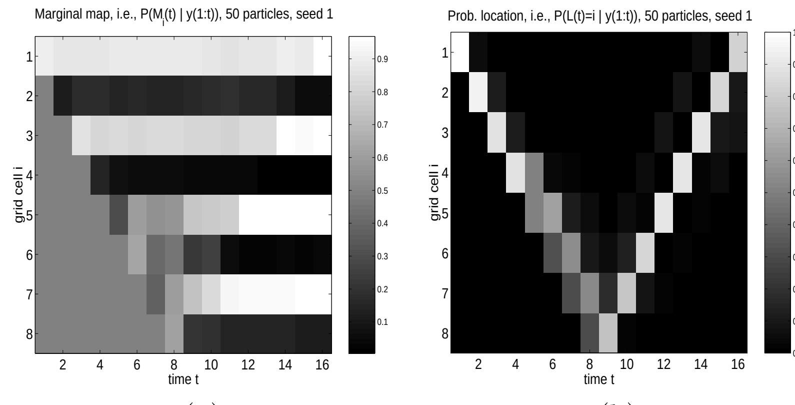
In Figure 8, we show the results obtained using RBPF. We see that it approximates the exact solution very closely, using only 50 particles. The



(a)

(b)

Figure 7. Results of exact inference on the 1D grid world. (a) A plot of $P(\mathbf{M}_t(i) = 1|y_{1:t})$, where i is the vertical axis and t is the horizontal axis; lighter cells are more likely to be color 1 (white). (b) A plot of $P(L_t = i|y_{1:t})$, i.e., the estimated location of the robot at each time step.



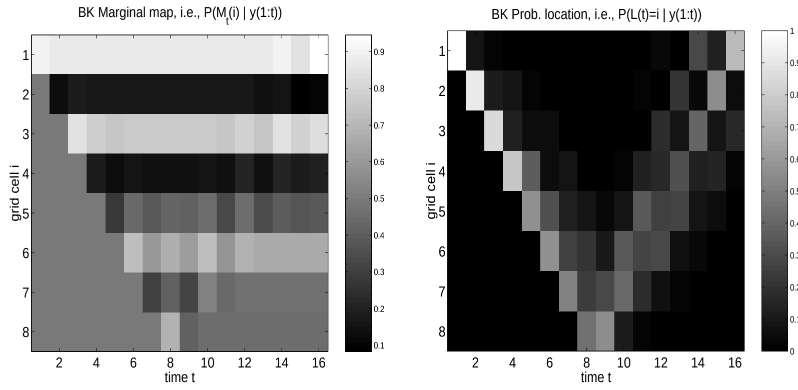
(a)

(b)

Figure 8. Results of the RBPF algorithm on the 1D grid world using 50 particles.

results shown are for a particular random number seed; other seeds produce qualitatively very similar results, indicating that 50 particles are in fact sufficient in this case. Obviously, as we increase the number of particles, the error and variance decrease, but the running time increases (linearly).

The question of how many particles to use is a difficult one: it depends both on the noise parameters and the structure of the environment (if every cell has a unique color, localization, and hence map learning, is easy). Since we are sampling trajectories, the number of hypotheses grows exponentially with time (as in a jump Markov linear system). In the worst case, the number



(a) (b)

Figure 9. Results of the BK algorithm on the 1D grid world.

of particles needed may depend on the length of the longest cycle in the environment, since this determines how long it might take for the robot to return to "familiar territory" and "kill off" some of the hypotheses (since a uniform prior on the map cannot be used to determine L_t when the robot is visiting places for the first time). In the above example, the robot was able to localize itself quite accurately when it reached the end of the corridor, since it knew that this corresponded to cell 8. In general, we may need to use a clever control policy, such as the one we discuss in the next section, to keep the number of particles tractable.

For comparison purposes, we also tried the Boyen-Koller (BK) algorithm (Boyen and Koller 1998), which is another popular approximate inference algorithm for discrete DBNs. In its simplest, fully factorised form, BK represents the belief state as a product of marginals:

$$P(\mathbf{M}_t, L_t | y_{1:t}) = P(L_t | y_{1:t}) \prod_{j=1}^{N_L} P(M_t(j) | y_{1:t})$$

The results of using BK are shown in Figure 9. As we can see, it performs very poorly in this case, because it ignores correlation between the cells. Of course, it is possible to use products of pairwise or higher-order marginals for tightly coupled sets of variables. Unfortunately, there is no natural subset of variables to use in this case, since all the grid cells are potentially correlated.

4.2 Results on a two-dimensional grid

We now consider the 10×10 grid world in Figure 10. We use four "colors", which represent closed doors, open doors, walls, and free space. Doors can toggle between open and closed independently with probability $p_c = 0.1$, but

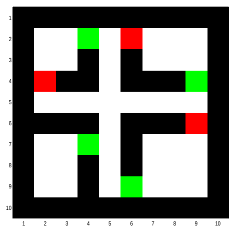


Figure 10. A simple 2D grid world. Grey cells denote doors (which are either open or closed), black denotes walls, and white denotes free space.

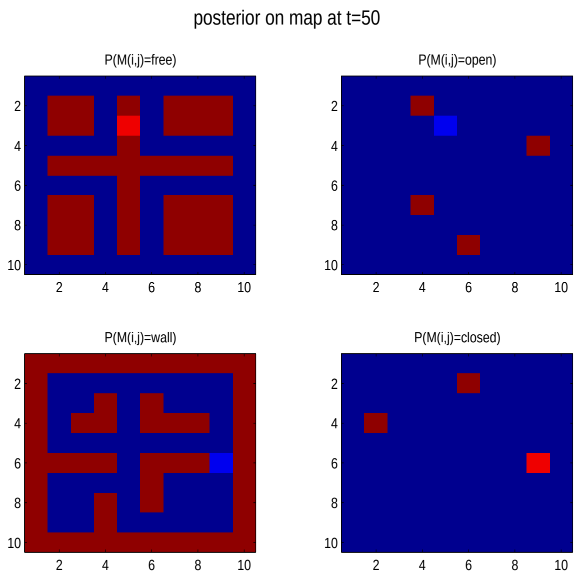


Figure 11. Results of RBPF on the 2D grid world using 200 particles.

the other "colors" remain static; hence the cell transition matrix T_M is

$$\begin{pmatrix} 1-p_c & p_c & 0 & 0 \\ p_c & 1-p_c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The robot observes a 3×3 neighborhood centered on its current location. The total probability that each pixel gets misclassified is $p_o = 0.1$. The robot can move north, east, south or west; there is a $p_a = 0.1$ chance it will accidentaly move in a direction perpendicular to the one specified by U_t.

We use a control policy that alternates between exploring new territory when it is confident of its location, and returning to familiar territory to relocalize itself when the entropy of $P(L_t|y_{1:t})$ becomes too high c.f., (Fox, Burgard and Thrun 1998): see (Murphy 2000) for details.

The results of applying the RBPF algorithm to this problem, using 200 particles, are shown in Figure 11. We see that by time 50, it has learnt an almost perfect map, even though the state-space has size 2^{100}.

5 Conclusions and future work

We have shown how to exploit "tractable substructure" in certain kinds of DBNs by combining particle filtering with exact inference. In the future, we hope to find more applications of this technique, and to extend the algorithm to the batch (offline) case.

References

Boutilier, C., Friedman, N., Goldszmidt, M. and Koller, D. (1996). Context-specific independence in Bayesian networks, *Proc. of the Conf. on Uncertainty in AI.*

Boyen, X. and Koller, D. (1998). Tractable inference for complex stochastic processes, *Proc. of the Conf. on Uncertainty in AI.*

Casella, G. and Robert, C. P. (1996). Rao-Blackwellisation of sampling schemes, *Biometrika* **83**(1): 81–94.

Chen, R. and Liu, S. (1999). Mixture Kalman filters, *Submitted.*

Cowell, R. G., Dawid, A. P., Lauritzen, S. L. and Spiegelhalter, D. J. (1999). *Probabilistic Networks and Expert Systems*, Springer.

Doucet, A. (1998). On sequential simulation-based methods for Bayesian filtering, *Technical Report CUED/F-INFENG/TR 310*, Department of Engineering, Cambridge University.

Doucet, A., de Freitas, N., Murphy, K. and Russell, S. (2000). Rao-blackwellised particle filtering for dynamic Bayesian networks. Submitted.

Doucet, A., Gordon, N. and Krishnamurthy, V. (1999). Particle filters for state estimation of jump markov linear systems, *Technical report*, Department of Engineering, Cambridge University.

Fox, D., Burgard, W. and Thrun, S. (1998). Active Markov localization for mobile robots, *Robotics and Autonomous Systems.*

Ghahramani, Z. and Jordan, M. (1997). Factorial hidden Markov models, *Machine Learning* **29**: 245–273.

Julier, S. and Uhlmann, J. (1997). A new extension of the Kalman filter to nonlinear systems, *Proc. of AeroSense: The 11th Intl. Symp. on Aerospace/Defence Sensing, Simulation and Controls.*

Moravec, H. and Elfes, A. (1985). High resolution maps from wide angle sonar, *IEEE Intl. Conf. on Robotics and Automation.*

Murphy, K. P. (1999). A variational approximation for Bayesian networks with discrete and continuous latent variables, *Proc. of the Conf. on Uncertainty in AI.*

Murphy, K. P. (2000). Bayesian map learning in dynamic environments, *in* S. Solla, T. Leen and K.-R. Müller (eds), *Advances in Neural Information Processing Systems 12*, MIT Press, pp. 1015–1021.

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

Smith, R., Self, M. and Cheeseman, P. (1988). Estimating uncertain spatial relationships in robotics, *in* Lemmer and Kanal (eds), *Uncertainty in Artificial Intelligence*, Vol. 2, Elsevier, pp. 435–461.

Smyth, P., Heckerman, D. and Jordan, M. I. (1997). Probabilistic independence networks for hidden Markov probability models, *Neural Computation* **9**(2): 227–269.

Thrun, S., Burgard, W. and Fox, D. (1998). A probabilistic approach to concurrent mapping and localization for mobile robots, *Machine Learning* **31**: 29–53.