Multi-Level Submap Based SLAM Using Nested Dissection

Kai Ni and Frank Dellaert

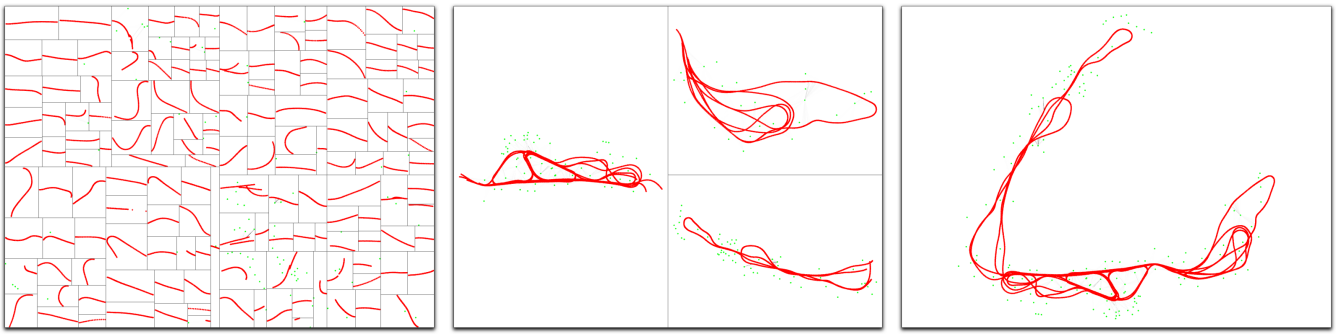

Fig. 1. **Our algorithm recursively partitions the SLAM graph into a submap tree, and the optimization runs from the leaves to the root.** *Following the treemap visualization [1], each rectangle represents a submap, and the sub-rectangles represent the submaps in the child level. The red and green dots are robot poses and landmarks respectively. From left to right: 1). the finest level of submaps; 2). the coarsest level of submaps; 3). the optimized full map.*

***Abstract*— We propose a novel batch algorithm for SLAM problems that distributes the workload in a hierarchical way. We show that the original SLAM graph can be recursively partitioned into multiple-level submaps using the nested dissection algorithm, which leads to the cluster tree, a powerful graph representation. By employing the nested dissection algorithm, our algorithm greatly minimizes the dependencies between two subtrees, and the optimization of the original SLAM graph can be done using a bottom-up inference along the corresponding cluster tree. To speed up the computation, we also introduce a base node for each submap and use it to represent the rigid transformation of the submap in the global coordinate frame. As a result, the optimization moves the base nodes rather than the actual submap variables. We demonstrate that our algorithm is not only exact but also much faster than alternative approaches in both simulations and real-world experiments.**

I. Introduction

Simultaneous localization and mapping (SLAM) has become the key to numerous applications for autonomous robots. Despite of the success of many SLAM projects, there are still some challenging scenarios in which most of the current algorithms are barely able to deliver an *exact* solution *fast* enough. One of these challenges is the size of SLAM problems, which has increased by several magnitudes over the last decade, and a lot of work has been done to push its limit, e.g. [2], [3], [4]. Another challenge for SLAM problems is the large amount of noise baked in the measurements, which often yields poor initializations and slows or even fails the optimization [5].

The divide-and-conquer scheme is one of the possible directions towards solving challenging SLAM problems, especially in the large-scale environments. The scheme is also referred as a *submap* based approach [6], [7], [8], [9] and in general consists of three steps. In the first step, submaps are created such that the dependencies between submaps are limited as small as possible. Next, each individual submap is optimized independently of other submaps, in a manner similar to non-submap approaches. At last, all the optimized submaps are joined together in a global optimization step.

One advantage of submap based approaches is that the computation can be done in an out-of-core manner. First, submaps make it possible to distribute most of work over multiple computation resources and increases the scalability in terms of both CPU time and the memory. Second, a lot of real data has different levels of noise from one portion to another, and the divide-and-conquer scheme can be easily used to allocate the computation resource smartly. In contrast, non-submap based approaches spend the resource evenly which leads to suboptimal workload scheduling.

The divide-and-conquer scheme also enable us to obtain a good initialization to batch approaches, which is one of the most crucial issues in nonlinear optimization. Compared to incremental approaches, typical batch approaches suffer from the bad initializations, e.g. those computed by composing robot odometries. By employing a divide-and-conquer approach, we can recursively compute the initializations from the optimized submaps, which are much accurate than those from conventional approaches.

Compared to previous incremental approaches in the same divide-and-conquer spectrum, we argue that a batch algorithm enables us to achieve better submap partitioning, which means less overlap between the submaps. A typical incremental approach creates a new submap whenever the size or the uncertainty of the current submap exceeds a threshold. Although this scheme works for the exploration scenarios, it becomes problematic and less efficient when the robot visits old places. In this case, there may exist multiple maps of the same area with possibly different estimations, and it is not only inefficient but also difficult to compute a

The authors are with the Georgia Institute of Technology, Atlanta, GA nikai,dellaert@gatech.edu.

consistent solution due to the local minima. From a global perspective, our batch algorithm is able to produce better partitioning, which in turn results in a much more efficient and robust approach compared to the traditional ones.

The divide-and-conquer scheme has been well studied for various optimization problems, especially in the SLAM community. As early as 1976, Brown [10] first employed the submap scheme in the aerotriangulation and mapping of city-scale areas. A recursive partitioning is used to exploit the band diagonal structure of the linear system in the project, and no nonlinearity is considered. The submap idea for SLAM problems was also investigated in the hierarchical SLAM [8] with a filtering-based local map building. However, their approach is carried out in two levels, and the map joining in the global level tends to become expensive with large maps. Later, Paz et al. [9] improved it by fusing the local maps in a hierarchical way, but the overall submap creating scheme is still suboptimal.

In both graph theory and linear algebra literature [11], [12], it has been shown that the efficiency of linear system solving depends heavily on the elimination ordering. More recently the same idea was also applied to the SLAM problem [13]. A good elimination ordering yields small cliques during graph triangulation and introduces fewer non-zero fill-ins during matrix factorizations. Although finding an optimal ordering is NP-complete [14], there are two successful schemes for finding a good ordering: *minimum degree* (its variants include MMD [15] and AMD [16]) and *nested dissection* [17], [18].

For submap based approaches, we found that nested dissection has more appealing properties than minimum degree. Although both schemes generate elimination orderings with comparable qualities, nested dissection tends to perform better on large graphs, and its elimination trees are typically lower and better balanced, which naturally fits our hierarchical partitioning. In addition, for solving SLAM problems with planar graphs [19], nested dissection has seen proven to be optimal. In the paper, we use the state-of-art hybrid ordering [20] that combines the advantages of AMD and nested dissection for small and large graphs respectively.

Graph-based SLAM algorithms, especially tree-based representations has recently gained in popularity but still lack on some aspects. Paskin [21] first employed a tree-based data structure, so called junction tree, to capture the belief state of robot poses and landmarks in SLAM problems. The algorithm is referred to as thin junction tree filter (TJTF). To speed up the inference, the tree gets "thinned" periodically by variable contraction, which is proven to minimize the KL divergence before or after the edge removal. Frese [2], [22] introduced another fast algorithm, called treemap, as a back-end for solving large-scale SLAM problems. In addition, treemap applies a sophisticated hierarchical tree partitioning (HTP) to re-balance the binary tree and reduce the cost of propagating the information from leaves to the root. Note that HTP during the run-time does not produce a junction tree, and hence its optimization is not necessary most efficient. Both TJTF and treemap marginalize the nodes to keep the tree sparse, such that the algorithms become inexact for the same reason as the other filtering approaches. We argue that the marginalization should be rather prevented for better accuracies. Our work also shares the same hierarchical idea as HOG-Man [23], but our algorithm is free of the overhead introduced by making virtual measurements and explicitly propagating the changes between successive levels. Therefore, it is more efficient as demonstrated in the experiments.

II. Tectonic SAM

Previously we introduced Tectonic Smoothing and Mapping (TSAM) [24], which is a two-level submap based approach. However, TSAM does not have the ability to maintain hierarchical maps hence does not scale well enough. In addition, TSAM uses edge separators to partition the SLAM graph and generates fully connected separators, which slows down the computation for large maps. TSAM also requires multiple iterations between the submaps and the separator to converge to exact minima. Our motivation is to address these issues in a new algorithm we are going to introduce.

In this paper, we propose TSAM 2, a novel *multi-level* SLAM approach that employs the *nested dissection* algorithm [17], [18] to solve SLAM problems in an efficient, robust, and *exact* manner. After recursively partitioning the original SLAM graphs (Figure 1), we can represent the decoupled SLAM problem by a *cluster tree.* As defined in [25], a cluster tree is a directed tree of clusters in which the running intersection property holds, and each factor in the original graph is associated with a cluster. In fact, the cluster tree is more general than the junction tree or the clique tree [12], which have already been widely used in the graphical model based inference. Here we simply use the cluster tree to organize our SLAM computation.

We also introduce *base nodes* to speed up the convergence of nonlinear optimization and fully exploit the power of the submap representation. The intuition here is rather straightforward: if individual submaps have been optimized properly, they only need to move by some unknown rigid transformations with respect to each other in the global optimization step. Such an effect can be achieved by introducing a base node for each submap. Instead of optimizing over all the variables as in the traditional approaches, we move all the variables in the submaps as bundles represented by their base nodes, hence our algorithm is able to work on a much smaller set of unknown variables and run faster.

III. Problem Formulation

In this paper, we use the same formulation as used by Dellaert and Kaess in [13]. We denote the robot poses as $X = \{x_i\}$ with $i \in [0, M]$ and the landmarks as $L = \{l_j\}$ with $j \in [1, N]$. The joint probability of the robot poses and the map can be then formulated as:

$$P(X, L, Z) \propto \prod_{i=1}^{M} P(x_i|x_{i-1}, u_i) \prod_{k=1}^{K} P(z_k|x_{i_k}, l_{j_k}) \quad (1)$$

where u_i is the control input at step i, and z_k is the measurement of landmark l_{j_k} at robot pose x_{i_k}.

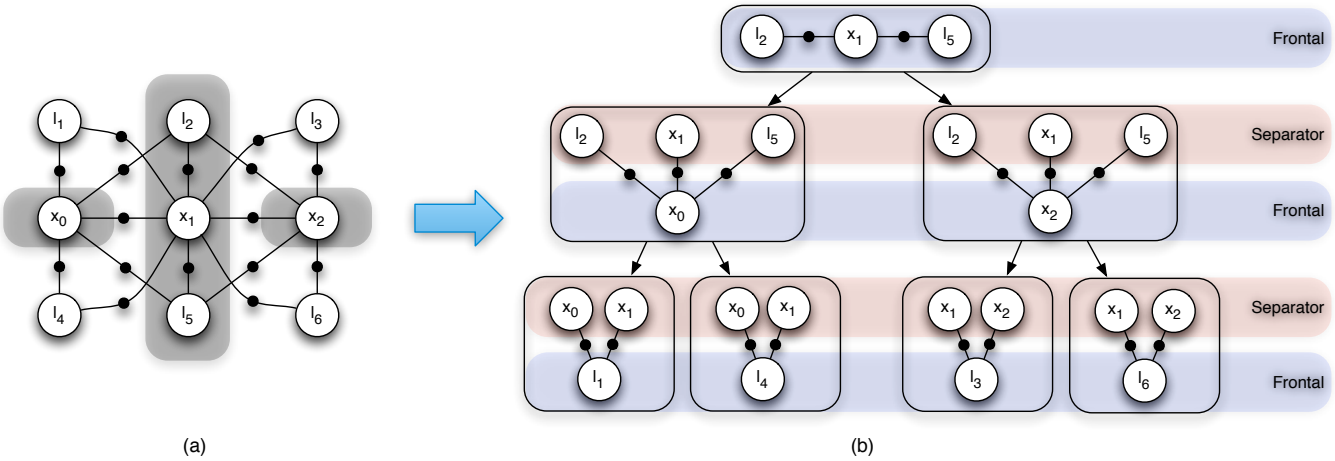

Fig. 2. **Recursive partitioning with nested dissection.** *(a). The input factor graph is partitioned using nested dissection. For each partitioning, the separator, highlighted by the shadow, partitions the subgraph into two pieces, which correspond to the two children in the cluster tree. (b) The resulting cluster tree representation of the original factor graph. Note that all the factors are captured by the clusters of the resulting cluster tree.*

We employ *factor graphs* [26] to represent the SLAM problem. As illustrated in Figure 2.a, the factor graph G is a bipartite graph and can be denoted as a tuple $(\mathcal{F}, \Theta, \mathcal{E})$, where $\mathcal{F}$ is the factor nodes corresponding to the constraints (odometers or landmark measurements), Θ is the variable nodes corresponding to unknowns X and L, and $\mathcal{E}$ is the edges connecting $\mathcal{F}$ and Θ. Indeed, the factor nodes define the joint probabilities over their involving variables in Equation 1, and the factor graph G defines its factorization:

$$P(X, L, Z) \propto f(G) = \prod_i f_i(\Theta_i) \tag{2}$$

where Θ_i is the set of variables connected to factor f_i.

Furthermore, we assume Gaussian noise in measurement models as is standard in SLAM literature, and the SLAM problem can be converted to a least square problem [27]:

$$\begin{aligned} -\log\left(f(\Theta)\right) \quad &\propto \quad \sum_i \|q_i(x_{i-1}, u_i) - x_i\|^2_{\Lambda_i} \\ &+ \quad \sum_k \|h_k(x_{i_k}, l_{j_k}) - z_k\|^2_{\Sigma_k} \end{aligned} \tag{3}$$

where q_i is the motion model, and h_k is the measurement model. Both the models have zero-mean Gaussian noise with covariance matrices Λ_i and Σ_k respectively. Here $\|e\|^2_\Sigma \triangleq e^T\Sigma^{-1}e$ denotes the squared Mahalanobis distance given the covariance matrix Σ.

The maximum a posteriori (MAP) estimate Θ^* of the robot poses X and the landmarks L can be obtained by maximizing the joint probability $P(X, L, Z)$, which is equivalent to minimizing the negative log-likelihood in Equation 3:

$$\Theta^* = \underset{\Theta}{\text{argmin}} \ -\log(f(\Theta))$$

IV. Partitioning with Nested Dissection

To create a hierarchical set of submaps, we *recursively partition* the SLAM graph using nested dissection. More specifically, we use METIS [28] to find a nested dissection ordering at the global level and then order the resulting subgraphs locally using AMD algorithm [16].

The basic idea of nested dissection is to recursively find small vertex separators such that at each level the remaining two subgraphs are disconnected. In other words, given the original graph G, nested dissection continuously partitions as follows: each time the current subgraph G_i is split into three sets A_i, B_i, and C_i, such that no vertex in A_i is connected to any vertex in B_i. Note that nested dissection does not guarantee that A_i or B_i is a connected graph. In the disconnected case, we simply make each disconnected component as a new subgraph. Hence the tree induced by nest dissection is usually a K-way tree rather than a binary tree. Without loss of generality, we use the nested dissection notations above and assume we always obtain two-way cuts.

The partitioning yields an ordering called a nested dissection ordering. The sets A_i, B_i are referred to as the *frontal variables*, and C_i is called the *separator* of A_i and B_i. The nodes in submap A_i and B_i are grouped together and ordered first, the separator nodes in C_i are ordered last. By ordering the variables in this manner, we can perform a graph elimination or the corresponding sparse matrix factorization in each submap and then in the separator. Because submap variables do not have connections to variables in other submaps, the inference tasks in individual submaps can be carried out in parallel.

In the context of the SLAM problem represented by a factor graph G, nested dissection distributes all the factors $\mathcal{F}$ along a cluster tree T [25], as shown in Figure 2.b. The recursive partitioning starts from the root and continuously builds the child subgraphs. For a certain subgraph in the cluster tree, let us assume that we have the remaining factors $\mathcal{F}_0$, the frontal variables Θ_0, and the separator Θ_S inherited from its parent (in the case of the root node, $\mathcal{F}_0 = \mathcal{F}$ and $\Theta_S = \emptyset$). The nested dissection algorithm groups the variables Θ_0 into three sets Θ_A, Θ_B and Θ_C, such that no variable in Θ_A shares any factor with variables in Θ_B.

Algorithm 1 Recursively partition the input factor graph $G = (\mathcal{F}, \Theta, \mathcal{E})$ and build a cluster tree which encapsulates all the factors. The recursive algorithm starts by calling *root=Partition*($\mathcal{F}, \Theta, \emptyset$).

Function: $\mathcal{M}$=***Partition***($\mathcal{F}_0$, Θ_0, Θ_S)
if $size(G) > \alpha$ **or** G_0 is not fully connected
  $(\Theta_A, \Theta_B, \Theta_C)$ =***Nested_Dissection***(Θ_0);
  $\mathcal{F}_0 \longrightarrow \mathcal{F}_A \cup \mathcal{F}_B \cup \mathcal{F}_C$;
  find Θ_{AC} and Θ_{BC};
  $\mathcal{M} = (\mathcal{F}_C, \Theta_C, \Theta_S)$;
  $\mathcal{M}$.*left_child*=***Partition***($\mathcal{F}_A, \Theta_A, \Theta_{AC}$));
  $\mathcal{M}$.*right_child*=***Partition***($\mathcal{F}_B, \Theta_B, \Theta_{BC}$));
else
  $\mathcal{M}$=$(\mathcal{F}_0, \Theta_0, \Theta_S)$;
end
return $\mathcal{M}$;

Based on the frontal variables Θ_C, we define submap $\mathcal{M}$ as a tuple of factors, the frontal variables, and the separator:

$$\mathcal{M} = (\mathcal{F}_C, \Theta_C, \Theta_S) \tag{4}$$

where $\mathcal{F}_C$ are the factors in $\mathcal{F}_0$ not connected to any variable in Θ_A and Θ_B, notated as $E(\mathcal{F}_C, \Theta_A) \cup E(\mathcal{F}_C, \Theta_B) = \emptyset$. The frontal variable set of the new submap $\mathcal{M}$ is Θ_C, and the new separator of $\mathcal{M}$ is Θ_S. The remaining factors can be grouped into two sets $\mathcal{F}_A$ and $\mathcal{F}_B$ with respect to Θ_A and Θ_B, such that $E(\mathcal{F}_A, \Theta_A) \neq \emptyset$ and $E(\mathcal{F}_B, \Theta_B) \neq \emptyset$. We may find the separators of two child nodes as $\Theta_{AC} \subset \Theta_C \cup \Theta_S$ with $E(\mathcal{F}_A, \Theta_{AC}) \neq \emptyset$ and $\Theta_{BC} \subset \Theta_C \cup \Theta_S$ with $E(\mathcal{F}_B, \Theta_{BC}) \neq \emptyset$. Such a factorization can be written as

$$f(G_0) = f(\Theta_A) f(\Theta_B) f(\Theta_C)$$

The recursive partitioning exits when the size of the current subgraph is below a certain threshold α ($\alpha = 40$ in our experiments). As all the factors in the original factor graph G have been distributed to the nodes of cluster tree T, we may format Equation 2 in terms of a cluster tree:

$$f(G) = \prod_{i \in T} \prod_{f_j \in \mathcal{F}_i} f_j(\Theta_{j_k})$$

where Θ_{j_k} are the variables associated with the factor f_j. All the nodes in the subgraphs can be further ordered using AMD. The pseudo-code is listed in Algorithm 1. The collection of all the clusters (submaps) forms the cluster tree.

V. Cluster Tree Inference

The main idea of submap based approaches is divide-and-conquer: first the submaps are optimized locally, and their relative positions and orientations are determined when optimizing their parent submap. The same idea applies to our cluster tree representation: the inference is first done locally inside each submap by eliminating frontal variables and then finished by a back-substitution step. For linear systems, this technique is also referred to as a multifrontal method [20].

Algorithm 2 The elimination of the frontal variables $F_\mathcal{M}$ in the submap $\mathcal{M}$ using AMD orderings. The recursive algorithm generates a set of new factors $\mathcal{F}_S$ propagated to the parent cluster of $\mathcal{M}$.

Function: $\mathcal{F}_S$=**EliminateNode**($\mathcal{M}$)
with $\mathcal{M} = (\mathcal{F}_\mathcal{M}, F_\mathcal{M}, S_\mathcal{M})$
if $\mathcal{M}$.**hasChildren**()
  $\mathcal{F}_A$ =**EliminateNode(**$\mathcal{M}$.*left_child*);
  $\mathcal{F}_B$ =**EliminateNode(**$\mathcal{M}$.*right_child*);
  (*clique*, $\mathcal{F}_S$)=**Triangulate**($\mathcal{F}_\mathcal{M} \cup \mathcal{F}_A \cup \mathcal{F}_B$);
else
  $\mathcal{F}_S$ =**Triangulate**($\mathcal{F}_\mathcal{M}$);
end
return $\mathcal{F}_S$;

A. Leaves-to-Root Elimination

First, we apply an elimination algorithm to the partitioned subgraphs obtained from Algorithm 1 in leaf-to-root order. The frontal variables $F_\mathcal{M}$ (Θ_C in Equation 4) are eliminated for each submap graph $\mathcal{M}$, which yields a directed subgraph as shown in Figure 3. The pseudo-code is in Algorithm 2.

The elimination is indeed refactorizing the factor graph contained in submap $\mathcal{M}$, and it is equivalent to applying the chain rule to the joint probability of the frontal variables $F_\mathcal{M}$ and the separator variables $S_\mathcal{M}$ (Θ_S in Equation 4) :

$$P(F_\mathcal{M}, S_\mathcal{M}) = P(F_\mathcal{M}|S_\mathcal{M}) P(S_\mathcal{M}) \tag{5}$$

Note that $P(S_\mathcal{M})$ corresponds to the new factors between the separator variables and is propagated to the parent submap, shown as squares in Figure 3. For each cluster $\mathcal{M}$, as $P(F_\mathcal{M}|S_\mathcal{M})$ is a Gaussian density, eliminating frontal variables is equivalent to factorizing the corresponding matrix:

$$P(F_\mathcal{M}|S_\mathcal{M}) \propto \exp -\frac{1}{2} \left\| R_\mathcal{M} F_\mathcal{M} + A_\mathcal{M} S_\mathcal{M} - d_\mathcal{M} \right\|^2_{\Sigma_\mathcal{M}} \tag{6}$$

in which $\begin{bmatrix} R_\mathcal{M} & A_\mathcal{M} \end{bmatrix}$ is the factorized matrix with $R_\mathcal{M}$ being upper triangular. $d_\mathcal{M}$ is the corresponding right-hand side (RHS), and $\Sigma_\mathcal{M}$ is the covariance matrix. The matrix factorization is currently done using SuiteSparseQR [29].

B. Root-to-Leaves Back-Substitution

Once the elimination step is done, the optimal values of all the variables can be obtained by performing back-substitutions in the root-to-leaf order. The process starts from the root cluster and recursively solves the children. For each cluster $\mathcal{M}$ with frontal variables $F_\mathcal{M}$, as all the parent clusters have been solved, we may compute frontal variables $F_\mathcal{M}$ using the estimate of separator variables $S_\mathcal{M}$:

$$F_\mathcal{M} = R_\mathcal{M}^{-1} (d_\mathcal{M} - A_\mathcal{M} S_\mathcal{M})$$

VI. Nonlinear Optimization

In this section, we extend the algorithm for linear systems we introduced in the last section to the nonlinear case. Due to the nonlinear nature of the SLAM problems, linearization errors prevent the system from converging within one pass

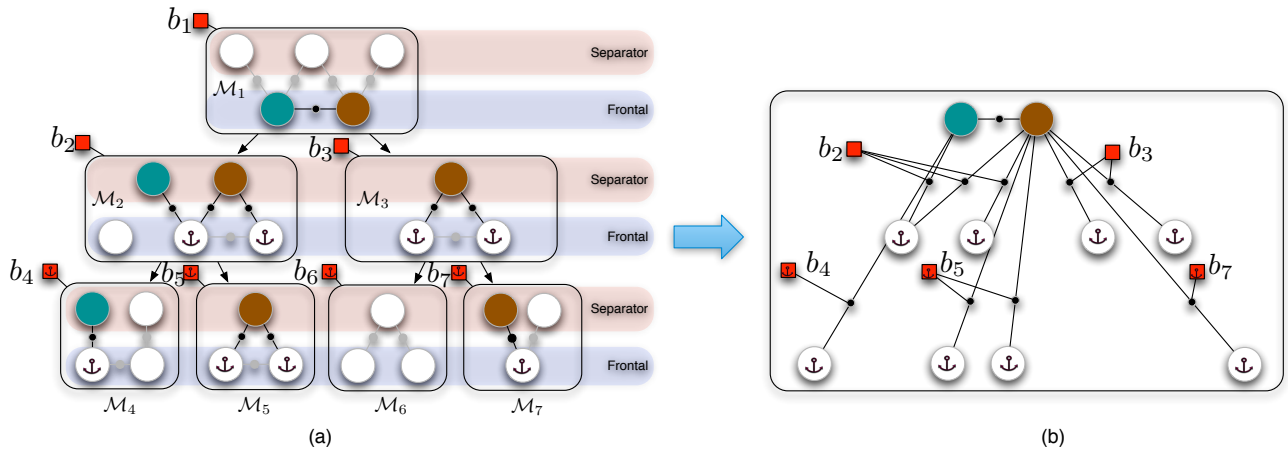

Fig. 4. **The optimization of submap $\mathcal{M}_1$ whose two frontal variables are colored in green and in brown.** *The base nodes are shown as red squares. Note that instead of moving all the variables in the figure, we only need to compute the two frontal variables and the two base nodes b_2 and b_3 of the direct child submaps $\mathcal{M}_2$ and $\mathcal{M}_3$.* ***(a)****: Each submap is assigned a base node, which is colored in red. Note that the factors are not used when optimizing submap $\mathcal{M}_1$ are greyed out. For the active factors, since we will only move the two frontal variables of $\mathcal{M}_1$ and its base nodes b_2 and b_3, all the other variables stay fixed , labelled with anchor labels.* ***(b)****: By removing all the uninvolved factors and variables on the left, we have the actual factor graph used for optimizing submap $\mathcal{M}_1$.*

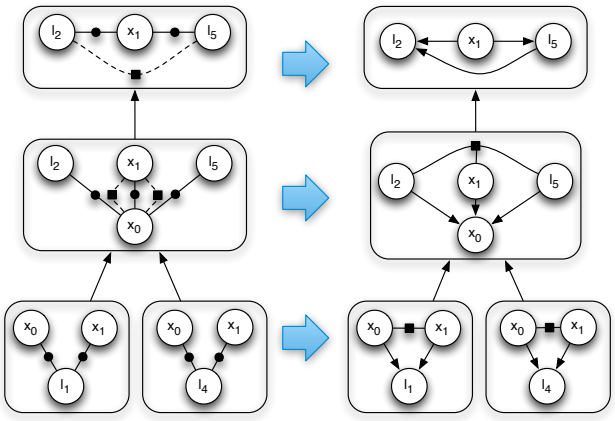

Fig. 3. **The elimination on the partitioned factor graphs (left) in Figure 2 yields the corresponding cluster tree (right).** *Note that the square factors with dashed edges are those propagated from the child clusters. For example, the elimination on the clusters $\{l_1 : x_0, x_1\}$ and $\{l_4 : x_0, x_1\}$ generates two new factors $\mathcal{F}_1(x_0, x_1)$ and $\mathcal{F}_2(x_0, x_1)$, which are sent to its parent cluster $\{x_0 : l_2, x_1, l_5\}$. Due to the constraint of the space, only the left subtree of the cluster tree T in Figure 2 is illustrated here.*

of elimination and back-substitution, hence iterative methods are often employed to tackle the linearized system at the latest estimate. Below we show how such nonlinear optimization adapts to the multiple-level submaps.

A. Subtree Optimization with Base Nodes

In the hierarchical context, the subtree optimization mainly serves two purposes. Assume the subtree with root cluster $\mathcal{M}$ is denoted as $\mathcal{T}_{\mathcal{M}}$, and $\mathcal{T}_{\mathcal{M}} = \mathcal{M} \cup \mathcal{T}_{\mathcal{M}_1} \cup \cdots \cup \mathcal{T}_{\mathcal{M}_s}$ in which s is the number of $\mathcal{M}$'s child clusters. First, subtrees $\{\mathcal{T}_{\mathcal{M}_k}\}$ are aligned together. Second, the relative positions between the frontal variables $F_{\mathcal{M}}$ of $\mathcal{M}$ are determined. The first task can be done by using the factors propagated from $\{\mathcal{T}_{\mathcal{M}_k}\}$, and the second task can be achieved by using the factors stored locally in the cluster $\mathcal{M}$. One of such examples is shown in Figure 4.a. In practice, All those factors constitute a new nonlinear factor graph used for the submap optimization.

One naive way to optimize over the resulting nonlinear factor graph is to optimize over the child variables $C_{\mathcal{M}}$ in $\{\mathcal{T}_{\mathcal{M}_k}\}$ and $F_{\mathcal{M}}$ together. However, this approach does not exploit the fact that each subtree $\mathcal{T}_{\mathcal{M}_k}$ has usually been well recovered during the previous optimization. If the size of variable set $C_{\mathcal{M}}$ is large, such an optimization process is not only inefficient but also tend to get stuck in local minima.

In this paper, we solve the problem by introducing base nodes and exploiting the local structure of the submaps. For each submap $\mathcal{M}_k$, we assign a base node b_k that represents the position and the orientation of the submap in its parent's coordinate system. Considering the structure of the cluster tree, if the submaps between the submap $\mathcal{M}_k$ and the root are $\mathcal{M}_p, \mathcal{M}_{p+1}, \cdots, \mathcal{M}_{k-1}$, we may see that a robot pose x_i in the submap $\mathcal{M}_k$ satisfies the following relationship:

$$x_i = b_p \otimes b_{p+1} \otimes \cdots \otimes b_{k-1} \otimes b_k \otimes \widetilde{x}_i$$

where $\widetilde{x}_i$ is the robot pose x_i in the local coordinate system of submap $\mathcal{M}_k$. The similar chained rigid transformation also applies to all the landmarks in submap $\mathcal{M}_k$.

In this way, the resulting optimization process only works on $F_{\mathcal{M}}$ and the base nodes of the child clusters, as shown in Figure 4. The variable set $C_{\mathcal{M}}$ will stay fixed as anchor variables (labelled by anchor icons in the figure). Note that only the base nodes of direct child submaps are movable (b_2 and b_3 in Figure 4), since the base nodes of other child submaps (b_4, b_5, and b_7) have already been optimized when optimizing maps $\mathcal{M}_2$ and $\mathcal{M}_3$.

The complexity of the new submap optimization is greatly reduced with respect to the naive approach. The complexity of the method without base nodes is $O(|C_{\mathcal{M}} \cup F_{\mathcal{M}}|^2)$. By

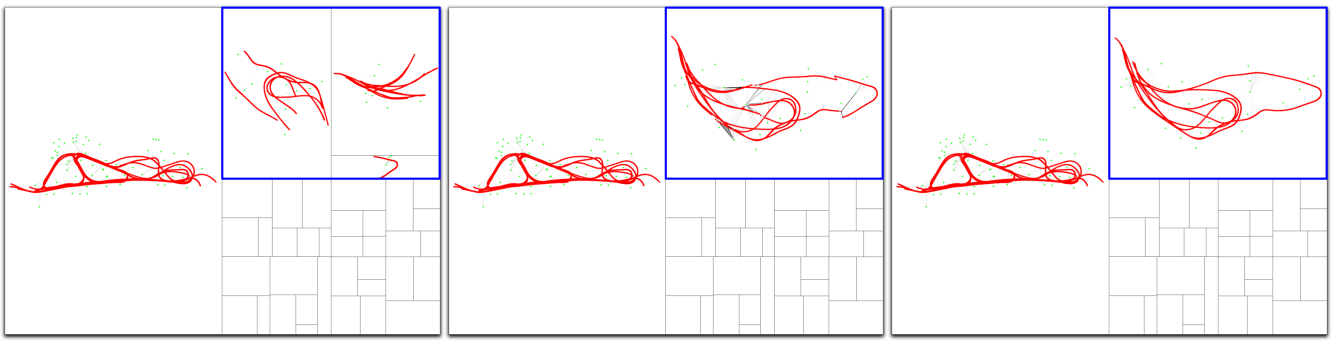

Fig. 5. **The optimization of one submap in Victoria Park data-set.** *The rectangle stroked in blue corresponds to the current submap. The intensity of the black lines indicates the amount of residuals on the corresponding measurements. From left to right: 1). Three child submaps that have been optimized before. 2). The new submap is created by aligning three child submaps using base nodes. Note that the three submaps are roughly aligned together, and a few constraints (black lines) are not satisfied very well. 3). The submap after the relaxation step. Nearly all the constraints are perfectly satisfied now.*

introducing the base nodes, we limit the complexity to $O((|F_{\mathcal{M}}| + s)^2)$ considering that there are s base nodes involved. In the experiments, we will show that $|F_{\mathcal{M}}|$ is usually much smaller than $|C_{\mathcal{M}}|$ and almost stays constant due to the minimized sizes of vertex separators. On the other hand, $|C_{\mathcal{M}}|$ increases with respect to the number of clusters in $\mathcal{T}_{\mathcal{M}}$. For the large-scale SLAM problems, $|C_{\mathcal{M}}|$ of the submaps close to the root is almost equal to the number of total variables and consequently dominates $|C_{\mathcal{M}} \cup F_{\mathcal{M}}|$.

B. Subtree Relaxation

Once the subtree alignment is finished, we may invoke a "relaxation step" to further balance all the variables in the subtree whose root is the current submap. In the relaxation step, all the base nodes stay fixed, and the resulting optimization is equivalent to the traditional SAM [13] approach. As both the local structure of child submaps and the current submap are well optimized, we found at most two iterations of the full optimization suffices for all clusters except the root. For the root cluster in cluster tree, we don't constrain the number of SAM iterations, i.e. using the same termination as SAM, which guarantees that TSAM 2 and SAM converge to the *exactly same* minimum.

VII. Experimental Results

A. Victoria Park Data-set

We first tested our algorithm on the public Victoria park data-set, which contains about 6900 laser scans with corresponding odometry readings. All the results reported in this paper were produced on a Macbook Pro with 2.8GHz CPU.

As shown in Figure 1, the SLAM graph is recursively partitioned into multiple submaps, and one of the submap optimization steps is illustrated in Figure 5. The advantage of using a nested dissection based recursive partitioning is shown in Figure 6. We can see that, despite of more than seven thousand nodes in the entire graph, the sizes of most submaps are between 10 to 60. In fact, the average size of all the submaps is 38.3, which is only 0.52% of the total size. The root submap, i.e. the vertex separator between three coarsest-level submaps only contains 19 nodes. Given those 19 nodes, the three submaps are statistically independent.

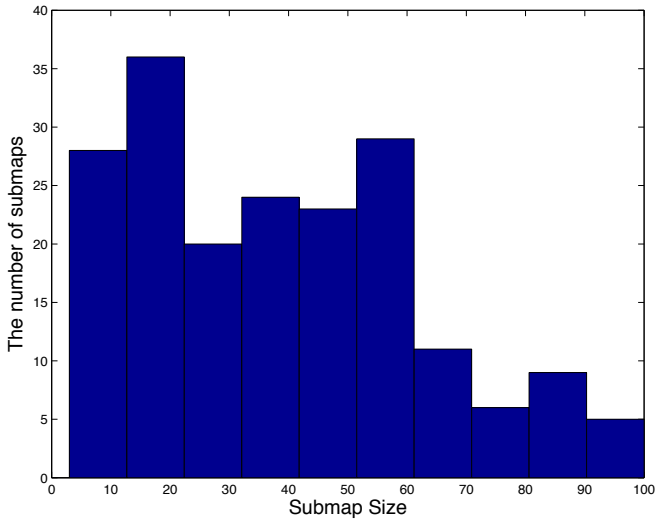

Fig. 6. **The histograms of submap sizes for the Victoria Park data-set.**

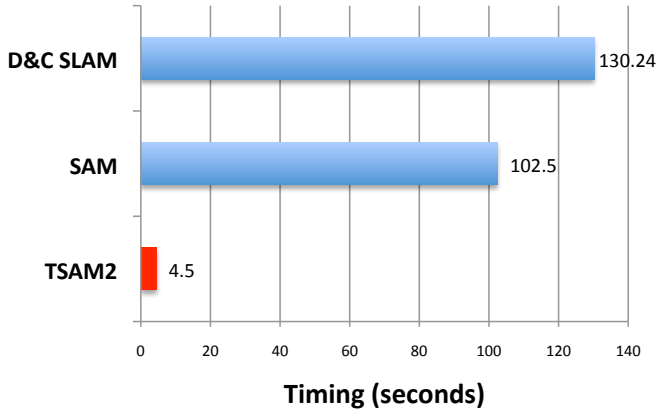

Fig. 7. **The comparison of timing results on Victoria Park data-set between TSAM 2, SAM and D&C SLAM.**

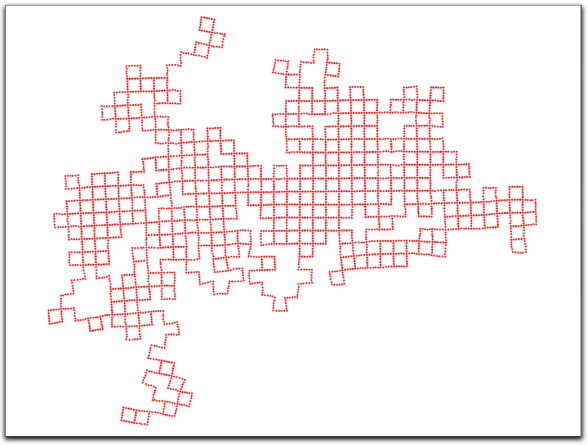

Fig. 8. **The optimized map of W-10000 data-set by TSAM 2.**

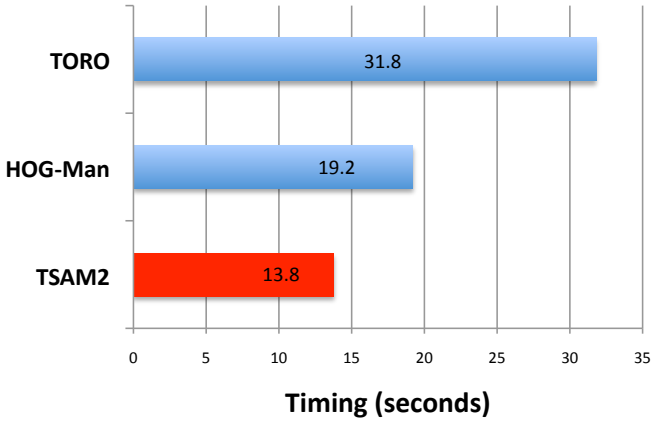

Fig. 9. **The comparison of timing results on W-10000 between TORO, HOG-Man, and TSAM 2.**

When assembling those submaps, we only need to optimize over those 19 nodes as well as the three base nodes.

In terms of efficiency, we compared our algorithm with SAM [13] and D&C SLAM [9], which is one of the fastest submap based algorithms available there. As the initial estimate is quite off due to the noisy measurements, we found that SAM does not converge if directly applied to the entire map. Hence we gradually increased the size of the active map and applied the algorithm sequentially. Also note that the timing of D&C SLAM was originally reported in [9] with 2.8GHz CPU and did include data-association overhead. The final comparisons are plotted in Figure 7. TSAM 2 was able to finish the entire optimization in less than 5 seconds.

In terms of accuracy, we observed that TSAM 2 converged to the exactly same minimum as SAM. In the relaxation steps of non-root clusters, our algorithm ran two iterations of full subtree optimization. For the root submap, it took one additional iteration, i.e. 3 iterations of SAM, to converge to the optimal point. This result verified that TSAM 2 is *exact*.

B. W-10000 Data-set

Another public data-set we used to verify our algorithm is W-10000 included in the HOG-Man release [23], a SLAM

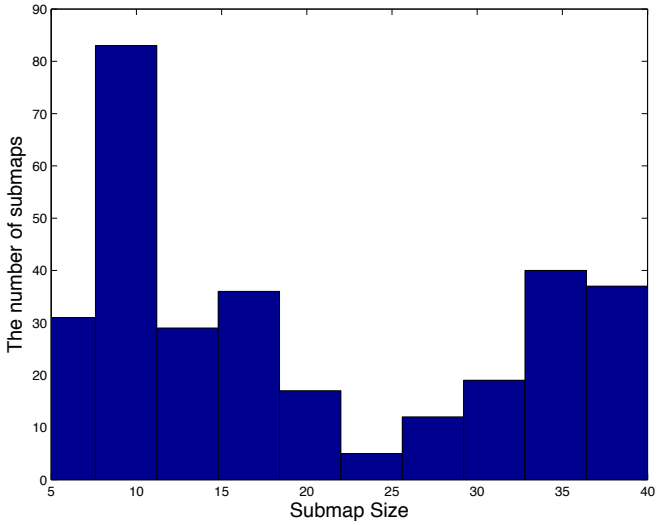

Fig. 11. **The histograms of submap sizes in the block-world simulation.**

algorithm that also exploits the hierarchical idea. The final map produced by TSAM 2 is illustrated in Figure 8. We found that the HOG-Man code ran a little faster in our experiments than what was reported in [23] due to the hardware difference. Overall, TSAM 2 is 1.4 times faster than HOG-Man (Figure 9), and the residual of HOG-Man after the convergence is 3% higher than our algorithm.

We also compared the results with another state-of-the-art batch algorithm, TORO [30]. We ran the batch version of TORO for 300 iterations, and its final residual is 6% higher than that from TSAM 2. Our algorithm not only produced better maps but also converged 2.3 times faster than TORO as shown in Figure 9.

C. Block-world Simulation

To further verify the performance of the proposed algorithm, we created a block-world simulation that contains a trajectory with 2640 robot poses and 3200 landmarks, as illustrated in Figure 10. We observed similar submap sizes as those found on the Victoria Park data-set, as shown in Figure 11. Since the map of the block-world simulation is more structured than the map of Victoria Park, the submap sizes are more consistent. Overall in both data-sets, the sizes remain small compared to the total map size. For example, the root submap contains 29 nodes, i.e. 0.50% of the total nodes. Having such small vertex separators is the key for our algorithm to achieve the high efficiency. We refer [19] to interested readers for more theoretical proofs.

We compared the efficiency of our algorithm to SAM. For each submap including the root submap, TSAM 2 ran two iterations of full optimization. TSAM 2 took 13.6 seconds to converge, while SAM took 67.9 seconds to finish. Both algorithms again converged to the same minimum point.

VIII. Conclusions

The main contribution of this paper is introducing an nested dissection based SLAM approach, which yields a hierarchical set of submaps. In this framework, the optimization

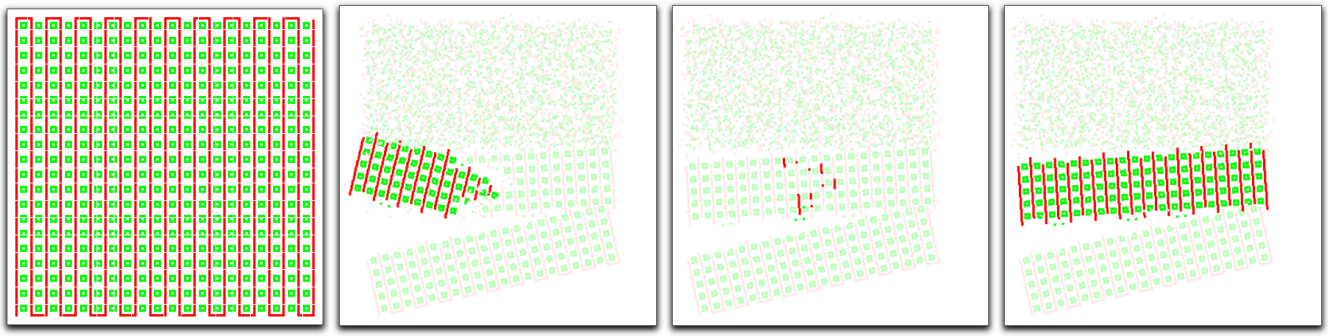

Fig. 10. **The block-world simulation in which the robot poses are rendered in red and the landmarks are rendered in green.** *The high-lighted nodes are the variables being optimized, and the other inactive nodes are rendered in the washed-out colors. From left to right: (1). The original block-world simulation. (2). The optimization of a certain submap $\mathcal{M}_k$. (3). The high-lighted part is the vertex separator between submap $\mathcal{M}_k$ and another submap $\mathcal{M}_{k+1}$ on the right. (4). The one-iteration relaxation for the quarter-size map. Note that there is no visible change between the third figure and the fourth figure, hence one iteration is suffice to re-balance all the nodes in the quarter-size map.*

can be carried out efficiently and exactly on the cluster tree representation. The usage of the base nodes further speeds up the computation by optimizing the submaps as bundles.

The future work includes the full exploitation of the parallel computation power introduced by the cluster tree representation as well as applying the algorithm to 3D problems. We also plan to make an online variant of the algorithm that incrementally builds the cluster tree and re-balance it whenever necessary.

IX. Acknowledgements

We gratefully acknowledge the support from the National Science Foundation, Awards No. 0713162 and the CAREER Award No. 0448111.

References

[1] B. Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," in *ACM Transactions on Graphics*, 1992.

[2] U. Frese and L. Schröder, "Closing a million-landmarks loop," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Oct 2006, pp. 5032–5039.

[3] R. Kummerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard, "Large scale graph-based slam using aerial images as prior information," in *Robotics: Science and Systems (RSS)*, 2009.

[4] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, "Building rome in a day," in *Intl. Conf. on Computer Vision (ICCV)*, Kyoto; Japan, October 2009.

[5] S. Julier and J. Uhlmann, "A counter example to the theory of simultaneous localization and map building," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 4, 2001, pp. 4238–4243.

[6] S. Williams, "Efficient solutions to autonomous mapping and navigation problems," Ph.D. dissertation, The University of Sydney, 2001.

[7] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework," *Intl. J. of Robotics Research*, vol. 23, no. 12, pp. 1113–1139, Dec 2004.

[8] C. Estrada, J. Neira, and J. D. Tardas, "Hierarchical SLAM: Real-time accurate mapping of large environments," *IEEE Trans. Robotics*, vol. 21, no. 4, pp. 588–596, Aug 2005.

[9] L. Paz, J. Tardos, and J. Neira, "Divide and conquer : Ekf slam in o(n)," *IEEE Trans. Robotics*, vol. 24, no. 5, October 2008.

[10] D. C. Brown, "The bundle adjustment - progress and prospects," *Int. Archives Photogrammetry*, vol. 21, no. 3, 1976.

[11] J. A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, 1981.

[12] J. Blair and B. Peyton, "An introduction to chordal graphs and clique trees," in *Graph Theory and Sparse Matrix Computations*, ser. IMA Volumes in Mathematics and its Applications, J. George, J. Gilbert, and J.-H. Liu, Eds. New York: Springer-Verlag, 1993, vol. 56, pp. 1–27.

[13] F. Dellaert and M. Kaess, "Square Root SAM: Simultaneous localization and mapping via square root information smoothing," *Intl. J. of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, Dec 2006.

[14] M. Yannakakis, "Computing the minimum fill-in is np-complete," vol. 2, no. 1, pp. 77–79, 1981.

[15] J. Liu, "Modification of the minimum-degree algorithm by multiple elimination," *ACM Trans. Math. Softw.*, vol. 11, no. 2, pp. 141–153, 1985.

[16] P. Amestoy, T. Davis, and I. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[17] A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 345–363, April 1973.

[18] R. Lipton and R. Tarjan, "Generalized nested dissection," *SIAM Journal on Applied Mathematics*, vol. 16, no. 2, pp. 346–358, 1979.

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

[20] L. Grigori, E. Boman, S. Donfack, and T. A. Davis, "Hypergraph-based unsymmetric nested dissection ordering for sparse lu factorization," University of Florida, Tech. Rep., April 2008.

[21] M. A. Paskin, "Thin junction tree filters for simultaneous localization and mapping," in *Intl. Joint Conf. on AI (IJCAI)*, 2003.

[22] U. Frese, "Efficient 6-dof slam with treemap as a generic backend," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2007.

[23] G. Grisetti, R. Kummerle, C. Stachniss, U. Frese, and C. Hertzberg, "Hierarchical optimization on manifolds for online 2d and 3d mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2010.

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

[25] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, August 2009.

[26] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, February 2001.

[27] H. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping (SLAM): Part I the essential algorithms," *Robotics & Automation Magazine*, Jun 2006.

[28] G. Karypis, V. Kumar, and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, pp. 96–129, 1998.

[29] T. A. Davis, "Multifrontal multithreaded rank-revealing sparse qr factorization," University of Florida, Tech. Rep., 2009.

[30] G. Grisetti, C. Stachniss, and W. Burgard, "Non-linear constraint network optimization for efficient map learning," *Trans. on Intelligent Transportation systems*, 2009.