Pose Graph Optimization using Matrix Sketching

AA 273 Spring 2021, Prof. Mac Schwager

Rohan Sinha and Emiko Soroka

***Abstract*—State-of-the-art algorithms for the simultaneous localization and mapping (SLAM) problem in robotics favor a pose graph-based representation of the problem over more traditional hidden Markov models. Graph-based SLAM algorithms rely on a front-end for measurement processing and data association, and a back-end solver that optimizes a large nonlinear least-squares problem, typically using iterative methods such as Gauss-Newton. Fast back-end implementations are critical for real-time state estimation. To this end, we proposed approximating the solutions of the Gauss-Newton iterations using matrix sketching, a method of dimensionality reduction for fast randomized linear algebra. We applied row sampling to several benchmark problems but determined that due to the sparsity of the pose graph matrices and lack of redundant information, Gauss-Newton for pose graph optimization is not a good candidate for sketching methods.**

I. Introduction

Simultaneous localization and mapping (SLAM) is a fundamental problem in robotics, enabling autonomous robots to build maps of unknown environments and localize within them. This has given modern autonomous systems the ability to navigate through a priori unmapped environments. Algorithms for SLAM have a long history spanning several decades and can broadly be classified as either filter-based or graph-based [15, 6]. Many state-of-the-art algorithms favor a graph-based approach, as its framework is more modular and can be applied to complex problems like monocular camera-based SLAM [12] where direct depth/spatial measurements (such as those from a LiDAR) are not available, but relative information can be extracted between frames.

A. Filter-based SLAM

Filter-based algorithms cast the SLAM problem as a more standard state estimation procedure for a Markovian dynamical system [15]. The goal is to infer the map m and state trajectory $x_{1:T}$ from an initial position x_0 and a sequence of measurements $y_{1:T}$. This can be done using a Markovian forward model of the dynamics $p(x_T|x_{0:T-1}) = p(x_T|x_{T-1})$ and a measurement likelihood model $p(y_t|x_t, m)$. By augmenting the state of the forward model with a data structure for the map, such as an occupancy grid, a state estimation algorithm like the extended Kalman Filter (EKF) can be used to perform SLAM [15].

B. Graph-Based SLAM

When a direct measurement model is not available, graph-based SLAM offers a solution. These algorithms produce a posterior map estimate m and trajectory $x_{0:T}$ by processing the data in a three step pipeline. The first step is to preprocess the observations in a consistent fashion. For example, in computer vision this typically entails identifying visually distinctive regions in an image and building unique descriptors of these keypoints using algorithms like SIFT [11]. These descriptors are then used to recognize the same locations in other measurements.

The second stage is data association, in which a pose graph is constructed from the preprocessed data. Each vertex of the graph represents a measurement and an associated unknown pose of the robot. An edge is added between the vertices associated with poses x_i and x_j if a "virtual measurement" z_{ij} can be identified between them [6]. Continuing our computer vision example, we would find a consistent affine transformation between shared keypoints in the images. This transformation represents a noisy estimate of the relative motion of the robot [12].

In the third step, the SLAM backend applies an optimization routine over the pose graph to find the maximum likelihood estimate of the poses $x_{1:T}$ given the virtual measurements. This process, referred to as pose graph optimization (PGO) involves solving a nonlinear least squares (NLLS) problem. Efficient solvers are critical to solve the resulting large-scale optimization problem for real-time SLAM.

In this work, we investigate the application of randomized algorithms to speed up the pose graph optimization problem. In particular, we apply randomized matrix sketching algorithms to decrease the size of the least-squares subproblems solved by an iterative Gauss-Newton solver. We first review related work on PGO and application of randomized methods. Then we review how the PGO problem can be solved using Gauss-Newton and how matrix sketching algorithms can be applied to a NLLS problem. We benchmark the performance of these algorithms on several existing datasets and conclude that the sparse nature of the pose graph generally means that matrix sketching techniques do not achieve significant benefits. However, if the pose-graph is sufficiently dense, sketching algorithms can improve computational speed and improve the convergence behavior of the solver.

II. Literature Review

Grisetti et al. [6] provide an overview of PGO with emphasis on how it can be implemented in software. Their paper outlines the Gauss-Newton method of solving the problem: a relatively simple algorithm. While not focused on PGO, Pilanci et al. [14] propose a sketched variant of Gauss-Newton to provide fast computation for optimization over large datasets. These two papers motivated us to apply the sketched Gauss-Newton algorithm to the PGO problem.

Other common methods to solve PGO include stochastic gradient descent and Gauss-Seidel relaxation [2]. Interestingly, several papers discuss randomized Gauss-Seidel approaches, where portions of the data are sampled probabilistically, for solving linear least-squares problems [13]. This approach has been shown to converge in the linear case [1]. Though PGO is a nonlinear least-squares problem, the results suggest randomized Gauss-Seidel as another candidate to reduce the cost of graph-based SLAM.

A more complex method is factor descent. Vallvé et al [16] take advantage of structure in the SLAM problem to seek a solution based on sparsification. Keeping the problem sparse reduces the computational cost. However, this method is complex and more difficult to implement.

Finally, bundle adjustment is a procedure that combines the SLAM front and back-ends, solving the data association and graph optimization problems simultaneously. This is typically applied to visual SLAM problems, where one must determine the most likely camera poses for a set of images [4] and presents a direction for future work on randomized optimization algorithms in SLAM.

Due to time constraints, in this paper we focus on the Gauss-Newton method for pose graph optimization.

Computational Cost

To reduce the computational cost of the PGO algorithm, we apply matrix sketching to approximate the Gauss-Newton update. Carlone [2] established theoretical properties of the PGO problem that make it surprisingly well-behaved despite being a nonlinear, nonconvex optimization problem. In particular, when orientation measurements are available for the robot, these results establish a convergence domain for the PGO problem and show that convergence can be improved by scaling relative position measurements. Well-behaved convergence suggests that even with approximation errors introduced by sketching, the algorithm can still converge. More recently, a probabilistic bound on sketched Gauss-Newton was established relating the size of the sketched problem (which is always less than the size of the original problem) to the probability that the result is within some ϵ of the true NLLS solution [14].

III. Graph SLAM and Pose Graph Optimization

In this section we detail how graph based SLAM algorithms construct maps of their environments and formulate the pose graph optimization problem that we will apply matrix sketching to. Modern SLAM architectures generally consist of a front-end that abstracts away the measurement process and a measurement agnostic back-end that constructs the map through a process called Pose Graph Optimization [6]. This allows modular development of SLAM algorithms for different sensing modalities, as the back-end can remain unchanged regardless of whether the measurements are made through cameras (as in [12]), or through LiDAR range finders (as in [6]). The presentation in this section closely follows that in [6], though we formulate the problem in a manner more amenable to the sketching approach we seek to apply.

In graph-based SLAM algorithms, it is assumed that the robot has recorded data for $T > 0$ timesteps, yielding a dataset of measurements $y_1, \dots y_T$. To identify the map, the SLAM algorithm must estimate *a posteriori* the robot states $x_1, \dots, x_T$ that produced the measurements. Instead of using an arbitrary state variable and transition function, graph-based SLAM algorithms are constructed as general-purpose packages for mobile robots. Therefore, these algorithms represent the trajectory taken by robots using state variables called *poses*. In 2D, a pose $x \in \mathsf{R}^3$ is defined as

$$x_t = \begin{bmatrix} p_t^x \\ p_t^y \\ \theta_t \end{bmatrix} = \begin{bmatrix} p_t \\ \theta_t \end{bmatrix},$$

where p_t^x and p_t^y denote the position of the robot in the $x - y$ plane and θ_t is the heading angle of the robot.

The front-end of the SLAM system identifies correlations between measurements to construct the *pose graph*. In the pose graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, vertices ($\mathcal{V}$) represent a measurement and robot pose pair (y_t, x_t) that were recorded at the same timestep. The robot pose x_t is unknown and needs to be estimated by the SLAM back-end. Edges ($\mathcal{E}$) between the vertices are added in the data association step: if a "virtual measurement" can be identified between two vertices, an edge is placed between them [6]. In vision-based SLAM, for example, this identification typically entails identifying visually distinctive keypoints (i.e. "corners") in each image. The keypoints are then associated with a descriptor extracted from the measurements so that the same keypoints can be recognized in multiple measurements. When two images y_i and y_j share many identical keypoints, we can be confident that the images record roughly the same region of space. The data-association algorithm then adds an edge between the vertices $i,\ j$ associated with the measurements, while the front-end identifies the virtual measurement as the *relative transformation* $z_{ij} \in \mathsf{R}^3$ between the measurements and its associated information matrix $\Omega_{ij} \succ 0$.

The virtual measurement z_{ij} is modelled as normally distributed according to a mean function $\hat{z}(x_i, x_j)$ so that $z_{ij} \sim \mathcal{N}(\hat{z}(x_i, x_j), \Omega_{ij}^{-1})$. The relative transformation between two poses x_i, x_j is straightforward to model, establishing the mean function as

$$\hat{z}(x_i, x_j) = \begin{bmatrix} R_{ij}^\top (R_i^\top p_j - p_i) \\ \theta_j - \theta_j \end{bmatrix}, \tag{1}$$

where R_i and R_{ij} are the 2D rotation matrices associated with θ_i and $\theta_j - \theta_i$. As a result, the negative log-likelihood of a virtual measurement then satisfies (2):

$$\begin{aligned} -\log(p(z_{ij}|x_i, x_j)) &\propto F_{ij}(x_i, x_j), \\ F_{ij}(x_i, x_j) &:= (z_{ij} - \hat{z}(x_i, x_j))^\top \Omega_{ij} (z_{ij} - \hat{z}(x_i, x_j)). \end{aligned} \tag{2}$$

The back-end of the SLAM system is then tasked with finding the maximum likelihood estimate (MLE) of the poses in the graph. Using (2), the MLE problem can be written as the nonlinear least-squares problem (3):

$$\min_{x_{1:T}} \sum_{(i,j) \in \mathcal{E}} F_{ij}(x_i, x_j) := F(x_{1:T}). \tag{3}$$

Solving for a local minimum of the nonconvex problem (3) will solve the PGO problem. The most common method of

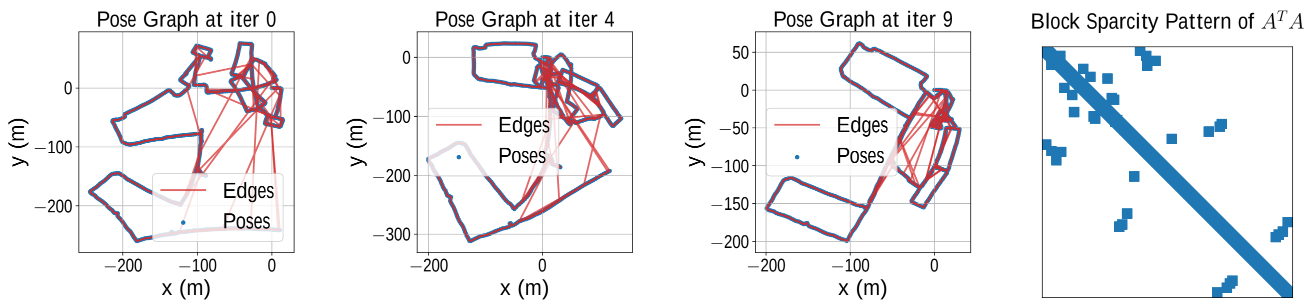

Fig. 1. Convergence of the Gauss-Newton algorithm on pose data of MIT's Killian Court [3], along with the sparsity pattern of the normal equations.

solving (3) is to apply the iterative Gauss-Newton algorithm [6, 10]. In Gauss-Newton, we iteratively construct quadratic approximations of the objective around candidate solutions $x^{(t)}$. After each iteration, $x^{(t+1)}$ is set to the minimizer of the quadratic approximation. To build this problem, we concatenate all the poses into a single decision variable $\mathbf{x} = [x_1^\top, \dots x_T^\top]^\top$ and optimize the quadratic approximation of the objective at $\mathbf{x}$ using the perturbation $\mathbf{\Delta x} = [\Delta x_1^\top, \dots, \Delta x_T^\top]^\top$. Then we have that the error

$$e_{ij}(x_i + \Delta x_i, x_j + \Delta x_j) = z_{ij} - \hat{z}_{ij}(x_i + \Delta x_i, x_j + \Delta x_j)$$

can be locally approximated using its first-order Taylor series

$$\underbrace{z_{ij} - \hat{z}(x_i, x_j)}_{y_{ij}} - \underbrace{\frac{\partial \hat{z}(x_i, x_j)}{\partial x_i}}_{A_{ij}} \Delta x_i - \underbrace{\frac{\partial \hat{z}(x_i, x_j)}{\partial x_j}}_{B_{ij}} \Delta x_j. \quad (4)$$

We can then use this expression to construct a convex quadratic approximation of the objective (3)

$$F_{ij}(\mathbf{x} + \mathbf{\Delta x}) \approx \|\Omega_{ij}^{1/2}(y_{ij} - A_{ij}\Delta x_i - B_{ij}\Delta x_j)\|_2^2, \quad (5)$$

where $\Omega_{ij}^{1/2}$ denotes the positive definite matrix square root of the information matrix Ω_{ij}. By summing the approximations for individual edges in (5), we get a quadratic approximation for the PGO objective (3). To do this, we define the matrix A with sparse block row structure

$$A = \begin{bmatrix} 0 & \dots & \Omega_{ij}^{1/2} A_{ij} & \dots & \Omega_{ij}^{1/2} B_{ij} & \dots & 0 \\ \vdots & & & \vdots & & \vdots & \\ 0 & \dots & \Omega_{ij}^{1/2} A_{ij} & \dots & \Omega_{ij}^{1/2} B_{ij} & \dots & 0 \end{bmatrix} \quad (6)$$

and the vector y as

$$y = \begin{bmatrix} \Omega_{ij}^{1/2} y_{ij}^\top & \dots & \Omega_{ij}^{1/2} y_{ij}^\top \end{bmatrix}^\top. \quad (7)$$

We emphasize that each block row of A and y are associated with a distinct pose graph edge $(i, j) \in \mathcal{E}$, but did not differentiate the indices to avoid cluttered notation. Then the local approximation to the PGO problem is given by the classic least-squares problem

$$\min_{\Delta x} \|A\mathbf{\Delta x} - y\|_2^2 \approx F(\mathbf{x} + \mathbf{\Delta x}). \quad (8)$$

The Gauss-Newton algorithm proceeds from an initial candidate solution $\mathbf{x}^{(0)}$ and iterates between constructing $A^{(k)}$ and $y^{(k)}$ by linearizing the objective around the current solution $\mathbf{x}^{(k)}$ and updating the the candidate solution using (8). The solution of (8) at iteration k can be found by solving the classical normal equations

$$(A^{(k)})^\top A^{(k)} \mathbf{\Delta x}^\star = (A^{(k)})^\top y^{(k)}, \quad (9)$$

after which we update the estimated poses as $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{\Delta x}^\star$. To develop fast solution methods for the PGO problem (3), it is of paramount importance to take advantage of the block sparsity structure in A and by extension of that in $(A^\top A)$. This is because robots map large areas that result in pose graphs consisting of tens of thousands of nodes and edges, making it intractable to use dense matrix algorithms to solve the PGO problem in real-time. Instead, to solve the normal equations, modern libraries such as `g2o` [10] favor sparse equation solvers – based on Cholesky decomposition – over methods such as conjugate gradient descent that are more sensitive to the conditioning of the problem. The convergence and sparsity pattern of the GN algorithm are shown in Figure 1.

IV. Overview of Matrix Sketching

The basic principle of matrix sketching is to reduce the size of a large matrix $A \in \mathsf{R}^{n \times d}$ before performing intensive computations by multiplying it with a randomly sampled sketching matrix S. The sketching matrix needs to be selected carefully to ensure the impact of the dimensionality reduction on the approximated result is low. In left-sketching, the pre-multiplication SA is computed for an $S \in \mathsf{R}^{m \times n}$. If $m << n$, then the sketched matrix $SA \in \mathsf{R}^{m \times d}$ is smaller, reducing computational complexity.

The sketching matrix S must have the property that

$$\mathsf{E}[S^\top S] = I \quad (10)$$

to guarantee that algebraic manipulations on the sketched matrix, SA, are identical in expectation to those on its unsketched counterpart, despite the reduction in size [8]. The sketching matrix can be constructed in a number of ways: it can sample more "important" entries of A more frequently, select rows from a uniformly random distribution, or even generate a Gaussian mixture of the columns. We briefly discuss these techniques to inform our approach for the PGO problem.

The simplest method of generating S is to generate rows or columns with one uniformly random nonzero entry of either -1 or 1. This is equivalent to selecting random rows or columns of A. Figure 2 demonstrates this approach.

The entries of S can also be drawn from a Gaussian distribution ($S_{ij} \sim \mathcal{N}(0,1)$, scaled so $\mathsf{E}[S^\top S] = 1$). This is equivalent to "mixing" values from different columns of the matrix A. Another popular method is the fast Johnson-Lindenstrauss transform, which works well if some columns of A are more "important" than others. The FJLT sketching matrix preconditions A by rescaling its columns; they can then be uniformly sampled with better performance [5].

Finally, row score sampling computes the row importance scores of A, then samples the rows with a probability proportional to their importance score. Let A^i be the ith row of A. Then the row score p_i is defined as

$$p_i = \|A^i\|_2^2. \tag{11}$$

The normalized row scores are then used to form a categorical probability distribution over the rows A^i. S is created by sampling from this distribution: if A^i contains all zeros, it will not be sampled, whereas a row with large norm $\|A^i\|_2^2$ is more likely to appear in SA. A visualization of this approach is seen in Figure 2.

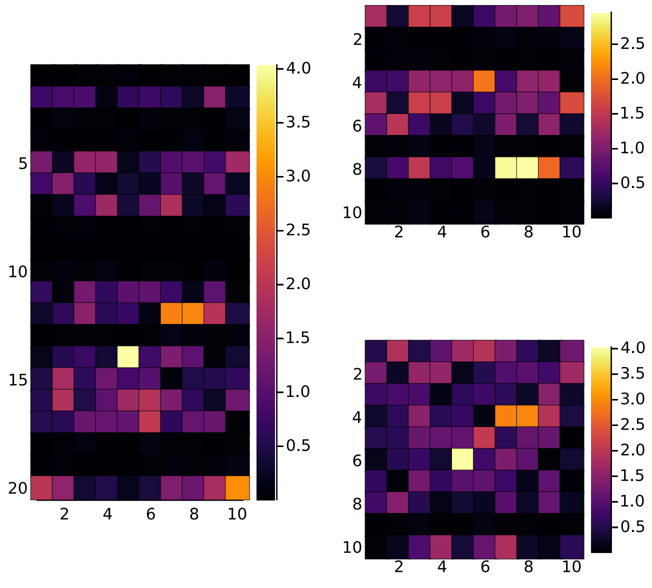

Fig. 2. A large matrix (left) with some "unimportant" rows ($\|A^{(i)}\|_2^2$ close to 0) is uniformly randomly sampled (yielding the matrix on the top right) and sampled using row scores (yielding the matrix on the bottom right).

When A is dense, Gaussian or FJLT sketching methods can be applied. However, if A is sparse, a sampling method is preferred.

A. Selecting a sketching method for PGO

Matrix sketching algorithms generally perform well when there is redundant information in a matrix: for example, in a large least-squares regression problem there are many more rows of data than columns. When some rows are more important, row score sampling reduces the likelihood that important information will be missing from the sketched matrix. However, if there is very little extra information, the sketched matrix cannot be made smaller and still approximate the true matrix. We noted in section III that the block sparsity structure in the PGO problem allows us to apply fast solution methods for the Gauss-Newton subproblems. Therefore, we apply row-importance sampling in our sketching algorithm for this problem, as Guassian sketching loses the sparse structure and naive row-sampling is likely to overlook the few denser rows in the PGO problem structure.

V. Applying Matrix Sketching to PGO

As described in section III, each Gauss-Newton step for a pose graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ requires solving a set of normal equations in $d = l|\mathcal{V}|$ decision variables for $n = l|\mathcal{E}|$ data samples, where l is the dimension of the poses (i.e., the matrix in (8) is $A \in \mathsf{R}^{n\times d}$). The complexity of inverting an $d \times d$ matrix (without taking advantage of any special structure) is $\mathcal{O}(d^3)$. Therefore, each iteration of the base Gauss-Newton algorithm, requiring us to compute the solution to the normal equations as

$$\boldsymbol{\Delta}\mathbf{x}^\star = (A^\top A)^{-1} A^\top y,$$

which has a time complexity of $\mathcal{O}(nd^2 + d^3)$. Initially, we hypothesized that we could apply right sketching to reduce the size of the Gauss-Newton inverse. This would reduce the computational cost of each iteration by updating a subset of the decision variables. However such (right) sketched approximate updates resulted in slower convergence and an increased failure rate (divergence of the objective). Therefore, we used a left-sketching method using row-importance sampling as outlined in IV instead, limiting the computational gains to graphs with large numbers of edges (many rows).

A. Implementation

We modified an existing implementation [7] of the algorithm described in section III and [6] as our baseline. This implementation is written in Python, allowing easy access to the optimization code.

To apply left-sketching to the Gauss-Newton subproblem (8) outlined in section III, we sample a sketching matrix S at each iteration and approximately solve the subproblem by minimizing

$$f(\boldsymbol{\Delta}\mathbf{x}) = \|S A \boldsymbol{\Delta}\mathbf{x} - Sy\|_2^2 \tag{12}$$

to find the pose update $\boldsymbol{\Delta}\mathbf{x}^\star$. The normal equations for the sketched least-squares problem (12) are then given as

$$A^\top S^\top S A \boldsymbol{\Delta}\mathbf{x} = A^\top S^\top S y. \tag{13}$$

Using the defining property (10) that $\mathsf{E}(S^\top S) = I$, we can see that the sketched subproblem (12) solves the original normal equations (9) in expectation.

if A is $n \times d$, computing this solution in the general (non-sparse) case requires multiplying $A^\top A$ with cost $\mathcal{O}(nd^2)$ and inverting a $d \times d$ matrix with cost $\mathcal{O}(d^3)$. By applying left-sketching, reducing the number of rows in A to $m << n$, we therefore achieve a linear time speedup per iteration from $\mathcal{O}(d^3 + nd^2)$ to $\mathcal{O}(d^3 + md^2)$. The additional sparsity structure retained from the row-sampling procedure can further speed up the computation. This approach is discussed extensively in [14].

Unfortunately, we found that there is too little redundancy in the matrices produced by the SLAM front-end for sketching to be effective in the benchmark suite [3]. In associating data with features, the front-end must prunes less likely connections. The resulting pose graph matrices are sparse and nearly square: they contain very little extra information that can be removed via sketching before the sketched normal equations (13) become singular. The sizes of several public datasets are listed in Table I.

Dataset	2D/3D	# Edges	# Vertices	Edges/Vertices
MIT Killian Court	2D	827	808	1.024
Intel Research Lab	2D	1483	1228	1.207
Sphere	3D	8647	2200	3.930
Torus	3D	9058	5000	1.8096
Cube	3D	22236	8000	2.7795

TABLE I
COMPARISON OF GRAPH CONNECTEDNESS ON PUBLIC DATASETS [3].

B. Determining the Sketching Dimension

Left sketching is usually applied to an overdetermined system with many more rows than columns. Recent results ([14]) have provided a bound to guide selection of the sketching dimension m as a function of a high-probability bound on the error ϵ between a sketched unconstrained NLLS solution and the true optimum. For an unconstrained problem, the sketching dimension m (recall if A is $n \times d$, S is $m \times d$) is bounded by

$$m \geq \frac{c}{\epsilon^2} \min\{n, d\}.$$

The paper states c is a constant, while ϵ is a user-defined error tolerance. This bound depends on $\min\{n, d\}$, suggesting overdetermined ($n >> d$) or underdetermined ($n << d$) systems are most effectively sketched. Unfortunately, we found that in most of our SLAM PGO datasets n/d is closer to 1, indicating sketching will be less effective. These n/d ratios are the edge/vertex ratios listed in Table I.

The "Sphere", "Torus", and "Cube" synthetic datasets have more edges than vertices: we primarily worked with the sphere because it has the highest ratio. However, these datasets still cannot be sketched efficiently. While some of the edges represent incorrect correlations between poses, it possible that sampling will drop the correct correlation, leading to a Gauss-Newton iteration that fails to converge.

C. Rejection of Poor Approximations

To mitigate this challenge, we introduced a rejection procedure: if the current cost is much larger than the previous cost, the iteration is rejected. This prevents one poor sketch of A from losing the previous iterations' progress towards an optimal solution.

D. Row Score Sampling of A

Initially, we tested random sampling: rows of A are sampled uniformly randomly to construct SA. This is a simple and efficient way to sample a matrix, but performs poorly if the row scores (11) are not similar: conceptually, if all rows of A are equally "important", the approximation SA can be constructed by randomly choosing rows of A. No prior information about A is required.

To understand why A cannot be effectively randomly sampled, we plotted the row scores (11) of A for the sphere in Figure 3. The presence of the higher values suggests some rows are more important: random sampling risks losing this data. This makes intuitive sense, as uninformative edges in the pose graph likely have an information matrix Ω_{ij} with small eigenvalues, representing an unreliable measurement.

We therefore applied row score sampling, in which a categorical distribution is constructed using the scores $\|A^i\|_2^2$ for $i = 1, \ldots, n$. The rows to sample are drawn from this distribution without replacement, ensuring the "most important" rows are likely to be sampled once, while the least important rows are likely to be excluded. Figure 3 shows a histogram of the row scores of A and SA. Notice the scores closest to 0 in A are excluded in SA while the higher values appearing in A continue to appear in SA.

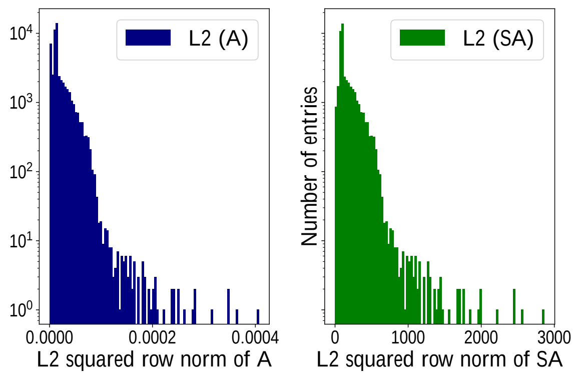

Fig. 3. Histogram of row scores for A and subsampled SA ($m = 0{:}85n$).

VI. Results

We benchmark our approach on the datasets in [3]. On "real-world" 2D datasets, the baseline Python GraphSLAM implementation [7] performed well. However, sketching failed to converge due to the lack of redundant data.

On the larger, more connected synthetic 3D data, we were able to apply row sampling with a very small size reduction $m = 0.95n$ but could not reduce the problem size enough to reduce the computational cost.

Interestingly, the baseline GraphSLAM implementation from [7] failed to converge on the sphere dataset. This is suspected to be a limitation of the simple GraphSLAM package as the more advanced g2o C++ package [9] was able to solve all of the test problems in Table I.

Despite the failure of the baseline GraphSLAM, applying sketching with no size reduction ($m = n$, equivalent to randomizing the rows of A and corresponding values in y) allowed our modified Gauss-Newton algorithm to converge in 40 iterations. We were able to achieve similar results with a very small size reduction: sampling 95% of the rows of A.

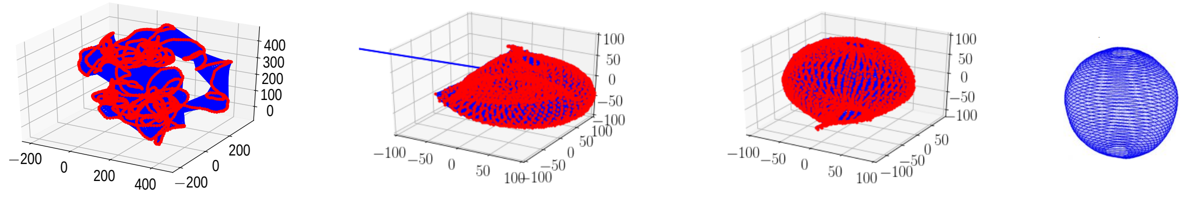

Fig. 4. From left to right: Initial pose-graph of the sphere dataset (red represents the poses, blue lines represent the edges). Sphere dataset after 40 iterations of randomized Gauss-Newton, sampling 90% of the rows of A. This sphere failed to fully converge. Sphere dataset after 40 iterations of randomized Gauss-Newton, sampling 95% of the rows of A. Actual shape of the sphere dataset (from [3]).

However, 90% and smaller sketch sizes failed to converge, with a few points remaining outside the sphere at each iteration as shown in Figure 4. This suggests that about 5% of the data in A is redundant: removing any more prevents the PGO problem from fully converging (Figure 4).

Unfortunately, the computational cost of sampling A overtakes any reduction gained by reducing the size of the least-squares problem. Matrix sketching is typically applied to problems where n can be reduced by large amounts: for example, $m = n/2$ or $m = n/4$. Our pose graph matrices are not good candidates for this procedure.

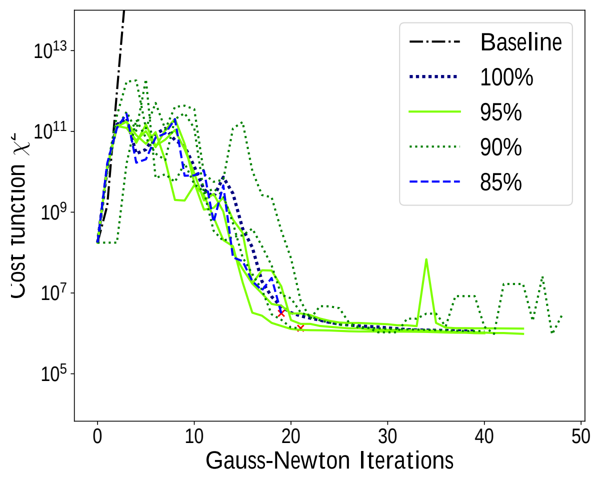

Fig. 5. Cost function for the baseline Python GraphSLAM on the 3D sphere dataset for several sketching sizes. Some runs were terminated early because the objective function failed to decrease or suffered a sudden increase.

We also tested the torus dataset, but were unable to get satisfactory results with any sampling size reduction (Figure 6).

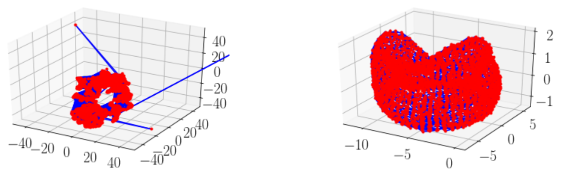

Fig. 6. Left: Torus dataset after 40 iterations of randomized Gauss=Newton, sampling 95% of the rows of A. Right: Torus dataset after 20 iterations of the baseline GraphSLAM with no sketching.

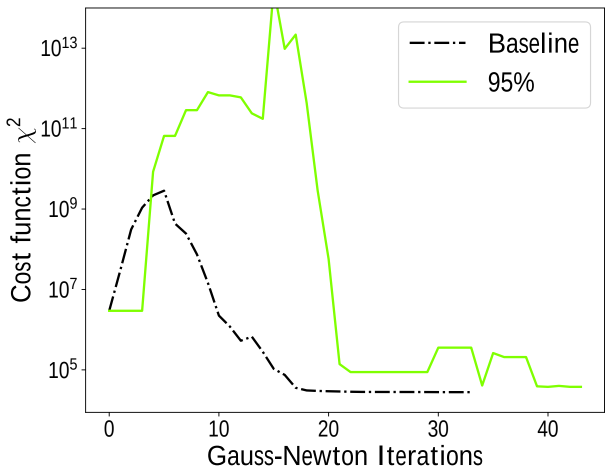

Fig. 7. Cost function for the baseline Python GraphSLAM on the 3D torus dataset for the baseline GraphSLAM and one sketching size.

VII. Conclusions

In this paper, we investigated the applicability of randomized linear algebra to reduce the computational cost of a SLAM back-end using the Gauss-Newton algorithm. However, we found that due to the lack of redundant information in the pose graph matrices generated by the SLAM front-end, the size of the matrix could not be reduced far enough to achieve an increase in computational speed. We analyzed the row scores of the pose graph matrices to determine the "importance" of each row for sampling purposes, and observed that while some rows have larger scores, about 95% of the rows must be sampled to achieve an acceptable result.

Because the SLAM front-end filters out spurious correlations, future work could include applying randomized methods to bundle adjustment, an approach in which the front-end and back-end operations are combined [4]. Alternative methods of speeding up the SLAM back-end by taking advantage of matrix sparsity and structure should also be considered.

VIII. Acknowledgements

This project was advised by Professor Mac Schwager for AA 273: State Estimation and Filtering. The application of sketching methods was inspired by Professor Mert Pilanci's class on randomized linear algebra: EE 270.

References

[1] Zhong-Zhi Bai, Lu Wang, and Wen-Ting Wu. "On convergence rate of the randomized Gauss-Seidel method". In: *Linear Algebra and its Applications* 611 (2021), pp. 237–252. ISSN: 0024-3795.

[2] Luca Carlone. "A convergence analysis for pose graph optimization via Gauss-Newton methods". In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 965–972.

[3] Luca Carlone et al. "Initialization techniques for 3D SLAM: A survey on rotation estimation and its use in pose graph optimization". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 4597–4604. DOI: 10.1109/ICRA.2015.7139836.

[4] Frank Dellaert. "Visual SLAM Tutorial: Bundle Adjustment". In: *CVPR 2014* (2014). URL: http://www.cs.cmu.edu/~kaess/vslam%5C_cvpr14/media/VSLAM-Tutorial-CVPR14-A13-BundleAdjustment-handout.pdf.

[5] Casper Benjamin Freksen. *An Introduction to Johnson-Lindenstrauss Transforms*. 2021. arXiv: 2103.00564 [cs.DS].

[6] Giorgio Grisetti et al. "A Tutorial on Graph-Based SLAM". In: *IEEE Intelligent Transportation Systems Magazine* (2010), pp. 31–43.

[7] Jeff Irion. *Python graphslam documentation*. 2019. URL: https://python-graphslam.readthedocs.io/en/stable/index.html (visited on 07/04/2021).

[8] Ravindran Kannan and Santosh Vempala. "Randomized algorithms in numerical linear algebra". In: *Acta Numerica* 26 (2017), pp. 95–135.

[9] R. Kümmerle et al. "G2o: A general framework for graph optimization". In: *2011 IEEE International Conference on Robotics and Automation* (2011), pp. 3607–3613.

[10] Rainer Kümmerle et al. "G2o: A general framework for graph optimization". In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 3607–3613. DOI: 10.1109/ICRA.2011.5979949.

[11] D.G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[12] Raúl Mur-Artal and Juan D. Tardós. "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras". In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262. DOI: 10.1109/TRO.2017.2705103.

[13] Yu-qi Niu and Bing Zheng. "A new randomized Gauss–Seidel method for solving linear least-squares problems". In: *Applied Mathematics Letters* 116 (2021), p. 107057. ISSN: 0893-9659.

[14] M. Pilanci and M. J. Wainwright. "Newton Sketch: A Linear-time Optimization Algorithm with Linear-Quadratic Convergence". In: *SIAM Jour. Opt.* 27.1 (Mar. 2017), pp. 205–245.

[15] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[16] Joan Vallvé, Joan Solà, and Juan Andrade-Cetto. "Pose-graph SLAM sparsification using factor descent". In: *Robotics and Autonomous Systems* 119 (2019), pp. 108–118. ISSN: 0921-8890.