View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by DSpace@MIT

iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree

Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John Leonard, and Frank Dellaert

Abstract

We present a novel data structure, the Bayes tree, that provides an algorithmic foundation enabling a better understanding of existing graphical model inference algorithms and their connection to sparse matrix factorization methods. Similar to a clique tree, a Bayes tree encodes a factored probability density, but unlike the clique tree it is directed and maps more naturally to the square root information matrix of the simultaneous localization and mapping (SLAM) problem. In this paper, we highlight three insights provided by our new data structure. First, the Bayes tree provides a better understanding of the matrix factorization in terms of probability densities. Second, we show how the fairly abstract updates to a matrix factorization translate to a simple editing of the Bayes tree and its conditional densities. Third, we apply the Bayes tree to obtain a completely novel algorithm for sparse nonlinear incremental optimization, named iSAM2, which achieves improvements in efficiency through incremental variable re-ordering and fluid relinearization, eliminating the need for periodic batch steps. We analyze various properties of iSAM2 in detail, and show on a range of real and simulated datasets that our algorithm compares favorably with other recent mapping algorithms in both quality and efficiency.

Keywords: graphical models, clique tree, junction tree, probabilistic inference, sparse linear algebra, nonlinear optimization, smoothing and mapping, SLAM

1 Introduction

Probabilistic inference algorithms are important in robotics for a number of applications, ranging from simultaneous localization and mapping (SLAM) for building geometric models of the world, to tracking people for human robot interaction. Our research is mainly in large-scale SLAM and hence we will use this as an example throughout the paper. SLAM is a core competency for autonomous robots, as it provides the necessary data for many other important tasks such as planning and manipulation, in addition to direct applications such as navigation, exploration, and 3D modeling. The uncertainty inherent in sensor measurements makes probabilistic inference algorithms the favorite choice for SLAM. Online operation is essential for most real applications, therefore our work focuses on efficient incremental online algorithms.

Taking a graphical model perspective to probabilistic inference in SLAM has a rich history (Brooks, 1985) and has especially led to several novel and exciting developments in the last years (Paskin, 2003; Folkesson and Christensen, 2004; Frese et al., 2005; Frese, 2006; Folkesson and Christensen, 2007; Ranganathan et al., 2007). Paskin (2003) proposed the thin junction tree filter (TJTF), which provides an incremental solution directly based on graphical models. However, filtering is applied, which is known to be inconsistent when applied to the inherently nonlinear SLAM problem (Julier and Uhlmann, 2001), i.e., the average taken over a large number of experiments diverges from the true solution. In contrast, *full SLAM* (Thrun et al., 2005) retains all robot poses and can provide an exact solution, which does not suffer from inconsistency. Folkesson and Christensen (2004) presented Graphical SLAM, a graph-based full SLAM solution that includes mechanisms for reducing the complexity by locally reducing the number of variables. More closely related, Treemap by Frese (2006) performs QR factorization within nodes of a tree. Loopy SAM (Ranganathan et al., 2007) applies loopy belief propagation directly to the SLAM graph.

The sparse linear algebra perspective has been explored in Smoothing and Mapping (SAM) (Dellaert, 2005; Dellaert and Kaess, 2006; Kaess et al., 2007, 2008). The matrices associated with smoothing are typically very sparse, and one can do much better than the cubic complexity associated with factorizing a dense matrix (Krauthausen et al., 2006). Kaess et al. (2008) proposed incremental smoothing and mapping (iSAM), which performs *fast incremental updates* of the square root information matrix, yet is able to compute the full map and trajectory at any time. New measurements are added using matrix update equations (Gill et al., 1974; Gentleman, 1973; Golub and Loan, 1996), so that previously calculated components of the square root information matrix are reused. However, to remain efficient and consistent, iSAM requires periodic batch steps to allow for variable reordering and relin-

Draft manuscript, April 6, 2011. Submitted to IJRR.

M. Kaess, H. Johannsson, and J. Leonard are with the Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), Cambridge, MA 02139, USA {kaess, hordurj, jleonard}@mit.edu.

R. Roberts, V. Ila, and F. Dellaert are with the School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA {richard, vila, frank}@cc.gatech.edu.

This work was presented in part at the International Workshop on the Algorithmic Foundations of Robotics, Singapore, December 2010, and in part at the International Conference on Robotics and Automation, Shanghai, China, May 2011.

earization, which is expensive and detracts from the intended online nature of the algorithm.

To combine the advantages of the graphical model and sparse linear algebra perspectives, *we propose a novel data structure, the Bayes tree*, first presented in Kaess et al. (2010). Our approach is based on viewing matrix factorization as eliminating a factor graph into a Bayes net, which is the graphical model equivalent of the *square root information matrix*. Performing marginalization and optimization in Bayes nets is not easy in general. However, a Bayes net resulting from elimination/factorization is *chordal,* and it is well known that a chordal Bayes net can be converted into a tree-structured graphical model in which these operations are easy. This data structure is similar to the *clique tree* (Pothen and Sun, 1992; Blair and Peyton, 1993; Koller and Friedman, 2009), also known as the *junction tree* in the AI literature (Cowell et al., 1999), which has already been exploited for distributed inference in SLAM (Dellaert et al., 2005; Paskin, 2003). However, the new data structure we propose here, the Bayes tree, is *directed* and corresponds more naturally to the result of the QR factorization in linear algebra, allowing us to analyze it in terms of conditional probability densities in the tree. We further show that incremental inference corresponds to a simple editing of this tree, and present a novel incremental variable ordering strategy.

Exploiting this new data structure and the insights gained, we propose iSAM2, *a novel incremental exact inference method that allows for incremental reordering and just-in-time relinearization.* iSAM2, first presented in Kaess et al. (2011), extends our original iSAM algorithm by leveraging these insights about the connections between graphical model and sparse linear algebra perspectives. To the best of our knowledge this is a completely novel approach to providing an efficient and exact solution to a sparse nonlinear optimization problem in an incremental setting, with general applications beyond SLAM. While standard nonlinear optimization methods repeatedly solve a linear batch problem to update the linearization point, our Bayes tree-based algorithm allows fluid relinearization of a reduced set of variables, which translates into higher efficiency, while retaining sparseness and full accuracy. Fig. 1 shows an example of the Bayes tree for a small SLAM sequence. As a robot explores the environment, new measurements only affect parts of the tree, and only those parts are re-calculated.

A detailed evaluation of iSAM2 and comparison with other state-of-the-art SLAM algorithms is provided. We explore the impact of different variable ordering strategies on the performance of iSAM2. Furthermore, we evaluate the effect of the relinearization and update thresholds as a trade-off between speed and accuracy, showing that large savings in computation can be achieved while still obtaining an almost exact solution. Finally, we present a detailed comparison with other SLAM algorithms in terms of computation and accuracy, using a range of 2D and 3D, simulated and real-world, pose-only and landmark-based datasets.

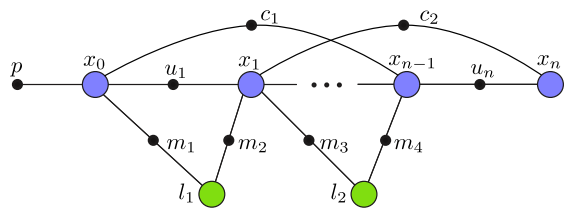


Fig. 2: Factor graph (Kschischang et al., 2001) formulation of the SLAM problem, where variable nodes are shown as large circles, and factor nodes (measurements) as small solid circles. The factors shown are odometry measurements u, a prior p, loop closing constraints c and landmark measurements m. Special cases include the pose-graph formulation (without l and m) and landmark-based SLAM (without c). Note that the factor graph can represent any cost function, involving one, two or more variables (e.g. calibration).

2 Problem Statement

This work focuses on how to efficiently solve a nonlinear estimation problem in an incremental and real-time approach. Incremental means that an updated estimate has to be obtained whenever new measurements are added, to reflect the most accurate model of the environment that can be derived from all measurements gathered so far. Online means that the estimate has to become available during robot operation, and not from a batch computation after the robot's task is finished, as the estimate is needed by the robot for navigation and planning to achieve a given goal.

We use a *factor graph* (Kschischang et al., 2001) to represent a given estimation problem in terms of graphical models. Formally, a factor graph is a bipartite graph $G = (\mathscr{F}, \Theta, \mathscr{E})$ with two node types: *factor nodes* $f_i \in \mathscr{F}$ and *variable nodes* $\theta_j \in \Theta$. Edges $e_{ij} \in \mathscr{E}$ are always between factor nodes and variables nodes. A factor graph G defines the factorization of a function $f(\Theta)$ as

$$f(\Theta) = \prod_i f_i(\Theta_i) \tag{1}$$

where Θ_i is the set of variables θ_j adjacent to the factor f_i, and independence relationships are encoded by the edges e_{ij}: each factor f_i is a function of the variables in Θ_i. Our goal is to find the variable assignment Θ^* that maximizes (1)

$$\Theta^* = \arg\max_{\Theta} f(\Theta) \tag{2}$$

The general factor graph formulation of the SLAM problem is shown in Fig. 2, where the landmark measurements m, loop closing constraint c and odometry measurements u are examples of factors. Note that this formulation allows our work to support general probability distributions or cost functions of any number of variables, allowing the inclusion of calibration parameters or spatial separators as used in T-SAM (Ni et al., 2007) and cooperative mapping (Kim et al., 2010).

Gaussian Case

When assuming Gaussian measurement models

$$f_i(\Theta_i) \propto \exp\left(-\frac{1}{2}\|h_i(\Theta_i) - z_i\|^2_{\Sigma_i}\right) \tag{3}$$

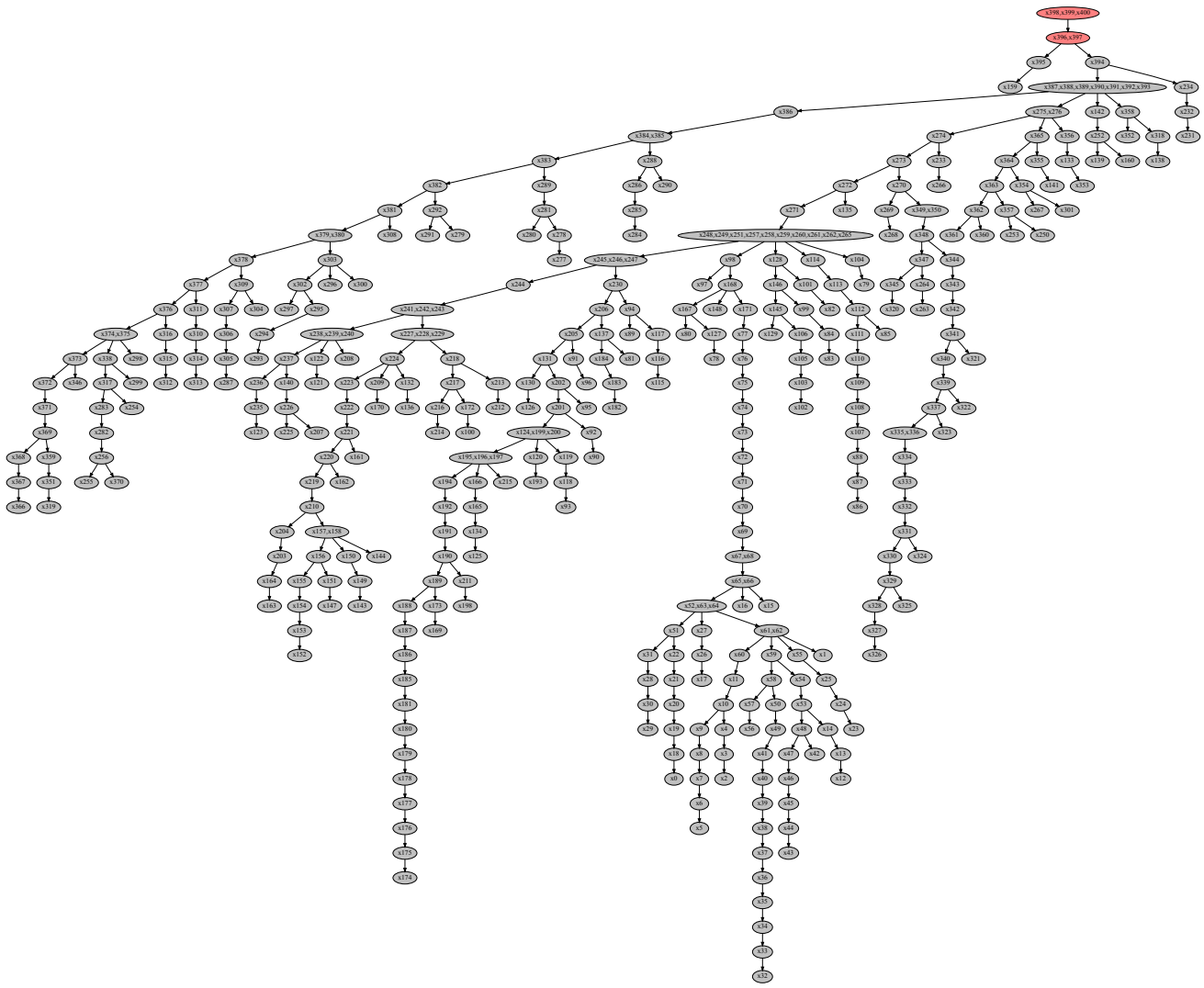


Fig. 1: An example of the Bayes tree data structure, showing step 400 of the Manhattan sequence (see Extension 1 in Appendix A for an animation of the full sequence together with the map). Our incremental nonlinear least-squares estimation algorithm iSAM2 is based on viewing incremental factorization as editing the graphical model corresponding to the posterior probability of the solution, the Bayes tree. As a robot explores the environment, new measurements often only affect small parts of the tree, and only those parts are re-calculated.

as is standard in the SLAM literature (Smith et al., 1987; Castellanos et al., 1999; Dissanayake et al., 2001), the factored objective function to maximize (2) corresponds to the nonlinear least-squares criterion

$$\arg\min_{\Theta}\left(-\log f(\Theta)\right) = \arg\min_{\Theta} \frac{1}{2}\sum_i \|h_i(\Theta_i) - z_i\|^2_{\Sigma_i} \qquad (4)$$

where $h_i(\Theta_i)$ is a measurement function and z_i a measurement, and $\|e\|^2_\Sigma \triangleq e^T\Sigma^{-1}e$ is defined as the squared Mahalanobis distance with covariance matrix Σ.

In practice one typically considers a linearized version of problem (4). For nonlinear measurement functions h_i in (3), nonlinear optimization methods such as Gauss-Newton iterations or the Levenberg-Marquardt algorithm solve a succession of linear approximations to (4) in order to approach the minimum. At each iteration of the nonlinear solver, we linearize around a linearization point Θ to get a new, *linear* least-squares problem in Δ

$$\arg\min_{\Delta}\left(-\log f(\Delta)\right) = \arg\min_{\Delta} \|A\Delta - \mathbf{b}\|^2 \qquad (5)$$

where $A \in \mathbb{R}^{m\times n}$ is the measurement Jacobian consisting of m measurement rows, and Δ is an n-dimensional vector. Note that the covariances Σ_i have been absorbed into the corresponding block rows of A, making use of

$$\|\Delta\|^2_\Sigma = \Delta^T\Sigma^{-1}\Delta = \Delta^T\Sigma^{-\frac{T}{2}}\Sigma^{-\frac{1}{2}}\Delta = \left\|\Sigma^{-\frac{1}{2}}\Delta\right\|^2 \qquad (6)$$

Once Δ is found, the new estimate is given by $\Theta \oplus \Delta$, which is then used as linearization point in the next iteration of the nonlinear optimization. The operator $\oplus$ is often simple addition, but for over-parameterized representations such as Quaternions for 3D orientations or homogeneous point representa-

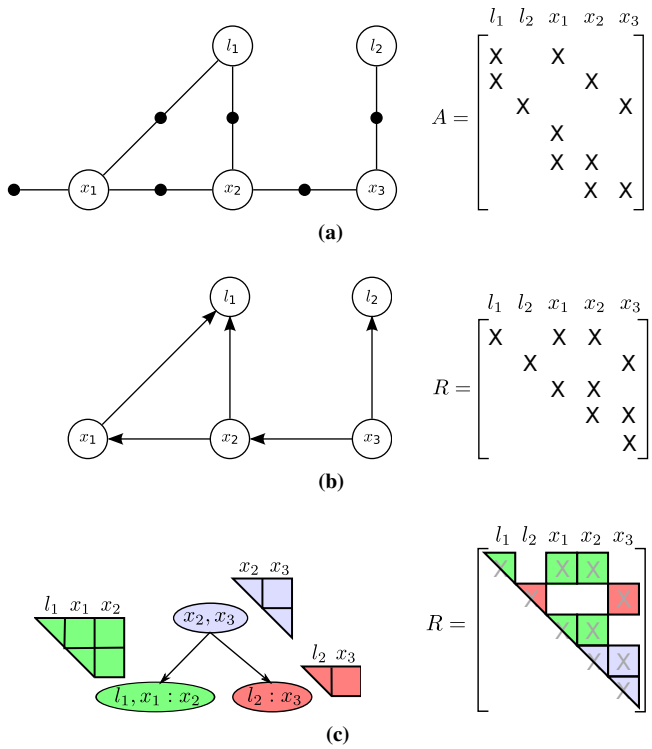


Fig. 3: (a) The factor graph and the associated Jacobian matrix A for a small SLAM example, where a robot located at successive poses x_1, x_2, and x_3 makes observations on landmarks l_1 and l_2. In addition there is an absolute measurement on the pose x_1. (b) The chordal Bayes net and the associated square root information matrix R resulting from eliminating the factor graph using the elimination ordering l_1, l_2, x_1, x_2, x_3. The last variable to be eliminated, here x_3, is called the root. (c) The Bayes tree and the associated square root information matrix R describing the clique structure in the chordal Bayes net. A Bayes tree is similar to a junction tree, but is better at capturing the formal equivalence between sparse linear algebra and inference in graphical models. The association of cliques and their conditional densities with rows in the R factor is indicated by color.

tions in computer vision, an exponential map based on Lie group theory (Hall, 2000) is used instead.

The matrix A above is a sparse block-matrix, and its graphical model counterpart is a *Gaussian* factor graph (i.e. the original factor graph linearized at Θ) with exactly the same structure as the nonlinear factor graph, see the small SLAM example in Fig. 3a. The probability density on Δ defined by this factor graph is the normal distribution

$$P(\Delta) \propto e^{-\log f(\Delta)} = \exp\left\{-\frac{1}{2}\|A\Delta - \mathbf{b}\|^2\right\} \tag{7}$$

The minimum of the linear system $A\Delta - \mathbf{b}$ can be obtained directly either by Cholesky or QR matrix factorization. By setting the derivative in Δ to zero we obtain the normal equations $A^T A\Delta = A^T\mathbf{b}$. Cholesky factorization yields $A^T A = R^T R$, and a forward and backsubstitution on $R^T\mathbf{y} = A^T\mathbf{b}$ and $R\Delta = \mathbf{y}$ first recovers $\mathbf{y}$, then the actual solution, the update Δ. Alternatively we can skip the normal equations and apply QR factorization, yielding $R\Delta = \mathbf{d}$, which can directly be solved by backsubstitution. Note that Q is not explicitly formed; instead $\mathbf{b}$ is modified during factorization to obtain $\mathbf{d}$, see Kaess et al. (2008) for details. Alg. 1 shows a summary of the necessary steps to solve the smoothing formulation of the SLAM problem with direct methods.

Alg. 1 General structure of the smoothing solution to SLAM with a direct equation solver (Cholesky, QR). Steps 3-6 can optionally be iterated and/or modified to implement the Levenberg-Marquardt algorithm.

Repeat for new measurements in each step:

1. Add new measurements.
2. Add and initialize any new variables.
3. Linearize at current estimate Θ.
4. Factorize with QR or Cholesky.
5. Solve by backsubstitution to obtain Δ.
6. Obtain new estimate $\Theta' = \Theta \oplus \Delta$.

Incremental and online smoothing can be achieved by our original iSAM algorithm (Kaess et al., 2008), but relinearization is only performed during periodic batch reordering steps. A batch solution, as proposed above, performs unnecessary calculations, because it solves the complete problem at every step, including all previous measurements. New measurements often have only a local effect, leaving remote parts of the map untouched. iSAM exploits that fact by incrementally updating the square root information matrix R with new measurements. The updates are performed with Givens rotations and often only affect a small part of the matrix, therefore being much cheaper than batch factorization. However, as new variables are appended, the variable ordering is far from optimal, and fill-in may occur. iSAM performs periodic batch steps, in which the variables are reordered, requiring a batch factorization. That solution is not optimal as linearization is only performed during batch steps, and because the frequency of the periodic batch steps is determined heuristically.

3 The Bayes Tree

In this section we describe how the estimation problem can be solved by directly operating on the graphical models, without converting the factor graph to a sparse matrix and then applying sparse linear algebra methods.

3.1 Inference and Elimination

A crucial insight is that *inference can be understood as converting the factor graph to a Bayes net using the elimination algorithm.* Variable elimination (Blair and Peyton, 1993; Cowell et al., 1999) originated in order to solve systems of linear equations, and was first applied in modern times by Gauss in the early 1800s (Gauss, 1809).

In factor graphs, elimination is done via a *bipartite elimination game*, as described by Heggernes and Matstoms (1996). This can be understood as taking apart the factor graph and transforming it into a *Bayes net* (Pearl, 1988). One proceeds by eliminating one variable at a time, and converting it into a node of the Bayes net, which is gradually built up. After

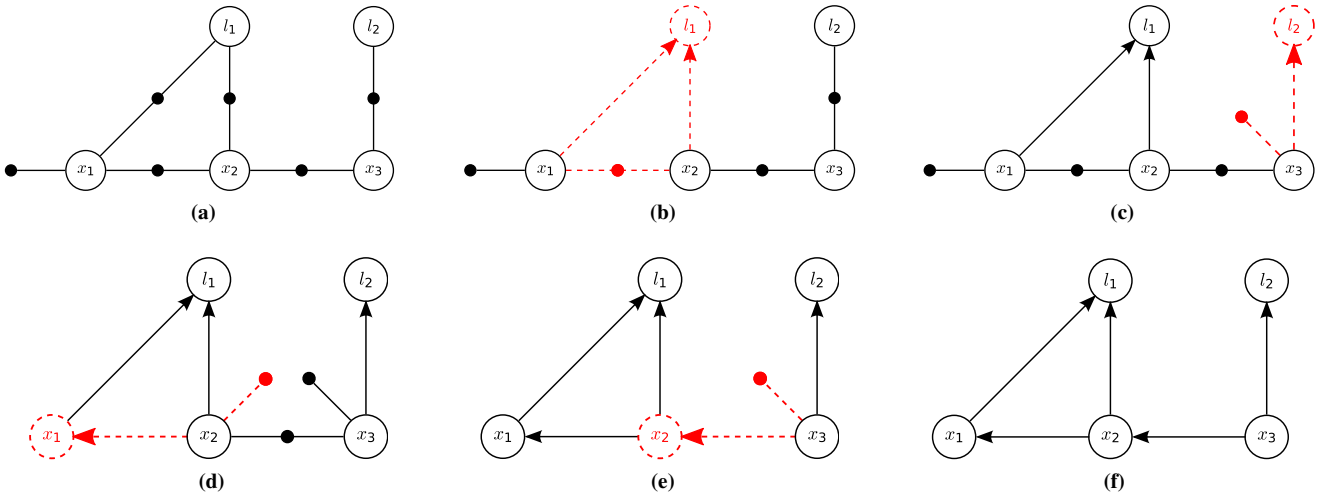


Fig. 4: Steps in the variable elimination process starting with the factor graph of Fig. 3a and ending with the chordal Bayes net of Fig. 3b. Following Alg. 2, in each step one variable is eliminated (dashed red circle), and all adjacent factors are combined into a joint distribution. By applying the chain rule, this joint density is transformed into conditionals (dashed red arrows) and a new factor on the separator (dashed red factor). This new factor represents a prior that summarizes the effect of the eliminated variables on the separator.

Alg. 2 Eliminating a variable θ_j from the factor graph.

1. Remove from the factor graph all factors $f_i(\Theta_i)$ that are adjacent to θ_j. Define the *separator* S_j as all variables involved in those factors, excluding θ_j.
2. Form the (unnormalized) joint density $f_{joint}(\theta_j, S_j) = \prod_i f_i(\Theta_i)$ as the product of those factors.
3. Using the chain rule, factorize the joint density $f_{joint}(\theta_j, S_j) = P(\theta_j|S_j) f_{new}(S_j)$. Add the conditional $P(\theta_j|S_j)$ to the Bayes net and the factor $f_{new}(S_j)$ back into the factor graph.

eliminating each variable, the reduced factor graph defines a density on the remaining variables. The pseudo-code for eliminating a variable θ_j is given in Alg. 2. After eliminating all variables, the Bayes net density is defined by the product of the conditionals produced at each step:

$$P(\Theta) = \prod_j P(\theta_j|S_j) \tag{8}$$

where S_j is the separator of θ_j, that is the set of variables that are directly connected to θ_j by a factor. Fig. 3 shows both the factor graph and the Bayes net resulting from elimination for a small SLAM example.

For illustration, the intermediate steps of the elimination process are shown in Fig. 4, and we explain the first step here in detail. The factor graph in Fig. 4(a) contains the following six factors: $f(x_1)$, $f(x_1,x_2)$, $f(x_2,x_3)$, $f(l_1,x_1)$, $f(l_1,x_2)$, $f(l_2,x_3)$. The first variable to be eliminated is the first landmark l_1. Following Alg. 2, first we remove all factors involving this landmark ($f(l_1,x_1)$, $f(l_1,x_2)$), and define the separator $S = \{x_1,x_2\}$. Second, we combine the removed factors into a joint factor $f_{joint}(l_1,x_1,x_2)$. Third, we apply the chain rule to split the joint factor into two parts: The first part is a conditional density $P(l_1|x_1,x_2)$ over the eliminated variable given the separator, which shows up as two new arrows in Fig. 4(b). The second part created by the chain rule is a new factor $f(x_1,x_2)$ on the separator as shown in the figure. Note that this factor can also be unary as is the case in the next step when the second landmark l_2 is eliminated and the separator is a single variable, x_3. In all intermediate steps we have both an incomplete factor graph and an incomplete Bayes net. The elimination is complete after the last variable is eliminated and only a Bayes net remains. Speaking in terms of probabilities, the factors $\prod_i f_i(\Theta_i)$ have been converted into an equivalent product of conditionals $\prod_j P(\theta_j|S_j)$.

Gaussian Case

In Gaussian factor graphs, elimination is equivalent to sparse QR factorization of the measurement Jacobian. The chain-rule-based factorization $f_{joint}(\theta_j, S_j) = P(\theta_j|S_j) f_{new}(S_j)$ in step 3 of Alg. 2 can be implemented using Householder reflections or a Gram-Schmidt orthogonalization, in which case the entire elimination algorithm is equivalent to QR factorization of the entire measurement matrix A. To see this, note that, for $\Delta_j \in \mathbb{R}$ and $\mathbf{s}_j \in \mathbb{R}^{n_j}$ (the set of variables S_j combined in a vector of length n_j), the factor $f_{joint}(\Delta_j, \mathbf{s}_j)$ defines a Gaussian density

$$f_{joint}(\Delta_j, \mathbf{s}_j) \propto \exp\left\{-\frac{1}{2}\left\|\mathbf{a}\Delta_j + \mathbf{A}_S\mathbf{s}_j - \mathbf{b}\right\|^2\right\} \tag{9}$$

where the dense, but small matrix $\mathbf{A}_j = [\mathbf{a}|\mathbf{A}_S]$ is obtained by concatenating the vectors of partial derivatives of all factors connected to variable Δ_j. Note that $\mathbf{a} \in \mathbb{R}^{m_j}$, $\mathbf{A}_S \in \mathbb{R}^{m_j \times n_j}$ and $\mathbf{b} \in \mathbb{R}^{m_j}$, with m_j the number of measurement rows of all factors connected to Δ_j. The desired conditional $P(\Delta_j|\mathbf{s}_j)$ is obtained by evaluating the joint density (9) for a given value of $\mathbf{s}_j$, yielding

$$P(\Delta_j|\mathbf{s}_j) \propto \exp\left\{-\frac{1}{2}(\Delta_j + \mathbf{r}\mathbf{s}_j - d)^2\right\} \tag{10}$$

with $\mathbf{r} \triangleq \mathbf{a}^{\dagger}\mathbf{A}_S$ and $d \triangleq \mathbf{a}^{\dagger}\mathbf{b}$, where $\mathbf{a}^{\dagger} \triangleq \left(\mathbf{a}^T\mathbf{a}\right)^{-1}\mathbf{a}^T$ is the *pseudo-inverse* of $\mathbf{a}$. The new factor $f_{new}(\mathbf{s}_j)$ is obtained by substituting $\Delta_j = d - \mathbf{r}\mathbf{s}_j$ back into (9):

$$f_{new}(\mathbf{s}_j) = \exp\left\{-\frac{1}{2}\left\|\mathbf{A}'\mathbf{s}_j - \mathbf{b}'\right\|^2\right\} \tag{11}$$

where $\mathbf{A}' \triangleq \mathbf{A}_S - \mathbf{a}\mathbf{r}$ and $\mathbf{b}' \triangleq \mathbf{b} - \mathbf{a}d$.

The above is one step of Gram-Schmidt, interpreted in terms of densities, and the sparse vector $\mathbf{r}$ and scalar d can be recognized as specifying a single joint conditional density in the Bayes net, or alternatively a single row in the sparse square root information matrix. The chordal Bayes net resulting from variable elimination is therefore equivalent to the square root information matrix obtained by variable elimination, as indicated in Fig. 3b. Note that alternatively an incomplete Cholesky factorization can be performed, starting from the information matrix A^TA.

Solving the least squares problem is finally achieved by calculating the optimal assignment Δ^* in one pass from the leaves up to the root of the tree to define all functions, and then one pass down to retrieve the optimal assignment for all frontal variables, which together make up the variables Δ. The first pass is already performed during construction of the Bayes tree, and is represented by the conditional densities associated with each clique. The second pass recovers the optimal assignment starting from the root based on (10) by solving

$$\Delta_j = d - \mathbf{r}\mathbf{s}_j \tag{12}$$

for every variable Δ_j, which is known as backsubstitution in sparse linear algebra.

3.2 Creating the Bayes Tree

In this section we introduce a new data structure, *the Bayes tree*, derived from the Bayes net resulting from elimination, to better capture the equivalence with linear algebra and enable new algorithms in recursive estimation. The Bayes net resulting from elimination/factorization is *chordal,* and it can be converted into a tree-structured graphical model, in which optimization and marginalization are easy. A Bayes tree is a directed tree where the nodes represent *cliques* C_k of the underlying chordal Bayes net. Bayes trees are similar to junction trees (Cowell et al., 1999), but a Bayes tree is directed and is closer to a Bayes net in the way it encodes a factored probability density. In particular, we define one conditional density $P(F_k|S_k)$ per node, with the *separator* S_k as the intersection $C_k \cap \Pi_k$ of the clique C_k and its parent clique Π_k, and the *frontal variables* F_k as the remaining variables, i.e. $F_k \triangleq C_k \setminus S_k$. We write $C_k = F_k : S_k$. This leads to the following expression for the joint density $P(\Theta)$ on the variables Θ defined by a Bayes tree,

$$P(\Theta) = \prod_k P(F_k|S_k) \tag{13}$$

Alg. 3 Creating a Bayes tree from the chordal Bayes net resulting from elimination (Alg. 2).

For each conditional density $P(\theta_j|S_j)$ of the Bayes net, in *reverse* elimination order:
If no parent ($S_j = \{\}$)
  start a new root clique F_r containing θ_j
else
  identify parent clique C_p that contains the first eliminated variable of S_j as a frontal variable
   if nodes $F_p \cup S_p$ of parent clique C_p are equal to separator nodes S_j of conditional
    insert conditional into clique C_p
  else
    start new clique C' as child of C_p containing θ_j

where for the root F_r the separator is empty, i.e., it is a simple prior $P(F_r)$ on the root variables. The way Bayes trees are defined, the separator S_k for a clique C_k is always a subset of the parent clique Π_k, and hence the directed edges in the graph have the same semantic meaning as in a Bayes net: conditioning.

Every chordal Bayes net can be transformed into a tree by discovering its cliques. Discovering cliques in chordal graphs is done using the maximum cardinality search algorithm by Tarjan and Yannakakis (1984), which proceeds in reverse elimination order to discover cliques in the Bayes net. The algorithm for converting a Bayes net into a Bayes tree is summarized in Alg. 3.

Gaussian Case

In the Gaussian case the Bayes tree is closely related to the square root information factor. The Bayes tree for the small SLAM example in Fig. 3a is shown in Fig. 3c. Each clique of the Bayes tree contains a conditional density over the variables of the clique, given the separator variables. All conditional densities together form the square root information matrix shown on the right-hand side of Fig. 3c, where the assignment between cliques and rows in the matrix are shown by color. Note that one Bayes tree can correspond to several different square root information factors, because the children of any node can be ordered arbitrarily. The resulting change in the overall variable ordering neither changes the fill-in of the factorization nor any numerical values, but just their position within the matrix.

3.3 Incremental Inference

We show that incremental inference corresponds to a simple editing of the Bayes tree, which also provides a better explanation and understanding of the otherwise abstract incremental matrix factorization process. In particular, we will now store and compute the square root information matrix R in the form of a Bayes tree $\mathscr{T}$. When a new measurement is added, for example a factor $f'(x_j, x_{j'})$, only the paths between the cliques containing x_j and $x_{j'}$ (respectively) and the root are affected. The sub-trees below these cliques are unaffected, as are any other sub-trees not containing x_j or $x_{j'}$. The affected

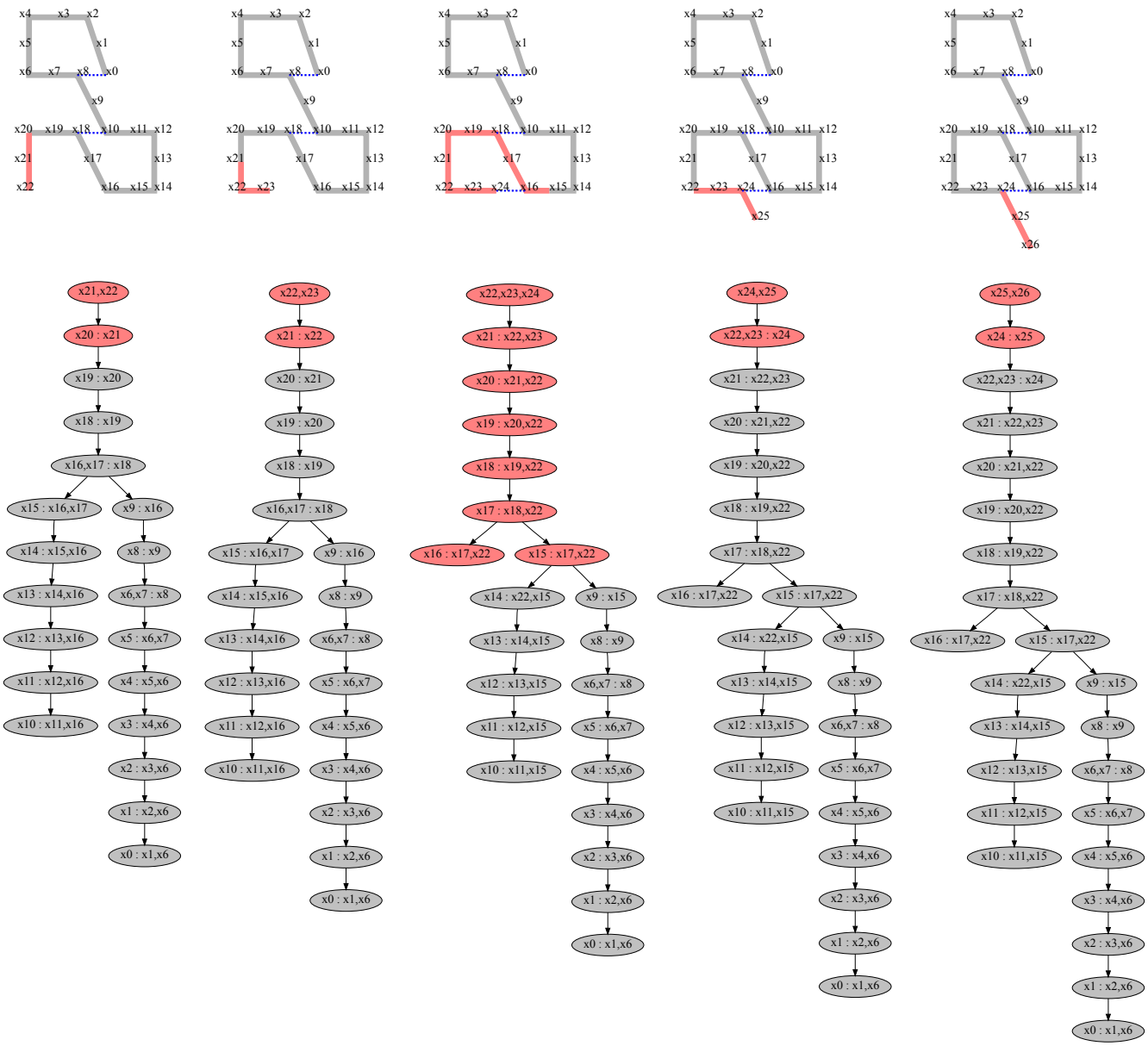


Fig. 5: Evolution of the Bayes tree: The columns represent five time steps for a small SLAM example. The top row shows the map with individual robot poses, with loop closures indicated in dashed blue. The bottom row depicts the Bayes tree, with modified cliques shown in red. Note the loop closure in the center that affects a subset of the variables, while two sub-trees remain unchanged. Also see Extension 1 in Appendix A for an animation for the full Manhattan sequence.

part of the Bayes tree is turned into a factor graph and the new factors are added to it. Using a new elimination ordering, a new Bayes tree is formed and the unaffected sub-trees are reattached. Fig. 6 shows how these incremental factorization/inference steps are applied to our small SLAM example in Fig. 3 for adding a new factor between x_1 and x_3, affecting only the left branch of the tree. The entire process of updating the Bayes tree with a new factor is described in Alg. 4.

In order to understand why only the top part of the tree is affected, we look at two important properties of the Bayes tree. These directly arise from the fact that it encodes the information flow during elimination. The Bayes tree is formed from the chordal Bayes net following the inverse elimination order. In this way, variables in each clique collect information *from their child cliques* via the elimination of these children. Thus, information in any clique propagates *only upwards* to the root. Second, the information from a factor enters elimination only when the first variable of that factor is eliminated. Combining these two properties, we see that a new factor cannot influence any other variables that are not successors of the factor's variables. However, a factor on variables having different (i.e. independent) paths to the root means that these paths must now be re-eliminated to express the new dependency between them.

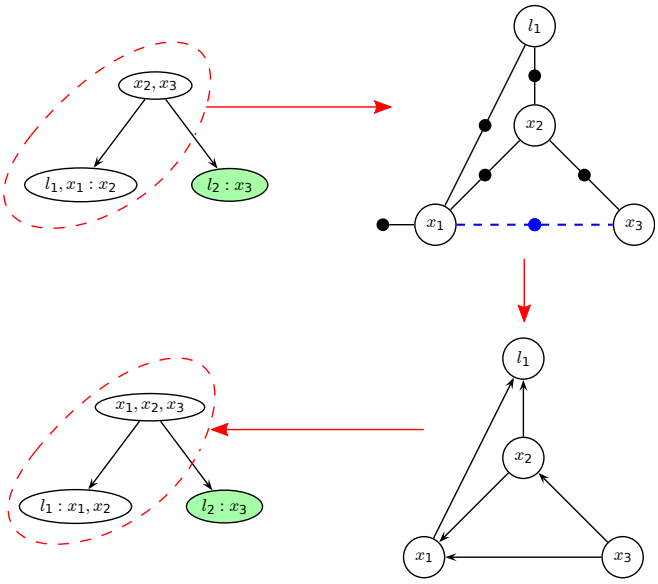


Fig. 6: Updating a Bayes tree with a new factor, based on the example in Fig. 3c. The affected part of the Bayes tree is highlighted for the case of adding a new factor between x_1 and x_3. Note that the right branch is not affected by the change. (top right) The factor graph generated from the affected part of the Bayes tree with the new factor (dashed blue) inserted. (bottom right) The chordal Bayes net resulting from eliminating the factor graph. (bottom left) The Bayes tree created from the chordal Bayes net, with the unmodified right "orphan" sub-tree from the original Bayes tree added back in.

Alg. 4 Updating the Bayes tree with new factors $\mathscr{F}'$.

In: Bayes tree $\mathscr{T}$, new linear factors $\mathscr{F}'$
Out: modified Bayes tree $\mathscr{T}$'

1. Remove top of Bayes tree and re-interpret it as a factor graph:
   (a) For each variable affected by new factors, remove the corresponding clique and all parents up to the root.
   (b) Store orphaned sub-trees $\mathscr{T}_{orph}$ of removed cliques.
2. Add the new factors $\mathscr{F}'$ into the resulting factor graph.
3. Re-order variables of factor graph.
4. Eliminate the factor graph (Alg. 2) and create a new Bayes tree (Alg. 3).
5. Insert the orphans $\mathscr{T}_{orph}$ back into the new Bayes tree.

3.4 Incremental Variable Ordering

Choosing a good variable ordering is essential for the efficiency of the sparse matrix solution, and this also holds for the Bayes tree approach. An optimal ordering of the variables minimizes the fill-in, which refers to additional entries in the square root information matrix that are created during the elimination process. In the Bayes tree, fill-in translates to larger clique sizes, and consequently slower computations. Fill-in can usually not be completely avoided, unless the original Bayes net already is chordal. While finding the variable ordering that leads to the minimal fill-in is NP-hard (Arnborg et al., 1987) for general problems, one typically uses heuristics such as the column approximate minimum degree (COLAMD) algorithm by Davis et al. (2004), which provide close to optimal orderings for many batch problems.

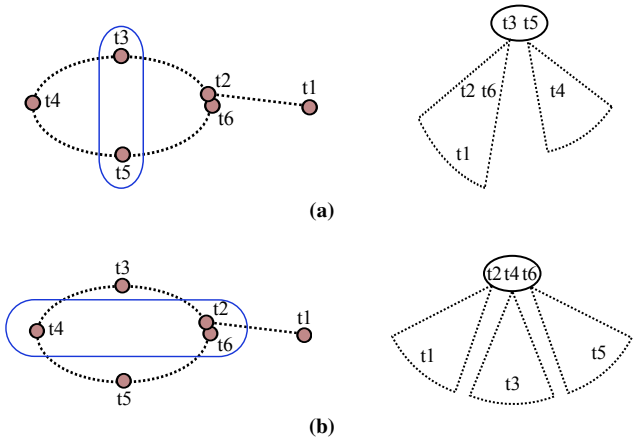


Fig. 7: For a trajectory with loop closing, two different optimal variable orderings based on nested dissection are shown on the left-hand side, with the variables to be eliminated marked in blue. For incremental updates the strategies are not equivalent as can be seen from the corresponding Bayes tree on the right-hand side. Adding factors connected to t_6 will affect (a) the left subtree and the root, (b) only the root. In the latter case incremental updates are therefore expected to be faster.

While performing incremental inference in the Bayes tree, variables can be reordered at every incremental update, eliminating the need for periodic batch reordering. This was not understood in (Kaess et al., 2008), because this is only obvious within the graphical model framework, but not for matrices. The affected part of the Bayes tree, for which variables have to be reordered, is typically small, as new measurements usually only affect a small subset of the overall state space represented by the variables of the estimation problem. Finding an optimal ordering for this subset of variables does not necessarily provide an optimal overall ordering. However, we have observed that some incremental orderings provide good solutions, comparable to batch application of COLAMD.

To understand how the local variable ordering affects the cost of subsequent updates, consider a simple loop example in Fig. 7. In the case of a simple loop, nested dissection (Lipton and Tarjan, 1979) provides the optimal ordering. The first cut can either (a) not include the loop closing, or (b) include the loop closing, and both solutions are equivalent in terms of fill-in. However, there is a significant difference in the incremental case: For the vertical cut in (a), which does not include the most recent variable t_6, that variable will end up further down in the tree, requiring larger parts of the tree to change in the next update step. The horizontal cut in (b), on the other hand, includes the most recent variable, pushing it into the root, and therefore leading to smaller, more efficient changes in the next step.

A similar problem occurs with applying COLAMD locally to the subset of the tree that is being recalculated. In the SLAM setting we can expect that a new set of measurements connects to some of the recently observed variables, be it landmarks that are still in range of the sensors, or the previous robot pose connected by an odometry measurement. The

expected cost of incorporating the new measurements, i.e. the size of the affected sub-tree in the update, will be lower if these variables are closer to the root. Applying COLAMD locally does not take this consideration into account, but only minimizes fill-in for the current step.

To allow for faster updates in subsequent steps, we therefore propose an incremental variable ordering strategy that forces the most recently accessed variables to the end of the ordering. We use the constrained COLAMD (CCOLAMD) algorithm (Davis et al., 2004) to both, force the most recently accessed variables to the end and still provide a good overall ordering. Subsequent updates will then only affect a small part of the tree, and can therefore be expected to be efficient in most cases, except for large loop closures.

We evaluate the merit of our proposed constrained ordering strategy in Fig. 8, by comparing it to the naive way of simply applying COLAMD. The top row of Fig. 8 shows a color coded trajectory of the Manhattan simulated dataset (Olson et al., 2006). The robot starts in the center, traverses the loop counter clockwise, and finally ends at the bottom left. The number of affected variables significantly drops from the naive approach (left) to the constrained approach (right), as red parts of the trajectory (high cost) are replaced by green (low cost). Particularly for the left part of the trajectory the number of affected variables is much smaller than before, which one would expect from a good ordering, as no large loops are being closed in that area. The remaining red segments coincide with the closing of the large loop in the right part of the trajectory. The second row of Fig. 8 shows that the constrained ordering causes a small increase in fill-in compared to the naive approach, which itself is close to the fill-in caused by the batch ordering. The bottom figure shows that the number of affected variables steadily increases for the naive approach, but often remains low for the constrained version, though the spikes indicate that a better incremental ordering strategy can likely be found for this problem.

4 The iSAM2 Algorithm

In this section we use the Bayes tree in a novel algorithm called iSAM2 for online mapping in robotic applications. Assuming Gaussian noise, the algorithm incrementally estimates a set of variables (robot positions and/or landmarks in the environment) given a set of nonlinear factors, both sets growing over time. We have already shown how the Bayes tree is updated with new linear factors. That leaves the question of how to deal with nonlinear factors and how to perform this process efficiently by only relinearizing where needed, a process that we call *fluid relinearization*. To further improve efficiency we restrict the state recovery to the variables that actually change, resulting in partial state updates.

4.1 Fluid Relinearization

The idea behind just-in-time or fluid relinearization is to keep track of the validity of the linearization point for each variable, and only relinearize when needed. This represents a departure from the conventional linearize/solve approach that currently represents the state-of-the-art for direct equation solvers. For a variable that is chosen to be relinearized, all relevant information has to be removed from the Bayes tree and replaced by relinearizing the corresponding original nonlinear factors. For cliques that are re-eliminated we also have to take into account any marginal factors that are passed up from their sub-trees. Caching those marginal factors during elimination allows restarting of the elimination process from the middle of the tree, rather than having to re-eliminate the complete system.

Alg. 5 Fluid relinearization: The linearization points of select variables are updated based on the current delta Δ.

In: linearization point Θ, delta Δ
Out: updated linearization point Θ, marked cliques M

1. Mark variables in Δ above threshold β: $J = \{\Delta_j \in \Delta | \left|\Delta_j\right| \geq \beta\}$.
2. Update linearization point for marked variables: $\Theta_J := \Theta_J \oplus \Delta_J$.
3. Mark all cliques M that involve marked variables Θ_J and all their ancestors.

Alg. 6 Updating the Bayes tree inclusive of fluid relinearization by recalculating all affected cliques. Note that the algorithm differs from Alg. 4 as it also includes the fluid relinearization; combining both steps is more efficient.

In: Bayes tree $\mathscr{T}$, nonlinear factors $\mathscr{F}$, affected variables $\mathscr{J}$
Out: modified Bayes tree $\mathscr{T}$'

1. Remove top of Bayes tree:
   (a) For each affected variable in $\mathscr{J}$ remove the corresponding clique and all parents up to the root.
   (b) Store orphaned sub-trees $\mathscr{T}_{orph}$ of removed cliques.
2. Relinearize all factors required to recreate top.
3. Add cached linear factors from orphans $\mathscr{T}_{orph}$.
4. Re-order variables, see Section 3.4.
5. Eliminate the factor graph (Alg. 2) and create a new Bayes tree (Alg. 3).
6. Insert the orphans $\mathscr{T}_{orph}$ back into the new Bayes tree.

Our fluid relinearization algorithm is shown in Alg. 5. The decision to relinearize a given variable is based on the deviation of its current estimate from the linearization point being larger than a threshold β. To be exact, the different units of variables have to be taken into account, but one simple solution is to take the minimum over all thresholds. For the Manhattan dataset, a nearly exact solution is provided for a threshold of 0.1, while the computational cost is significantly reduced, as can be seen from Fig. 9. Note that because we combine the relinearization and update steps for efficiency, the actual changes in the Bayes tree are performed later, which differs from the original algorithm in Kaess et al. (2010). The modified update algorithm is presented in Alg. 6.

4.2 Partial State Updates

Computational cost can be reduced significantly by realizing that recovering a nearly exact solution in every step does not

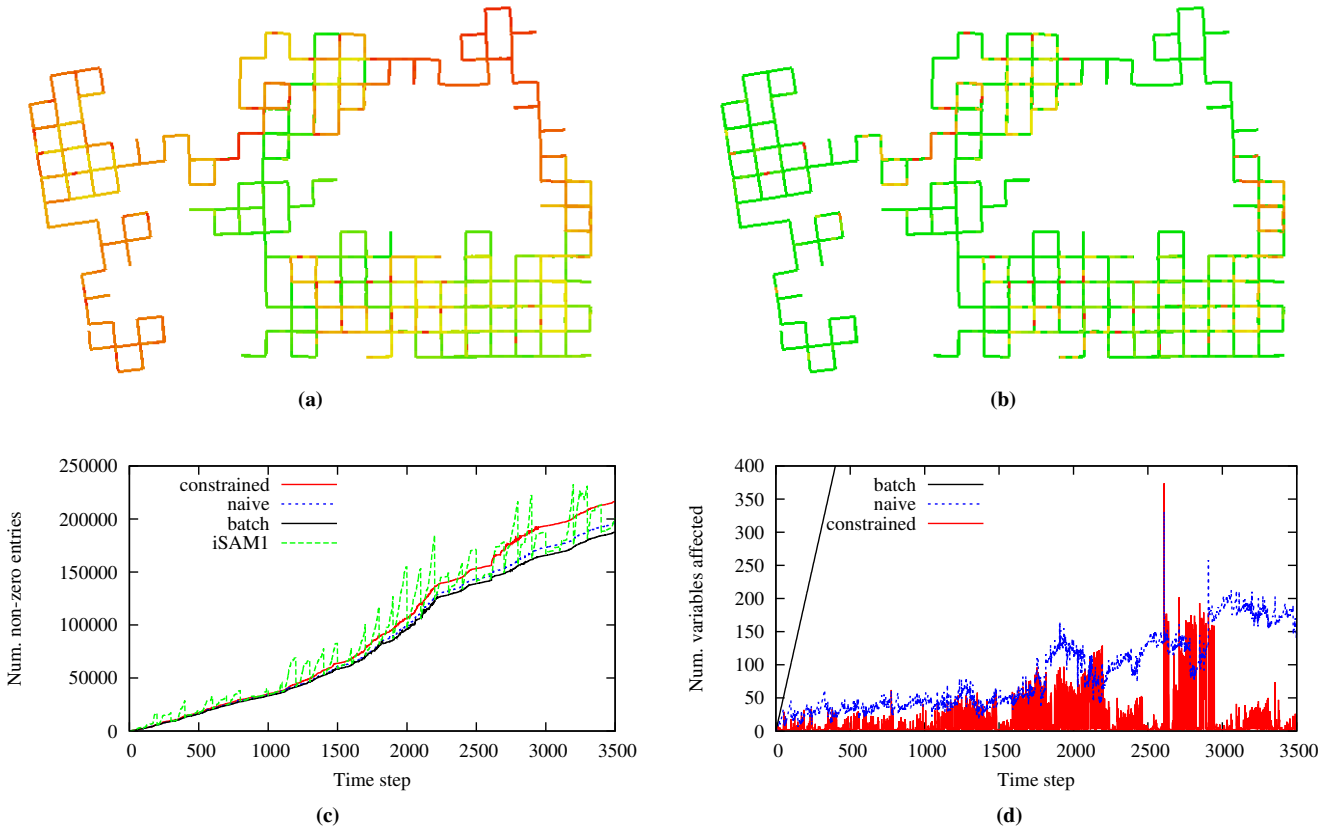


Fig. 8: Comparison of variable ordering strategies using the Manhattan world simulated environment (Olson et al., 2006). By color coding, the top row shows the number of variables that are updated for every step along the trajectory. Green corresponds to a low number of variables, red to a high number. (a) The naive approach of applying COLAMD to the affected variables in each step shows a high overall cost. (b) Forcing the most recently accessed variables to the end of the ordering using constrained COLAMD (Davis et al., 2004) yields a significant improvement in efficiency. (c) Fill-in over time for both strategies as well as the batch ordering and iSAM1. (d) Comparing the number of affected variables in each step clearly shows the improvement in efficiency achieved by the constrained ordering.

require solving for all variables. Updates to the Bayes tree from new factors and from relinearization only affect the top of the tree, however, changes in variable estimates occurring here can still propagate further down to all sub-trees. But the effect of changes in the top is often limited, as new measurements have only a local effect if no large loops are being closed, leaving spatially remote parts of the estimate unchanged. Consider the example of mapping a large building with many rooms: Measurements taken inside a room usually do not affect the estimates previously obtained for other rooms. Solving only for variables that actually change should therefore significantly reduce computational cost.

How do we update only variables that actually change, i.e. perform a partial state update? Full backsubstitution starts at the root and continues to all leaves, obtaining a delta vector Δ that is used to update the linearization point Θ. The partial state update starts by solving for all variables contained in the modified top of the tree. As shown in Alg. 7, we continue processing all sub-trees, stopping when a clique is encountered that does not refer to any variable for which Δ changed by more than a small threshold α. The running intersection property guarantees that none of the variables that changed significantly can occur in any sub-tree of that clique. Note that the threshold refers to a change in the delta vector Δ, not the absolute value of the recovered delta Δ itself. The absolute values of the entries in Δ can be quite large, because, as described above, the linearization point is only updated when a larger threshold β is reached. For simplicity we again use the same threshold for all variables, though that could be refined. For variables that are not reached by this process, the previous estimate Δ is kept. A nearly exact solution is obtained with significant savings in computation time, as can be seen from Fig. 10.

Alg. 7 Partial state update: Solving the Bayes tree in the nonlinear case returns an update Δ to the current linearization point Θ.

In: Bayes tree $\mathscr{T}$
Out: update Δ
Starting from the root clique $C_r = F_r$:

1. For current clique $C_k = F_k : S_k$
   compute update Δ_k of frontal variables F_k from the local conditional density $P(F_k|S_k)$.
2. For all variables Δ_{k_j} in Δ_k that change by more than threshold α:
   recursively process each descendant containing such a variable.

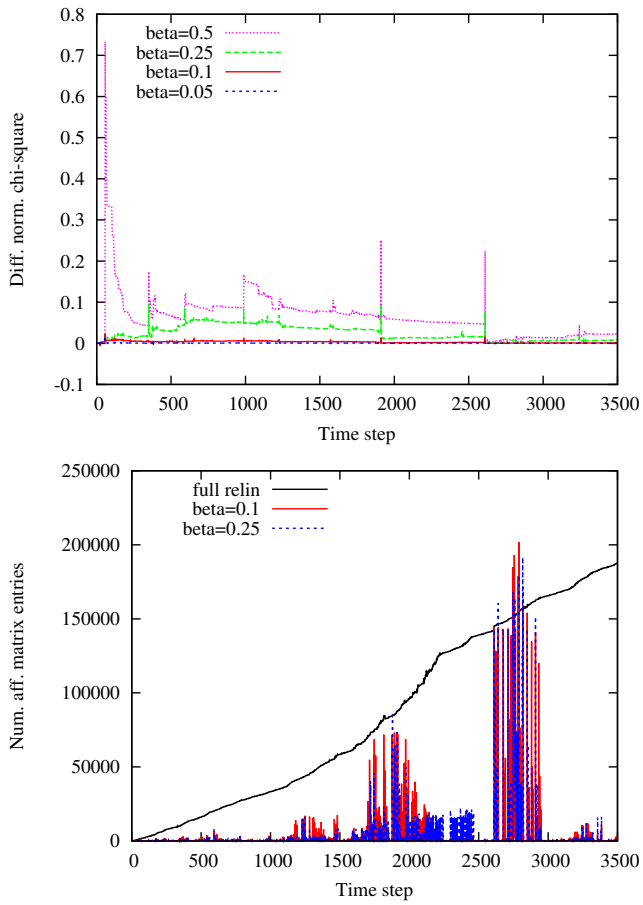


Fig. 9: How the relinearization threshold β affects accuracy (top) and computational cost (bottom) for the Manhattan dataset. For readability of the top figure, the normalized χ^2 value of the least-squares solution was subtracted. A threshold of 0.1 has no notable effect on the accuracy, while the cost savings are significant as can be seen in the number of affected nonzero matrix entries. Note that the spikes extend beyond the curve for full relinearization, because there is a small increase in fill-in over the batch variable ordering (compare with Fig. 8).

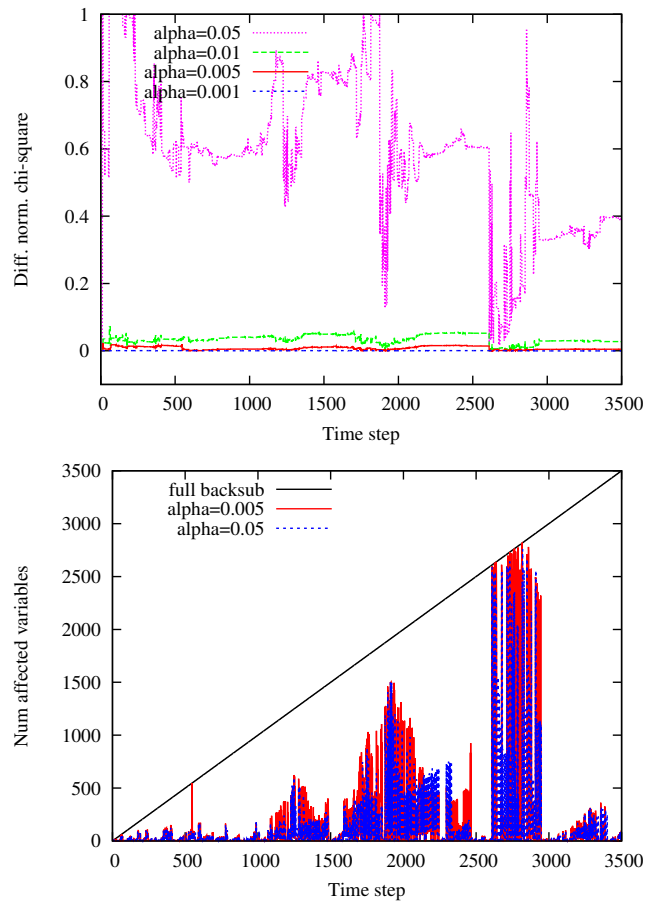


Fig. 10: How the backsubstitution threshold α affects accuracy (top) and computational cost (bottom) for the Manhattan dataset. For readability of the top figure, the normalized χ^2 value of the least-squares solution was subtracted. A small threshold such as 0.005 yields a significant increase in speed, while the accuracy is nearly unaffected.

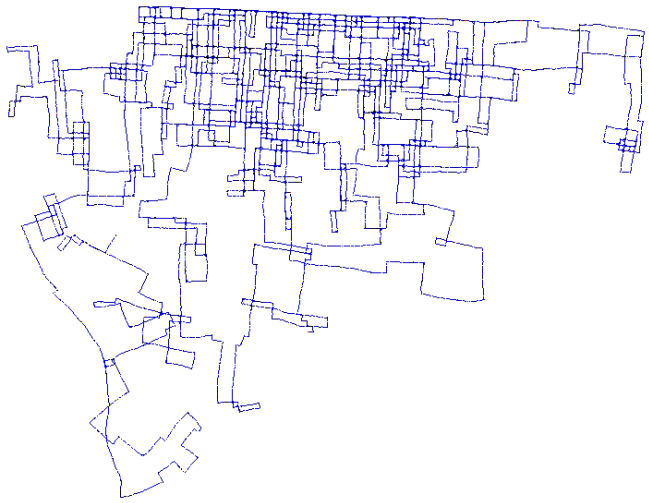
4.3 Algorithm and Complexity

The iSAM2 algorithm is summarized in Alg. 8. The goal of our algorithm is to obtain an estimate Θ for the variables (map and trajectory), given a set of nonlinear constraints that expands over time, represented by nonlinear factors $\mathscr{F}$. New factors $\mathscr{F}'$ can arrive at any time and may add new variables Θ' to the estimation problem. Based on the current linearization point Θ we solve a linearized system as a subroutine in an iterative nonlinear optimization scheme. The linearized system is represented by the Bayes tree $\mathscr{T}$.

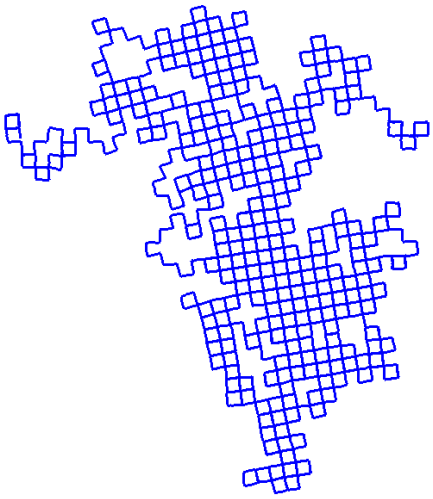
Here we provide some general complexity bounds for iSAM2. The number of iterations needed to converge is typically fairly small, in particular because of the quadratic convergence properties of Gauss-Newton iterations near the minimum. We assume here that the initialization of variables is close enough to the global minimum to allow convergence - that is a general requirement of any direct solver method. For exploration tasks with a constant number of constraints per pose, the complexity is $O(1)$ as only a constant number of variables at the top of the tree are affected and have to be re-eliminated, and only a constant number of variables are solved for. In the case of loop closures the situation becomes more difficult, and the most general bound is that for full factorization, $O(n^3)$, where n is the number of variables (poses and landmarks if present). Under certain assumptions that hold for many SLAM problems, batch matrix factorization and back-substitution can be performed in $O(n^{1.5})$ (Krauthausen et al., 2006). It is important to note that this bound does not depend on the number of loop closings. Empirically, complexity is usually much lower than these upper bounds because most of the time only a small portion of the matrix has to be refactorized in each step, as we show below.

5 Comparison to Other Methods

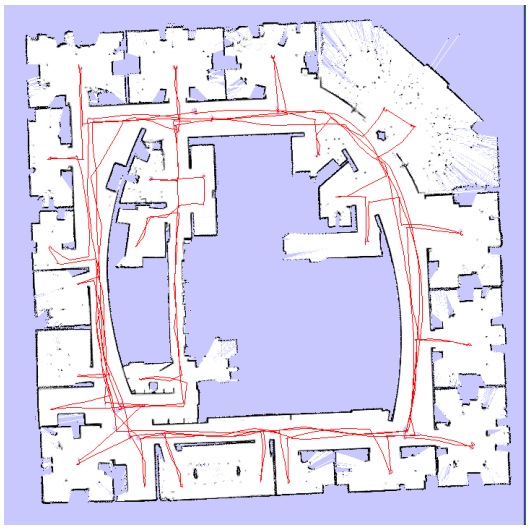
We compare iSAM2 to other state-of-the-art SLAM algorithms, in particular the iSAM1 algorithm (Kaess et al., 2008), HOG-Man (Grisetti et al., 2010) and SPA (Konolige et al.,



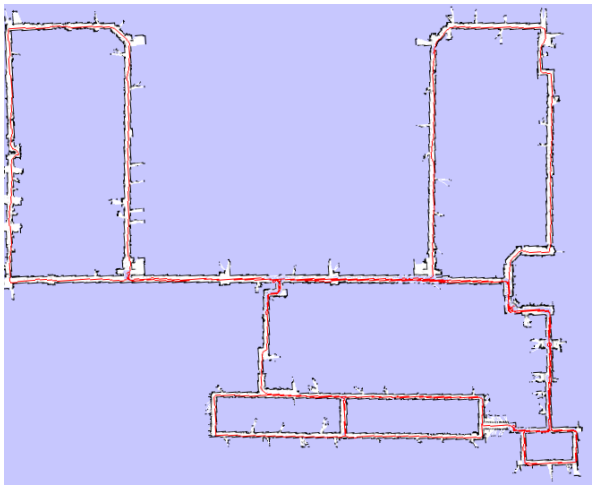
(a) City20000



(b) W10000



(c) Intel



(d) Killian Court

Fig. 11: 2D pose-graph datasets, including simulated data (City20000, W10000), and laser range data (Killian Court, Intel). See Fig. 8 for the Manhattan sequence.

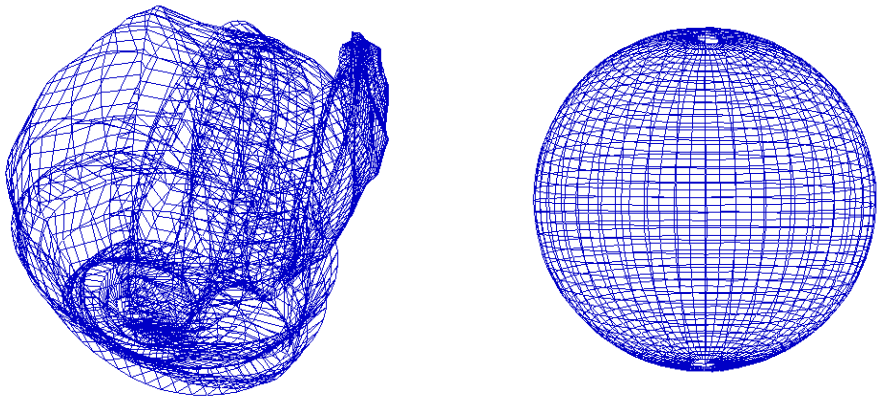
Alg. 8 One step of the iSAM2 algorithm, following the general structure of a smoothing solution given in Alg. 1.

In/out: Bayes tree $\mathscr{T}$, nonlinear factors $\mathscr{F}$, linearization point Θ, update Δ
In: new nonlinear factors $\mathscr{F}'$, new variables Θ'
Initialization: $\mathscr{T} = \emptyset$, $\Theta = \emptyset$, $\mathscr{F} = \emptyset$

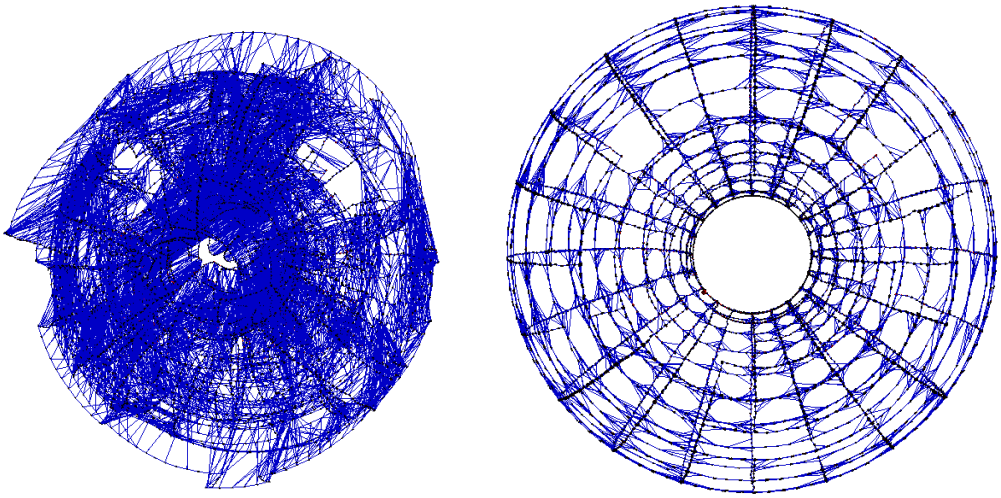
1. Add any new factors $\mathscr{F} := \mathscr{F} \cup \mathscr{F}'$.
2. Initialize any new variables Θ' and add $\Theta := \Theta \cup \Theta'$.
3. Fluid relinearization with Alg. 5 yields marked variables M, see Section 4.1.
4. Redo top of Bayes tree with Alg. 6 with $\mathscr{J}$ the union of M and all variables affected by new factors.
5. Solve for delta Δ with Alg. 7, see Section 4.2.
6. Current estimate given by $\Theta \oplus \Delta$.

2010). We use a wide variety of simulated and real-world datasets shown in Figs. 11, 12 and 13 that feature different sizes and constraint densities, both pose-only and with landmarks. All timing results are obtained on a laptop with Intel 1.6 GHz i7-720 processor. For iSAM1 we use version 1.6 of the open source implementation available at `http://people.csail.mit.edu/kaess/isam` with standard parameters, i.e. solving in every step. For HOG-Man, we use svn revision 14 available at `http://openslam.org/` with command line option "-update 1" to force solving in every step. For SPA, we use svn revision 36438 of ROS at `http://www.ros.org/` with standard parameters.

For iSAM2 we use a research C++ implementation running single-threaded, using the CCOLAMD algorithm by Davis et al. (2004), with parameters $\alpha = 0.001$ and $\beta = 0.1$. For improved efficiency, relinearization is performed every 10 steps. Source code for iSAM2 is available as part of the gtsam li-



(a) Sphere2500



(b) Torus10000

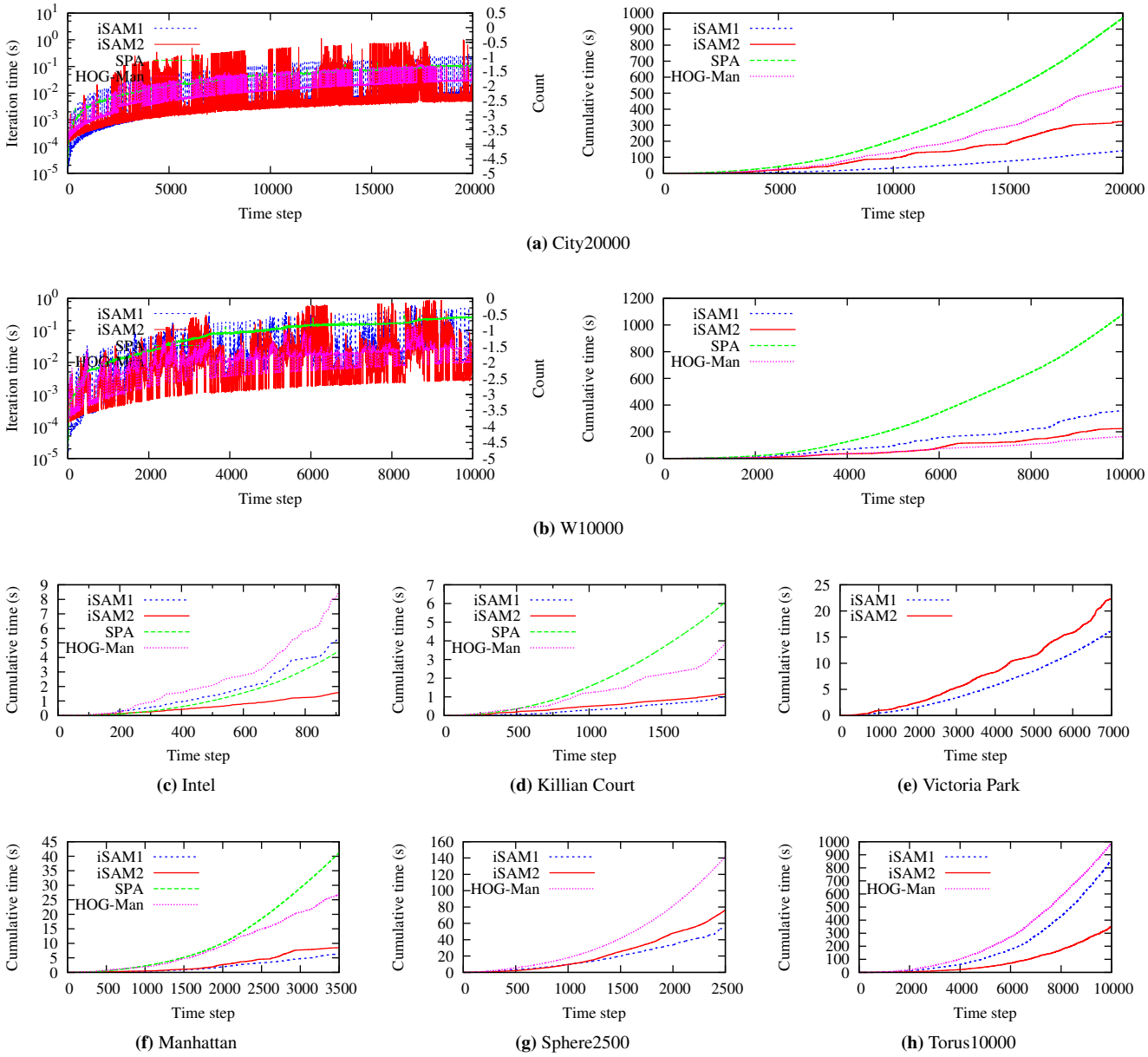
Fig. 13: Simulated 3D datasets (sphere2500 and torus10000, included in iSAM1 distribution). The left column shows the data based on noisy odometry, the right column the estimate obtained from iSAM2. Note that a large range of orientations is traversed, as the robot is simulated to drive along the surface of the sphere and torus, respectively.

brary at `https://collab.cc.gatech.edu/borg/gtsam/`. For efficiency we use incomplete Cholesky instead of QR factorization within each node of the tree. For optimization over 3D orientations, the $\oplus$ operator is implemented using exponential maps based on the theory of Lie groups (Hall, 2000). Our original SAM work (Dellaert and Kaess, 2006) used local updates of Euler angles for visual SLAM. Here, as representation, we use rotation matrices in iSAM2 and Quaternions in iSAM1 (Grassia, 1998). We have found that, depending on the application, each representation has its own advantages.

Comparing the computational cost of different algorithms is not a simple task. Tight complexity bounds for SLAM algorithms are often not available. Even if complexity bounds are available, they are not necessarily suitable for comparison because the involved constants can make a large difference in practical applications. On the other hand, speed comparison for the implementations of the algorithms depend on the implementation itself and any potential inefficiencies or wrong choice of data structures. We will therefore discuss not only the timing results obtained from the different implementations, but also compare some measure of the underlying cost, such as how many entries of the sparse matrix have to be recalculated. That again on its own is also not a perfect measure, as recalculating only parts of a matrix might occur some overhead that cannot be avoided.

5.1 Timing

We compare execution speed of implementations of the various algorithms on all datasets in Fig. 14, with detailed results in Table 1. The results show that a batch solution using sparse matrix factorization (SPA, SAM) quickly gets expensive, emphasizing the need for incremental solutions. iSAM1 performs very well on sparse datasets, such as Manhattan, Killian Court and City20000, while performance degrades on datasets with denser constraints (number of constraints at least 5 times the number of poses), such as W10000 and Intel, because of



(a) City20000

(b) W10000

(c) Intel

(d) Killian Court

(e) Victoria Park

(f) Manhattan

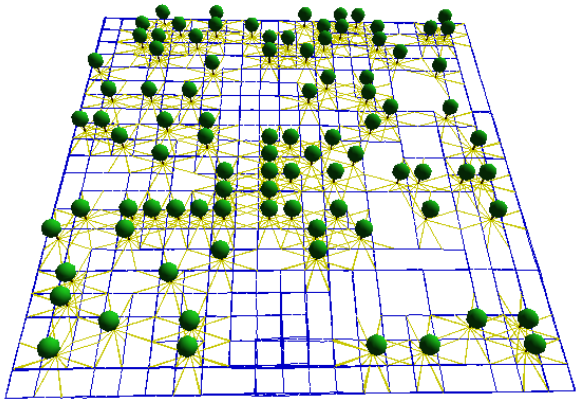
(g) Sphere2500

(h) Torus10000

Fig. 14: Timing comparison between the different algorithms for all datasets, see Fig. 11. The left column shows per iteration time and the right column cumulative time. The bottom row shows cumulative time for the remaining datasets.

Table 1: Runtime comparison for the different approaches (P: number of poses, M: number of measurements, L: number of landmarks). Listed are the average time per step together with standard deviation and maximum in milliseconds, as well as the overall time in seconds (fastest result shown in red).

			Algorithm	iSAM2		iSAM1		HOG-Man		SPA	
Dataset	P	M	L	avg/std/max [ms]	time [s]	avg/std/max [ms]	time [s]	avg/std/max [ms]	time [s]	avg/std/max [ms]	[s]
City20000	20000	26770	-	16.1 / 65.6 / 1125	323	7.05 / 14.5 / 308	141	27.4 / 27.8 / 146	548	48.7 / 32.6 / 140	977
W10000	10000	64311	-	22.4 / 64.6 / 901	224	35.7 / 58.8 / 683	357	16.4 / 14.9 / 147	164	108 / 75.6 / 287	1081
Manhattan	3500	5598	-	2.44 / 7.71 / 133	8.54	1.81 / 3.69 / 57.6	6.35	7.71 / 6.91 / 33.8	27.0	11.8 / 8.46 / 28.9	41.1
Intel	910	4453	-	1.74 / 1.76 / 9.13	1.59	5.80 / 8.03 / 48.4	5.28	9.40 / 12.5 / 79.3	8.55	4.89 / 3.77 / 14.9	4.44
Killian Court	1941	2190	-	0.59 / 0.80 / 12.5	1.15	0.51 / 1.13 / 16.6	0.99	2.00 / 2.41 / 11.8	3.88	3.13 / 1.89 / 7.98	6.07
Victoria Park	6969	10608	151	2.34 / 7.75 / 316	16.3	2.35 / 4.82 / 80.4	16.4	N/A	N/A	N/A	N/A
Trees10000	10000	14442	100	4.24 / 6.52 / 124	42.4	2.98 / 6.70 / 114	29.8	N/A	N/A	N/A	N/A
Sphere2500	2500	4950	-	30.4 / 25.5 / 158	76.0	21.7 / 31.3 / 679	54.3	56.7 / 40.8 / 159	142	N/A	N/A
Torus10000	10000	22281	-	35.2 / 45.7 / 487	352	86.4 / 119 / 1824	864	99.0 / 82.9 / 404	990	N/A	N/A



(a) Trees10000

(b) Victoria Park

Fig. 12: 2D datasets with landmarks, both simulated (Trees10000), and laser range data (Victoria Park).

local fill-in between the periodic batch reordering steps (see Fig. 8 center). Note that the spikes in the iteration time plots are caused by the periodic variable reordering every 100 steps, which is equivalent to a batch Cholesky factorization as performed in SPA, but with some overhead for the incremental data structures. The performance of HOG-Man is between SPA and iSAM1 and 2 for most of the datasets, but performs better on W10000 than any other algorithm. Performance is generally better on denser datasets, where the advantages of hierarchical operations dominate their overhead.

iSAM2 consistently performs better than SPA, and similar to iSAM1. While iSAM2 saves computation over iSAM1 by only performing partial backsubstitution, the fluid relinearization adds complexity. Relinearization typically affects many more variables than a linear update (compare Figs. 8 and 9), resulting in larger parts of the Bayes tree having to be

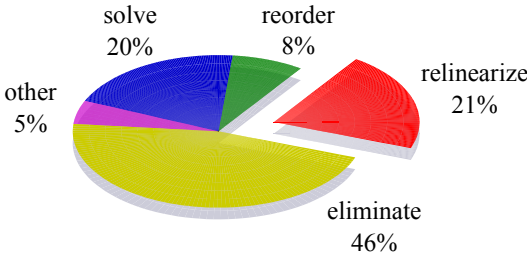


Fig. 15: How time is spent in iSAM2: Percentage of time spent in various components of the algorithm for the W10000 dataset.

recalculated. Interesting is the fact that the spikes in iSAM2 timing follow SPA, but are higher by almost an order of magnitude, which becomes evident in the per iteration time plots for City20000 and W10000 in Fig. 14ab. That difference can partially be explained by the fact that SPA uses the well optimized CHOLMOD library (Chen et al., 2008) for batch Cholesky factorization, while for the algorithms underlying iSAM2 no such library is available yet and we are using our own research implementation. Fig. 15 shows how time is spent in iSAM2, with elimination being the dominating part.

5.2 Number of Affected Entries

We also provide a computation cost measure that is more independent of specific implementations, based on the number of variables affected, and the number of entries of the sparse square root information matrix that are being recalculated in each step. The bottom plots in Figs. 10 and 9 show the number of affected variables in backsubstitution and the number of affected non-zero entries during matrix factorization. The red curve shows the cost of iSAM2 for thresholds that achieve an almost exact solution. When compared to the batch solution shown in black, the data clearly shows significant savings in computation of iSAM2 over Square Root SAM and SPA.

In iSAM2 the fill-in of the corresponding square root information factor remains close to that of the batch solution as shown in Fig. 8. The same figure also shows that for iSAM1 the fill-in increases significantly between the periodic batch steps, because variables are only reordered every 100 steps. This local fill-in explains the higher computational cost on datasets with denser constraints, such as W10000. iSAM2 shows no significant local variations of fill-in owing to the incremental variable ordering.

5.3 Accuracy

We now focus on the accuracy of the solution of each SLAM algorithm. There are a variety of different ways to evaluate accuracy. We choose the normalized χ^2 measure that quantifies the quality of a least-squares fit. Normalized χ^2 is defined as $\frac{1}{m-n}\sum_i \|h_i(\Theta_i) - z_i\|^2_{\Lambda_i}$, where the numerator is the weighted sum of squared errors of (4), m is the number of measurements and n the number of degrees of freedom. Normalized χ^2 measures how well the constraints are satisfied, approach-

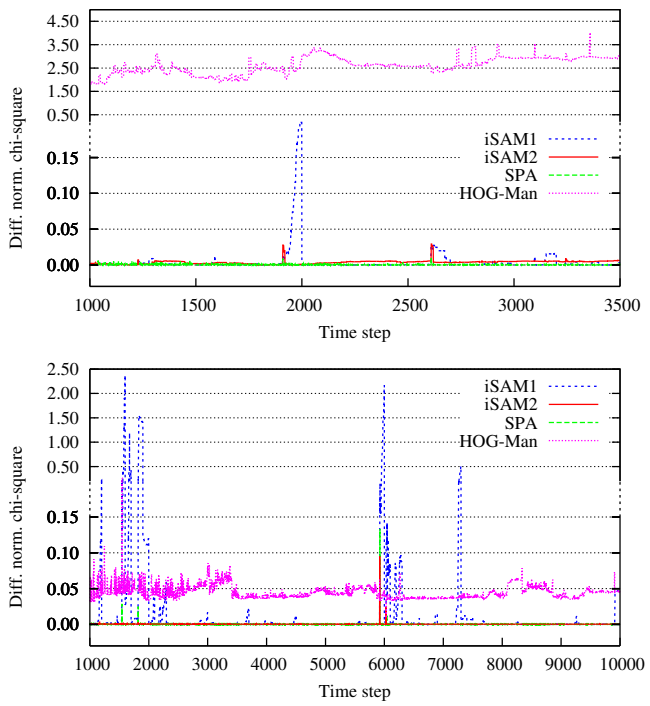


Fig. 16: Step-wise quality comparison of the different algorithms for the Manhattan world (top) and W10000 dataset (bottom). For improved readability, the difference in normalized χ^2 to the least squares solution is shown (i.e. ground truth given by $y = 0$).

ing 1 for a large number of measurements sampled from a normal distribution.

The results in Fig. 16 show that the iSAM2 solution is very close to the ground truth. The ground truth is the least-squares estimate obtained by iterating until convergence in each step. Small spikes are caused by relinearizing only every 10 steps, which is a trade-off with computational speed. iSAM1 shows larger spikes in error that are caused by relinearization only being done every 100 steps. HOG-Man is an approximate algorithm exhibiting consistently larger errors, even though visual inspection of the resulting map showed only minor distortions. Accuracy improves for more dense datasets, such as W10000, but is still not as good as iSAM2.

6 Related Work

The first smoothing approach to the SLAM problem was presented by Lu and Milios (1997), where the estimation problem is formulated as a network of constraints between robot poses. The first solution was implemented using matrix inversion (Gutmann and Nebel, 1997). A number of improved and numerically more stable algorithms have since been developed, based on well known *iterative* techniques such as relaxation (Duckett et al., 2002; Bosse et al., 2004; Thrun et al., 2005), gradient descent (Folkesson and Christensen, 2004, 2007), preconditioned conjugate gradient (Konolige, 2004; Dellaert et al., 2010), multi-level relaxation (Frese et al., 2005), and belief propagation (Ranganathan et al., 2007).

Direct methods, such as QR and Cholesky matrix factorization, provide the advantage of faster convergence, at least if a good initialization is available. They have initially been ignored, because a naive dense implementation is too expensive. An efficient sparse factorization for SLAM has first been proposed by Dellaert (2005), but is now widely used (Dellaert and Kaess, 2006; Frese, 2006; Folkesson et al., 2007; Kaess et al., 2008; Mahon et al., 2008; Grisetti et al., 2010; Konolige et al., 2010; Strasdat et al., 2010). Square Root SAM (Dellaert, 2005; Dellaert and Kaess, 2006) performs smoothing by Cholesky factorization of the complete, naturally sparse information matrix in every step using the Levenberg-Marquardt algorithm. Konolige et al. (2010) recently presented Sparse Pose Adjustment (SPA) using Cholesky factorization, that introduces a continuable Levenberg-Marquardt algorithm and focuses on a fast setup of the information matrix, often the most costly part in batch factorization.

Smoothing is closely related to structure from motion (Hartley and Zisserman, 2000) and bundle adjustment (Triggs et al., 1999) in computer vision. Both bundle adjustment and the smoothing formulation of SLAM keep all poses and landmarks in the estimation problem (pose-graph SLAM is a special case that omits landmarks). The key difference between bundle adjustment and SLAM is that bundle adjustment is typically solving the batch problem, while for robotics applications online solutions are required because data is continuously collected. Our iSAM2 algorithm achieves online bundle adjustment, at least up to some reasonable size of datasets. The question of creating a "perpetual SLAM engine" to run indefinitely remains open. Note that the number of landmarks per frame is typically much lower for laser range-based applications than for visual SLAM applications (Eade and Drummond, 2007; Konolige and Agrawal, 2008; Paz et al., 2008; Strasdat et al., 2010). However, smoothing has recently also been shown to be the method of choice for visual SLAM in many situations (Strasdat et al., 2010).

In an incremental setting, the cost of batch optimization can be sidestepped by applying matrix factorization updates, with the first SLAM applications in (Kaess et al., 2007; Folkesson et al., 2007; Wang, 2007; Mahon et al., 2008). The iSAM algorithm (Kaess et al., 2007) performs incremental updates using Givens rotations, with periodic batch factorization and relinearization steps. (Folkesson et al., 2007) only keeps a short history of robot poses, avoiding the reordering problem. Wang (2007) mentions Cholesky updates as an option for the D-SLAM information matrix that only contains landmarks. iSAM2 is similar to these methods, but has the advantage that both reordering and relinearization can be performed incrementally in every step. Note that iSAM2 does not simply apply existing methods such as matrix factorization updates, but introduces a completely novel algorithm for solving sparse nonlinear least-squares problems that grow over time.

The relative formulation in Olson et al. (2006) expresses poses relative to previous ones, replacing the traditional

global formulation. The resulting Jacobian has significantly more entries, but is solved efficiently by stochastic gradient descent. The relative formulation avoids local minima in poorly initialized problems. A hierarchical extension by Grisetti et al. (2007) called TORO provides faster convergence by significantly reducing the maximum path length between two arbitrary nodes. However, the separation of translation and rotation leads to inaccurate solutions (Grisetti et al., 2010) that are particularly problematic for 3D applications.

Relative formulations have also been used on a different level, to split the problem into submaps, for example Atlas (Bosse et al., 2004) or Tectonic SAM (Ni et al., 2007; Ni and Dellaert, 2010). In some way, iSAM2 provides a separation into submaps represented by different sub-trees, even though they are not completely separated in a way, as new measurements can change that topology at any time, so that the complexity is not explicitly bounded.

Sibley et al. (2009) couples the relative formulation with locally restricted optimization, operating on a manifold similar to Howard et al. (2006). Restricting optimization to a local region allows updates in constant time. The produced maps are locally accurate, but a globally metric map can only be obtained offline. While such maps are sufficient for some applications, we argue that tasks such as planning require an accurate globally metric map to be available at every step: For example, to detect/decide if an unexplored direct path (such as a door) might exist between places A and B requires globally metric information. In iSAM2 we therefore focus on making online recovery of globally metric maps more efficient - however, our update steps are not constant time, and for large loop closings can become as expensive as a batch solution.

Grisetti et al. (2010) recently presented HOG-Man, a hierarchical pose graph formulation using Cholesky factorization that represents the estimation problem at different levels of detail. Computational effort is focused on affected areas at the most detailed level, while any global effects are propagated to the coarser levels. In particular for dense sequences, HOG-Man fares well when compared to iSAM2, but only provides an approximate solution.

The thin junction tree filter (TJTF) by Paskin (2003) provides an incremental solution directly based on graphical models. A junction tree is maintained incrementally for the filtering version of the SLAM problem, and data is selectively omitted in order to keep the data structure sparse (filtering leads to fill-in) and the complexity of solving manageable. iSAM2 in contrast solves the full SLAM problem, and does not omit any information. Furthermore, construction of the Bayes tree differs from the junction tree, which first forms a clique graph and then finds a spanning tree. The Bayes tree, in contrast, is based on a given variable ordering, similar to the matrix factorization. Though it gains some flexibility because the order of the sub-trees of a clique can be changed comparing to the fixed variable ordering of the square root information matrix.

Treemap by Frese (2006) performs QR factorization within nodes of a tree, which is balanced over time. Sparsification prevents the nodes from becoming too large, which introduces approximations by duplication of variables. Treemap is also closely related to the junction tree, though the author approached the subject "from a hierarchy-of-regions and linear-equation-solving perspective" Frese (2006). Our work formalizes this connection in a more comprehensive way through the Bayes tree data structure.

7 Conclusion

We have presented a novel data structure, the Bayes tree, which provides an algorithmic foundation that enables new insights into existing graphical model inference algorithms and sparse matrix factorization methods. These insights have led us to iSAM2, a fully incremental algorithm for nonlinear least-squares problems as they occur in mobile robotics. Our new algorithm is completely different from the original iSAM algorithm as both variable reordering and relinearization are now done incrementally. In contrast, iSAM can only update linear systems incrementally, requiring periodic batch steps for reordering and relinearization. We have used SLAM as an example application, even though the algorithm is also suitable for other incremental inference problems, such as object tracking and sensor fusion. We performed a systematic evaluation of iSAM2 and a comparison with three other state-of-the-art SLAM algorithms. We expect our novel graph-based algorithm to also allow for better insights into the recovery of marginal covariances, as we believe that simple recursive algorithms in terms of the Bayes tree are formally equivalent to the dynamic programming methods described in (Kaess and Dellaert, 2009). The graph based structure is also suitable for exploiting parallelization that is becoming available in newer processors.

Acknowledgments

M. Kaess, H. Johannsson and J. Leonard were partially supported by ONR grants N00014-06-1-0043 and N00014-10-1-0936. F. Dellaert and R. Roberts were partially supported by NSF, award number 0713162, "RI: Inference in Large-Scale Graphical Models". V. Ila has been partially supported by the Spanish MICINN under the Programa Nacional de Movilidad de Recursos Humanos de Investigación.

We thank E. Olson for the Manhattan dataset, E. Nebot and H. Durrant-Whyte for the Victoria Park dataset, D. Haehnel for the Intel dataset and G. Grisetti for the W10000 dataset.

References

Arnborg, S., Corneil, D., and Proskurowski, A. (1987). Complexity of finding embeddings in a *k*-tree. *SIAM J. on Algebraic and Discrete Methods*, 8(2):277–284.

Blair, J. and Peyton, B. (1993). An introduction to chordal graphs and clique trees. In George, J., Gilbert, J., and Liu, J.-H., editors,

Graph Theory and Sparse Matrix Computations, volume 56 of *IMA Volumes in Mathematics and its Applications*, pages 1–27. Springer-Verlag, New York.

Bosse, M., Newman, P., Leonard, J., and Teller, S. (2004). Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework. *Intl. J. of Robotics Research*, 23(12):1113–1139.

Brooks, R. (1985). Visual map making for a mobile robot. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 2, pages 824 – 829.

Castellanos, J., Montiel, J., Neira, J., and Tardós, J. (1999). The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Trans. Robot. Automat.*, 15(5):948–953.

Chen, Y., Davis, T., Hager, W., and Rajamanickam, S. (2008). Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):22:1–22:14.

Cowell, R., Dawid, A., Lauritzen, S., and Spiegelhalter, D. (1999). *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer-Verlag.

Davis, T., Gilbert, J., Larimore, S., and Ng, E. (2004). A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):353–376.

Dellaert, F. (2005). Square Root SAM: Simultaneous location and mapping via square root information smoothing. In *Robotics: Science and Systems (RSS)*.

Dellaert, F., Carlson, J., Ila, V., Ni, K., and Thorpe, C. (2010). Subgraph-preconditioned conjugate gradient for large scale slam. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

Dellaert, F. and Kaess, M. (2006). Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *Intl. J. of Robotics Research*, 25(12):1181–1203.

Dellaert, F., Kipp, A., and Krauthausen, P. (2005). A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping. In *Proc. 22nd AAAI National Conference on AI*, Pittsburgh, PA.

Dissanayake, M., Newman, P., Durrant-Whyte, H., Clark, S., and Csorba, M. (2001). A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Trans. Robot. Automat.*, 17(3):229–241.

Duckett, T., Marsland, S., and Shapiro, J. (2002). Fast, on-line learning of globally consistent maps. *Autonomous Robots*, 12(3):287–300.

Eade, E. and Drummond, T. (2007). Monocular SLAM as a graph of coalesced observations.

Folkesson, J. and Christensen, H. (2004). Graphical SLAM - a self-correcting map. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 1, pages 383–390.

Folkesson, J. and Christensen, H. (2007). Closing the loop with Graphical SLAM. *IEEE Trans. Robotics*, 23(4):731–741.

Folkesson, J., Leonard, J., Leederkerken, J., and Williams, R. (2007). Feature tracking for underwater navigation using sonar. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3678–3684.

Frese, U. (2006). Treemap: An $O(\log n)$ algorithm for indoor simultaneous localization and mapping. *Autonomous Robots*, 21(2):103–122.

Frese, U., Larsson, P., and Duckett, T. (2005). A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Trans. Robotics*, 21(2):196–207.

Gauss, C. (1809). *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Mabientium [Theory of the Motion of the Heavenly Bodies Moving about the Sun in Conic Sections]*. Perthes and Besser, Hamburg, Germany. English translation available at http://name.umdl.umich.edu/AGG8895.0001.001.

Gentleman, W. (1973). Least squares computations by Givens transformations without square roots. *IMA J. of Appl. Math.*, 12:329–336.

Gill, P., Golub, G., Murray, W., and Saunders, M. (1974). Methods for modifying matrix factorizations. *Mathematics and Computation*, 28(126):505–535.

Golub, G. and Loan, C. V. (1996). *Matrix Computations*. Johns Hopkins University Press, Baltimore, third edition.

Grassia, F. S. (1998). Practical parameterization of rotations using the exponential map. *J. Graph. Tools*, 3:29–48.

Grisetti, G., Kuemmerle, R., Stachniss, C., Frese, U., and Hertzberg, C. (2010). Hierarchical optimization on manifolds for online 2D and 3D mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Anchorage, Alaska.

Grisetti, G., Stachniss, C., Grzonka, S., and Burgard, W. (2007). A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Robotics: Science and Systems (RSS)*.

Gutmann, J.-S. and Nebel, B. (1997). Navigation mobiler Roboter mit Laserscans. In *Autonome Mobile Systeme*, Berlin. Springer Verlag.

Hall, B. (2000). *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Springer.

Hartley, R. and Zisserman, A. (2000). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

Heggernes, P. and Matstoms, P. (1996). Finding good column orderings for sparse QR factorization. In *Second SIAM Conference on Sparse Matrices*.

Howard, A., Sukhatme, G., and Matarić, M. (2006). Multi-robot mapping using manifold representations. *Proceedings of the IEEE - Special Issue on Multi-robot Systems*, 94(9):1360 – 1369.

Julier, S. and Uhlmann, J. (2001). A counter example to the theory of simultaneous localization and map building. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 4, pages 4238–4243.

Kaess, M. and Dellaert, F. (2009). Covariance recovery from a square root information matrix for data association. *Journal of Robotics and Autonomous Systems*, 57:1198–1210.

Kaess, M., Ila, V., Roberts, R., and Dellaert, F. (2010). The Bayes tree: An algorithmic foundation for probabilistic robot mapping. In *Intl. Workshop on the Algorithmic Foundations of Robotics*, pages 157–173, Singapore.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F. (2011). iSAM2: Incremental smoothing and mapping

with fluid relinearization and incremental variable reordering. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, China. To appear.

Kaess, M., Ranganathan, A., and Dellaert, F. (2007). Fast incremental square root information smoothing. In *Intl. Joint Conf. on AI (IJCAI)*, pages 2129–2134, Hyderabad, India.

Kaess, M., Ranganathan, A., and Dellaert, F. (2008). iSAM: Incremental smoothing and mapping. *IEEE Trans. Robotics*, 24(6):1365–1378.

Kim, B., Kaess, M., Fletcher, L., Leonard, J., Bachrach, A., Roy, N., and Teller, S. (2010). Multiple relative pose graphs for robust cooperative mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3185–3192, Anchorage, Alaska.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. The MIT press, Cambridge, MA.

Konolige, K. (2004). Large-scale map-making. In *Proc.* 21th *AAAI National Conference on AI*, San Jose, CA.

Konolige, K. and Agrawal, M. (2008). FrameSLAM: from bundle adjustment to realtime visual mapping. *IEEE Trans. Robotics*, 24(5):1066–1077.

Konolige, K., Grisetti, G., Kuemmerle, R., Burgard, W., Benson, L., and Vincent, R. (2010). Sparse pose adjustment for 2D mapping. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

Krauthausen, P., Dellaert, F., and Kipp, A. (2006). Exploiting locality by nested dissection for square root smoothing and mapping. In *Robotics: Science and Systems (RSS)*.

Kschischang, F., Frey, B., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory*, 47(2).

Lipton, R. and Tarjan, R. (1979). Generalized nested dissection. *SIAM Journal on Applied Mathematics*, 16(2):346–358.

Lu, F. and Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, pages 333–349.

Mahon, I., Williams, S., Pizarro, O., and Johnson-Roberson, M. (2008). Efficient view-based SLAM using visual loop closures. *IEEE Trans. Robotics*, 24(5):1002–1014.

Ni, K. and Dellaert, F. (2010). Multi-level submap based SLAM using nested dissection. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

Ni, K., Steedly, D., and Dellaert, F. (2007). Tectonic SAM: Exact; out-of-core; submap-based SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Rome; Italy.

Olson, E., Leonard, J., and Teller, S. (2006). Fast iterative alignment of pose graphs with poor initial estimates. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

Paskin, M. (2003). Thin junction tree filters for simultaneous localization and mapping. In *Intl. Joint Conf. on AI (IJCAI)*.

Paz, L., Pinies, P., Tardós, J., and Neira, J. (2008). Large scale 6DOF SLAM with stereo-in-hand. *IEEE Transactions on Robotics*, 24(5):946–957.

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

Pothen, A. and Sun, C. (1992). Distributed multifrontal factorization using clique trees. In *Proc. of the Fifth SIAM Conf. on Parallel Processing for Scientific Computing*, pages 34–40. Society for Industrial and Applied Mathematics.

Ranganathan, A., Kaess, M., and Dellaert, F. (2007). Loopy SAM. In *Intl. Joint Conf. on AI (IJCAI)*, pages 2191–2196, Hyderabad, India.

Sibley, G., Mei, C., Reid, I., and Newman, P. (2009). Adaptive relative bundle adjustment. In *Robotics: Science and Systems (RSS)*.

Smith, R., Self, M., and Cheeseman, P. (1987). A stochastic map for uncertain spatial relationships. In *Int. Symp on Robotics Research*.

Strasdat, H., Montiel, J., and Davison, A. (2010). Real-time monocular SLAM: Why filter? In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

Tarjan, R. and Yannakakis, M. (1984). Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT press, Cambridge, MA.

Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (1999). Bundle adjustment – a modern synthesis. In Triggs, W., Zisserman, A., and Szeliski, R., editors, *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag.

Wang, Z. (2007). *Exactly Sparse Information Filters for Simultaneous Localization and Mapping*. PhD thesis, The University of Technology, Sydney.

A Index to Multimedia Extensions

The multimedia extensions to this article are at: http://www.ijrr.org.

Extension	Type	Description
1	Video	The Bayes tree data structure and the map as they evolve over time for the Manhattan sequence.