The Bayes Tree: An Algorithmic Foundation for Probabilistic Robot Mapping

Michael Kaess, Viorela Ila, Richard Roberts 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 batch 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, that combines incremental updates with fluid relinearization of a reduced set of variables for efficiency, combined with fast convergence to the exact solution. We also present a novel strategy for incremental variable reordering to retain sparsity. We evaluate our algorithm on standard datasets in both landmark and pose SLAM settings.

Key words: graphical models, clique tree, probabilistic inference, sparse linear algebra, nonlinear optimization, smoothing and mapping, SLAM, iSAM

Acknowledgements This work was partially funded under NSF grant 0713162 (RI: Inference in Large-Scale Graphical Models) and under ONR grant N00014-06-1-0043. V. Ila has been partially supported by the Spanish MICINN under the *Programa Nacional de Movilidad de Recursos Humanos de Investigación*. M. Kaess would like to thank J. Leonard for his support of this work.

Michael Kaess
CSAIL, MIT, Cambridge, Massachusetts, e-mail: kaess@mit.edu

Viorela Ila, Richard Roberts and Frank Dellaert
School of Interactive Computing, Georgia Institute of Technology, Atlanta, Georgia, e-mail: {vila,richard,frank}@cc.gatech.edu

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 in mobile robotics, as it provides the necessary data for many other important tasks such as planning and manipulation, in addition to direct applications such as 3D modeling, exploration, and reconnaissance. The uncertainty inherent in sensor measurements makes probabilistic inference algorithms the favorite choice for SLAM. And because online operation is essential for most real applications, efficient incremental online algorithms are important and are at the focus of this paper.

Taking a graphical model perspective to probabilistic inference in SLAM has a rich history [2] and has especially led to several novel and exciting developments in the last years [27, 10, 13, 12, 11, 31]. Paskin proposed the thin junction tree filter (TJTF) [27], 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 [20], i.e., the average taken over a large number of experiments diverges from the true solution. In contrast, *full SLAM* [34] retains all robot poses and can provide an exact solution, which does not suffer from inconsistency. Folkesson and Christensen presented Graphical SLAM [10], a graph-based full SLAM solution that includes mechanisms for reducing the complexity by locally reducing the number of variables. More closely related, Frese's Treemap [12] performs QR factorization within nodes of a tree that is balanced over time. Sparsification is applied to prevent nodes from becoming too large, introducing approximations by duplication of variables.

The sparse linear algebra perspective has been explored by Dellaert et al. [6, 7, 23] in Smoothing and Mapping (SAM), an approach that exploits the sparsity of the smoothing information matrix. 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 [24]. Kaess et al. [22, 23] 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 [16, 15, 17], 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 relinearization, 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 perspective, **we propose a novel data structure, the Bayes tree.** 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. The most well-known such data structure is the *clique tree* [30, 1], also known as the *junction tree* in the AI literature [4], which has already been exploited for distributed inference in SLAM [8, 27]. 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 **a novel incremental exact inference method that allows for incremental reordering and just-in-time relinearization**. 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. We compare our new method to iSAM using multiple publicly available datasets in both landmark and pose SLAM settings.

2 Problem Statement

We use a *factor graph* [25] to represent the SLAM 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. An example of a SLAM factor graph is shown in Fig. 1(top).

When assuming Gaussian measurement models

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

as is standard in the SLAM literature [32, 3, 9], the factored objective function we want to maximize (1) corresponds to the nonlinear least-squares criterion

$$\arg\min_{\Theta}\left(-\log f(\Theta)\right) = \arg\min_{\Theta}\frac{1}{2}\sum_i \left\|h_i(\Theta_i) - z_i\right\|^2_{\Sigma_i} \tag{3}$$

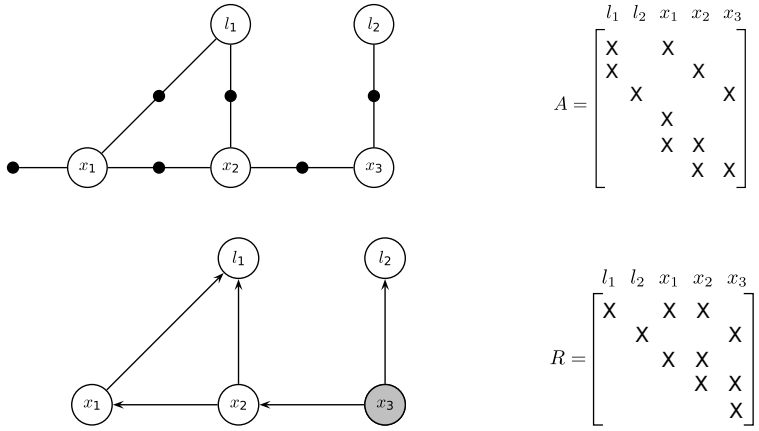

Fig. 1 (top) The factor graph and the associated Jacobian matrix A for a small SLAM example, where a robot located at successive poses x_1, x_2, and x_3 makes observations on landmarks l_1 and l_2. In addition there is an absolute measurement on the pose x_1. (bottom) The chordal Bayes net and the associated 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. Note that the root, the last variable to be eliminated, is shaded darker.

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

A crucial insight is that **inference can be understood as converting the factor graph to a Bayes net using the elimination algorithm.** Variable elimination [1, 4] originated in order to solve systems of linear equations, and was first applied in modern times by Gauss in the early 1800s [14].

Algorithm 1 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.

In factor graphs, elimination is done via a *bipartite elimination game*, as described by Heggernes and Matstoms [19]. This can be understood as taking apart the factor graph and transforming it into a *Bayes net* [29]. 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 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 Algorithm 1. After eliminating all variables, the Bayes net density is defined by the product of the conditionals produced at each step:

$$P(\Theta) = \prod_j P(\theta_j | S_j) \tag{4}$$

The result of this process for the example in Fig. 1(top) is shown in Fig. 1(bottom).

3 The Bayes Tree

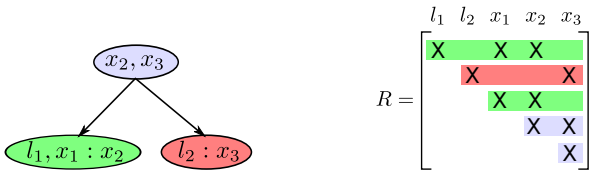

Fig. 2 The Bayes tree and the associated square root information matrix R describing the clique structure in the Bayes net from Fig. 1. A Bayes tree is similar to a clique tree, but is better at capturing the formal equivalence between sparse linear algebra and inference in graphical models. The association of cliques with rows in the R factor is indicated by color.

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. In this paper we introduce a new data structure, *the Bayes tree*, to better capture the equivalence with linear algebra and enable new algorithms in recursive estimation. A Bayes tree is a directed tree where the nodes represent *cliques* C_k of the underlying chordal Bayes net. In this respect Bayes trees are similar to clique trees, 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,

$$P(\Theta) = \prod_k P(F_k | S_k) \tag{5}$$

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 [33], 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 Algorithm 2 and the corresponding Bayes tree for the small SLAM example in Fig. 1 is shown in Fig. 2.

Algorithm 2 Creating a Bayes tree from the chordal Bayes net resulting from elimination (Algorithm 1).

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

Gaussian Case In practice one always considers a linearized version of problem (3). If the measurement models h_i in equation (2) are nonlinear and a good linearization point is not available, nonlinear optimization methods such as Gauss-Newton iterations or the Levenberg-Marquardt algorithm solve a succession of linear approximations to (3) 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 $\mathbf{x}$ with the objective function

$$-\log f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2 \tag{6}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ is the measurement Jacobian consisting of m measurement rows and $\mathbf{x}$ is an n-dimensional tangent vector of a minimal representation [18]. Note that the covariances Σ_i have been absorbed into the corresponding block rows of $\mathbf{A}$, making use of $\|\mathbf{x}\|_\Sigma^2 = \mathbf{x}^T\Sigma^{-1}\mathbf{x} = \mathbf{x}^T\Sigma^{-\frac{T}{2}}\Sigma^{-\frac{1}{2}}\mathbf{x} = \left\|\Sigma^{-\frac{1}{2}}\mathbf{x}\right\|^2$. The matrix $\mathbf{A}$ above is a sparse block-matrix, and its graphical model counterpart is a *Gaussian* factor graph with exactly the same structure as the nonlinear factor graph, see Fig. 1. The probability density on $\mathbf{x}$ defined by this factor graph is the normal distribution

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

In Gaussian factor graphs, elimination is equivalent to sparse QR factorization of the measurement Jacobian. In Gaussian factor graphs, the chain rule $f_{joint}(\theta_j, S_j) = P(\theta_j|S_j)f_{new}(S_j)$ in step 3 of Algorithm 1 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 $\mathbf{A}$. To see this, note that, for $x_j \in \mathbb{R}$ and $\mathbf{s}_j \in \mathbb{R}^l$ (the set of variables S_j combined in a vector of length l), the factor $f_{joint}(x_j, \mathbf{s}_j)$ defines a Gaussian density

$$f_{joint}(x_j, \mathbf{s}_j) \propto \exp\left\{-\frac{1}{2}\left\|\mathbf{a}x_j + \mathbf{A}_S\mathbf{s}_j - \mathbf{b}\right\|^2\right\} \tag{8}$$

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 x_j. Note that $\mathbf{a} \in \mathbb{R}^k$, $\mathbf{A}_S \in \mathbb{R}^{k\times l}$ and $\mathbf{b} \in \mathbb{R}^k$, with k the number of measurement rows of all factors connected to x_j. The desired conditional $P(x_j|\mathbf{s}_j)$ is obtained by evaluating the joint (8) for a given value of $\mathbf{s}_j$, yielding

$$P(x_j|\mathbf{s}_j) \propto \exp\left\{-\frac{1}{2}\left(x_j + \mathbf{r}\mathbf{s}_j - d\right)^2\right\} \tag{9}$$

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 $x_j = d - \mathbf{r}\mathbf{s}_j$ back into (8):

$$f_{new}(\mathbf{s}_j) = \exp\left\{-\frac{1}{2}\left\|\mathbf{A}'\mathbf{s}_j - \mathbf{b}'\right\|^2\right\} \tag{10}$$

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 as indicated in Fig. 2.

Solving The optimal assignment $\mathbf{x}^*$ of the linear least-squares solution is the one that maximizes the joint density $P(\mathbf{x})$ from (7). The optimal assignment $\mathbf{x}^*$ can be computed in dynamic programming style 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 $\mathbf{x}$. 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 (9) by solving

$$x_j = d - \mathbf{r}\mathbf{s}_j \tag{11}$$

for every variable x_j, which is equivalent to backsubstitution in sparse linear algebra.

4 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 $\mathscr{T}$. Incremental factorization/inference is performed by reinterpreting the top part of the Bayes tree again as a factor graph, adding to this the new factors, creating with a new elimination order a new Bayes tree from this "top", then reattaching to it the unaffected subtrees. 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)

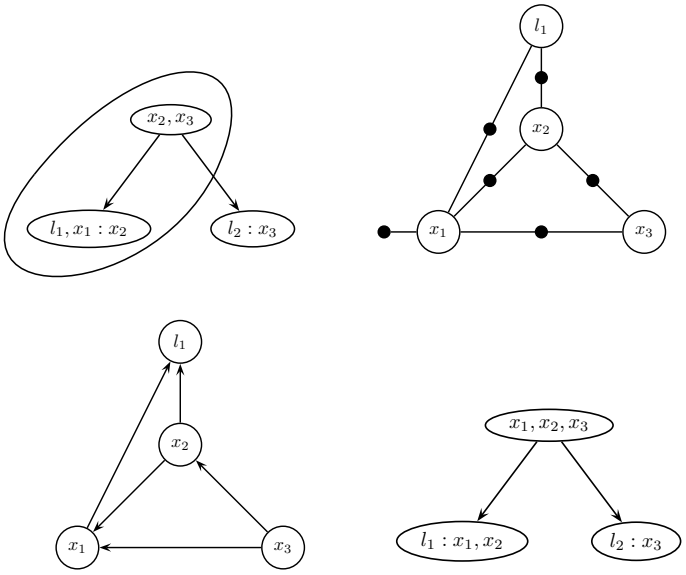

Fig. 3 Updating a Bayes tree with a new factor, based on the example in Fig. 2. (top left) The affected part of the Bayes tree is highlighted for the case of adding a new factor between x_1 and x_3. Note that the right branch is not affected by the change. (top right) The factor graph generated from the affected part of the Bayes tree. (bottom left) The chordal Bayes net resulting from eliminating the factor graph. (bottom right) The Bayes tree created from the chordal Bayes net, with the unmodified right "orphan" subtree from the original Bayes tree added back in.

and the root are affected. The sub-trees below these cliques are unaffected, as are any other sub-trees not containing x_j or $x_{j'}$. Fig. 3 shows how these steps are applied to our small SLAM example (originally in Fig. 2). The upper-left shows that adding the new factor between x_1 and x_3 only affects the left branch of the tree. The entire process of updating the Bayes tree with a new factor is described in Algorithm 3.

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 it encoding information flow during elimination. First, during elimination, 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.

Algorithm 3 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 affected variable, 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 and eliminate the factor graph into a Bayes net (Algorithm 1), and re-assemble into a new Bayes tree (Algorithm 2).
4. Insert the orphans $\mathscr{T}_{orph}$ back into the new Bayes tree.

5 Incremental Reordering

Choosing the right variable ordering is essential for the efficiency of a sparse matrix solution, and this also holds for the Bayes tree approach. An optimal ordering 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. Finding the variable ordering that leads to the minimal fill-in is NP-hard. One typically uses heuristics such as the column approximate minimum degree (COLAMD) algorithm by Davis et al. [5], which provide close to optimal orderings for many problems.

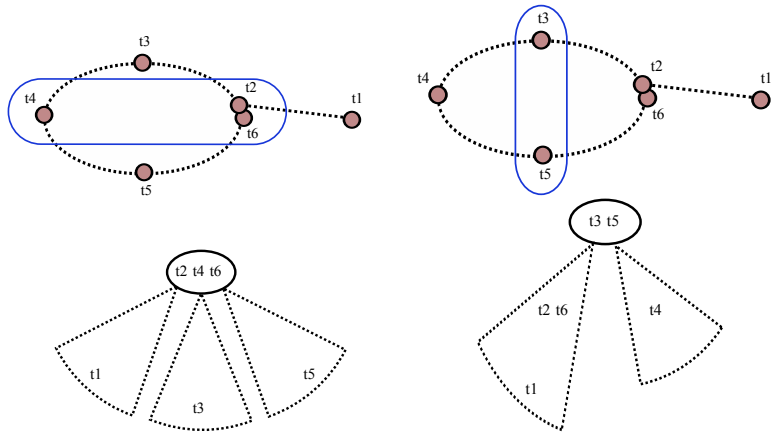

Fig. 4 For a trajectory with loop closing, two different optimal variable orderings based on nested dissection are shown in the top row, with the corresponding Bayes tree structure in the bottom row. For the incremental setting the left choice is preferable, as the most recent variables end up in the root, minimizing work in future updates.

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 [23], because this is only obvious within the graphical model framework, but not for matrices. Reordering is only performed for the variables affected by the new factors. 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.

One particularly good ordering forces the affected variables to be eliminated last. This strategy provides a good ordering because new measurements almost always connect to recently observed variables. In particular, odometry measurements always connect to the previous pose. In the exploration mode it is clear that if the most recent variables end up in the root, only a small part of the tree (optimally only the root) has to be reorganized in the next step. The more difficult case of a loop closure is visualized in Fig. 4. In the case of a simple loop, nested dissection provides the optimal ordering. The first cut can either (a) include the root, or (b) not include the root, and both solutions are equivalent in terms of fill-in. However, there is a significant difference in the incremental case: For the horizontal cut that 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 vertical cut, on the other hand, includes the last variable in the first cut, pushing it into the root, and therefore leading to smaller, more efficient changes in the next step. In order to deal with more general topologies than this simple example, we use a constrained version of the COLAMD algorithm, that allows keeping the last variables in the root while still obtaining a good overall ordering.

6 Exact Incremental Inference with Fluid Relinearization

In this section we use the Bayes tree in a novel algorithm for optimizing a set of nonlinear factors that grows over time, which is directly applicable to online mapping. We have already shown how the Bayes tree is updated with new linear factors. We now discuss how to perform relinearization where needed, a process that we call *fluid relinearization*. Then we present a combined algorithm for adding nonlinear factors over time, while keeping the Bayes tree and the estimate up-to-date.

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 $\mathcal{F}$. New factors $\mathcal{F}'$ can arrive at any time and may add new variables Θ' to the estimation problem. We take the most recent estimate Θ as linearization point to solve a linearized system as a subroutine in an iterative nonlinear optimization scheme. The linearized system is represented by the Bayes tree $\mathcal{T}$.

Solving When solving in a nonlinear setting, we obtain a delta vector Δ that is used to update the linearization point Θ, as shown in Algorithm 4. Updates are often local operations that do not affect the solution of other parts of the map. Therefore

Algorithm 4 Solving the Bayes tree in the nonlinear case returns an update Δ that can be added to the current linearization point Θ to obtain the current estimate for all variables $\Theta + \Delta$.

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 using already computed values of parents S_k and the local conditional density $P(F_k|S_k)$.
2. For all variables $\Delta_{k,j}$ in Δ_k that change by more than a threshold α:
   recursively process each descendant containing such a variable.

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

In: nonlinear factors $\mathscr{F}$, linearization point Θ, Bayes tree $\mathscr{T}$, delta Δ
Out: updated Bayes tree $\mathscr{T}$, updated linearization point Θ

1. Mark variables in Δ above threshold β: $J = \{\Delta_j \in \Delta | \Delta_j \geq \beta\}$.
2. Update linearization point for marked variables: $\Theta_J := \Theta_J + \Delta_J$.
3. Mark all cliques that involve marked variables Θ_J and all their ancestors.
4. From the leaves to the top, if a clique is marked:
   a. Relinearize the original factors in $\mathscr{F}$ associated with the clique.
   b. Add cached marginal factors from any unmarked children.
   c. Re-eliminate.

we will consider variables unchanged for which the recovered delta changes by less than a small threshold α. For a clique that does not contain any variables that are considered changed, the subtrees will not be traversed. 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.

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, and can be viewed as a completely new algorithm for nonlinear optimization. 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. Cliques that are re-eliminated have to take into account also the marginal factors that get passed up from subtrees. We cache those marginals during elimination to avoid having to re-eliminate unmarked cliques to obtain them. Algorithm 5 shows the overall fluid relinearization process.

Now we have all components for a fully incremental nonlinear optimization algorithm that allows exact incremental inference for sparse nonlinear problems such as SLAM. The algorithm is summarized in Algorithm 6, and we provide a brief dis-

Algorithm 6 Nonlinear iteration with incremental variable reordering and fluid relinearization.

In / out: Bayes tree $\mathscr{T}$, linearization point Θ, nonlinear factors $\mathscr{F}$
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. Linearize new factors $\mathscr{F}'$ to obtain $\mathscr{F}'_{lin}$.
4. Linear update step, applying Algorithm 3 to $\mathscr{F}'_{lin}$.
5. Solve for delta Δ with Algorithm 4.
6. Iterate Algorithm 5 until no more relinearizations occur.

cussion of its complexity here. We assume here that initialization is available and it is close enough to the global minimum to allow convergence - that is a general requirement of any direct solver method. The number of iterations needed to converge is typically fairly small, in particular because of the quadratic convergence properties of our algorithm near the minimum. For exploration tasks with a constant number of constraints per pose, the complexity is $O(1)$. 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, the complexity is bounded by $O(n^{1.5})$ [24]. It is important to note that this bound does not depend on the number of loop closings. It should also be noted that our incremental algorithm is often much faster than a full factorization, as we show below.

7 Experimental Results

This section describes the experiments that validate the presented approach, using both synthetic and real datasets that are publicly available. We compare our estimation and timing results with a state of the art incremental algorithm [23] in order to highlight the advantages of fluid relinearization and incremental reordering. We have implemented the batch and iSAM algorithms using the same Bayes tree library to provide a comparison of the algorithms, rather than a comparison of different implementations. All results are obtained with a research C++ implementation, running single-threaded on a laptop with Intel Core 2 Duo 2.2 GHz processor, and using the COLAMD algorithm by Davis et al. [5]. We use the thresholds $\alpha = 0.005$ and $\beta = 0.05$ for solving and relinearization, respectively.

We evaluate the timing of our Bayes tree algorithm on the Victoria Park dataset, an often-used benchmark SLAM dataset [23, 28] courtesy of H. Durrant-Whyte and E. Nebot. This dataset includes 6969 laser scans with the corresponding odometry readings. The laser scans are processed to detect the trunks of the trees in the park, which are used as landmarks. Fig. 5 shows the final trajectory estimate together

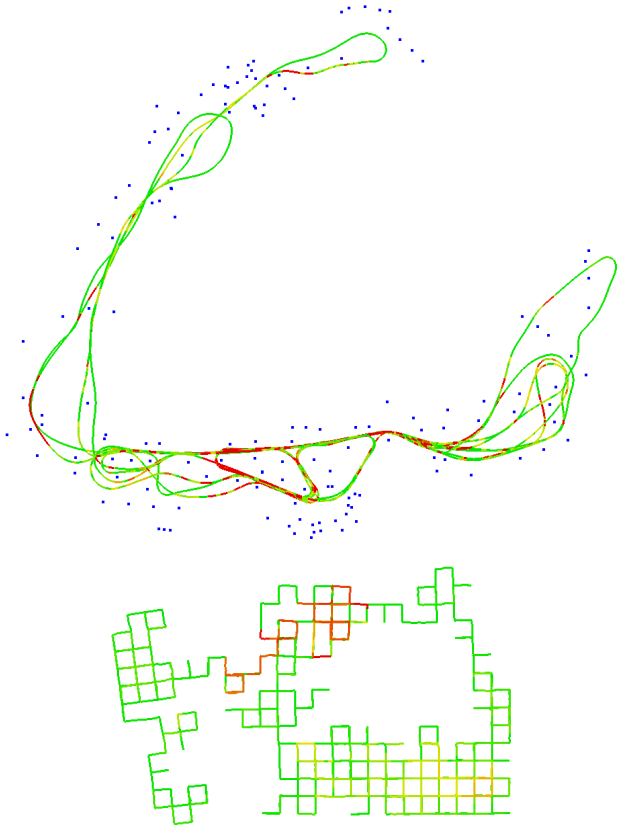

Fig. 5 The Victoria Park dataset (top) and the simulated Manhattan world dataset (bottom) after optimization, color coded with the number of variables that are updated for every step along the trajectory. Green corresponds to a low number of variables, red to a high number.

with the detected landmarks. In this figure the trajectory is colored according to the number of variables our algorithm had to recalculate at each step, where green represents a small number of variables (order of 10), yellow a moderate number, and red finally a large number (order of hundreds of variables). A relatively small portion of the trajectory is colored red, mainly the part at the bottom where the vehicle closed loops multiple times, re-visiting the same location up to eight times. In Fig. 6, we compare per-step timing on the Victoria Park dataset between our algorithm and the original iSAM algorithm [23] (both implemented using the same library as noted above). The results show that our fully incremental algorithm does not suffer from the periodic spikes in iSAM. Our algorithm also performs better in cumulative time, while providing the additional advantage of continuously updating the linearization point of all variables having significant changes.

To evaluate the accuracy of our algorithm, we use the simulated Manhattan world from [26], courtesy of E. Olson. Fig. 7 shows that the normalized chi-square values

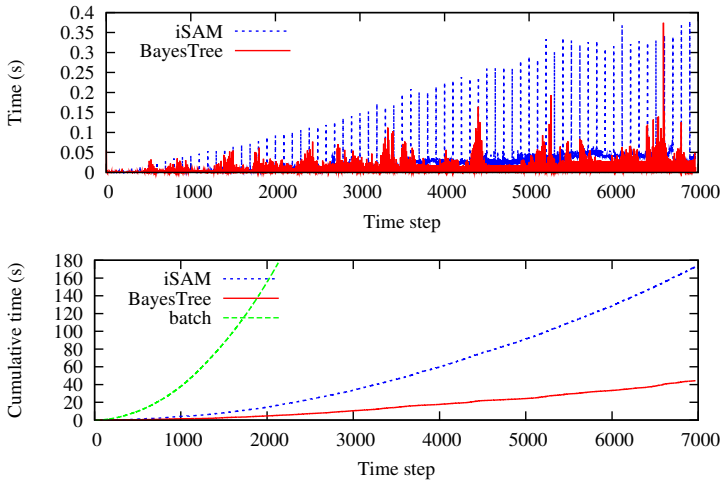

Fig. 6 Timing comparison for the Victoria Park dataset. The top row shows per step timing and the bottom row shows the cumulative time. Our new algorithm (red) provides an improvement in speed over the original iSAM algorithm (blue), in addition to its advantages of eliminating periodic batch factorization and performing fluid relinearization. The original iSAM algorithm included a batch step every 100 iterations, which is clearly visible from the spikes.

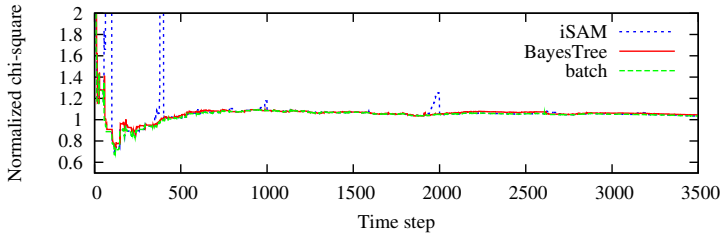

Fig. 7 Comparison of normalized χ^2 for the simulated Manhattan world. iSAM shows some spikes where it deviates from the least squares solution because relinearization is only performed every 100 steps. The Bayes tree solution is always very close to the least squares solution because of the fluid relinearization ($\beta = 0.05$).

follow the least-squares batch solution, providing a nearly exact solution in every step. While iSAM also converges to the exact solution, it shows some spikes related to relinearization only being performed in the periodic batch steps. Final cumulative times for providing a full solution in every step are 19.4s and 47.6s for our algorithm and iSAM, respectively. Fig. 5 shows the estimated trajectory for the simulated Manhattan world, again using the same color coding for the number of variables that had to be recalculated in each step.

Finally, we evaluated timing results on the Intel dataset, courtesy of D. Haehnel and D. Fox. This dataset was preprocessed by laser scan-matching, resulting in a

pose graph formulation without landmarks, containing about 4000 poses. The final cumulative times are 44.4*s* and 172.6*s* for our algorithm and iSAM, respectively.

8 Conclusion

We have presented a novel data structure, the Bayes tree, that provides an algorithmic foundation which enables new insights into existing graphical model inference algorithms and sparse matrix factorization methods. These insights have led us to a fully incremental algorithm for nonlinear least-squares problems as they occur in mobile robotics. 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. Our novel graph-based algorithm should 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 [21]. The graph based structure also provides a starting point for exploiting parallelization that is becoming available in newer processors.

References

1. Blair, J., Peyton, B.: An introduction to chordal graphs and clique trees. In: J. George, J. Gilbert, J.H. Liu (eds.) Graph Theory and Sparse Matrix Computations, *IMA Volumes in Mathematics and its Applications*, vol. 56, pp. 1–27. Springer-Verlag, New York (1993)
2. Brooks, R.: Visual map making for a mobile robot. In: IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 2, pp. 824 – 829 (1985)
3. Castellanos, J., Montiel, J., Neira, J., Tardós, J.: The SPmap: A probabilistic framework for simultaneous localization and map building. IEEE Trans. Robot. Automat. **15**(5), 948–953 (1999)
4. Cowell, R.G., Dawid, A.P., Lauritzen, S.L., Spiegelhalter, D.J.: Probabilistic Networks and Expert Systems. Statistics for Engineering and Information Science. Springer-Verlag (1999)
5. Davis, T., Gilbert, J., Larimore, S., Ng, E.: A column approximate minimum degree ordering algorithm. ACM Trans. Math. Softw. **30**(3), 353–376 (2004)
6. Dellaert, F.: Square Root SAM: Simultaneous location and mapping via square root information smoothing. In: Robotics: Science and Systems (RSS) (2005)
7. Dellaert, F., Kaess, M.: Square Root SAM: Simultaneous localization and mapping via square root information smoothing. Intl. J. of Robotics Research **25**(12), 1181–1203 (2006)
8. Dellaert, F., Kipp, A., Krauthausen, P.: A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping. In: Proc. 22^{nd} AAAI National Conference on AI. Pittsburgh, PA (2005)
9. Dissanayake, M., Newman, P., Durrant-Whyte, H., Clark, S., Csorba, M.: A solution to the simultaneous localization and map building (SLAM) problem. IEEE Trans. Robot. Automat. **17**(3), 229–241 (2001)
10. Folkesson, J., Christensen, H.: Graphical SLAM - a self-correcting map. In: IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 1, pp. 383–390 (2004)
11. Folkesson, J., Christensen, H.: Closing the loop with Graphical SLAM. IEEE Trans. Robotics **23**(4), 731–741 (2007)

12. Frese, U.: Treemap: An $O(\log n)$ algorithm for indoor simultaneous localization and mapping. Autonomous Robots **21**(2), 103–122 (2006)
13. Frese, U., Larsson, P., Duckett, T.: A multilevel relaxation algorithm for simultaneous localisation and mapping. IEEE Trans. Robotics **21**(2), 196–207 (2005)
14. Gauss, C.: 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 (1809). English translation available at http://name.umdl.umich.edu/AGG8895.0001.001
15. Gentleman, W.: Least squares computations by Givens transformations without square roots. IMA J. of Appl. Math. **12**, 329–336 (1973)
16. Gill, P., Golub, G., Murray, W., Saunders, M.: Methods for modifying matrix factorizations. Mathematics and Computation **28**(126), 505–535 (1974)
17. Golub, G., Loan, C.V.: Matrix Computations, third edn. Johns Hopkins University Press, Baltimore (1996)
18. Hall, B.: Lie Groups, Lie Algebras, and Representations: An Elementary Introduction. Springer (2000)
19. Heggernes, P., Matstoms, P.: Finding good column orderings for sparse QR factorization. In: Second SIAM Conference on Sparse Matrices (1996)
20. Julier, S., Uhlmann, J.: A counter example to the theory of simultaneous localization and map building. In: IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 4, pp. 4238–4243 (2001)
21. Kaess, M., Dellaert, F.: Covariance recovery from a square root information matrix for data association. Journal of Robotics and Autonomous Systems **57**, 1198–1210 (2009). DOI 10.1016/j.robot.2009.06.008
22. Kaess, M., Ranganathan, A., Dellaert, F.: Fast incremental square root information smoothing. In: Intl. Joint Conf. on AI (IJCAI), pp. 2129–2134. Hyderabad, India (2007)
23. Kaess, M., Ranganathan, A., Dellaert, F.: iSAM: Incremental smoothing and mapping. IEEE Trans. Robotics **24**(6), 1365–1378 (2008)
24. Krauthausen, P., Dellaert, F., Kipp, A.: Exploiting locality by nested dissection for square root smoothing and mapping. In: Robotics: Science and Systems (RSS) (2006)
25. Kschischang, F., Frey, B., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Trans. Inform. Theory **47**(2) (2001)
26. Olson, E., Leonard, J., Teller, S.: Fast iterative alignment of pose graphs with poor initial estimates. In: IEEE Intl. Conf. on Robotics and Automation (ICRA) (2006)
27. Paskin, M.: Thin junction tree filters for simultaneous localization and mapping. In: Intl. Joint Conf. on AI (IJCAI) (2003)
28. Paz, L.M., Pinies, P., Tardós, J.D., Neira, J.: Large scale 6DOF SLAM with stereo-in-hand. IEEE Transactions on Robotics **24**(5), 946–957 (2008)
29. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
30. Pothen, A., Sun, C.: Distributed multifrontal factorization using clique trees. In: Proc. of the Fifth SIAM Conf. on Parallel Processing for Scientific Computing, pp. 34–40. Society for Industrial and Applied Mathematics (1992)
31. Ranganathan, A., Kaess, M., Dellaert, F.: Loopy SAM. In: Intl. Joint Conf. on AI (IJCAI), pp. 2191–2196. Hyderabad, India (2007)
32. Smith, R., Self, M., Cheeseman, P.: A stochastic map for uncertain spatial relationships. In: Int. Symp on Robotics Research (1987)
33. Tarjan, R., Yannakakis, M.: 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 (1984)
34. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. The MIT press, Cambridge, MA (2005)