Complexity Analysis and Efficient Measurement Selection Primitives for High-Rate Graph SLAM

Kristoffer M. Frey[1], Ted J. Steiner[2], and Jonathan P. How[1]

***Abstract*— Sparsity has been widely recognized as crucial for efficient optimization in graph-based SLAM. Because the sparsity and structure of the SLAM graph reflect the set of incorporated measurements, many methods for sparsification have been proposed in hopes of reducing computation. These methods often focus narrowly on reducing edge count without regard for structure at a global level. Such structurally-naïve techniques can fail to produce significant computational savings, even after aggressive pruning. In contrast, simple heuristics such as measurement decimation and keyframing are known to reliably produce significant computation reductions. To demonstrate why, we propose a quantitative metric called *elimination complexity* (EC) that bridges the existing analytic gap between graph structure and computation. EC quantifies the complexity of the primary computational bottleneck: the factorization step of a Gauss-Newton iteration. Using this metric, we show analytically that decimation and keyframing impose favorable global structures and therefore achieve computation reductions on the order of $r^2/9$ and r^3, respectively, where r is the pruning rate. We additionally present numerical results that show EC provides a good approximation of computation in both batch and incremental (iSAM2) optimization and demonstrate that pruning methods promoting global sparsity patterns outperform those that do not.**

I. Introduction

Graph-based approaches to the Simultaneous Localization and Mapping (SLAM) problem have gained popularity in recent years [1]–[6]. These problems are generally formulated as nonlinear least-squares (NLLS) optimizations over the set of robot poses and landmark positions. Though in certain cases non-iterative solutions exist [7], general SLAM problems are usually solved iteratively via a form of Gauss-Newton (GN). As recognized in [1], SLAM problems demonstrate a naturally sparsity that can be leveraged to significantly reduce computation with each GN iteration. This sparsity manifests itself as a large number of zero entries in the graph adjacency matrix, or equivalently a large number of "missing" edges relative to a complete graph. Nevertheless, this paper will emphasize that computation is a function of graph structure and not simply edge count.

Because measurements correspond to edges in the SLAM graph, the choice of measurements included in the optimization directly influences sparsity and graph structure. This has motivated numerous sophisticated measurement-selection and sparsification strategies for computation reduction in recent years [4], [6], [8], [9]. However, these methods narrowly focus on reducing edge count (i.e., treat all edges as equivalent from a computation perspective) [4], [6], [10] or enforcing locally-sparse structure [8], [9]. Because they do not curate global structure, these methods may only achieve limited computational savings, even after aggressive pruning.

To better connect graph structure to computation, this paper proposes *elimination complexity* (EC) as an approximation of the FLOP count associated with each GN iteration. As a function purely of graph structure, EC is independent of the particular numeric values of the optimization at any time. It is shown experimentally that EC trends linearly with computation time in both batch and incremental modes of operation.

We apply EC-based analysis to keyframing and decimation, two common measurement selection primitives in high-rate vision-based SLAM [5], [12]–[15]. These two heuristics are shown to produce *globally* efficient graph structures that achieve significant computational savings. In particular, it is demonstrated both analytically and numerically that decimation reduces EC at a rate of $\sim r^2/9$, while keyframing reduces EC at an increased rate of r^3. Here, r refers to "rate" of pruning in either case and is defined in more detail in the body of this paper. These insights can guide application of these techniques in systems with significant computation constraints and high required optimization rates.

As an analytic tool, EC provides a key link between graph structure and computation that has thus far been lacking from the discussion of measurement selection in SLAM. Furthermore, because it can be evaluated directly from the iSAM2 Bayes Tree [2], EC can potentially be used in an online sense to adapt computation management policies as the SLAM graph evolves.

A. Related Work

The most relevant discussion of computation in graph SLAM comes from the seminal paper by Dellaert et al. [1], and the following iSAM2 paper by Kaess et al. [2]. There, they connect GN optimization to sparse factorization of the corresponding system matrix, relating SLAM to linear algebra. They also make the important observation that choice of elimination ordering is crucial for efficient optimization. However, they do not explicitly connect the underlying Bayes Tree structure [2] to a metric of computational complexity. While the complexity of sparse matrix factorization is well-understood in the linear algebra literature [16]–[18], such

This work was supported by the Defense Research Advanced Projects Agency as part of the Fast Lightweight Autonomy program. The views expressed here those of the authors, and do not reflect the official views or policies of the Department of Defense or the U.S. Government.

[1]Kristoffer M. Frey and Jonathan P. How are with the Department of Aeronautics and Astronautics, MIT, Cambridge, MA 02139, USA kfrey@mit.edu jhow@mit.edu

[2]Ted J. Steiner is a Senior Member of the Technical Staff at Draper, Cambridge, MA 02139, USA tsteiner@draper.com

results have not been tailored for SLAM-specific solvers nor applied to measurement selection in SLAM.

Computation is often managed in SLAM by bounding the number of variables being estimated, usually via sliding-window approaches [5], [19] or by merging nearby pose variables [8], [12], [20]. Nonetheless, as measurements (and corresponding edges) are added the graph grows denser, and further steps must often be taken to promote sparsity.

Many methods of measurement selection exist in the literature with the specific goal of promoting sparsity, though none are guaranteed to produce computationally-efficient global graph structures. The Sparse Extended Information Filter [10] or the $\mathcal{L}_1$ regularization of [4] directly sparsify the information matrix, although without regard for resulting structure. Alternatively, [8], [9] make use of pre-selected sparse topologies to replace dense regions of the graph, and structural considerations are limited to local regions of the graph. Heuristics for applying these sparsification optimizations in online pose-graph SLAM were suggested in [8], but applying such techniques to the more general graphs common in landmark-SLAM is less straightforward. Following a general graph-theoretic approach, [6] provides max-spanning-tree-algorithms which select a fixed number of edges to retain or prune. Besides the fact that none of these approaches explicitly consider global structure from a computational standard, none achieve both practicality for computationally-constrained systems and direct applicability to general landmark-SLAM graphs. In contrast, keyframing and decimation policies are generally inexpensive and have been used extensively in real-time SLAM [12]–[14], [21].

II. Estimating Computation

The graph SLAM optimization problem with Gaussian factors can be expressed as a NLLS minimization [1]

$$\underset{X}{\operatorname{argmin}} \frac{1}{2}\sum_{i=0}^{m}(\mathbf{h}_i(X_i)-\mathbf{z}_i)^T\mathbf{\Sigma}_i^{-1}(\mathbf{h}_i(X_i)-\mathbf{z}_i). \quad (1)$$

Using iterative GN [22], optimization of (1) requires multiple solves of the (assumed positive-definite) *linearized* system

$$\mathbf{A}^T\mathbf{A}\mathbf{x}=\mathbf{A}^T\mathbf{b}, \quad (2)$$

The graph representations of the original (1) and linearized systems (2) are identical, meaning sparsity is preserved through linearization.

The number of GN iterations required for convergence depends fundamentally on the initialization points, measurement values (which are assumed random), and ultimately the actual environment [22]. Thus, it is difficult to predict if convergence will occur or how many iterations will be needed in general.

In contrast, linear systems of the form (2) are well-characterized, and is a function of the corresponding graph structure rather than numeric values. As the total GN computation is essentially the sum of these linear solves, computational savings at the linear level corresponds to multi-fold savings in total. Thus, quantifying the computation involved in (2) provides a link from structure to total computation.

The two fundamental steps involved in solving the linear system (2) are *elimination* and *back-substitution*. Elimination is equivalent to factorization of the system $\mathbf{A}^T\mathbf{A} = \mathbf{R}^T\mathbf{R}$ into the upper-triangular square matrix $\mathbf{R}$ [1]. Back-substitution refers to solving the remaining triangular system, and has complexity linear in the number of non-zero elements in $\mathbf{R}$. Because elimination of the $n \times n$ system $\mathbf{A}^T\mathbf{A}$ carries a worst-case $\mathcal{O}(n^3)$ complexity, it often represents the majority of computation in practice, even for incremental algorithms [2]. This motivates the use of *elimination complexity* as a representative measure of the inherent complexity represented by a graph.

A. Elimination Complexity

As noted by [1], [11], (2) can be solved efficiently by sparse factorization of $\mathbf{A}^T\mathbf{A}$. Sparse factorization follows the pattern of node elimination on the graph, illustrated in Figure 1. In node elimination, variable nodes are eliminated one-by-one from the graph, corresponding to the marginalization of the corresponding variable from the joint distribution over the remaining variables. When the i-th node is eliminated, it is removed from the elimination graph and edges are induced such that all its remaining neighbors form a fully-connected clique in elimination graph $G^{(i+1)}$. These new edges did not exist in the original graph and thus constitute *fill*, representing intermediate dependencies between variables induced by a particular elimination ordering. In the final upper-triangular $\mathbf{R}$ factor, these fill edges correspond to nonzero "filled-in" entries that were zero in the original system matrix $\mathbf{A}^T\mathbf{A}$.

As is well-known from the sparse linear algebra literature [1], [23], [24], fill depends on the chosen variable ordering $\mathcal{P}$. Though the solution itself is unaffected by ordering, different orderings can result in widely differing fill at each step of the optimization. However, determining the optimal (min-fill or min-FLOP) ordering is NP-complete [18], [24]. In practice, efficient heuristics such as Column-Approximate Min-Degree (`COLAMD`) [23] are widely used [1], [2].

For certain graph structures and elimination orders, fill-in can be catastrophic, destroying sparsity. For example, Duff [25] showed that as elimination progresses in random matrices with initially very few non-zero entries, the probability of the remaining elements being non-zero (due to fill-in) rapidly approaches 1.

Each step of node elimination corresponds to computing one step of the corresponding sparse QR or Cholesky factorization [1], [16], [17], and involves computation scaling with the dimension of the neighbors of the eliminated node. From this perspective, solving the full system (2) is decomposed into solving a series of smaller, dense sub-problems. The complexity of factorizing the full sparse system is then simply the sum of the complexities of the individual dense sub-problems.

Rose [16] showed that computing the $\mathbf{R}^T\mathbf{R}$ decomposition of a sparse $n \times n$ matrix can be performed in

$$\frac{1}{2}\sum_{i=1}^{n-1} d(i,\mathcal{P})(d(i,\mathcal{P})+3) \sim \sum_{i=1}^{n-1} d(i,\mathcal{P})^2 \quad (3)$$

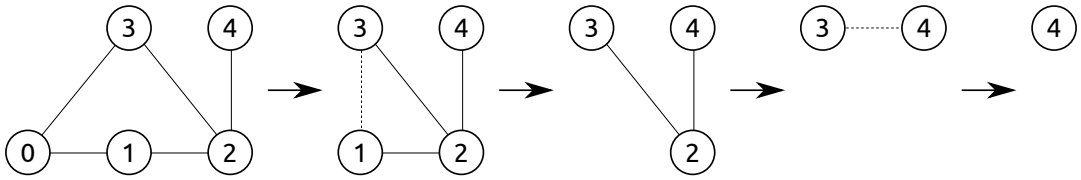

Fig. 1: The node elimination algorithm executed on a simple graph $G^{(0)}$. Nodes are eliminated in the order (0, 1, 2, 3, 4), producing a series of elimination graphs $G^{(i)}$. Induced edges are shown with dotted lines.

multiplications, where $d(i, \mathcal{P})$ refers to the degree of the i-th eliminated node in the elimination graph $G^{(i)}$ produced by ordering $\mathcal{P}$. The asymptotic form of (3) is equivalent to the approximate Cholesky FLOP count used by [18]. Note that for a fully dense matrix (corresponding to a fully-connected graph), $d(i) \sim n$ (the full system dimension) and sparse factorization approaches the full n^3 complexity for dense matrices.

In a conventional linear algebra approach, factorization of the system (2) occurs one row or column at a time. The corresponding graph $G^{(0)}$ includes n nodes, matching the scalar dimension of the system. However, (2) has additional block structure for typical SLAM systems [1]. In SLAM, the variables of interest are often multi-dimensional quantities such as positions and rotations, and measurements generally are defined on the level of these "macro-variables". In this case, it is the *block* sparsity pattern of (2) that is represented in the factor graph.

By applying ordering heuristics such as COLAMD [23] on the block structure directly, [1] showed improved performance and less fill. Following this observation, modern SLAM solvers such iSAM2 [2] apply elimination directly on the "macro-variables" of the factor graph. This motivates the definition of a version of (3) that accounts for the block structure of SLAM.

We define the *elimination complexity* (EC) $\mathcal{C}(G, \mathcal{P})$ of a factor graph G with variables $\mathcal{X}$ and ordering $\mathcal{P}$ as

$$\mathcal{C}(G, \mathcal{P}) \triangleq \sum_{i=1}^{|\mathcal{X}|} d_f(i)\Big(d_f(i) + d_s(i, G, \mathcal{P})\Big)^2 \qquad (4)$$

where $d_f(i, \mathcal{P})$ and $d_s(i, G, \mathcal{P})$ are the total scalar dimension of the i-th frontal variable $\mathbf{x}_f$ and its corresponding separator set $\mathbf{x}_s$ according to G and $\mathcal{P}$, respectively. Note that due to induced fill and the removal of previously-eliminated nodes, the dimension d_s of the i-th-eliminated variable will generally not match the neighborhood of that variable in the original graph G. Thus, computation of $\mathcal{C}$ in general requires simulation of the elimination process using ordering $\mathcal{P}$. Under scalar elimination, which corresponds to frontal variables of singular dimension $d_f(i) = 1$, the EC (4) reduces to the asymptotic form of (3).

Lemma 1: For a fixed elimination ordering $\mathcal{P}$ and graph G, let G^+ be constructed by adding an edge to G. Then, $\mathcal{C}(G, \mathcal{P}) \leq \mathcal{C}(G^+, \mathcal{P})$.

Proof: Let G^+ refer to the graph constructed by adding an edge to G. Following the elimination process described in Section II-A, the elimination neighborhood at each step i cannot be smaller for G^+ than for G

$$d_s(i, G, \mathcal{P}) \leq d_s(i, G^+, \mathcal{P}) \qquad \forall i \qquad (5)$$

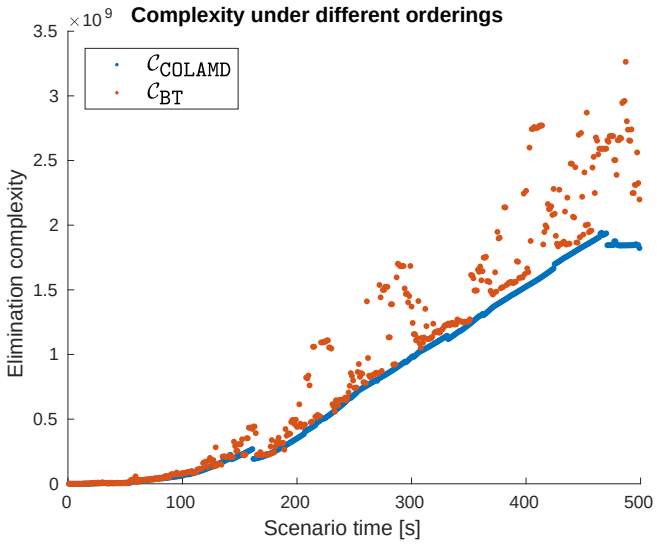

Fig. 2: Elimination complexity (EC) during a simulated SLAM experiment. EC grows over time as more variables and measurements are added. $\mathcal{C}_{\text{COLAMD}}$ is computed from the graph using a COLAMD variable ordering, and $\mathcal{C}_{\text{BT}}$ is computed directly from the Bayes Tree. iSAM2 uses a constrained form of COLAMD and only updates the ordering when nodes are re-eliminated. For this reason $\mathcal{C}_{\text{BT}}$ is often greater than $\mathcal{C}_{\text{COLAMD}}$.

Substituting this into the definitions of $\mathcal{C}(G, \mathcal{P})$ and $\mathcal{C}(G^+, \mathcal{P})$ results in

$$\mathcal{C}(G, \mathcal{P}) \leq \mathcal{C}(G^+, \mathcal{P}) \qquad (6)$$

■

Lemma 1 confirms the intuition that, for a fixed ordering, adding an edge to the graph cannot decrease elimination complexity. Equivalently, removing an edge cannot *increase* complexity, which confirms the conventional intuition applied by many existing measurement pruning techniques. Importantly, however, it will be demonstrated experimentally that arbitrary edge pruning (with no regard for global structure) can be surprisingly *ineffective* at reducing complexity, and thus EC-naive pruning techniques may only achieve limited computation reduction even after aggressive pruning.

B. Relationship to the iSAM2 Bayes Tree

Characterizing the update-time computation of incremental solvers such as iSAM2 [2] is generally difficult. An iSAM2 incremental update attempts to avoid re-elimination of the entire graph by representing the current solution and elimination process in a Bayes Tree structure. Because it

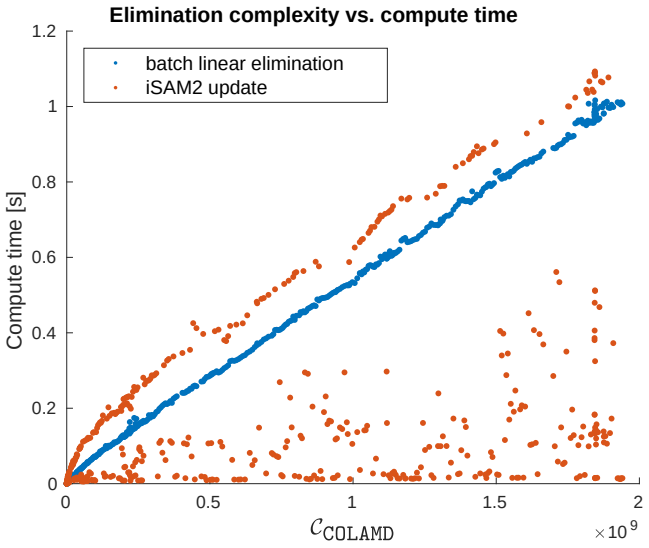

Fig. 3: Our elimination complexity metric, $\mathcal{C}_{\text{COLAMD}}$, is directly proportional to the batch elimination time of the linearized system (blue). In this experiment, iSAM2 (red) often performed the full nonlinear update without fully re-eliminating the graph, resulting in a greatly reduced compute time. However, the in-practice worst-case computation involves re-eliminating the entire graph and tracks nearly linearly with elimination complexity.

performs re-linearization and re-ordering only as needed, the computation required for each update depends on the the full update history and numerics of the problem. Nonetheless, because the Bayes Tree fundamentally represents the elimination process of the system, elimination complexity still serves as a useful predictor of computation.

EC can be computed directly from the Bayes Tree, although some subtleties must be acknowledged. Though generally guided by a COLAMD ordering, the implicit variable ordering represented in the Bayes Tree is semi-static and may not necessarily match $\mathcal{P}_{\text{COLAMD}}(G)$ at any time. Furthermore, cliques in the Bayes Tree represent a *multifrontal* factorization [26], in which multiple variables may in certain cases be eliminated together rather than sequentially, taking advantage of optimized dense matrix operations.

For a given iSAM2 instance represented by the Bayes Tree $\mathcal{T}_{\text{BT}}$, elimination complexity can be computed

$$\mathcal{C}_{\text{BT}}(\mathcal{T}_{\text{BT}}) \triangleq \sum_{C \in \mathcal{T}_{\text{BT}}} d_f(C)\Big(d_f(C) + d_s(C)\Big)^2 \qquad (7)$$

where $d_f(C)$ and $d_s(C)$ are the total scalar dimension of the frontal and separator variables in clique C, respectively.

C. Numerical Results

As a measure of graph computational complexity, EC $\mathcal{C}(G, \mathcal{P})$ should correlate linearly with the actual computation time. This was verified in an incremental SLAM simulation using iSAM2 and with the elimination complexity evaluated using the COLAMD heuristic. Figure 2 shows that both $\mathcal{C}_{\text{COLAMD}}$ and $\mathcal{C}_{\text{BT}}$ grow over time as more poses, landmarks, and measurements are added to the SLAM system. iSAM2 employs a lazy re-ordering scheme that attempts to maintain a near-COLAMD ordering, which often makes $\mathcal{C}_{\text{BT}}$ larger than $\mathcal{C}_{\text{COLAMD}}$.

Figure 3 shows that the batch elimination time of the linearized system at each step is proportional to $\mathcal{C}_{\text{COLAMD}}$. Furthermore, the incremental update time of iSAM2 displays an "in-practice" worst-case computation which also follows $\mathcal{C}_{\text{COLAMD}}$ linearly. Though iSAM2 often avoids re-eliminating much of the graph in order to produce relatively low-cost updates, in this experiment it still re-eliminated much or all of the graph the majority of the time. These results demonstrate that elimination complexity provides an approximation of computation even for incremental solvers. This motivates the use of EC as an analytic link between graph structure and computation, allowing for quantitative evaluation of measurement selection strategies.

III. Measurement Selection Primitives

Armed with a measure of complexity relating graph structure to computation, we can now assess measurement selection strategies analytically for their affects on computation. We will focus on two of the most ubiquitous selection policies in use in high-rate SLAM: decimation and keyframing, particularly in the context of landmark-SLAM. First, we will define a general case of the landmark-SLAM graph and approximate the worst-case elimination complexity. Then we will examine both keyframing and decimation, providing quantitative estimates of computation reduction.

A. The Landmark-SLAM Graph

A typical landmark-SLAM problem is shown in Figure 4. Landmarks are represented as green nodes l_j, and robot poses (sampled along the trajectory) as blue nodes x_i. As is typical of mobile robots, odometry measurements relate sequential poses, and could arise from inertial sensors or wheel encoders. Landmark observations connect the robot's pose at a particular moment in time to a particular landmark, and could represent measurements taken from LIDAR, ranging data, or a camera image.

Due to real-world sensor limitations, most landmarks will only be observable from a small subset of poses. However, the following analysis will assume the worst-case (from a sparsity perspective) – that every landmark is observed from every pose, as illustrated in Figure 4. As is standard in SLAM approaches [5], [27], [28], it is assumed that landmark positions are uncorrelated *a priori*, and thus no landmark-landmark edges exist in the graph. Furthermore, to simplify the following discussion, direct pose-pose loop closures are not explicitly taken into account here. Nonetheless, generalizing the results here to allow for this form of loop closure is straightforward.

Let there be n_l landmarks and n_x poses in landmark-SLAM graph G. The elimination complexity of this form of landmark-SLAM can be estimated by simulating the elimination process using a heuristic landmark-then-pose

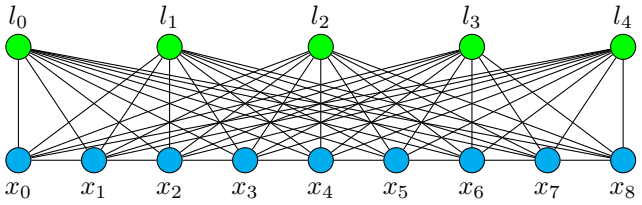

Fig. 4: A small landmark-SLAM example, in which every landmark l_j is observed from every pose x_i. Odometry constraints (e.g., from inertial sensors or wheel encoders) connect consecutive poses. Assuming no direct pose-pose loop closures or direct landmark-landmark correlations are allowed (as is typical of visual-inertial odometry systems), this is the *worst-case* graph from a sparsity perspective.

ordering $\bar{\mathcal{P}}$. In practice, the *de facto* `COLAMD` ordering is based on a min-degree heuristic, and will depend on the realized graph structure, which in turn depends on sensor limitations, the distribution of landmarks in the environment, and the robot's trajectory. However, by analyzing the worst-case graph in which all landmarks are observed by all poses, and assume that $n_l > n_x$, then $\bar{\mathcal{P}} = \mathcal{P}_{\texttt{min degree}} \approx \mathcal{P}_{\texttt{COLAMD}}$. Thus, it follows that

$$\mathcal{C}(G, \bar{\mathcal{P}}) \sim (d_l n_l + d_x n_x)(d_x n_x)^2 \tag{8}$$

where d_x and d_l are the scalar dimensions of the pose and landmark variables, respectively.

To derive (8), note that eliminating each of the n_l landmarks incurs a complexity of $d_l(d_l + n_x d_x)^2$, as each landmark is adjacent to all n_x poses. Because the landmark nodes are non-adjacent in G, fill is only induced between pose nodes, leaving a fully-connected clique of the n_x pose nodes. This fully-connected clique then can be eliminated in $\mathcal{O}(d_x^3 n_x^3)$ operations, producing the total asymptotic operation count shown in (8).

Also note that the inherent sparsity of SLAM has already produced a significant savings here compared to the $\mathcal{O}((d_x n_x + d_l n_l)^3)$ bound which would be achieved for a fully-dense system. Nonetheless, computation is still cubic with the number of poses n_x, and thus reducing the number of pose nodes promises to reduce computation significantly.

B. Keyframing

Keyframing approaches aim to do exactly that. By selecting only a subset of measurement frames to incorporate in the SLAM system, the number of included poses is reduced, having a cubic impact on computation. Here, a *frame* can refer to an image frame in a video stream, or more generally any set of measurements produced at the same time and corresponding to a single robot pose.

Letting $G_k(r)$ refer to the graph produced from G by keeping only n_x/r pose nodes, it is easy to see that

$$\mathcal{C}(G_k(r), \bar{\mathcal{P}}) \sim \left(d_l n_l + d_x \frac{n_x}{r}\right)\left(d_x \frac{n_x}{r}\right)^2 \tag{9}$$

Compared to (8), (9) shows an asymptotic complexity reduction of between r^2 and r^3. It will be demonstrated numerically that this in practice can be much closer to r^3.

Algorithm 1 Decimation

```
procedure TestDecimate(Obs, r)
    i ← Obs.PoseIdx
    k ← Offset[Obs.LandmarkIdx]
    if i mod r = k then
        Keep Obs
    else
        Discard Obs
```

As noted by Ila et al. [12], reducing the number of redundant poses in the SLAM system has the added benefit of improving estimator consistency, because in GN approaches each pose corresponds to a first-order noise propagation. All in all, this makes keyframing an ideal computation-reduction strategy if intermediate pose estimates are not of direct interest.

C. Decimation

In contrast to keyframing, *decimation* can be applied per measurement, and is particularly useful if the set of poses in the graph is fixed – i.e., if estimation of the pose at each frame is of direct interest. Algorithm 1 defines the decimation used here. By taking every r-th observation of each landmark in a *non-aligned* fashion, decimated graphs demonstrate a pattern of offset partitioning as shown in Figure 5. Here r is the *decimation rate* and k_j refers to the offset associated with a particular landmark j. Usually, the decimation offset k_j is determined by the offset of the first observation available for landmark j. For a landmark observed by a set of consecutive frames, this ensures that the maximum number of observations are accepted under the decimation constraint, and allows for landmark measurements to be included in the optimization as early as possible.

This partitioning structure is shown in a more general form in Figure 6. Pose and landmark nodes are divided into r disjoint subsets, Π_k and $\mathcal{L}_k$, respectively, with $k \in \{0, 1, \ldots, r-1\}$. As before, a "worst-case" approach is taken from a sparsity perspective, meaning that we assume each landmark $j \in \mathcal{L}_k$ is adjacent to each pose $i \in \Pi_k$ for a given k. Additionally, to simplify the analysis, we assume each pose $i \in \Pi_k$ is adjacent to all poses in $\Pi_{k-1} \cup \Pi_k \cup \Pi_{k+1}$. By construction, for each k, $|\Pi_k| \leq n_x/r$ (with equality if r divides n_x evenly).

Applying the elimination ordering $\bar{\mathcal{P}}$ to this graph $G_{\texttt{dec}}$ yields

$$\mathcal{C}(G_{\texttt{dec}}, \bar{\mathcal{P}}) \sim (n_l d_l + 9 n_x d_x)\left(d_x \frac{n_x}{r}\right)^2 \tag{10}$$

To obtain (10), after $\bar{\mathcal{P}}$ we eliminate the n_l landmarks first, involving a total of $n_l d_l (d_l + n_x d_x / r)^2 \sim n_l d_l (n_x d_x)^2 / r^2$ operations. This leaves each Π_k as a fully-connected clique, with additional connections to its neighbors Π_{k-1} and Π_{k+1}. Eliminating each of these cliques one-by-one involves com-

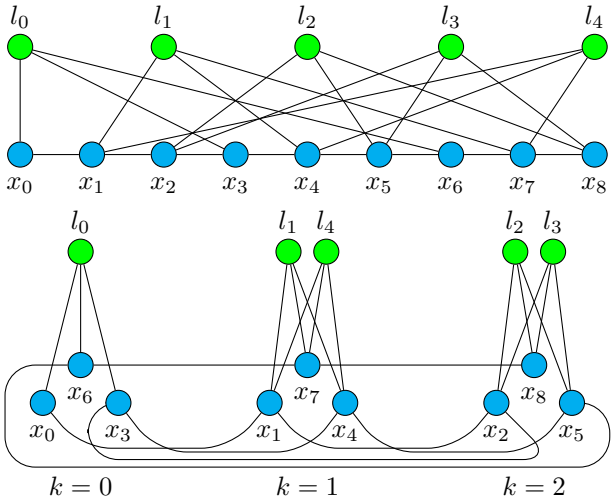

Fig. 5: [top] A small landmark-SLAM example after observation decimation with $r = 3$. Each landmark track is associated with a particular offset $k \in \{0, 1, \ldots, r-1\}$, which together with the rate r defines which observations are accepted. In this example, l_0 has offset 0, l_1 and l_4 have offset 1, and l_2 and l_3 have offset 2. [bottom] The same graph, re-drawn to highlight the partitioned structure produced by decimation. Note that these r subgraphs have limited interconnectivity, only arising from the odometry edges.

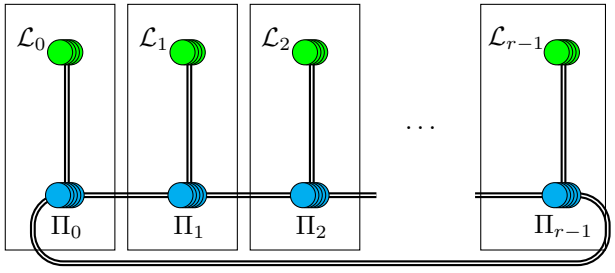

Fig. 6: The general partitioning structure $G_{\texttt{dec}}$ producing by decimation.

putation upper-bounded by

$$\sum_{k=0}^{r-1} \sum_{i=1}^{|\Pi_k|} d_x \left(d_x + 3 d_x \frac{n_x}{r} \right)^2 \sim r \frac{n_x}{r} d_x \left(3 d_x \frac{n_x}{r} \right)^2 \quad (11)$$

$$= \frac{9}{r^2} (d_x n_x)^3 \quad (12)$$

and adding this computation to the previous step gives (10).

Thus, we see that decimation produces a partitioned superstructure which reduces graph complexity asymptotically by $r^2/9$ compared to (8). Comparing to keyframing (9), decimation is clearly less effective at reducing computation. Nonetheless, decimation can have other advantages, particularly if estimating the full trajectory (including intermediate poses) is of direct interest. A full comparison of keyframing and decimation is outside the scope of this paper.

The experimental results in the next section emphasize that it is the unique partitioned structure shown in Figure 6, rather than simply the reduction in number of measurements, which produces this computational savings.

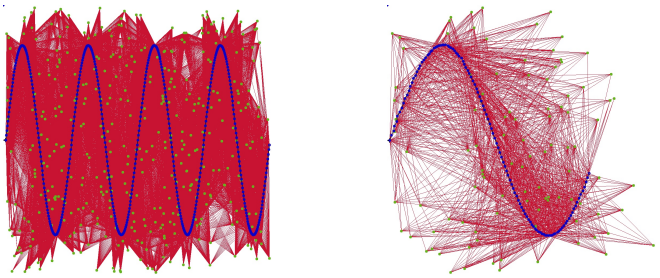

Fig. 7: Simulated visual-odometry dataset. [left] Top-down view of the full trajectory and measurements. The robot trajectory is shown as the series of blue pose nodes, and landmarks are shown in green. The robot moves in a sinusoidal pattern along a plane, starting from the left and moving right. Red edges indicate the full set of monocular vision observations. [right] A partially-completed trajectory, with landmark observations decimated at a rate of $r = 4$.

D. Experimental Results

A suite of simulation experiments were performed to verify the analytic results discussed above in a full 3D, incremental visual SLAM setting. In the simulation, a robot drives a sinusoidal trajectory, observing nearby landmarks according via a monocular visual sensor, using a standard pinhole camera model [27]. At each step of the simulation, a new pose node is added to the graph, and newly-triangulated landmarks are added to the graph. Because of the underrank nature of monocular measurements, landmarks are not initialized (i.e. added to the graph) until they have been observed a minimum number of times. Poses are represented as elements of SE(3), and landmarks as Cartesian points in $\mathbb{R}^3$.

The full trajectory, landmark distribution, and set of available landmark observations are shown in Figure 7. As this simulation accounts for realistic sensor limitations, any particular landmark is only observed by a subset of robot poses. This means that the complexity results derived in Section III, which assume all landmarks are observable from all poses, will be significant overestimates of the realized complexity $\mathcal{C}_{\texttt{COLAMD}}$.

Several pruning strategies are evaluated here, all parameterized by pruning rate r.

- `rand`: Random pruning, parameterized to remove a similar number of observations to the other methods.
- `tgreedy`: Greedy algorithm of [6] which attempts to maximize the number of spanning trees in the graph.
- `kf`: Simple keyframing strategy which represents only every r-th timestep in the optimization.
- `dec`: Decimation strategy described in Section III-C, always accepting the first available observation of each landmark.

As a comparison, results with no pruning are shown as `full`. In all cases, only visual observations (pose-landmark edges) are considered for pruning.

The goal of these experiments is to the verify the complexity reduction estimates derived in the previous section. As expected, the predicted complexity reduction of both keyframing and decimation methods are well-supported by

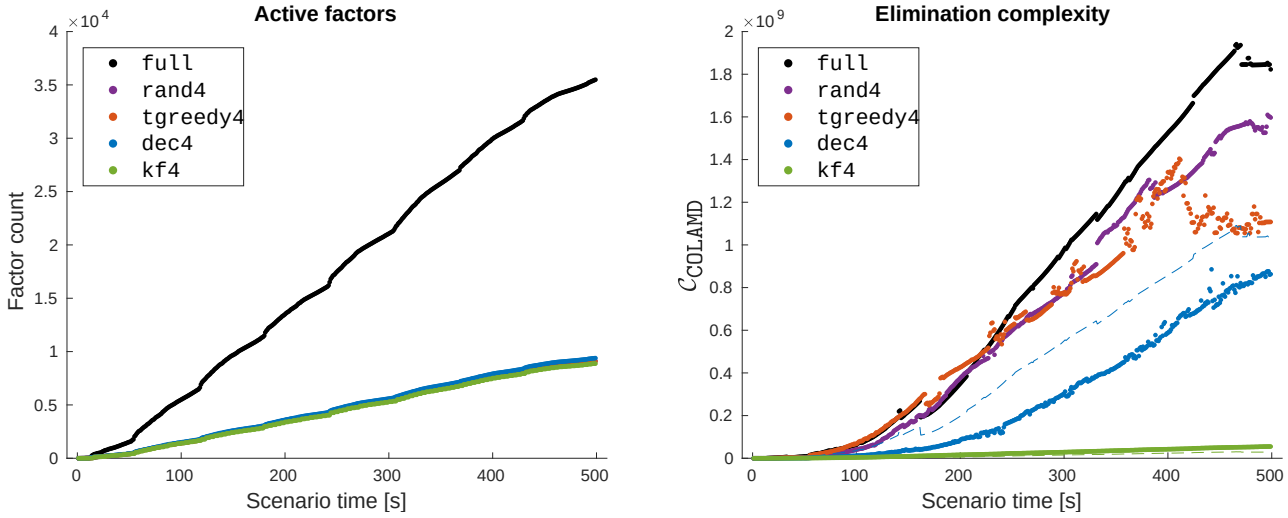

Fig. 8: Elimination complexity during a simulated SLAM experiment with $r = 4$. [left] All three pruning methods maintain a similar number of factors in the graph. [right] In all cases, elimination complexity grows over time. Though the three pruned estimators maintain a similar number of measurements, they have significantly-varying elimination complexities. Because it prunes without any regard for global structure, `rand` produces underwhelming complexity reduction compared to `dec` or `kf`. The dashed lines demonstrate the predicted complexities for `dec` and `kf`, based on scalings of `full` derived from (10) and (9). The actual complexities match these predictions well, verifying the method of analysis presented in this paper.

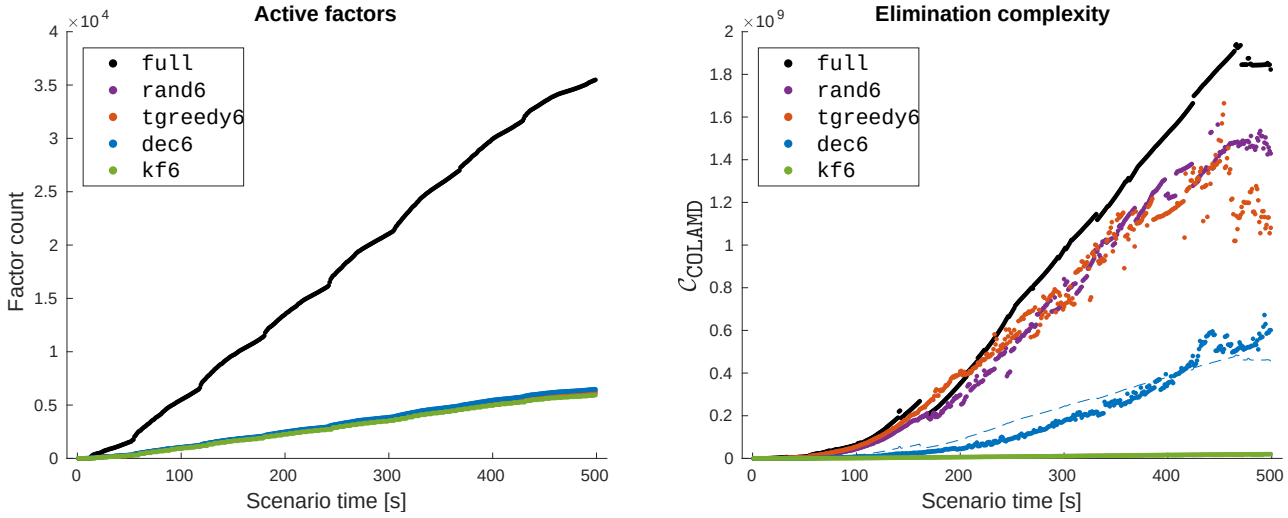

Fig. 9: Elimination complexity during a simulated SLAM experiment with $r = 6$. The trends here are very similar to those demonstrated in Figure 8.

the experimental results shown in Figures 8 and 9. In each case, the plotted dashed lines represent a complexity prediction produced with a scaling of the elimination complexity of `full`. For keyframing this prediction is scaled by $1/r^3$, and for decimation by $9/r^2$, based on (9) and (10) respectively. As can be seen, the realized complexity follows this prediction well, verifying the method of analysis presented in Section III.

Furthermore, average iSAM2 update computation time is shown in Table I. As indicated there and in the EC plots in Figures 8 and 9, the computational savings achieved by `rand` and `tgreedy` are relatively small, and do not scale with increasing pruning. In contrast, `dec` and `kf` provide significant reduction at $r = 4$ and continue to improve at $r = 6$.

IV. Conclusions

Many existing measurement selection techniques focus narrowly on edge count reduction [4], [6], [10] or locally-sparse structure [8], [9]. By neglecting global structure, these methods can produce limited computation reduction even after very aggressive sparsification. This paper proposes the use of *elimination complexity* (EC) as a link between graph structure and computation, and demonstrates how simple heuristics like decimation and keyframing can produce dramatic computation savings.

As an analytic tool, the EC framework is used to predict asymptotic computation reduction scaling with $r^2/9$ and r^3

TABLE I: iSAM2 performance in simulation shown in Figure 7. Note that `rand` and `tgreedy` fail to improve mean iSAM2 update times significantly between $r = 4$ and $r = 6$, while `dec` and `kf` continue to reduce computation with increased pruning.

Method	Avg. iSAM2 update time [s]
`full`	0.205
`rand4`	0.154
`tgreedy4`	0.138
`dec4`	0.087
`kf4`	**0.023**
`rand6`	0.131
`tgreedy6`	0.137
`dec6`	0.062
`kf6`	**0.012**

for decimation and keyframing, respectively. These predictions are confirmed numerically, and shown to far outperform structurally naïve methods which remove the same number of edges. In addition to the fact that many sophisticated selection approaches can be computationally impractical for high-rate, realtime use, this demonstrates that they also may not be as effective at reducing computation as these much simpler policies.

Ultimately, this motivates the search for new measurement selection strategies which *efficiently* reduce EC, rather than focusing naively on edge count or local sparsification. As was seen here, EC itself can be used to evaluate existing strategies, and provides the link between graph structure and computation that has thus far been lacking from the SLAM measurement selection literature.

Because it can be evaluated from and updated alongside the iSAM2 Bayes Tree, EC can also be applicable for adaptation of computation-management policies online. For example, if a robot using a sliding-window or fixed-lag estimator [3], [5], [19] enters a new environment with relatively fewer landmarks, the active SLAM graph will decrease in complexity. By monitoring EC, the estimator could choose to increase the window length to fill its computation budget and ensure sufficient landmarks are actively tracked at all times.

ACKNOWLEDGMENT

The authors thank Dr. Kasra Khosoussi for the productive conversations and input over the course of this work.

REFERENCES

[1] Frank Dellaert and Michael Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *Int. J. of Robotics Research*, 25(12):1181–1203, 2006.

[2] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. iSAM2: incremental smoothing and mapping using the bayes tree. *Int. J. of Robotics Research*, 2011.

[3] Gabe Sibley, Larry Matthies, and Gaurav Sukhatme. Sliding window filter with application to planetary landing. *Journal of Field Robotics*, 27(5):587–608, 2010.

[4] Guoquan Huang, Michael Kaess, and John J Leonard. Consistent sparsification for graph optimization. In *Proc. European Conf. on Mobile Robots (ECMR)*, pages 150–157, 2013.

[5] Ted J Steiner, Robert D Truax, and Kristoffer Frey. A vision-aided inertial navigation system for agile high-speed flight in unmapped environments. In *Proc. IEEE Aerospace Conf.*, 2017.

[6] Kasra Khosoussi, Gaurav S Sukhatme, Shoudong Huang, and Gamini Dissanayake. Maximizing the weighted number of spanning trees: Near-t-optimal graphs. *arXiv preprint arXiv:1604.01116*, 2016.

[7] David M. Rosen, Luca Carlone, Afonso S. Bandeira, and John J. Leonard. SE-Sync: A certifiably correct algorithm for synchronization over the special Euclidean group, 2016.

[8] Nicholas Carlevaris-Bianco and Ryan M Eustice. Long-term simultaneous localization and mapping with generic linear constraint node removal. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1034–1041. IEEE, 2013.

[9] Mladen Mazuran, Wolfram Burgard, and Gian Diego Tipaldi. Nonlinear factor recovery for long-term SLAM. *Int. J. of Robotics Research*, 35(1-3):50–72, 2016.

[10] Gamini Dissanayake, Hugh Durrant-Whyte, and Tim Bailey. A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem. In *Proc. IEEE Conf. Robot. Autom. (ICRA)*, volume 2, pages 1009–1014. IEEE, 2000.

[11] Michael Kaess, Viorela Ila, Richard Roberts, and Frank Dellaert. The bayes tree: An algorithmic foundation for probabilistic robot mapping. In *Proc. Workshop on Alg. Foundations of Rob. (WAFR)*, pages 157–173. Springer, 2010.

[12] Viorela Ila, Josep M Porta, and Juan Andrade-Cetto. Information-based compact pose SLAM. *IEEE Trans. Robot.*, 26(1):78–93, 2010.

[13] John Stalbaum and Jae-bok Song. Keyframe and inlier selection for visual SLAM. In *Proc. Int. Conf. on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 391–396, 2013.

[14] Luca Carlone and Sertac Karaman. Attention and anticipation in fast visual-inertial navigation. In *Proc. IEEE Conf. Robot. Autom. (ICRA)*, pages 3886–3893, 2017.

[15] Raul Mur-Artal and Juan D Tardós. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Trans. Robot.*, 2017.

[16] Donald J Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. *Graph theory and computing*, 183, 1972.

[17] Pinar Heggernes and Pontus Matstoms. *Finding good column orderings for sparse QR factorization*. University of Linköping, Department of Mathematics, 1996.

[18] Robert Luce and Esmond G Ng. On the minimum FLOPs problem in the sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 35(1), 2014.

[19] Han-Pang Chiu, Stephen Williams, Frank Dellaert, Supun Samarasekera, and Rakesh Kumar. Robust vision-aided navigation using sliding-window factor graphs. In *Proc. IEEE Conf. Robot. Autom. (ICRA)*, pages 46–53, 2013.

[20] Hordur Johannsson, Michael Kaess, Maurice Fallon, and John J Leonard. Temporally scalable visual slam using a reduced pose graph. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 54–61. IEEE, 2013.

[21] Yue Wang, Rong Xiong, Qianshan Li, and Shoudong Huang. Kullback-Leibler divergence based graph pruning in robotic feature mapping. In *Proc. European Conf. on Mobile Robots (ECMR)*, pages 32–37, 2013.

[22] Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[23] Timothy A Davis, John R Gilbert, Stefan I Larimore, and Esmond G Ng. Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 30(3):377–380, 2004.

[24] Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

[25] Iain S Duff. On the number of nonzeros added when Gaussian elimination is performed on sparse random matrices. *Mathematics of Computation*, 28(125):219–230, 1974.

[26] Frank Dellaert, Alexander Kipp, and Peter Krauthausen. A multifrontal QR factorization approach to distributed inference applied to multirobot localization and mapping. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, Pittsburgh, PA, 2005.

[27] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *Proc. IEEE Conf. Robot. Autom. (ICRA)*, pages 3565–3572. IEEE, 2007.

[28] Agostino Martinelli. Vision and IMU data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination. *IEEE Trans. Robot.*, 28(1):44–60, 2012.