# Thin Junction Tree Filters for Simultaneous Localization and Mapping

**Mark A. Paskin**
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
paskin@cs.berkeley.edu

## Abstract

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, at the same time, localize itself within that map. One popular solution is to treat SLAM as an estimation problem and apply the Kalman filter; this approach is elegant, but it does not scale well: the size of the belief state and the time complexity of the filter update both grow quadratically in the number of landmarks in the map. This paper presents a filtering technique that maintains a tractable approximation of the belief state as a thin junction tree. The junction tree grows under filter updates and is periodically "thinned" via efficient maximum likelihood projections so inference remains tractable. When applied to the SLAM problem, these thin junction tree filters have a linear-space belief state and a linear-time filtering operation. Further approximation yields a filtering operation that is often constant-time. Experiments on a suite of SLAM problems validate the approach.

## 1 Introduction

Simultaneous Localization and Mapping (SLAM)—where a robot navigating in an unknown environment must incrementally build a map of its surroundings and localize itself within that map—has attracted significant attention because it is required by many applications in mobile robotics [Thrun, 2002]. Typically the environment is idealized so that it consists of an unknown number of stationary "landmarks"; for example, in a given SLAM application these landmarks may be low-level visual features or structural features such as walls and corners. SLAM can then be viewed as the problem of incrementally estimating the locations of the robot and landmarks from noisy and incomplete observations.

One popular approach treats SLAM as a filtering problem [Smith *et al.*, 1990]. The hidden state of the system at time $t$ is represented by a random variable $\boldsymbol{m}_t$ which includes $\boldsymbol{x}_t$, the state of the robot at time $t$, and $\boldsymbol{l}_1, \ldots, \boldsymbol{l}_{n_t}$, the locations of the $n_t$ landmarks observed up to time $t$. Thus, the size of the state vector is linear in the number of observed landmarks and grows over time. The Kalman filter is used to compute the filtered belief state $p(\boldsymbol{m}_t \mid \text{observations to time } t)$, which in this case takes the form of a multivariate Gaussian distribution $\mathcal{N}(\mu_t, \Sigma_t)$. We regard the mean $\mu_t$ as the estimate of the map and the covariance matrix $\Sigma_t$ a measure of confidence.

The Kalman filter solution is elegant, but it does not scale well to large SLAM problems. Because $\Sigma_t$ explicitly represents correlations between all pairs of variables, the size of the belief state grows as $O(n_t^2)$; and because each of these correlations must be updated whenever a landmark is reobserved, the time complexity of its filter operation is also $O(n_t^2)$. This quadratic complexity renders the Kalman filter inapplicable to large SLAM problems and gives rise to the need for principled, efficient approximations.

Unfortunately, the simplest approach—discarding correlations so each variable is estimated independently—presents problems. Ignoring correlations between the robot state and the landmarks' states leads to overconfidence and divergence because correlated observations are treated as if they conveyed independent information [Hébert *et al.*, 1996]. Furthermore, correlations between pairs of landmark states are required for quick convergence when the robot *closes a loop*, i.e., it reobserves known landmarks after an extended period of mapping unknown territory (see Figure 1). When the robot closes a loop, it reobserves landmarks whose positions are known with relative certainty; this helps the robot localize. The robot-landmark correlations translate this improved localization estimate into improved position estimates for recently-observed landmarks. The inter-landmark correlations translate these improved position estimates into improvements for the remaining landmarks on the tour.[1] Thus, the correlations give the Kalman filter a valuable property normally associated with smoothing algorithms: it can use current observations to improve estimates "from the past".

Because quadratically many correlations are necessary to close loops, we view the challenge of scalable SLAM filtering as that of *estimating and reasoning with quadratically many correlations without quadratic time or space complexity.* In this paper, we present a novel and general approximate filtering method satisfying this criterion. Our point of departure

[1]Robot-landmark correlations, which decay over time due to motion noise, cannot translate an improved localization estimate into improvements for landmarks observed in the distant past; in contrast, inter-landmark correlations do not decay over time.

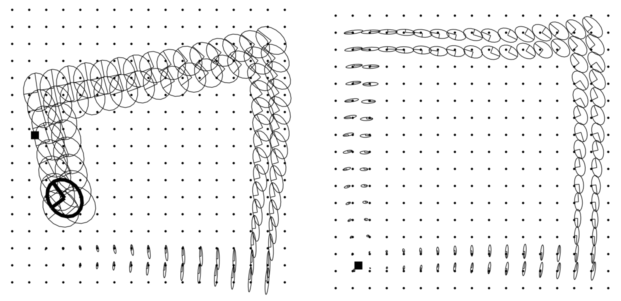


Figure 1: The robot is travelling counter-clockwise on a square path. Dots represent landmarks; the true position of the robot is shown by a square; the filter belief state is visualized using the 95% confidence ellipses of the variable marginals (bold for the robot). Left: accumulated noise and error has led to uncertain and drifted estimates for the robot and landmark positions. Right: after closing the loop, all of the position estimates improve and their confidences increase.

(in Section 2) is to view the filtered belief state of the Kalman filter as a Gaussian graphical model [Cowell *et al.*, 1999] that evolves over time; this allows us to express correlations in terms of direct dependencies (edges) and indirect dependencies (paths). Analyzing the evolution of this graphical model reveals that filter updates add edges to the graphical model, making inference more expensive. This motivates an approximation scheme in which weak or redundant edges are periodically removed to improve the complexity of inference. Note that edge removal is very different than simply discarding correlations; because other edges are left intact, paths—and thus correlations—persist between each pair of variables.

Graphical models give us valuable insight into how good approximate filters can be designed, but using them to represent the belief state presents problems. First, variable marginals like the robot's current position would not be immediately available as they are in the Kalman filter representation; we would require inference to obtain them. Second, while it is possible to remove edges from a Gaussian graphical model using the Iterative Proportional Fitting algorithm [Speed and Kiiveri, 1986], its application in this context would be prohibitively slow. Finally, choosing edges whose removal leaves a distribution for which inference is tractable is itself a complicated process [Kjærulff, 1993].

Our solution to these problems is to use a different representation of the belief state. Exact inference in graphical models is often implemented by message passing on a *junction tree* [Cowell *et al.*, 1999]. Rather than view the junction tree algorithm as an "inference engine", we use the junction tree itself as our representation of the belief state. This representation has many advantages: the belief state has a "built-in" inference algorithm (namely, message passing); it gives immediate access to the marginal distribution over any variable; and as we demonstrate, it gives us efficient methods of selecting edges to prune and pruning them.

To implement such a *junction tree filter*, we develop methods for updating the junction tree to reflect filtering updates in Section 3. These updates can cause the width of the junction tree to grow, making inference more expensive; in Section 4 we present a novel "thinning" operation over junction trees called *variable contraction*. We prove that each variable contraction is a maximum likelihood projection that removes a *set* of edges from the corresponding graphical model. The approximation error introduced by a variable contraction can be computed efficiently, which allows us to choose which edges to remove at each time step so as to minimize the error.

In Section 5 we apply these techniques to the SLAM problem and obtain a *thin junction tree filter* (TJTF) with a $O(n_t)$ space belief state representation and a $O(n_t)$ time filter operation. By delaying the incorporation of recent evidence into the majority of the map, we can improve the filter's time complexity; we present a method of evaluating the significance evidence has on different portions of the map, which can be used to adaptively interpolate between constant and linear-time filter operations. Empirically, we find that these adaptive filters choose constant-time updates when mapping new territory, and when closing a loop, they use time linear in the length of the loop. This is perhaps the best time complexity one would hope for in the SLAM problem, since linearly-many estimates cannot be improved in constant time. Section 6 presents the results of simulation experiments that compare TJTF to other SLAM filters and Section 7 concludes. A companion technical report contains proofs of all propositions as well as additional background, analysis, and experiments [Paskin, 2002].

## 1.1 Related work

Significant interest in the SLAM complexity problem has led to a number of approaches [Thrun, 2002]. For example, there are several *submap* approaches that decompose the problem into a set of small mapping problems yielding a block-diagonal landmark covariance matrix. These techniques can achieve constant time complexity, but converge slowly because information cannot pass between the submaps.

Recently, the FastSLAM algorithm [Montemerlo *et al.*, 2002]—a Rao-Blackwellized particle filter—has attracted attention because of its logarithmic time complexity. However, our experiments show FastSLAM is susceptible to divergence in large, noisy SLAM problems. We believe this is because the number of particles required for a satisfactory solution can grow exponentially over time; see [Paskin, 2002] for details.

*Sparse extended information filters* (SEIF) [Thrun *et al.*, 2002] can be viewed in terms of the graphical model representation described above; at each time-step, edges are removed so that a constant-time filter operation can be guaranteed. To avoid the additional complexity of inference, SEIF employs approximate inference over this approximate model. Thus, the SEIF paper provided the valuable insight that sparse graphical models can constitute an efficient solution to SLAM. Implementing this insight while avoiding additional approximation was one of the primary motivations of this work.

Each of these approaches described above uses a sublinear-time filter update, and therefore, none can improve all of the landmark estimates in a single update (like the Kalman filter). TJTF has the best of both worlds: its update step takes constant time unless the observation is significant enough to warrant a linear-time update.

Outside of the SLAM literature, there are two works that are especially relevant. Kjærulff [1993] investigated edge removal as a means of reducing the complexity of inference in graphical models. Our approach is somewhat simpler, as it operates directly on the junction tree without referring to the underlying graphical model. Kjærulff's analysis of the approximation error inspired ours, and several of his results apply directly to our case.

Thin junction tree filtering is an *assumed density filtering* (ADF) algorithm because it periodically "projects" the filter's belief state to some tractable family of distributions—in this case, the family of Gaussian distributions characterized by thin junction trees. This makes other work on ADF relevant, especially that of Boyen and Koller [1998], in which the belief state of a dynamic Bayesian network is periodically projected to a product-of-marginals approximation. In fact, the connection to this work is stronger: Boyen and Koller [1999] extended their earlier analysis to filters where the belief state is represented by a junction tree whose structure evolves over time; however, no algorithms were presented. To our knowledge, TJTF is the first algorithm to which this analysis applies. Here we apply TJTF to a Gaussian graphical model, but nothing prevents its application to the discrete variable networks considered by Boyen and Koller.

## 2 A graphical model perspective on SLAM

We begin by presenting the SLAM model and then formulating SLAM filtering in terms of graphical models.

### 2.1 The SLAM model

We assume a general SLAM model where in each time step the robot moves, obtains an odometry measurement of its motion, and makes several observations of landmarks. As in the Kalman filter context, we assume that the motion and measurement models are known and that they are linear-Gaussian.[2] The robot motion at time $t$ is governed by

$$\boldsymbol{x}_{t+1} = A_t \boldsymbol{x}_t + \boldsymbol{v}_t, \qquad \boldsymbol{v}_t \sim \mathcal{N}(b_t, Q_t) \tag{1}$$

and the odometry measurement $y_t$ at time $t$ is governed by

$$\boldsymbol{y}_t = C_t \boldsymbol{x}_t + \boldsymbol{w}_t, \qquad \boldsymbol{w}_t \sim \mathcal{N}(d_t, R_t). \tag{2}$$

$y_t$ is typically a noisy measurement of the robot's velocities.

Landmark measurements are typically assumed to depend only upon the state of the robot and the state of the observed landmark; for example the observation may consist of the range and bearing to the landmark in the robot's coordinate frame. If the $i$th landmark measurement at time $t$ issued from landmark $j$, it is governed by

$$\boldsymbol{z}_t^i = E_t^i \boldsymbol{x}_t + F_t^i \boldsymbol{l}_j + \boldsymbol{u}_t^i, \qquad \boldsymbol{u}_t^i \sim \mathcal{N}(g_t^i, S_t^i) \tag{3}$$

For simplicity, we assume the correspondence between each measurement and the landmark from which it issued is known. This question of *data association*, while critically important in SLAM, is largely orthogonal to the issues we address here; in particular, the standard technique of choosing the maximum likelihood data association applies without change in our treatment. When a landmark is first observed, its state variable is added to the belief state with a uninformative (infinite variance, zero covariance) prior; the measurement update yields an informed posterior estimate of its state.

[2]When these models are not linear-Gaussian, they can be approximated as such as in the Extended or Unscented Kalman Filter.

### 2.2 Gaussian graphical models

Under the assumptions outlined above, the filtered belief state $p(\boldsymbol{m}_t \mid y_{1:t}, z_{1:t})$ is a multivariate Gaussian distribution. The Kalman filter represents this distribution using the *moment parameters*—the mean vector $\mu_t$ and covariance matrix $\Sigma_t$. If $\boldsymbol{u} \sim \mathcal{N}(\mu, \Sigma)$ then its probability distribution is

$$p(u) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left\{-\frac{1}{2}(u-\mu)^T \Sigma^{-1} (u-\mu)\right\} \tag{4}$$

where $d$ is the length of $\boldsymbol{u}$. In contrast, Gaussian graphical models are usually based upon the *canonical parameters*—the information vector $\eta$ and matrix $\Lambda$. If $\boldsymbol{u} \sim \mathcal{N}^{-1}(\eta, \Lambda)$

$$p(u) = \exp\left\{a + \eta^T u - \frac{1}{2} u^T \Lambda u\right\} \tag{5}$$

where $a = -\frac{1}{2}(d \log(2\pi) - \log|\Lambda| + \eta^T \Lambda^{-1} \eta)$ is the (log) normalization constant. The canonical and moment parameters are related by $\Lambda = \Sigma^{-1}$ and $\eta = \Sigma^{-1}\mu$. An advantage of the canonical parameterization is that multiplication/division of Gaussians reduces to addition/subtraction of the parameters.

Let $\{\boldsymbol{u}_i : i \in V\}$ be a set of random variables indexed by elements of the finite set $V$. We will call a subset of $V$ a *family*. For a family $a \subseteq V$, let $\boldsymbol{u}_a = \{\boldsymbol{u}_i : i \in a\}$ be the associated set of random variables. A *potential* over a family $a \subseteq V$ is a non-negative function of $\boldsymbol{u}_a$. Let $F$ be a set of families and let $\Psi = \{\psi_a : a \in F\}$ be a set of potential functions over these families. $(F, \Psi)$ defines a distribution

$$p(u) = \frac{1}{Z} \prod_{a \in F} \psi_a(u_a) \tag{6}$$

when the normalizer $Z = \int \prod_{a \in F} \psi_a(u_a)\, \mathrm{d}u$ is finite.

The *Markov graph* associated with $(F, \Psi)$ has vertex set $V$ and a clique of edges over each $a \in F$, i.e., there is an edge between $i$ and $j$ iff $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ are bound by a potential. The primary value of the Markov graph representation comes from the Hammersley-Clifford theorem, which states that $s$ separates $a$ from $b$ in the Markov graph (for $a, b, s \subseteq V$) iff $\boldsymbol{u}_a \perp\!\!\!\perp \boldsymbol{u}_b \mid \boldsymbol{u}_s$ (provided $p > 0$). In other words, graph separation in the Markov graph encodes the conditional independence properties of $p$. Because conditional independence properties often translate into efficient inference algorithms (e.g., junction tree), the Markov graph gives good intuitions into the design of efficient approximations.

We can represent the Gaussian (5) by a Markov graph, since if we partition the vector $\boldsymbol{u} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$

$$p(u) = \frac{1}{Z} \prod_{i=1}^{n} \psi_{\{i\}}(u_i) \prod_{j=i+1}^{n} \psi_{\{i,j\}}(u_i, u_j) \tag{7}$$

where

$$\psi_{\{i\}}(u_i) = \exp\left\{\eta_i u_i^T - \frac{1}{2} u_i^T \Lambda_{ii} u_i\right\} \tag{8}$$

$$\psi_{\{i,j\}}(u_i, u_j) = \exp\left\{-u_i^T \Lambda_{ij} u_j\right\} \tag{9}$$

and $Z = e^{-a}$ is the normalization constant. Thus, all the potentials of a Gaussian graphical model are either unary (*node potentials*) or binary (*edge potentials*). We also have the important interpretation that if $\Lambda_{ij} = 0$, then $\psi_{\{i,j\}}(u_i, u_j)$ is unity (and therefore superfluous), meaning there is no edge between $i$ and $j$ in the corresponding Markov graph.

## 2.3 Filtering in Gaussian graphical models

Filtering can be viewed as a three-step procedure: *estimation*, in which we incorporate the current time step's measurements; *prediction*, in which we augment the model with the state variables of the next time step; and *roll-up*, in which we marginalize out the state variables from the past time step. When the measurement and motion models are linear-Gaussian, the prediction and estimation steps reduce to multiplying small Gaussian potentials into the model; these updates are summarized by

**Proposition 1.**[3] *Ignoring irrelevant normalization constants, the motion update of equation* (1) *can be implemented by multiplying the potential*

$$\psi\begin{pmatrix} \boldsymbol{x}_t \\ \boldsymbol{x}_{t+1} \end{pmatrix} = \mathcal{N}^{-1}\left(\begin{bmatrix} -b^T Q^{-1} A \\ b^T Q^{-1} \end{bmatrix}, \begin{bmatrix} A^T Q^{-1} A, & A^T Q^{-1} \\ Q^{-1} A, & Q^{-1} \end{bmatrix}\right)$$

*into the model; the odometry measurement update of equation* (2) *can be implemented by multiplying in the potential*

$$\psi(\boldsymbol{x}_t) = \mathcal{N}^{-1}(C^T R^{-1}(y_t - d), C^T R^{-1} C)$$

*and the landmark measurement update of equation* (3) *can be implemented by multiplying in the potential*

$$\psi\begin{pmatrix} \boldsymbol{x}_t \\ \boldsymbol{l}_j \end{pmatrix} = \mathcal{N}^{-1}\left(\begin{bmatrix} E^T S^{-1}(z_t^i - g) \\ F^T S^{-1}(z_t^i - g) \end{bmatrix}, \begin{bmatrix} E^T S^{-1} E, & E^T S^{-1} F \\ F^T S^{-1} E, & F^T S^{-1} F \end{bmatrix}\right)$$

The final step of filtering is roll-up, or marginalizing out the past state. The standard rule for marginalization in the canonical parameterization is given by [Cowell *et al.*, 1999]

**Fact 1.** *If* $a = V \setminus i$ *and*

$$\begin{bmatrix} \boldsymbol{u}_a \\ \boldsymbol{u}_i \end{bmatrix} \sim \mathcal{N}^{-1}\left(\begin{bmatrix} \eta_a \\ \eta_i \end{bmatrix}, \begin{bmatrix} \Lambda_{aa} & \Lambda_{ai} \\ \Lambda_{ia} & \Lambda_{ii} \end{bmatrix}\right) \tag{10}$$

*then* $\boldsymbol{u}_a \sim \mathcal{N}^{-1}(\eta_a^m, \Lambda_a^m)$ *where*

$$\eta_a^m = \eta_a - \Lambda_{ai}\Lambda_{ii}^{-1}\eta_i \tag{11}$$

$$\Lambda_a^m = \Lambda_{aa} - \Lambda_{ai}\Lambda_{ii}^{-1}\Lambda_{ia} \tag{12}$$

The time complexity of computing (11) and (12) is quadratic in the dimension of $\boldsymbol{u}_a$ and cubic in the dimension of $\boldsymbol{u}_i$.

The additive updates above can also be viewed as multiplying in a new potential $\mathcal{N}^{-1}(-\Lambda_{ai}\Lambda_{ii}^{-1}\eta_i, -\Lambda_{ai}\Lambda_{ii}^{-1}\Lambda_{ia})$ into the model. The *Markov blanket* $n(i)$ of $i \in V$ is the set of $i$'s neighbors in the Markov graph. Because missing edges in the Markov graph correspond to zeros in $\Lambda$, we can infer that this is really a potential over $n(i)$, and therefore that marginalizing $\boldsymbol{u}_i$ out of the model places a clique of edges over the Markov blanket of $i$.

[3]Model parameter indices are omitted for notational simplicity.

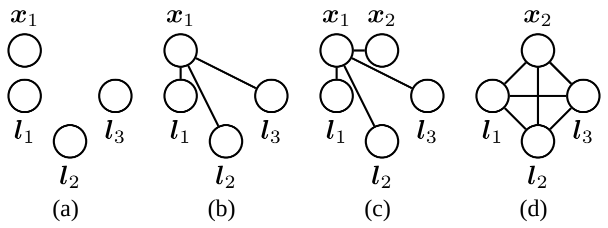


Figure 2: Example evolution of a SLAM graphical model. (a) In the initial belief state, the robot's state $\boldsymbol{x}_1$ and the landmarks' states $\boldsymbol{l}_1$, $\boldsymbol{l}_2$, and $\boldsymbol{l}_3$ are marginally independent. (b) Observing each landmark $j$ induces a correlation between $\boldsymbol{x}_1$ and $\boldsymbol{l}_j$, resulting in a new edge. (c) The prediction update adds the new robot state $\boldsymbol{x}_2$ to the model and joins it to the current robot state $\boldsymbol{x}_1$. (d) The roll-up phase marginalizes $\boldsymbol{x}_1$ out of the model, adding a clique edges over all of $\boldsymbol{x}_1$'s neighbors.

## 2.4 Filtering the SLAM graphical model

Using these results we can characterize how the structure of the SLAM belief state evolves over time (see Figure 2). For each observed landmark $j$, we multiply a measurement potential $\psi(\boldsymbol{x}_t, \boldsymbol{l}_j)$ into the graphical model; this adds an edge between $\boldsymbol{x}_t$ and $\boldsymbol{l}_j$. Thus, after the estimation phase, the robot's state will be connected to the states of all landmarks it has observed. The prediction phase then connects $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$. Finally, the roll-up phase marginalizes out $\boldsymbol{x}_t$; this places a potential over the Markov blanket of $\boldsymbol{x}_t$, which now includes all observed landmarks and $\boldsymbol{x}_{t+1}$. Now the SLAM graphical model takes the form of a complete graph—i.e., *the belief state has no conditional independencies*. By induction, this will be true after every time step.

An intuition for why the graphical model becomes dense over time is valuable. When the robot measures a landmark, the landmark's state becomes directly correlated with that of the robot, and thus *indirectly* correlated with all covariates of the robot state, e.g., other landmark states. When the robot's state is eliminated from the model during roll-up, these indirect correlations must be expressed *directly* via new edges.

Importantly, these indirect correlations are often much weaker than the direct ones. Thus, even though the SLAM belief state has no true conditional independencies, there are many "approximate" conditional independencies; e.g., the landmarks observed at the beginning and end of a tour are almost independent given those observed in the middle. By removing "weak" edges from the graphical model we can enforce these approximate conditional independencies so they can be used to speed inference.

## 3 Junction tree filtering

As discussed in the introduction, the graphical model representation is valuable for motivating our approximate filter, but it is not an appropriate representation for its implementation. Instead, we represent the belief state of the filter using a junction tree. We begin by briefly summarizing the relevant concepts; see [Cowell *et al.*, 1999] for details.

## 3.1 Junction trees

Let $p$ be a distribution of the form (6) with families $F$ and potentials $\Psi = \{\psi_a : a \in F\}$. Let $T = (C, E)$ be an undirected graph where each vertex (or *cluster*) $c \in C$ is a subset of $V$; $T$ is a *junction tree for* $p$ if the following three properties hold:

1. *Singly connected property:* $T$ is a tree.
2. *Potential property:* For every family $a \in F$ there is some cluster $c$ such that $c \supseteq a$.
3. *Running intersection property:* If $i \in V$ is present in two clusters $c_i$ and $c_j$ of $T$, it is also present in all clusters on the (unique) path between $c_i$ and $c_j$.

With each edge $\{c_i, c_j\} \in E$ we associate a *separator* $s = c_i \cap c_j$; let $S$ be the set of $T$'s separators.

Given a junction tree $T$, we can perform inference in the model $p$ by passing messages between the clusters of $T$. We begin by associating with $T$ a set of potential functions $\Phi = \{\phi_c : c \in C\} \cup \{\phi_s : s \in S\}$, one for each cluster and separator. The *charge on* $T$ is defined to be

$$\phi_T(u) = \frac{\prod_{c \in C} \phi_c(u_c)}{\prod_{s \in S} \phi_s(u_s)} \tag{13}$$

We initialize $\Phi$ by setting all cluster and separator potentials to unity, multiplying each potential $\psi \in \Psi$ into $\phi_c$ for some $c \in C$ (which is possible by the potential property), and multiplying $Z^{-1}$ into an arbitrary $\phi_c$; then $\phi_T = p$.

Let $c$ and $d$ be adjacent clusters with separator $s = c \cap d$. Passing a message from $c$ to $d$ updates the separator potential $\phi_s$ and the cluster potential $\phi_d$ as follows:

$$\phi_s^*(u_s) = \int \phi_c(u_c)\,\mathrm{d}u_{c-s} \tag{14}$$

$$\phi_d^*(u_d) = \phi_d(u_d)\frac{\phi_s^*(u_s)}{\phi_s(u_s)} \tag{15}$$

Importantly, these updates leave the charge (13) invariant, so $\phi_T^* = p$. Thus, we can view them as reparameterizing the distribution $p$. When messages are passed along every edge in both directions (in an appropriate schedule), the cluster and separator potentials are updated so that they are marginals of $p$ over their respective variables. A junction tree in this state is called *consistent* and it can be used to obtain marginals over any set of variables that reside together in some cluster.

When $T$ has no nonmaximal clusters, $|C| \leq |V|$ so the number of messages required for inference is bounded by $2 \cdot |V|$. In the case of a Gaussian graphical model, the cluster and separator potentials are Gaussians; if they are represented by their canonical parameters, the time complexity of passing a message is dominated by the cost of the marginalization in (14) which is implemented via (11) and (12); thus, it is at worst cubic in the size of the cluster. In sum, inference is linear in $|V|$ and cubic in the *width* of $T$, traditionally defined as the size of the largest cluster minus one.

## 3.2 Incremental junction tree maintenance

We adopt consistent junction trees as the belief state representation of our filter; i.e., the belief state will be represented by the charge (13) of a consistent junction tree. Recall from Section 2.3 that the prediction and estimation phases of the filter update can be implemented by multiplying in small, simple potentials to the probability distribution, and that the roll-up phase is implemented by marginalizing variables out of the model. In this section we describe how to incrementally maintain a consistent junction tree under these updates.

In what follows we will make use of three nonstandard operations to restructure a consistent junction tree.

- *Cloning:* To clone a cluster $c$, we create a copy $d$, attach $d$ to $c$ with separator $s = c = d$, and set $\phi_s^* = \phi_d^* = \phi_c$.
- *Merging:* Let $c$ and $d$ be neighboring clusters with separator $s$. To *merge* $d$ into $c$, we: (1) update $c^* = c \cup d$; (2) update $\phi_c^* = \phi_c \phi_d / \phi_s$; (3) swing all edges incident to $d$ over to $c$; and (4) remove $d$ from $C$ and $s$ from $S$.
- *Pushing:* Let $c$ and $d$ be neighboring clusters with separator $s$ such that $i \in c$ but $i \notin d$. To *push* $i$ *from* $c$ *to* $d$ we update $s^* = s \cup \{i\}$ and $d^* = d \cup \{i\}$ and pass a message from $c$ to $d$ to update $\phi_s$ and $\phi_d$. By extension we can push $i$ from $c$ to a nonadjacent cluster $d$ by successive pushes along the unique path from $c$ to $d$. (Any nonmaximal clusters created by pushing are subsequently merged into their subsuming neighbors.)

It is easy to check that all of these operations preserve the three structural constraints as well as the charge and consistency of a junction tree.

**Multiplying in potentials**

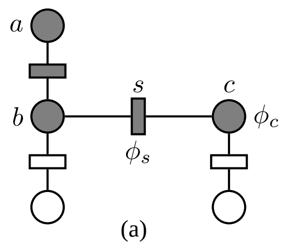
Assume our belief state is represented by a consistent junction tree $T$. In order to update the charge of $T$ to reflect the multiplication of a potential $\psi_a(u_a)$ into $p$, we must find a cluster $c \supseteq a$ and multiply $\psi_a$ into $\phi_c$. To restore consistency, we could pass messages throughout $T$, but this is twice the work needed: a simple consequence of the message-passing updates (14) and (15) is that we need only *distribute evidence* from $c$, i.e, we must pass messages along edges in a preorder traversal from $c$.

If there is no cluster that covers the family $a$ of the new potential, then we must first modify the junction tree $T$ to create one. Draper [1995] presents several techniques to do this; in the Gaussian case the problem is somewhat simpler, since the potentials bind at most two variables. When multiplying in an edge potential $\psi(u_i, u_j)$ requires us to create a cluster covering $\{i, j\}$, we find the closest pair of clusters $c$ and $d$ such that $i \in c$ and $j \in d$ and push $i$ from $c$ to $d$. We then multiply $\psi(u_i, u_j)$ into $\phi_d(u_d)$ and distribute evidence from $d$.

It is worth noting that in several cases, conditional independencies obviate the evidence distribution step. This is a significant optimization, since message passing is by far the most expensive operation. This occurs, for example, when performing the prediction step (because $\boldsymbol{x}_{t+1}$ is an unobserved directed leaf of the graphical model and therefore does not impact the distributions of the other nodes), when observing a landmark for the first time (due to its uninformative prior), and in certain types of odometry updates.

**Marginalizing out variables**

Assume again that we have a consistent junction tree $T$ representing $p$. As described in Section 2.3, marginalizing $\boldsymbol{u}_i$ out of $p$ places a potential over the Markov blanket of $\boldsymbol{u}_i$. Because the junction tree must have a cluster that covers this new



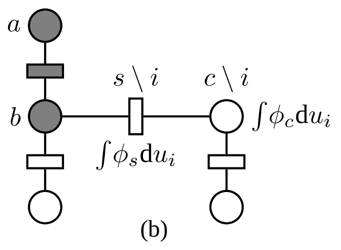


Figure 3: Illustration of variable contraction. Clusters are circles and separators are rectangles; $T_i$ is shaded. (a) $i$ can be contracted from $a$ or $c$ because they are leaves of $T_i$; $i$ cannot be contracted from $b$ because the running intersection property would be violated. (b) Contracting $i$ from $c$ removes it from $c$ and $s$ and marginalizes $\boldsymbol{u}_i$ out of $\phi_c$ and $\phi_s$.

potential, marginalizing out $\boldsymbol{u}_i$ is not as simple as marginalizing it out of all the cluster potentials that bind it.

Let $n(i)$ be the Markov blanket of $i$. Because $T$ has the potential property, we are guaranteed that $n(i) \subseteq \cup_{c \in C, c \ni i} c$, i.e., that $i$'s Markov blanket is covered by the clusters containing $i$. (In fact, in the sequel this containment will often be strict equality.) Moreover, because $T$ has the running intersection property, all clusters containing $i$ constitute a subtree of $T$, which we denote $T_i$. By successively merging the clusters of $T_i$ into each other, we can obtain a new junction tree where $i$ resides in a single cluster $c^* \supseteq n(i)$. Marginalizing $\boldsymbol{u}_i$ out of this junction tree is simple: we remove $i$ from $c^*$ and marginalize $\boldsymbol{u}_i$ out of $\phi_{c^*}$. It is simple to check that this operation results in a consistent junction tree for $\int p(u)\, \mathrm{d}u_i$.

## 4 Thinning the junction tree

The updates described in the previous section can cause the clusters of the junction tree to grow; in particular, the merging of clusters required by marginalizations can cause the width of the junction tree to increase quickly. The complexity of message passing scales with the width of the junction tree, and therefore our goal is to define a "thinning" operation that reduces the width (see Figure 3):

**Definition 1.** Let $i \in V$ appear in more than one cluster of the consistent junction tree $T$, let $c$ be a leaf of $T_i$ (the subtree of $T$ induced by $i$), and let $s$ be the separator joining $c$ to $T_i$. A *variable contraction of* $i$ *from* $c$ removes $i$ from $c$ and $s$ and marginalizes $\boldsymbol{u}_i$ out of $\phi_c$ and $\phi_s$. ($c$ is merged into its subsuming neighbor if it becomes nonmaximal.)

We now consider some properties of variable contraction.

**Proposition 2.** *Variable contractions preserve consistency and the singly connected and running intersection properties.*

Thus, the new junction tree is valid for some distribution, although perhaps not $p$: the potential property may be violated. Variable contraction is local and efficient: it requires marginalizing a variable out of one cluster potential and one separator potential, which in the Gaussian case can be accomplished in $O(|c|^2)$ time using (11) and (12). Also, variable contraction is a general method of "thinning" a junction tree:

**Proposition 3.** *In combination with cloning, variable contraction can reduce the width of any junction tree.*

The following proposition relates the original distribution and the distribution resulting from a variable contraction:

**Proposition 4.** *Let $\hat{T}$ be the junction tree obtained from the variable contraction of Definition 1. Then $\phi_{\hat{T}}$ minimizes the Kullback-Liebler divergence $D(\phi_T \,||\, \cdot)$ over all distributions in which $\boldsymbol{u}_i \perp\!\!\!\perp \boldsymbol{u}_{c\,-\,s} \,|\, \boldsymbol{u}_{s \setminus i}$.*

Alternatively, the probability distribution represented by $\hat{T}$ has maximum likelihood (under the original junction tree's distribution) over all distributions in which $\boldsymbol{u}_i$ is conditionally independent of $\boldsymbol{u}_{c-s}$ given $\boldsymbol{u}_{s \setminus i}$. Thus, we can consider each variable contraction to be a *maximum likelihood projection* that cuts edges between $i$ and $c - s$.

To reduce the width of a given junction tree, we should choose the variable contraction that minimizes the approximation error, which we take to be the Kullback-Liebler divergence from the original to the approximate distribution, $D(\phi_T \,||\, \phi_{\hat{T}})$. This approximation error can be computed efficiently, as shown by the following result (cf. [Kjærulff, 1993, Theorem 11]):

**Proposition 5.** *Let $\hat{T}$ be the junction tree obtained from the variable contraction of Definition 1. Then*

$$D(\phi_T \,||\, \phi_{\hat{T}}) = I(\boldsymbol{u}_i; \boldsymbol{u}_{c-s} \,|\, \boldsymbol{u}_{s \setminus i}) \tag{16}$$

To compute the conditional mutual information (16) we need only the marginal $p(\boldsymbol{u}_c)$. In a consistent junction tree, this marginal is simply $\phi_c$, and therefore the approximation error of a variable contraction can be computed locally. When $p(\boldsymbol{u}_c)$ is a Gaussian distribution, the computation is especially efficient: its cost depends only the dimension of $\boldsymbol{u}_i$.

**Proposition 6.** *Let $c$ index a set of Gaussian random variables $\mathbf{u}_c$, let $s \subseteq c$, and let $i \in s$. Then*

$$I(\boldsymbol{u}_i; \boldsymbol{u}_{c-s} \,|\, \boldsymbol{u}_{s \setminus i}) = \frac{1}{2} \left( \log |\Lambda^c_{ii}| - \log |\Lambda^s_{ii}| \right) \tag{17}$$

*where* $\Lambda^c = \mathrm{Cov}\,(\boldsymbol{u}_c)^{-1}$ *and* $\Lambda^s = \mathrm{Cov}\,(\boldsymbol{u}_s)^{-1}$.

$\Lambda^c$ and $\Lambda^s$ are parameters of the potentials $\phi_c$ and $\phi_s$, so we can simply extract from each the sub-blocks corresponding to $\boldsymbol{u}_i$ and compute the difference of their log determinants.

## 5 Thin junction tree filters for SLAM

We have now assembled most of the machinery required to design a thin junction tree filter for the SLAM problem. All that remains is the logic to decide into which clusters new potentials are multiplied and also how variable contractions are employed to thin the junction tree. There are many possibilities; the method below presents a nice compromise between simplicity and performance. We then describe a refinement that can reduce the time complexity from linear to constant.

### 5.1 Linear-time approximate filtering

Recall from Section 3.1 that if the width of our junction tree is $k$, then it will require $O(|V|k^2)$ space and message passing will take $O(|V|k^3)$ time. In SLAM $|V| = n_t + 1$, so we can obtain a $O(n_t)$ space filter with a $O(n_t)$ time filter operation by periodically thinning the junction tree so that its width remains bounded by a constant $k$.

We start with roll-up. When the robot state $\boldsymbol{x}_t$ 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 clusters, in which case our belief state will collapse to one large cluster. To prevent this, we iteratively contract $\boldsymbol{x}_t$ (choosing the contraction that minimizes the error (17) each time) until it resides in only one cluster $c$. Then, we perform the time update, which consists of multiplying the motion potential $\psi(\boldsymbol{x}_t, \boldsymbol{x}_{t+1})$ into $\phi_c$, marginalizing $\boldsymbol{x}_t$ out of $c$, multiplying the odometry potential $\psi(\boldsymbol{x}_{t+1})$ into $\phi_c$, and distributing evidence from $c$.

When multiplying in a landmark measurement potential $\psi(\boldsymbol{x}_t, \boldsymbol{l}_j)$ for a landmark that is currently in the model, we use the method of Section 3.2, i.e., we push $\boldsymbol{x}_t$ until it resides in a cluster $c$ with $\boldsymbol{l}_j$. (This may increase the sizes of some clusters, but the subsequent contraction of $\boldsymbol{x}_t$ in the next roll-up ensures this increase in cluster size is temporary.) We then multiply $\psi(\boldsymbol{x}_t, \boldsymbol{l}_j)$ into $\phi_c$ and distribute evidence from $c$.

If instead the landmark $j$ has not previously been observed, we must add the new variable $\boldsymbol{l}_j$ to the model. If the smallest cluster that contains $\boldsymbol{x}_t$ (call it $c$) can admit another variable without violating the width limit $k$, we add $\boldsymbol{l}_j$ to $c$ and multiply $\psi(\boldsymbol{x}_t, \boldsymbol{l}_j)$ into $\phi_c$. If not, then we clone $c$ to obtain $d$, contract $\boldsymbol{x}_t$ until it resides only in $d$, and thin $d$ via a sequence of greedy optimal variable contractions. A *cluster overlap* parameter $h$ governs the size to which $d$ is thinned, and therefore how many variables reside in the separator $s$ that joins it to $c$ (since in this case $s = d \setminus \boldsymbol{x}_t$). If $h$ is small, $d$ will admit more new landmark variables before another cloning is required; the trade-off is that its separator $s$ will shrink, reducing the amount of information it can transmit.

### 5.2 Constant-time approximate filtering

The linear time complexity of the filter above arises mainly because we pass messages to every cluster each time we distribute evidence from some cluster $c$. We can get a constant-time filter operation by employing a *lazy message passing* scheme, where we distribute evidence only to constantly many nearby clusters; the approximation is that the marginals of the remaining clusters will not be conditioned on the observation. This introduces minimal error when the observation is uninformative about distant variables; this occurs, e.g., when the robot is mapping new territory. Moreover, because we are still updating the charge correctly, this approximation is temporary: at any later time a full round of message passing (taking linear time) will yield the same estimate we would have obtained by passing all messages at every time step.

Alternatively, we can interpolate between the linear-time update and this constant-time update by employing an *adaptive message passing* scheme in which messages are propagated only as long as they induce significant changes in the belief state. If we define "significant" sensibly, this scheme will take constant time when mapping new territory; when closing loops, it will take time linear in the length of the loop.

We measure the significance of a message $c_i \to c_j$ over the separator $s$ by $D(\phi_s^* \,||\, \phi_s)$, the Kullback-Liebler divergence from the new separator marginal to the original separator marginal. In the Gaussian case this is

$$\frac{1}{2}\left[\log\frac{|\Sigma|}{|\Sigma^*|} - |s| + \mathrm{tr}(\Sigma^{-1}(\Sigma^* + (\mu^* - \mu)(\mu^* - \mu)^T))\right]$$

where $\phi_s = \mathcal{N}(\mu, \Sigma)$ and $\phi_s^* = \mathcal{N}(\mu^*, \Sigma^*)$. Importantly, the significance of evidence propagated from a cluster $c_i$ to another cluster $c_k$ (measured in this way) decreases with the distance between them in the junction tree [Kjærulff, 1993, Theorem 13]. Thus, if a message was not significant for a cluster, it need not continue the evidence distribution.

## 6 Experiments

Here we present a summary of our findings; the technical report contains more detail and further experiments.

We compared TJTF, the Kalman filter, and FastSLAM on large-scale SLAM simulations in which a robot moves around in a square world that is populated with uniformly distributed point-landmarks. Its motion and measurement models are all subject to significant noise and are linearized using the unscented transformation. We used two types of trajectories: a square loop (similar to that a robot mapping an indoor environment might travel) and a switchback trajectory (which could be used to map a large open area). Noise and controls were determined in advance so the robot followed the desired path and each filter received identical observations.

Figure 4 shows two examples of our simulations. The filters are evaluated by their computational cost (millions of floating point operations), localization error (the distance from the robot's position to the filter's estimate) and map error (the average distance from each landmark to the filter's estimate). TJTF was run with the width limit $k = 16$, the cluster overlap $h = 4$, and adaptive message passing with the significance threshold set at $0.1$ nats; FastSLAM was run with 100 particles, as recommended in [Montemerlo *et al.*, 2002].

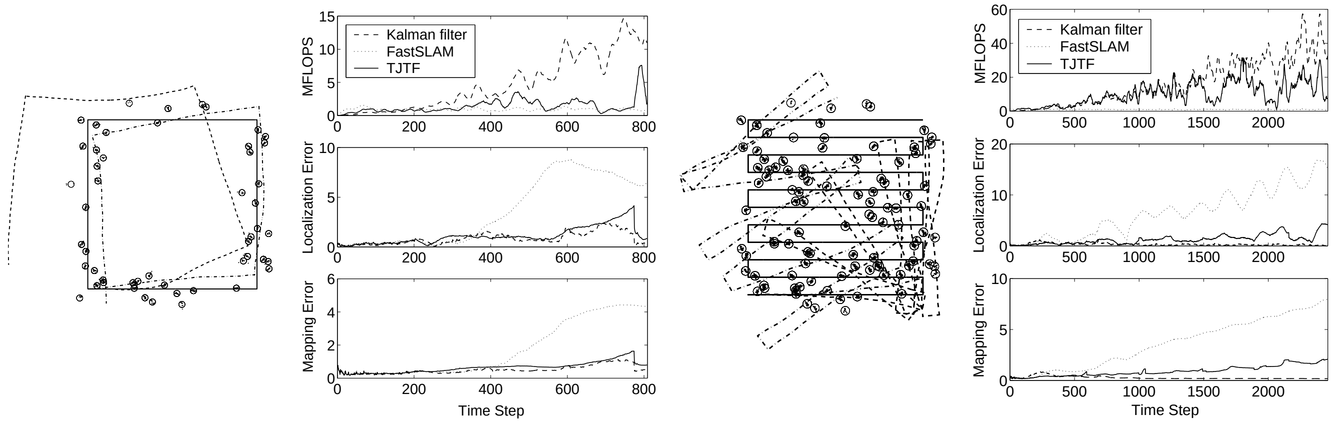
We found that the estimation error of TJTF with maximum cluster sizes as small as 16 can be comparable with the Kalman filter, and that it gets smaller as $k$ increases. This indicates that the edges removed by TJTF indeed carry little information; it also suggests that the estimation error of TJTF will be at least competitive with that of SEIF (an approximate form of the Kalman filter) and less than that of the submap approaches (which neglect long-distance correlations).

We also found that TJTF is good at closing loops; in Figure 4(b) we can see the localization and mapping error of TJTF suddenly drop at $t = 780$, when the robot first reobserves its starting point; also evident is a sudden increase in the computational cost: the filter is choosing to update the entire map in linear time rather than using cheaper constant-time updates. We found that FastSLAM had difficulty closing large loops (notice its divergence in Figure 4(b)) and that its estimation error in general was larger than that of TJTF.

Finally, using accurate counts of floating point operations, we found that TJTF can be as fast as FastSLAM, and that it becomes more efficient than the Kalman filter when the map contains a few hundred or more landmarks.

## 7 Conclusion

We believe thin junction tree filters are a promising approximation technique for dynamic probabilistic inference. First, they are flexible, in that they allow the practitioner to trade computational complexity for approximation accuracy by varying the width of the junction tree and the depth of



(a) a square problem (b) filter profile of (a) (c) a switchback problem (d) filter profile of (c)

Figure 4: In (a) and (c): the solid line is the actual robot path; the dashed line is the integrated odometry; the dash-dotted line is the integrated control signal; circles are landmarks; dots are landmark observations (relative to the unknown actual viewpoint); for clarity, only some of the 1000 landmarks are plotted. In (b) and (d) the floating point counts are time-averaged for clarity.

evidence propagation. Second, the error of each local approximation can be computed exactly, giving an important indication of how trustworthy the approximate estimates are. Finally, the TJTF approximation is context sensitive in that it is not chosen in advance; rather, the approximation is chosen adaptively to minimize the approximation error.

When applied to the SLAM problem, TJTF performs competitively with the exact filter, but with superior asymptotic space and time complexity. Interestingly, the approach presented here has significant connections to both the submap approach and SEIF. First, like SEIF, TJTF cuts "weak" edges from the graphical model to speed inference; however, in TJTF we can use exact inference over this approximate model, whereas SEIF must use approximate inference. Second, the belief state of a TJTF has 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. The TJTF 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.

**Acknowledgements**

I gratefully acknowledge Intel Corporation for supporting this research via an Intel Research Internship, as well as Barbara Engelhardt, Kevin Murphy, Stuart Russell, and Sekhar Tatikonda for valuable comments on a draft of this paper.

## References

[Boyen and Koller, 1998] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 14th Annual Conference on Uncertainty in AI*, pages 33–42, 1998.

[Boyen and Koller, 1999] X. Boyen and D. Koller. Exploiting the architecture of dynamic systems. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 313–320, 1999.

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

[Draper, 1995] D. Draper. Clustering without (thinking about) triangulation. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference (UAI-95)*, pages 125–133, 1995.

[Hébert *et al.*, 1996] P. Hébert, S. Betgé-Brezetz, and R. Chatila. Probabilistic map learning: Necessity and difficulties. In L. Dorst, M. van Lambalgen, and F. Voorbraak, editors, *Reasoning with Uncertainty in Robotics*, volume 1093 of *Lecture Notes in Computer Science*. Springer, 1996.

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

[Montemerlo *et al.*, 2002] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 593–598, 2002.

[Paskin, 2002] M. Paskin. Thin junction tree filters for simultaneous localization and mapping. Technical Report CSD-02-1198, University of California, Berkeley, September 2002.

[Smith *et al.*, 1990] R. C. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I. J. Cox and G. T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag, 1990.

[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] S. Thrun, D. Koller, Z. Ghahmarani, H. Durrant-Whyte, and A. Ng. Simultaneous localization and mapping with sparse extended information filters: Theory and initial results. Technical Report CMU-CS-02-112, Carnegie Mellon University, September 2002.

[Thrun, 2002] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.