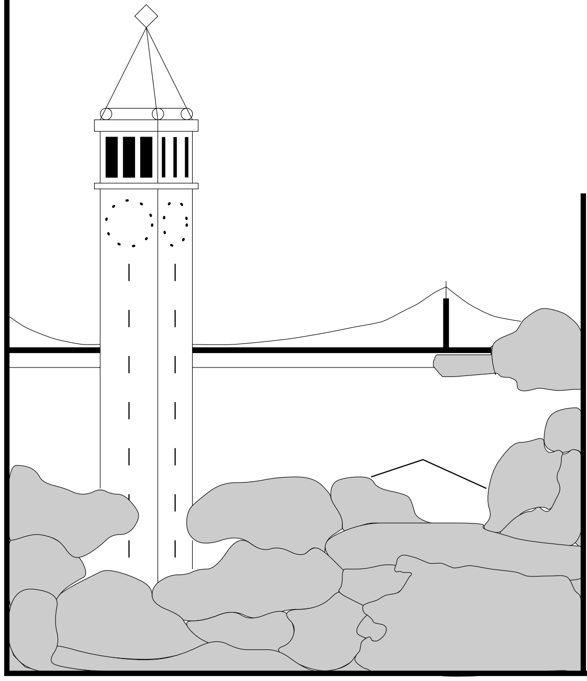
Thin Junction Tree Filters for Simultaneous Localization and Mapping

Mark A. Paskin



Report No. UCB/CSD-02-1198

September 2002

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Thin Junction Tree Filters for Simultaneous Localization and Mapping

Mark A. Paskin
paskin@cs.berkeley.edu

Abstract

Simultaneous Localization and Mapping is a fundamental problem in mobile robotics: while a robot navigates in an unknown environment, it must incrementally build a map of its surroundings and localize itself within that map. Traditional approaches to the problem are based upon Kalman filters, but suffer from complexity issues: first, the belief state grows quadratically in the size of the map; and second, the filtering operation can take time quadratic in the size of the map. I present a linear-space filter that maintains a tractable approximation of the filtered belief state as a thin junction tree. The junction tree grows under measurement and motion updates and is periodically "thinned" to remain tractable. The time complexity of the filter operation is linear in the size of the map. I also present simple enhancements that permit constant-time approximate filtering.

1 Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in mobile robotics: while a robot navigates in an unknown environment, it must incrementally build a map of its surroundings and localize itself within that map [Thrun, 2002]. Traditional approaches to the problem are based upon Kalman filters, but suffer from complexity issues: first, the belief state grows quadratically in the size of the map; and second, the filtering operation can take time quadratic in the size of the map.

In this technical report, I present a novel approximate filtering technique that avoids the quadratic space and time requirements of the Kalman filter technique. This approach is based upon representing the filtered belief state at each time step with a consistent junction tree. When the size of the junction tree grows beyond some predefined size, it is "thinned", using a maximum likelihood approximation. Thus, the size of the junction tree and the complexity of inference remain linear in the size of the map. Further approximation can be employed to obtain a constant-time filter operation, at the expense of delaying the incorporation of observations into the majority of the map.

The remainder of the introduction presents the traditional Kalman filtering approach to the SLAM problem and surveys existing approaches to improving the complexity of SLAM filtering. Section 2 recasts the filtering problem in terms of a graphical model and describes how the structure and parameters of the graphical model evolve over time. Section 3 describes how these changes can be applied incrementally to a junction tree that corresponds to the graphical model. The size of this junction tree grows rapidly; Section 4 describes a technique of "thinning" the junction tree in order to reduce the complexity of inference. Putting these results together, Section 5 describes a linear-time and linear-space approximate filter for the SLAM problem, and also how further approximations can be introduced to obtain a filter operation that is frequently or always constant-time. Finally, Section 6 concludes by drawing connections between the present proposal and other approaches.

1.1 The Kalman filter approach

It is natural to formulate SLAM as a filtering problem, where the hidden state is the map of the world, including the locations of all landmarks and the pose of the robot. Specifically, our state variable at time t is

$$\boldsymbol{m}_t = \begin{bmatrix} \mathbf{x}_t \\ \boldsymbol{\ell}_1 \\ \vdots \\ \boldsymbol{\ell}_{n_t} \end{bmatrix} \quad (1)$$

where $\boldsymbol{x}_t$ is the pose of the robot at time t, $\boldsymbol{\ell}_i$ is the state of landmark i (landmarks are assumed stationary) and n_t is the number of landmarks observed up to time t.[1]

The traditional filtering approach endows $\boldsymbol{m}$ with a multivariate Gaussian distribution. In the moment parameterization, this is given by

$$p(\boldsymbol{m}) = \frac{1}{Z} \exp\left\{ -\frac{1}{2}(\boldsymbol{m} - \mu)^T \Sigma^{-1} (\boldsymbol{m} - \mu) \right\} \quad (2)$$

[1] Boldface roman letters denote random variables and the corresponding plain letters denote specific values for these variables; parameters are denoted by Greek letters. Where it will not cause confusion, time subscripts are omitted.

where μ is the expectation of $\boldsymbol{m}$, Σ is its covariance, and $Z = (2\pi)^{N/2}(\det\Sigma)^{1/2}$ is a normalization constant (N being the length of the vector $\boldsymbol{m}$). The Gaussian assumption, in combination with linear (or more generally, affine) Gaussian measurement and motion models, allows us to use a Kalman filter to maintain the belief state over time. There are two operations the filter must support: measurement updates (in which the robot observes some landmarks), and motion updates (in which the robot moves and receives an odometry measurement).

At time step t, the robot makes k_t landmark measurements, which we will denote by $z_t^1, z_t^2, \ldots, z_t^{k_t}$. Landmark measurement z_t^i originates from landmark δ_t^i, where $\boldsymbol{\delta}_t$ is the data association variable at time t. Given δ_t, landmark measurements are modeled as noisy affine functions of the state:

$$\boldsymbol{z}_t^i \quad = \quad \hat{z}_t^i + A(\boldsymbol{x}_t - \mu^x) + B(\boldsymbol{\ell}_j - \mu^{\ell_j}) + \boldsymbol{u} \qquad (3)$$

where $j = \delta_t^i$ and $\boldsymbol{u} \sim \mathcal{N}(\mathbf{0}, R)$ is a white noise variable. Thus, all measurements are conditionally independent given $\boldsymbol{m}_t$ and $\boldsymbol{\delta}_t$.

When the data association $\boldsymbol{\delta}_t$ is unknown, it is popular to approximate it with its maximum likelihood value under $p(\boldsymbol{m}_t)$; i.e., we choose δ_t to be that value that associates each measurement with the landmark that was most likely to generate it. Since all measurements are conditionally independent given the map and the data association, this amounts to evaluating the likelihoods $p(z_t^i \,|\, \boldsymbol{x}_t, \boldsymbol{\ell}_j, \boldsymbol{\delta}_t^i = j)$ for all measurements i and landmarks j, and performing a global matching (using, say, the Hungarian algorithm).

New landmarks can be added to the map by a technique called *gating*, wherein measurements whose likelihoods under all data associations are below some threshold cause new landmarks to be added to the map. In this case, the landmark state is added to the map with an uninformative (infinite variance, zero covariance) prior, and the measurement update yields an informed posterior estimate of its state. Landmarks that were erroneously added to the map can be detected by recording how frequently they are observed and then removed by marginalizing them out of the map.

After observing the landmarks, the robot moves based upon a noisy affine model:

$$\boldsymbol{x}_{t+1} \quad = \quad \hat{x}_{t+1} + C(\boldsymbol{x}_t - \mu^{x_t}) + \boldsymbol{v} \qquad (4)$$

where $\boldsymbol{v} \sim \mathcal{N}(\mathbf{0}, S)$ is a white noise variable. (The parameters of this model, $\hat{x}_{t+1}$, C, and S, may depend implicitly upon a control u_t applied at time t.) Then it obtains an odometry measurement y_{t+1}, which is a noisy affine function of its new state:

$$\boldsymbol{y}_{t+1} \quad = \quad \hat{y}_{t+1} + D(\boldsymbol{x}_{t+1} - \mu^{x_{t+1}}) + \boldsymbol{w} \qquad (5)$$

where $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, T)$ is a white noise variable.

In the common case where the measurement and motion functions are not affine, it is standard to employ either the Extended Kalman Filter (EKF), in which these functions are approximated by a first-order Taylor expansion about their expected input values, or the Unscented Kalman Filter (UKF), in which the posterior distribution of the function output is approximated by a conditional Gaussian distribution [Wan and van der Merwe, 2000]. Either technique results in affine (approximate) measurement and motion equations. The UKF is typically more precise than the EKF (and simpler, as it does not require computation of Jacobians), but it also requires the marginal distribution over the function's input arguments.

Using a Kalman filter to recursively compute $p(\boldsymbol{m}_T \,|\, z_{1:T}, y_{1:T})$, our filtered estimate of the current map given all of the measurements, quickly runs into two problems. First, the representation of this belief state (or more specifically, its covariance matrix) grows quadratically with the number of landmarks. Second, the time complexity of recursively computing $p(\boldsymbol{m}_T \,|\, z_{1:T}, y_{1:T})$ from $p(\boldsymbol{m}_{T-1} \,|\, z_{1:T-1}, y_{1:T-1})$, z_T and y_T (via the Kalman filtering equations) also grows quadratically in the number of landmarks. These problems make the Kalman filter inapplicable to large scale mapping problems, and give rise to the need for principled, efficient approximations.

1.2 Approaches to reducing the complexity

Significant interest in this problem has led to a number of approaches. One family of techniques involves using approximate inference. From the model description above, we know that conditioned on the path of the robot, the state estimates of all the landmarks are independent of each other. This motivates a Rao-Blackwellized scheme, in which a particle representation is used for estimates of the robot's path, and the estimates of the landmarks are represented exactly and independently, conditioned on each particle [Doucet *et al.*, 2000]. This representation, combined with a clever particle representation that avoids unnecessary copying of the landmark estimates, comprises the FastSLAM algorithm, which has an impressively fast $O(k \log n)$ time filter operation (where k is the number of particles) [Montemerlo *et al.*, 2002]. There primary drawback of this approach is that each particle has a different view of the map, including the estimates and even the number of landmarks; integrating these views to obtain a single map is nontrivial, and more importantly, data association must be performed for each particle independently. While this gives the representation a great deal of flexibility (in that the model need not commit to a particular maximum likelihood data association), it also introduces a significant computational burden (because data association can be expensive).

Rather than using approximate inference on the exact model, we can also use exact inference on tractable approximations of the true model. Another family of techniques involves a decomposition of the map into a set of submaps, each of which maintains its own estimate of the robot's state. Leonard and Feder (2000) present a heuristic first attempt at such a system, and Guivant and Nebot (2000) present a more sophisticated technique. The chief disadvantage of such approaches is that information cannot pass between the submaps; as a result, how the world is divided can significantly affect the results.

Sparse Extended Information Filters (SEIF) [Thrun *et al.*, 2002] represent the belief state using the canonical parameterization of the multivariate Gaussian rather than the moment parameterization of (2). In the canonical parameterization, we have

$$p(\boldsymbol{m}) \quad = \quad \exp\left\{a + \eta^T \boldsymbol{m} - \frac{1}{2}\boldsymbol{m}^T \Lambda \boldsymbol{m}\right\} \qquad (6)$$

where $a = -(N \log(2\pi) - \det \Lambda + \eta^T \Lambda^{-1} \eta)/2$ is the normalization constant. (The moment parameters are related to the canonical parameters by $\Sigma = \Lambda^{-1}$ and $\mu = \Lambda^{-1}\eta$.) SEIF is based upon the observation that even if Σ is dense, Λ may be sparse or many of its entries may be small, and that when Λ is sparse, measurement and motion updates can be made to happen very efficiently. Thus, at each time step SEIF selectively zeros-out small entries of Λ, and thereby obtains a constant-time approximate filter update.[2]

In spite of its unbeatable time complexity, SEIF has some representational problems. First, the means of the robot and landmark states (which are required to implement the EKF) can be recovered only approximately in constant time (by a hill-climbing procedure). Second, the marginal covariances of the robot and landmark states (which are required for maximum likelihood data association and to implement the UKF) cannot be recovered efficiently. And third, while a constant time update seems desireable, some measurement updates *should* cause a linear-time update. Consider the case where the robot "closes the loop", reobserving a landmark after a long period of mapping unknown territory: all landmark location estimates can be improved because the robot learns more about its current location, and therefore more about its tour and thus the locations of the landmarks it observed.

Like the submap approach and SEIF, the approach presented here also employs exact inference over an approximate model. Indeed, as we will see in Section 6, the present approach can be viewed as a generalization of both the submap approach and SEIF. Importantly, it improves significantly over both techniques, culminating in an efficient, principled approximation algorithm. The easiest way to motivate the present approach is in the graphical model framework.

[2]It is worth noting that the maximum likelihood approximation to Λ with an arbitrary sparsity pattern can be computed by the Iterative Proportional Fitting algorithm [Speed and Kiiveri, 1986]; however, as this algorithm is iterative, it is unsuitable for a projection operation that must be applied at every time step. SEIF avoids this cost by allowing new edges to be introduced when others are taken away; these new edges do not significantly affect the time complexity.

2 A graphical model perspective

As we have seen, the measurement and motion models for the SLAM problem exhibit strong conditional independencies. Because of this significant conditional independence structure, we will represent the belief state via a graphical model—more specifically, a dynamic Bayesian network (DBN).[3] Figure 1 depicts how a directed graphical model may be used to represent three time steps of an example SLAM inference problem. For example, each measurement $\boldsymbol{z}_t^i$ depends only upon the state of the robot $\boldsymbol{x}_t$ and the state of the landmark that was observed $\boldsymbol{\ell}_{\delta_t^i}$ (assuming a known data association δ_t). Hence, in Figure 1, each observed measurement node takes only these nodes as parents. Also, the state of the robot at time $t+1$ depends only upon its previous state $\boldsymbol{x}_t$. Finally, the odometry measurement $\boldsymbol{y}_t$ at time t depends only upon the current state of the robot $\boldsymbol{x}_t$.

In this section, we discuss how the filtered belief state can be represented as a graphical model that evolves over time. We start with the structural changes caused by measurement and motion updates and "roll-up", the process of marginalizing out past state variables. Then we describe the changes these updates cause to the parameters of the graphical model.

2.1 Structural updates during filtering

Representing the belief state using graphical models has some advantages over the traditional multivariate Gaussian representation. The first advantage is that the quadratic growth of the size of the belief state is easily understood by viewing filtering (which is nothing more than marginalizing out variables from past time steps) as an instance of the variable elimination algorithm [Russell and Norvig, 2002].

Let us begin with the first time slice of our DBN. Figure 2 depicts the landmark measurement portion of the first time step. As we can see, eliminating each measurement node adds an elimination edge between the robot's state and those of the observed landmarks. Intuitively, each measurement imposes a weak constraint between the state of the robot and the observed landmark, and this coupling is reflected by the undirected edge. We will call the set of landmarks that are conditionally dependent upon $\boldsymbol{x}_t$ the *active landmarks*.

Thus far, our model is still tractable, since the graph is in the form of a tree. However, when we perform a motion update (depicted in Figure 3), the situation changes. Incorporating the odometry measurement can be done in constant time, but eliminating the robot's previous state variable creates an elimination clique over the robot's new state variable and *all* of the active landmarks. The intuition here is more subtle, but worth understanding: in the previous model, weak constraints existed between all pairs of active landmarks via their individual couplings to the previous state of the robot;

[3]See [Cowell *et al.*, 1999] for a rigorous introduction to graphical models, and [Russell and Norvig, 2002] for a gentle introduction to directed graphical models, including DBNs.

when the robot's previous state variable is eliminated, these indirect couplings must be represented by explicit edges. If there are k active landmarks at time t, then the filtered distribution after the time-t motion update will have a clique of size $O(k)$; it will take $O(k^2)$ space to represent the parameters of this clique, and inference will require $O(k^3)$ operations.

2.2 Parameter updates during filtering

While the graphical model we have given in Figure 1 is directed, we will focus more on undirected grapical models: they are the natural method of representing conditional independence statements about multivariate Gaussian distributions, and, the graphical models we are left with after measurement and motion updates are undirected (see Figures 2(c) and 3(d)).

In order to represent our belief state with an undirected graphical model, we must express the multivariate Gaussian distribution (6) as a normalized product of potential functions. So that our notation for the state vector's components is uniform, let $\boldsymbol{m}^{(0)} = x$ and $\boldsymbol{m}^{(i)} = \boldsymbol{\ell}_i$, let $\eta^{(i)}$ be the subvector of η corresponding to $\boldsymbol{m}^{(i)}$, and let $\Lambda^{(ij)}$ be the submatrix of Λ corresponding to $\boldsymbol{m}^{(i)}$ and $\boldsymbol{m}^{(j)}$. Then we can write

$$p(\boldsymbol{m}) = \frac{1}{Z} \prod_{i=0}^{n} \psi_i(\boldsymbol{m}^{(i)}) \prod_{j=i+1}^{n} \psi_{ij}(\boldsymbol{m}^{(i)}, \boldsymbol{m}^{(j)}) \tag{7}$$

where

$$\psi_i(\boldsymbol{m}^{(i)}) = \exp\left\{ (\eta^{(i)})^T \boldsymbol{m}^{(i)} - \frac{1}{2} (\boldsymbol{m}^{(i)})^T \Lambda^{(ii)} \boldsymbol{m}^{(i)} \right\} \tag{8}$$

and

$$\psi_{ij}(\boldsymbol{m}^{(i)}, \boldsymbol{m}^{(j)}) = \exp\left\{ -(\boldsymbol{m}^{(i)})^T \Lambda^{(ij)} \boldsymbol{m}^{(j)} \right\} \tag{9}$$

and $Z = e^{-a}$ is the normalization constant. Thus, all the potential functions in the graphical model are either unary (*node potentials*) or binary (*edge potentials*). We also have the important interpretation that if $\Lambda^{(ij)} = \mathbf{0}$, then $\psi_{ij}(\boldsymbol{m}^{(i)}, \boldsymbol{m}^{(j)})$ is unity (and therefore superfluous), meaning there is no edge between $\boldsymbol{m}^{(i)}$ and $\boldsymbol{m}^{(j)}$ in the graphical model. We will alternatingly view the parameters of a Gaussian graphical model as a set of node potentials ψ_i and edge potentials ψ_{ij}, or as the vector η and matrix Λ that underlie them. Thus, when we speak of "multiplying in" new potentials, this corresponds to straightforward additive updates to η and Λ.

There are three types of changes made to the graphical model during filtering: measurement updates, motion updates, and roll-up, where we marginalize out the robot's state at the previous time step. We now show how each of these updates can be implemented by multiplying in new potentials into the current graphical model. First, we present the following fact:

Proposition 1. *Let $\boldsymbol{q}$ be a random n-vector distributed according to $p(\boldsymbol{q}, \boldsymbol{r})$ (for any arbitrary random variable $\boldsymbol{r}$) and let*

$$\boldsymbol{s} = s_0 + L\boldsymbol{q} + \boldsymbol{t} \tag{10}$$

where $\boldsymbol{t} \sim \mathcal{N}(0, U)$. Then

$$p(\boldsymbol{q}, \boldsymbol{r}, \boldsymbol{s}) \propto p(\boldsymbol{q}, \boldsymbol{r}) \cdot \phi_q(\boldsymbol{q}) \cdot \phi_s(\boldsymbol{s}) \cdot \phi_{qs}(\boldsymbol{q}, \boldsymbol{s}) \tag{11}$$

where

$$\phi_q(\boldsymbol{q}) = \exp\left\{ -s_0^T U^{-1} L\boldsymbol{q} - \frac{1}{2} \boldsymbol{q}^T L^T U^{-1} L \boldsymbol{q} \right\} \tag{12}$$

$$\phi_s(\boldsymbol{s}) = \exp\left\{ s_0^T U^{-1} \boldsymbol{s} - \frac{1}{2} \boldsymbol{s}^T U^{-1} \boldsymbol{s} \right\} \tag{13}$$

$$\phi_{qs}(\boldsymbol{q}, \boldsymbol{s}) = \exp\left\{ \boldsymbol{s}^T U^{-1} L \boldsymbol{q} \right\} \tag{14}$$

Furthermore,

$$p(\boldsymbol{q}, \boldsymbol{r} \mid s) \propto p(\boldsymbol{q}, \boldsymbol{r}) \cdot \phi'_q(\boldsymbol{q}) \tag{15}$$

where

$$\phi'_q(\boldsymbol{q}) = \exp\left\{ (s - s_0)^T U^{-1} L\boldsymbol{q} - \frac{1}{2} \boldsymbol{q}^T L^T U^{-1} L \boldsymbol{q} \right\} \tag{16}$$

Proof. (Sketch) $p(\boldsymbol{s} \mid \boldsymbol{q})$ is a conditional Gaussian distribution with mean $s_0 + L\boldsymbol{q}$ and covariance U. Thus, the canonical parameters of its conditional Gaussian distribution are $U^{-1}(s_0 + L\boldsymbol{q})$ and U^{-1}. Writing out the distribution using these canonical parameters and collecting terms reveals that $p(\boldsymbol{s} \mid \boldsymbol{q})$ is proportional to the product $\phi_q(\boldsymbol{q}) \cdot \phi_s(\boldsymbol{s}) \cdot \phi_{qs}(\boldsymbol{q}, \boldsymbol{s})$, which in combination with the chain rule yields (11). (15) is obtained by simply substituting $\boldsymbol{s} = s$ and eliminating constant terms. □

The import of this proposition is that we can update the graphical model to reflect landmark measurements, odometry measurements, and robot motion by multiplying in small, simple potentials. We describe this process in more detail below.

Landmark measurement updates

Since all measurements are assumed independent, we will deal with the case of a single measurement z originating from the landmark $\boldsymbol{\ell}$. By applying Proposition 1 with $\boldsymbol{s} = \boldsymbol{z}$ and $\boldsymbol{q} = (\boldsymbol{x}, \boldsymbol{\ell})$ (and $s_0 = \hat{z} - A\mu^x - B\mu^\ell$ and L the block-diagonal matrix $\mathrm{diag}(A, B)$), we find that we can condition on z by multiplying the binary potential

$$\phi'(\boldsymbol{x}, \boldsymbol{\ell}) = \exp\left\{ \begin{array}{l} (z - \hat{z} + A\mu^x + B\mu^\ell)^T R^{-1} (A\boldsymbol{x} + B\boldsymbol{\ell}) \\ -\frac{1}{2} (A\boldsymbol{x} + B\boldsymbol{\ell})^T R^{-1} (A\boldsymbol{x} + B\boldsymbol{\ell}) \end{array} \right\}$$

into the graphical model. Equivalently, we can update the parameters η and Λ as follows:

$$\begin{array}{rcl} \eta_x & \overset{+}{\leftarrow} & (z - \hat{z} + A\mu^x + B\mu^\ell)^T R^{-1} A \\ \eta_\ell & \overset{+}{\leftarrow} & (z - \hat{z} + A\mu^x + B\mu^\ell)^T R^{-1} B \\ \Lambda_{xx} & \overset{+}{\leftarrow} & A^T R^{-1} A \\ \Lambda_{\ell\ell} & \overset{+}{\leftarrow} & B^T R^{-1} B \\ \Lambda_{x\ell} & \overset{+}{\leftarrow} & A^T R^{-1} B \end{array} \tag{17}$$

where $a \overset{+}{\leftarrow} b$ means "increment a by the value b." We can see here how the new edges in Figure 2(c) arise, since $\Lambda_{x\ell}$ becomes non-zero after this update.

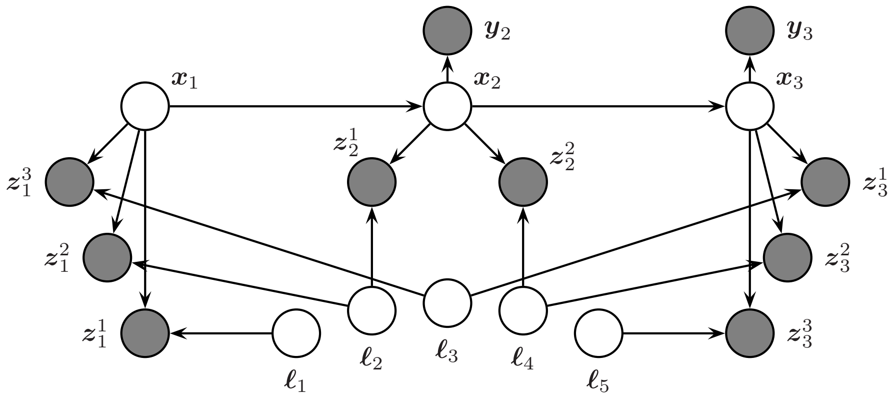


Figure 1: A DBN representing three time steps of an example SLAM problem. The edges from landmarks to measurements are determined by data association; e.g., $\delta_2^1 = 2$ in this example, and so there is an edge from ℓ_2 to z_2^1.

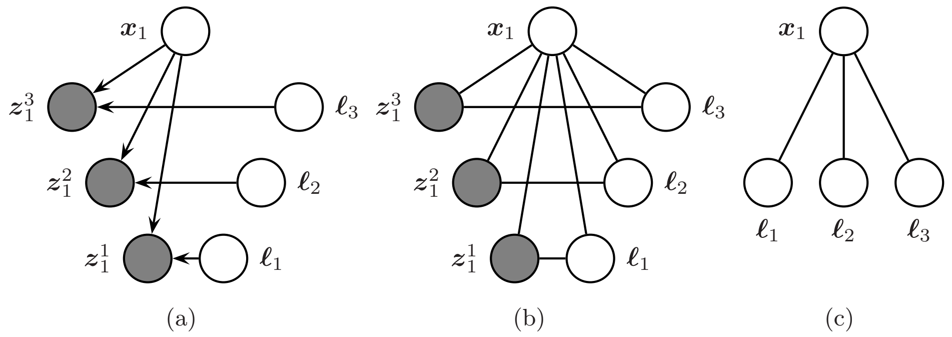


Figure 2: Incorporating landmark measurements. (a) At time one, three landmark measurements, z_1^1, z_2^1, and z_3^1, are made of landmark states ℓ_1, ℓ_2, and ℓ_3. (b) Moralizing the directed graph produces the undirected graphical model in which the landmarks are coupled with the robot state. (c) Eliminating the observed measurement variables leaves a graph in which the robot state is conditionally dependent upon the states of all landmarks it has measured.

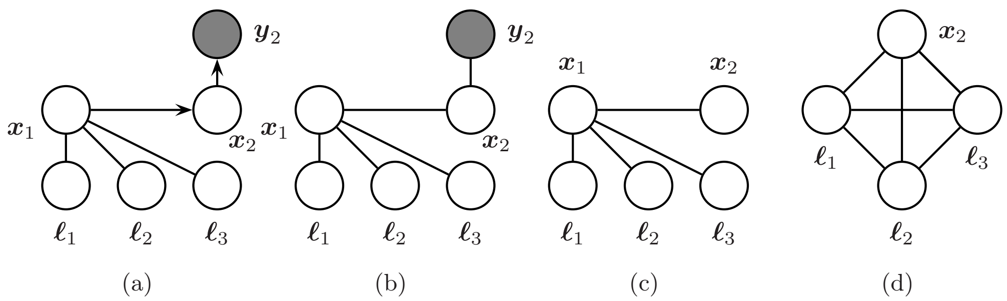


Figure 3: Incorporating robot motion. (a) The robot's state at the next time step and its odometry measurement y_2 are added to the model. (b) Moralizing this graph adds no new edges. (c) Eliminating the odometry variables adds no new edges. (d) Eliminating the state at the previous time step adds an elimination clique that includes the current state x_2 and all the active landmarks.

Motion update

The motion update consists of adding a new node $\boldsymbol{x}_{t+1}$ and connecting it to $\boldsymbol{x}_t$ via the potential function $p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, y_{t+1})$, and then incorporating the odometry measurement $p(y_{t+1} \mid \boldsymbol{x}_{t+1})$. Using Proposition 1, we find that the motion model $p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t)$ can be multiplied into the graphical model via the following updates:

$$\begin{array}{rcl} \eta_{x_t} & \overset{+}{\leftarrow} & (C\mu^{x_t} - \hat{x}_{t+1})^T S^{-1} C \\ \eta_{x_{t+1}} & \leftarrow & (\hat{x}_{t+1} - C\mu^{x_t}) S^{-1} \\ \Lambda_{x_t x_t} & \overset{+}{\leftarrow} & C^T S^{-1} C \\ \Lambda_{x_{t+1} x_{t+1}} & \leftarrow & S^{-1} \\ \Lambda_{x_t x_{t+1}} & \leftarrow & C^T S^{-1} \end{array} \tag{18}$$

where $a \leftarrow b$ means "assign a the value b." A second application of the proposition gives that the odometry model $p(y_{t+1} \mid \boldsymbol{x}_{t+1})$ can be multiplied in via the following updates:

$$\begin{array}{rcl} \eta_{x_{t+1}} & \overset{+}{\leftarrow} & (y_{t+1} + D\mu^{x_{t+1}} - \hat{y}_{t+1})^T T^{-1} D \\ \Lambda_{x_{t+1} x_{t+1}} & \overset{+}{\leftarrow} & D^T T^{-1} D \end{array} \tag{19}$$

Roll-up

To "roll up" the DBN, we must marginalize out the previous state variable $\boldsymbol{x}_t$. This too can be implemented by multiplying potentials onto the graphical model.

Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two vector variables that are jointly distributed according to $\mathcal{N}^{-1}(\eta, \Lambda)$. Then $\boldsymbol{a}$ is marginally distributed according to $\mathcal{N}^{-1}(\eta^{\mathrm{m}}, \Lambda^{\mathrm{m}})$, where

$$\eta^{\mathrm{m}} = \eta_a - \Lambda_{ab} \Lambda_{bb}^{-1} \eta_a \tag{20}$$

$$\Lambda^{\mathrm{m}} = \eta_a - \Lambda_{ab} \Lambda_{bb}^{-1} \Lambda_{ba} \tag{21}$$

Because zero entries in Λ_{ab} correspond to missing edges in the graphical model, we can infer that only those entries of η and Λ corresponding to $\boldsymbol{b}$'s Markov blanket in $\boldsymbol{a}$ change. Let $\bar{\boldsymbol{b}} = \textsc{mb}(\boldsymbol{b})$; then marginalizing out $\boldsymbol{b}$ causes the parameters for $\bar{\boldsymbol{b}}$ to change as follows:

$$\eta_{\bar{b}} \overset{+}{\leftarrow} -\Lambda_{\bar{b}b} \Lambda_{bb}^{-1} \eta_b \tag{22}$$

$$\Lambda_{\bar{b}\bar{b}} \overset{+}{\leftarrow} -\Lambda_{\bar{b}b} \Lambda_{bb}^{-1} \Lambda_{b\bar{b}} \tag{23}$$

After this update is completed, we can discard all potentials involving $\boldsymbol{b}$ and eliminate the node from our graphical model. Note that although we have written this update in terms of the parameters η and Λ, we could equally well express it in terms of multiplying in node and edge potentials over the Markov blanket of $\boldsymbol{b}$.

3 Incremental junction tree maintenance

Exact inference in graphical models is frequently based upon message passing in a junction tree, which is a special data structure based upon the graphical model. In our context, the graphical model changes over time, and so we must consider methods of maintaining the junction tree incrementally over time. From the discussion above, we see that we require a junction tree representation that allows us to (1) add new node and edge potentials (for observations and motion) and to (2) marginalize out nodes (specifically, the robot state at the previous time step). In this section, we describe how all of these updates can be applied to the junction tree incrementally. We begin by reviewing some basic facts about junction trees [Cowell *et al.*, 1999].

Recall that a *junction tree* $\mathcal{T}$ of a graphical model G is a graph with vertex set $\mathcal{C}$ such that each vertex (or *cluster*) $C \in \mathcal{C}$ is a subset of the variables in G and the following properties hold:

1. *Singly-connected property:* $\mathcal{T}$ is a tree.
2. *Potential property:* Each subset of G's variables that is coupled by a potential function is present in at least one cluster of $\mathcal{T}$.
3. *Running-intersection property:* A variable $\boldsymbol{x}$ present in two clusters C_1 and C_2 of $\mathcal{T}$ is also present in all clusters on the path between C_1 and C_2.

$\mathcal{T}$ also has an associated set of *separators* $\mathcal{S}$, one per edge $\{C_i, C_j\}$, such that $S_{ij} = C_i \cap C_j$. Each cluster $C \in \mathcal{C}$ (and separator $S \in \mathcal{S}$) has an associated potential function ϕ_C (or ϕ_S) over its variables. The *charge on* $\mathcal{T}$ is defined to be

$$\phi_{\mathcal{T}}(\boldsymbol{v}) = \frac{\prod_{C \in \mathcal{C}} \phi_C(\boldsymbol{v}_C)}{\prod_{S \in \mathcal{S}} \phi_S(\boldsymbol{v}_S)} \tag{24}$$

$\mathcal{T}$ is *initialized* by setting all potential functions to unity and then multiplying each potential of the graphical model G into one of the clusters (which is possible by the potential property) and the inverse of the normalizing constant Z into an arbitrary cluster; then $\phi_{\mathcal{T}}(\boldsymbol{v}) = p(\boldsymbol{v})$. Inference in the graphical model G can be carried out by a message passing algorithm over $\mathcal{T}$ in which each cluster sends messages to its neighbors through the separators, and absorbs the messages from its neighbors, updating its potential. Passing messages always leaves the charge on $\mathcal{T}$ invariant; moreover when it is performed in an appropriate schedule, it updates the cluster and separator potentials so that they are the marginal distributions over their respective variables. A junction tree in this state is called *consistent*. (Passing messages when $\mathcal{T}$ is consistent does not affect the cluster potentials.) In the case of a Gaussian graphical model, the message passing algorithm takes time $O(d(\mathcal{T})^3)$, where $d(\mathcal{T})$ is the *diameter* of $\mathcal{T}$, i.e., the size of $\mathcal{T}$'s largest cluster.

We will adopt consistent junction trees as our representation of the belief state. This gives us efficient access the filtered marginals of any sets of variables contained in the same cluster.[4] To do this, we must develop tech-

[4]This is sufficient for most tasks involved in SLAM. The notable exception is when using the UKF for measurement updates or data association. In these cases, the UKF requires the joint distribution over $\boldsymbol{x}$ and $\boldsymbol{\ell}$, which may not reside in the same cluster. There are two solutions: the exact solution entails adding a vacuous edge to the junction tree (so that $\boldsymbol{x}$ and $\boldsymbol{\ell}$ are in the same cluster) and then passing messages to make $\mathcal{T}$ consistent; a (maximum likelihood) approximate solution is to use the product of their marginal distributions.

niques of updating consistent junction trees to reflect potential additions and variable marginalizations.

3.1 Adding node and edge potentials

Adding a node potential to the graphical model requires no updates to the structure of the junction tree; we need only multiply the potential into one of the clusters containing the variable in order to update the charge.

In contrast, it is more complicated to update the junction tree to reflect the addition of an edge potential between variables $\boldsymbol{x}$ and $\boldsymbol{y}$ in the graphical model. If $\boldsymbol{x}$ and $\boldsymbol{y}$ do not appear in the same cluster, then the potential property will be violated. In order to restore it, we can add $\boldsymbol{y}$ to any cluster C containing $\boldsymbol{x}$. However, this update may violate the running-intersection property, as C may not be a neighbor of another cluster containing $\boldsymbol{y}$. To restore this property, we can add an edge from C to any other cluster C' containing $\boldsymbol{y}$. However, this will violate the singly-connected property, which can be difficult to restore (although see [Draper, 1995] for techniques).

Instead, consider the following update: after adding $\boldsymbol{y}$ to C, we find the cluster C' that contains $\boldsymbol{y}$ and is closest to C (in path length), and add $\boldsymbol{y}$ to every cluster along the path between C and C'. The resulting junction tree does not violate the singly-connected property or the running-intersection property, and therefore is correct for the graph with the new edge. This update can be shown to be equivalent to loop-cutset conditioning on $\boldsymbol{y}$ [Shachter *et al.*, 1994]. To update the charge on the junction tree, we simply multiply the edge potential into the potential of C.

3.2 Maintaining consistency under edge additions

Even though the charge on the junction tree is correct after this update, it may no longer be consistent. In general, restoring consistency requires a round of message passing. However, in some important special cases, conditional independence properties present in the graphical model allow us to avoid message passing.

For example, when adding $\boldsymbol{x}_{t+1}$ to the model and multiplying on the potential $p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, y_{t+1})$, we need not pass messages from the (unique) clique containing $\boldsymbol{x}_{t+1}$ to any other cliques. This is because $\boldsymbol{x}_{t+1}$ is an unobserved (directed) leaf of the graphical model—a *barren* node—and therefore does not impact the marginal distributions of the other nodes [Geiger *et al.*, 1990]. As a result, we need only renormalize the cluster potential to make the junction tree consistent.

A similar, but more complicated situation occurs when observing a landmark for the first time. Intuitively, observing a landmark whose absolute position is unknown cannot give you any information about your location. In this case, we cannot appeal to a graphical criterion to avoid message passing, since the newly added landmark node $\boldsymbol{\ell}$ is not a barren child of $\boldsymbol{x}$ (see Figure 1). Rather, the independence of $\boldsymbol{x}$ (and all other variables) from $\boldsymbol{\ell}$ is an artifact of the completely uninformative prior on $\boldsymbol{\ell}$ before the measurement z is made. In this case, "explaining away" leads to $\boldsymbol{\ell}$ assuming all responsibility for z, and therefore $\boldsymbol{x}$'s distribution remains unchanged. This can easily be verified numerically: when ℓ is first added to the map, its marginal covariance with $\boldsymbol{x}$ is given by

$$\Sigma \quad = \quad \begin{bmatrix} \Sigma_{xx} & \mathbf{0} \\ \mathbf{0} & \Sigma_{\ell\ell} \end{bmatrix} \tag{25}$$

where $\Sigma_{\ell\ell} = \mathrm{diag}(\infty, \ldots, \infty)$ and the zero off-diagonal blocks reflect that our prior over $\boldsymbol{\ell}$ is uninformative. The posterior parameters of $\boldsymbol{x}$ and $\boldsymbol{\ell}$ given the measurement z are

$$\mu' \quad = \quad \mu + K(z - F\mu) \tag{26}$$

$$\Sigma' \quad = \quad (I - KF)\Sigma \tag{27}$$

where $F = [A^T B^T]^T$ is the combined measurement model and K is the Kalman gain matrix:

$$K \quad = \quad \Sigma F^T (F \Sigma F^T + R)^{-1} \tag{28}$$

We can demonstrate that the marginal distribution of $\boldsymbol{x}$ is unaffected by the measurement by showing that $\mu_x = \mu_x$ and $\Sigma'_{xx} = \Sigma_{xx}$. Consider the form of the Kalman gain matrix:

$$\begin{aligned}
K &= \Sigma F^T (F \Sigma F^T + R)^{-1} \\
&= \Sigma F^T (A \Sigma_{xx} A^T + B \Sigma_{\ell\ell} B^T + R)^{-1} \\
&= \Sigma F^T (A \Sigma_{xx} A^T + \boldsymbol{\infty} + R)^{-1} \\
&= \Sigma F^T (\boldsymbol{\infty})^{-1} \\
&= \Sigma F^T \mathbf{0} \\
&= \begin{bmatrix} \Sigma_{xx} A^T \\ \Sigma_{\ell\ell} B^T \end{bmatrix} \mathbf{0} = \begin{bmatrix} \Sigma_{xx} A^T \\ \boldsymbol{\infty} \end{bmatrix} \mathbf{0} = \begin{bmatrix} \mathbf{0} \\ \mathbf{?} \end{bmatrix}
\end{aligned}$$

where we have used the loose notation $\boldsymbol{\infty}$ for a matrix with norm ∞. (We have assumed that B is not degenerate.) The lower block of K is, of course, a reasonable constant, but a limit argument would be required to compute it. Plugging this expression into (26) and (27) indeed give that the marginal distribution for $\boldsymbol{x}$ does not change. Thus, to perform such an update, we simply multiply on the observation potential $p(z \mid \boldsymbol{x}, \boldsymbol{\ell})$ onto the appropriate cluster and renormalize it to make the junction tree consistent.

In fact, the only type of edge addition in the SLAM application that requires message passing is when the robot reobserves a landmark. That a full round of message passing is required is intuitive, since the reobserved landmark was previously (indirectly) correlated to all other landmarks (and the robot); learning more about the its position gives us more information about the locations of the robot and all other landmarks in the map. Later we will leverage this fact, by delaying our use of this additional information, to obtain a filter operation that is commonly constant time.

3.3 Marginalizing nodes

As described in Section 2.2, marginalizing out a variable $\boldsymbol{x}$ creates a new potential over the Markov blanket of $\boldsymbol{x}$, MB($\boldsymbol{x}$). While we could add each of the edges of this clique independently using the technique described above (because the update potential factors into a product of unary and binary potentials), we instead derive a simpler and more efficient method.

In order to preserve the potential property, the structure of the junction tree must be updated so that it has a cluster containing $\text{MB}(\boldsymbol{x})$. The Markov blanket of $\boldsymbol{x}$ may not be efficiently computable from the junction tree, but we are guaranteed that it is a subset of the union of all clusters containing $\boldsymbol{x}$ (since the junction tree has the potential property). Because the junction tree also has the running intersection property, all clusters containing $\boldsymbol{x}$ must lie in a subtree of the junction tree. We can therefore merge all of these clusters to obtain a new cluster that contains $\boldsymbol{x}$ and $\text{MB}(\boldsymbol{x})$; this new cluster will inherit all of the edges incident to all of the merged clusters. It is straightforward to verify that the resulting junction tree is valid for the original graphical model.

We can preserve the charge on the junction tree during this cluster merging by setting the potential on the new cluster to the product of the potentials of the merged clusters divided by the product of the potentials of the eliminated separators. Then, to marginalize $\boldsymbol{x}$ out of the junction tree, we simply marginalize $\boldsymbol{x}$ out of this new cluster, which is the only cluster that contains it. If the junction tree was previously consistent, then this operation preserves its consistency without message passing.

4 Thinning the junction tree

As discussed in Section 2.1, the size of the largest clique of the junction tree will grow linearly in the number of landmarks. Thus, if we are to avoid cubic-time inference, we must resort to an approximation. In this section, we consider a method of thinning the junction tree so that the complexity of message passing remains manageable. The approach here is related to that of Kjærulff (1993); that work describes how removing an edge from a graphical model can sometimes result in a thinner junction tree. The approach presented here is simpler and more direct, as it operates on the junction tree without making reference to the underlying graphical model.

4.1 Variable contraction

The complexity of message passing is determined by the sizes of the clusters in the junction tree, and therefore our goal will be to define a thinning operation that reduces the size of clusters in the junction tree. We now describe such an operation, which we call *variable contraction.*

Let $\boldsymbol{x}$ be a variable that appears in more than one cluster of the junction tree $\mathcal{T}$. $\boldsymbol{x}$ induces the subtree of $\mathcal{T}$ containing those clusters in which $\boldsymbol{x}$ resides; let this subtree be denoted $\mathcal{T}_{\boldsymbol{x}}$. When we contract on $\boldsymbol{x}$, we remove $\boldsymbol{x}$ from a leaf C of $\mathcal{T}_{\boldsymbol{x}}$ (and its corresponding separator S in that subtree). If C becomes empty, we remove it.

Variable contraction has some attractive properties:

1. It preserves the singly-connectedness of the junction tree, as well as the running intersection property. Thus, the new junction tree is a valid junction tree for some graph, although not that of the original junction tree.
2. If the junction tree was consistent before a variable contraction, then it can easily be made consistent afterwards by marginalizing out $\boldsymbol{x}$ from the potentials of C and S. Thus variable contraction is a completely local operation in the junction tree.
3. In combination with cluster duplication (in which we duplicate a cluster C and attach the clone C' to C), variable contraction can reduce the size of an arbitrary cluster C by an arbitrary amount. To do this, pick a variable $\boldsymbol{x}$ in C; if $\boldsymbol{x}$ appears only in C, then we can (a) clone C to obtain C', (b) contract $\boldsymbol{x}$ from C to C', and then (c) contract some other variable $\boldsymbol{y}$ from C'; otherwise, when $\boldsymbol{x}$ appears elsewhere, we can contract $\boldsymbol{x}$ from other clusters until C is a leaf of $\mathcal{T}_{\boldsymbol{x}}$ and then contract $\boldsymbol{x}$ from C, reducing its size.

Most importantly, variable contraction can easily be motivated as a maximum likelihood approximation. By reverse engineering the graphical model corresponding to junction trees before and after we contract on a variable $\boldsymbol{x}$, we can develop an interpretation of variable contraction as a *maximum likelihood projection* to a more tractable family of graphical models.

A decomposable probability distribution p has the factored form

$$p(\boldsymbol{x}) = \frac{\prod_{C \in \mathcal{C}} p(\boldsymbol{x}_C)}{\prod_{S \in \mathcal{S}} p(\boldsymbol{x}_S)}$$

where each $C \in \mathcal{C}$ is a *maximal irreducible component* of p and each $S \in \mathcal{S}$ is a complete intersection of two irreducible components [Whittaker, 1989]. Thus, decomposable models have the very important property that they can be efficiently parameterized in terms of marginals over cliques of variables. The graphical model corresponding to a decomposable model is obtained by creating a clique over each $C \in \mathcal{C}$.

Any consistent junction tree can be viewed as a representation of a decomposable graphical model, since the expression for the charge of a consistent junction tree is exactly the same as the factored expression for a decomposable model. Thus, when we contract a variable $\boldsymbol{x}$ from some cluster C, this is equivalent to a maximum likelihood projection of one decomposable model onto another, simpler one. In particular, we are removing the edges between $\boldsymbol{x}$ and those variables in C that do not co-reside with $\boldsymbol{x}$ in some other cluster.

From this discussion, we see that while deleting single edges from the graphical model may be computationally intensive (requiring Iterative Proportional Fitting), deleting sets of edges (so that the resultant model is decomposable) is efficient. Moreover, a junction tree compactly describes sets of edges that can be efficiently deleted from a graphical model. Thus, in settings such as SLAM where a graphical model becomes denser over time, periodically thinning the junction tree via variable contraction is an attractive means of ensuring the model remains tractable.

4.2 Thinning cliques with contractions

Consider the problem of reducing of a given junction tree cluster C by one. (A solution to this problem can be applied recursively to thin the junction tree by an arbitrary amount.) Let C_{shared} be the subset of variables in C that appear elsewhere in the junction tree. (If $C_{\text{shared}} = \emptyset$, then we first clone C so that $C_{\text{shared}} = C$.) The question we must now ask is: which of the variables in C_{shared} should be contracted from C?

The following result, inspired by Theorem 11 of [Kjærulff, 1993], proves that the error introduced by contracting $\boldsymbol{x}$ from a cluster C can be computed using only the marginal over C:

Proposition 2. *Let the distribution $p(\boldsymbol{u})$ be represented by a junction tree $\mathcal{T}$ with cluster set $\mathcal{C}$ and separator set $\mathcal{S}$. Let $q(\boldsymbol{u})$ be the distribution represented by the junction tree $\mathcal{T}'$ obtained from $\mathcal{T}$ by contracting $\boldsymbol{x}$ from a leaf cluster C of $\mathcal{T}_{\boldsymbol{x}}$. Then $D(p \,||\, q) = I(\boldsymbol{x}; C \backslash \boldsymbol{x} \,|\, S \backslash \boldsymbol{x})$.*

Proof.

$$\begin{aligned}
D(p \,||\, q) &= \int p(u) \log \frac{p(U)}{q(U)} du \\
&= \int p(u) \log \frac{\prod_{C' \in \mathcal{C}} p(C')/q(C')}{\prod_{S' \in \mathcal{S}} p(S')/q(S')} du \\
&= \int p(u) \log \frac{p(C)q(S)}{q(C)p(S)} du \\
&= \int p(C) \log \frac{p(C)p(S \backslash \boldsymbol{x})}{p(C \backslash \boldsymbol{x})p(S)} dC \\
&= \int p(C) \log \frac{p(C \,|\, S)}{p(C \backslash \boldsymbol{x} \,|\, S \backslash \boldsymbol{x})} dC \\
&= H(C \backslash \boldsymbol{x} \,|\, S \backslash \boldsymbol{x}) - H(C \,|\, S) \\
&= I(\boldsymbol{x}; C \backslash \boldsymbol{x} \,|\, S \backslash \boldsymbol{x})
\end{aligned}$$

□

Thus, of all the variables $\boldsymbol{x} \in C_{\text{shared}}$ such that C is a leaf of $\mathcal{T}_{\boldsymbol{x}}$, we should contract the one that minimizes $I(\boldsymbol{x}; C \backslash \boldsymbol{x} \,|\, S \backslash \boldsymbol{x})$. While it is possible to *eventually* contract on all variables in C_{shared}, those variables for which C is not a leaf of $\mathcal{T}_{\boldsymbol{x}}$ will first have to be contracted from other clusters. The error introduced by a sequence of contractions is additive (cf. Theorem 12 of [Kjærulff, 1993]); thus, the error introduced by (eventually) contracting $\boldsymbol{x}$ from C can be computed by adding the errors of each of the successive contractions of $\boldsymbol{x}$.

In the case where $p(C)$ is a Gaussian distribution, the conditional mutual information $I(\boldsymbol{x}; C \backslash \boldsymbol{x} \,|\, S \backslash \boldsymbol{x})$ can be computed from the precision matrix Λ that parameterizes $p(C)$. In particular, the precision matrix of $p(C \,|\, S \backslash \boldsymbol{x})$ is simply that sub-block of Λ corresponding to $D = (C - S) \cup \{\boldsymbol{x}\}$; inverting this sub-block allows us to compute the conditional covariance Σ of D given $S \backslash \boldsymbol{x}$, from which we can compute the conditional mutual information as

$$I(\boldsymbol{x}; C \backslash \boldsymbol{x} \,|\, S \backslash \boldsymbol{x}) = \frac{1}{2} \log \frac{|\Sigma_{D \backslash \boldsymbol{x}}||\Sigma_{\boldsymbol{x}}|}{|\Sigma|} \qquad (29)$$

5 Thin junction tree filters for SLAM

We have now assembled all the machinery we require to build a approximate SLAM filters based upon thin junction trees. We first present a linear-time filter, and then present further approximations that result in a constant-time filtering operation.

5.1 Linear-time approximate filtering

We can easily design a linear-time and linear-space approximate filter for SLAM by noticing that if the diameter of our junction tree is constant with respect to the number of landmarks, then the junction tree will require space linear in the number of landmarks, and message passing will take time linear in the number of landmarks. Our task, then, is to periodically thin the junction tree so that its diameter remains constant with respect to the number of landmarks.

The measurement update poses no special problem for us. In the worst case, the robot will reobserve a landmark whose state variable $\boldsymbol{\ell}$ is very far from the robot's state variable $\boldsymbol{x}$ in the junction tree, requiring $\boldsymbol{x}$ to be added to every cluster to preserve the running intersection property. However, this, and the subsequent message passing, require only linear time. If adding $\boldsymbol{x}$ to these clusters has increased the diameter beyond some threshold, then we perform variable contractions until the junction tree is thin enough.

The motion update, however, is more complicated. When $\boldsymbol{x}_t$ is marginalized out, we must merge all of the clusters in which it resides. In the worst case, $\boldsymbol{x}_t$ can reside in all of the junction tree's clusters, in which case our belief state will collapse to one large cluster. In order to prevent this from happening, we first contract $\boldsymbol{x}_t$ until it resides in only constantly many clusters; then, we perform the motion update. If the cluster caused by marginalizing out $\boldsymbol{x}_t$ is too large, then we employ variable contractions (and possibly cluster duplication) to reduce the diameter of the junction tree.

As an extended example, Figures 4, 5, 6, and 7 demonstrate a thin junction tree filter for the example of Figure 1 in which we constrain the diameter of the junction tree to be three or less.

5.2 Constant-time approximate filtering

The linear-time filter presented above would take constant time, were it not for the fact that reobserving landmarks necessitates a round of message passing. Intuitively, we expect that landmarks whose state variables are far from the robot's state variable (in the junction tree) are those that were last observed in the distant past. Thus, in the common case where the robot observes new landmarks and reobserves some recently-viewed landmarks, only a small portion of the junction tree's structure must be changed.

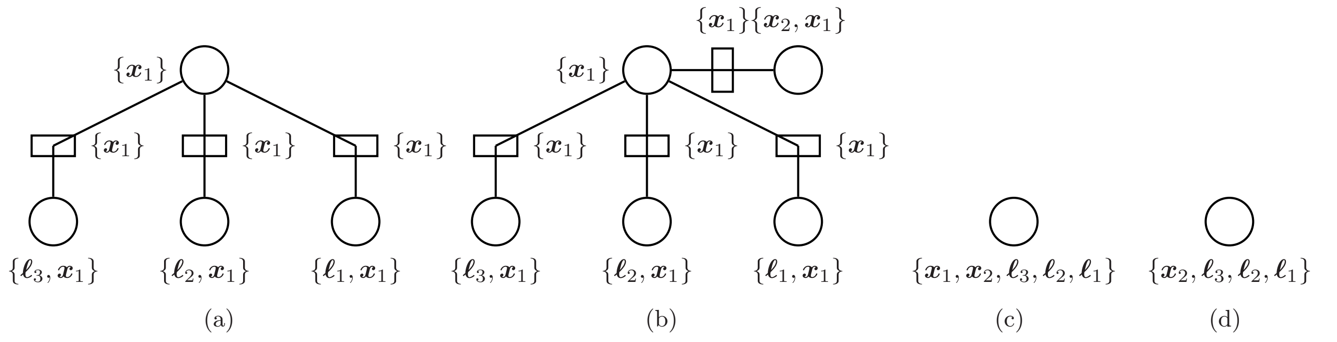
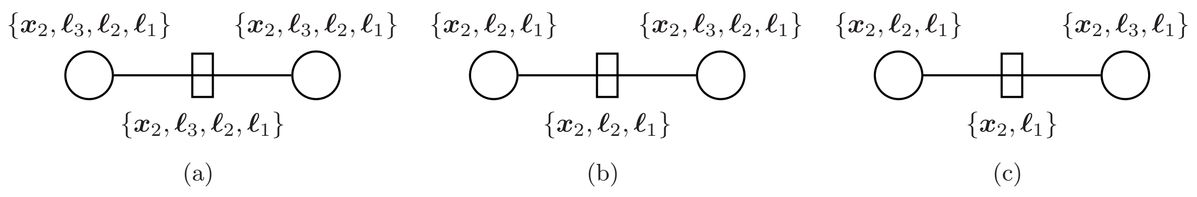
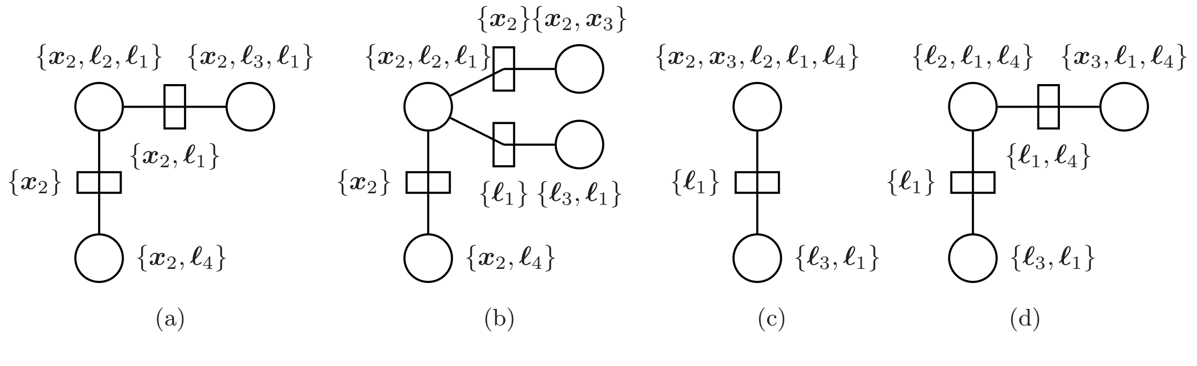


Figure 4: Incrementally maintaining the junction tree during the measurement and motion updates at time $t = 1$. (a) The junction tree after the measurement update. (b) x_2 is added to the model. To marginalize out x_1, (c) all of the clusters containing x_1 are merged together and then (d) x_1 is marginalized out of the new cluster.



$\{x_2, \ell_3, \ell_2, \ell_1\}$
$\{x_2, \ell_3, \ell_2, \ell_1\}$
$\{x_2, \ell_3, \ell_2, \ell_1\}$
(a)
$\{x_2, \ell_2, \ell_1\}$
$\{x_2, \ell_3, \ell_2, \ell_1\}$
$\{x_2, \ell_2, \ell_1\}$
(b)
$\{x_2, \ell_2, \ell_1\}$
$\{x_2, \ell_3, \ell_1\}$
$\{x_2, \ell_1\}$
(c)

Figure 5: Thinning the junction tree after the updates at time $t = 1$. (a) The single cluster is duplicated. (b) ℓ_3 is contracted from the left cluster. (c) ℓ_2 is contracted from the right cluster, resulting in a junction tree with diameter three.



$\{x_2, \ell_2, \ell_1\}$
$\{x_2, \ell_3, \ell_1\}$
$\{x_2, \ell_1\}$
$\{x_2\}$
$\{x_2, \ell_4\}$
(a)
$\{x_2\}\{x_2, x_3\}$
$\{x_2, \ell_2, \ell_1\}$
$\{x_2\}$
$\{\ell_1\}$ $\{\ell_3, \ell_1\}$
$\{x_2, \ell_4\}$
(b)
$\{x_2, x_3, \ell_2, \ell_1, \ell_4\}$
$\{\ell_1\}$
$\{\ell_3, \ell_1\}$
(c)
$\{\ell_2, \ell_1, \ell_4\}$
$\{x_3, \ell_1, \ell_4\}$
$\{\ell_1, \ell_4\}$
$\{\ell_1\}$
$\{\ell_3, \ell_1\}$
(d)

Figure 6: (a) The measurement of landmark 2 does not affect the junction tree, as x_2 and ℓ_2 reside in the same cluster; observing landmark 4 adds a new cluster containing x_2 and ℓ_4 to the junction tree. (b) x_3 is added to the model, and in preparation for the motion update, x_2 is contracted so that it resides in constantly many clusters. (c) The motion update merges the three clusters containing x_2 to yield a larger cluster. (d) x_2 is marginalized out of the new cluster and then it is split, using a cluster duplication and a sequence of variable contractions, as in Figure 5.

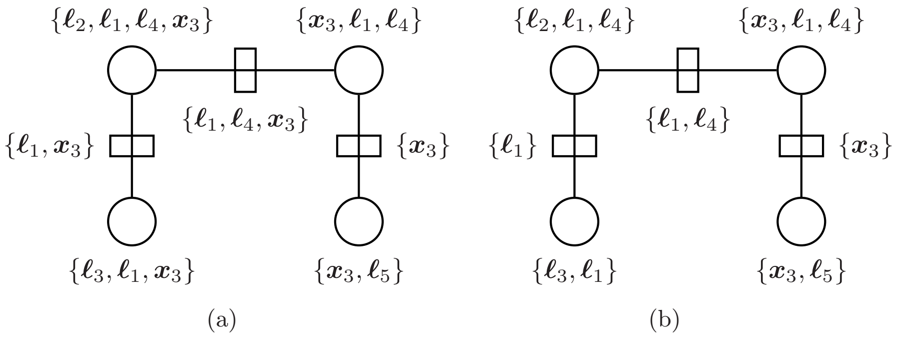

$\{\boldsymbol{\ell}_2, \boldsymbol{\ell}_1, \boldsymbol{\ell}_4, \boldsymbol{x}_3\}$
$\{\boldsymbol{x}_3, \boldsymbol{\ell}_1, \boldsymbol{\ell}_4\}$
$\{\boldsymbol{\ell}_1, \boldsymbol{\ell}_4, \boldsymbol{x}_3\}$
$\{\boldsymbol{\ell}_1, \boldsymbol{x}_3\}$
$\{\boldsymbol{x}_3\}$
$\{\boldsymbol{\ell}_3, \boldsymbol{\ell}_1, \boldsymbol{x}_3\}$
$\{\boldsymbol{x}_3, \boldsymbol{\ell}_5\}$
(a)
$\{\boldsymbol{\ell}_2, \boldsymbol{\ell}_1, \boldsymbol{\ell}_4\}$
$\{\boldsymbol{x}_3, \boldsymbol{\ell}_1, \boldsymbol{\ell}_4\}$
$\{\boldsymbol{\ell}_1, \boldsymbol{\ell}_4\}$
$\{\boldsymbol{\ell}_1\}$
$\{\boldsymbol{x}_3\}$
$\{\boldsymbol{\ell}_3, \boldsymbol{\ell}_1\}$
$\{\boldsymbol{x}_3, \boldsymbol{\ell}_5\}$
(b)

Figure 7: The measurement update at time $t = 3$. (a) Observing landmark 5 for the first time adds a new leaf to the junction tree; reobserving landmark 4 has no effect, as $\boldsymbol{x}_3$ and $\boldsymbol{\ell}_4$ are already in the same cluster; and reobserving landmark 3 necessitates adding $\boldsymbol{x}_3$ to all clusters on the path between the cluster containing $\boldsymbol{\ell}_3$ and the closest cluster containing $\boldsymbol{x}_3$. (b) In order to respect the diameter constraint (and to prepare for the next motion update), $\boldsymbol{x}_3$ is contracted to constantly many clusters.

If we are willing to sacrifice current estimates of the marginals over all cliques, then we can employ a *lazy message passing* scheme, in which we ensure that only constantly many messages are processed per time step. In cases where we have "closed the loop" (which can easily be detected via data association), we can use a linear-time filtering operation to update our map entirely. Note that as soon as we employ a linear-time step, the filtered estimate becomes the same estimate that would have obtained from using the linear-time filter at all steps. Our sacrifice has simply been a period during which the majority of the map was slightly out of date.

6 Conclusions

Thin junction tree filters are a novel approach to maintaining tractable approximations to a belief state that can be represented by graphical models with changing dependencies. As such, they represent a principled, flexible, and efficient solution to the SLAM complexity problem. Having now worked through the details of the filter, we can now offer a detailed comparison of it with the other methods described in Section 1.2.

Thin junction tree filters can (frequently) be made to operate in constant-time, making their time complexity competitive with that of the FastSLAM algorithm. Thin junction tree filters also represent a single estimate of the map, in contrast to FastSLAM; while this improves the complexity of data association and extracting map marginals, it significantly curbs the representation's expressiveness.

Interestingly, the approach presented here has significant connections to both the submap approach and SEIF. First, thin junction tree filters have a natural interpretation as a coupled set of local maps, just as in the submap approaches. In particular, each cluster of the junction tree can be viewed as a submap, and the consistency of the junction tree can be viewed as a constraint that overlapping submaps must agree on their shared content. The thin junction tree filter formulation gives concrete semantics to the relationships between the maps, including how they must be updated, how consistency is maintained, and how the set of local maps can be determined online to minimize the approximation error subject to a complexity constraint. Most importantly, thin junction tree filters allow information to be communicated between submaps.

Second, since thinning the junction tree is equivalent to removing sets of edges from the graphical model, which in turn is equivalent to zeroing out entries of Λ, thin junction tree filtering can be viewed as a technique for making Λ sparse; thus, its goal is identical to that of SEIF. As stated above, the maximum likelihood approach to deleting single edges requires Iterative Proportional Fitting, and is computationally complex; SEIF employs an approximation that is constant time, but leads to representational problems. In contrast, the approach presented here is a maximum likelihood sparse approximation to Λ that can be computed analytically and efficiently; it permits constant-time access to marginal distributions, and it allows both constant-time and linear-time updates. As we have seen, these improvements arise because we restrict ourselves to a distinguished set of sparsity patterns—those that correspond to the set of decomposable models.

Acknowledgments

I gratefully acknowledge Intel Corporation for supporting this research via an Intel Research Internship, as well as Francis Bach and Andrew Ng for helpful discussions.

References

[Cowell *et al.*, 1999] R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems.* Springer, New York, NY, 1999.

[Doucet *et al.*, 2000] Arnaud Doucet, Nando de Freitas, Kevin Murphy, and Stuart Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 176–183, 2000.

[Draper, 1995] Denise L. Draper. Clustering without (thinking about) triangulation. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference (UAI-1995)*, pages 125–133, San Francisco, CA, 1995. Morgan Kaufmann Publishers.

[Geiger *et al.*, 1990] D. Geiger, T. S. Verma, and J. Pearl. Identifying independence in bayesian networks. *Networks*, 20:507–534, 1990.

[Guivant and Nebot, 2000] J. E. Guivant and E. M. Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Transactions on Robotic and Automation*, 17(3):242–257, 2000.

[Kjærulff, 1993] Uffe Kjærulff. Approximation of bayesian networks through edge removals. Dept. of Mathematics and Computer Science Research Report IR-93-2007, Aalborg University, 1993.

[Leonard and Feder, 2000] J. J. Leonard and H. J. S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In *Proceedings of the Ninth International Symposium on Robotics Research*, pages 169–176. Springer-Verlag, 2000.

[Montemerlo *et al.*, 2002] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 593–598. AAAI Press, 2002.

[Russell and Norvig, 2002] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2002.

[Shachter *et al.*, 1994] R. D. Shachter, S. K. Andersen, and P. Szolovits. Global conditioning for probabilistic inference in belief networks. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 514–522, San Francisco, 1994. Morgan Kaufmann.

[Speed and Kiiveri, 1986] T. P. Speed and H. T. Kiiveri. Gaussian markov distributions over finite graphs. *Annals of Statistics*, 14(1):138–150, March 1986.

[Thrun *et al.*, 2002] Sebastian Thrun, Daphne Koller, Zoubin Ghahmarani, and Hugh Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters: Theory and initial results. Computer Science Technical Report CMU-CS-02-112, Carnegie Mellon University, Pittsburgh, PA 15213, September 2002.

[Thrun, 2002] Sebastian Thrun. Robotic mapping: A survey. Technical Report CMU-CS-02-111, Carnegie Mellon University, Pittsburgh, PA 15213, February 2002.

[Wan and van der Merwe, 2000] E. Wan and R. van der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of IEEE Symposium 2000 (AS-SPCC)*, 2000.

[Whittaker, 1989] Joe Whittaker. *Graphical Models in Applied Mathematical Multivariate Statistics*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 1989.