Generalized Subgraph Preconditioners for Large-Scale Bundle Adjustment

Yong-Dian Jian, Doru C. Balcan, and Frank Dellaert

College of Computing, Georgia Institute of Technology
ydjian@gatech.edu, {dbalcan,dellaert}@cc.gatech.edu

Abstract. We propose the Generalized Subgraph Preconditioners (GSP) to solve large-scale bundle adjustment problems efficiently. In contrast with previous work using either direct or iterative methods alone, GSP combines their advantages and is significantly faster on large datasets. Similar to [12], the main idea is to identify a sub-problem (subgraph) that can be solved efficiently by direct methods and use its solution to build a preconditioner for the conjugate gradient method. The difference is that GSP is more general and leads to more effective preconditioners. When applied to the "*bal*" datasets [2], our method shows promising results.

1 Introduction

Large-scale visual modeling with Structure from Motion (SfM) algorithms is an important problem. Recently, systems capable of handling millions of images have been built to realize this task [1,13,23], enabling automated 3D model generation from unstructured internet photo collections.

Bundle adjustment is used to find the optimal estimates of camera poses and 3-D points [26]. Mathematically speaking, it refers to the problem of minimizing the total reprojection error of the 3-D points in the images. The classical strategy to solve this problem is to apply a damped Newton's method (e.g., Levenberg-Marquardt) and solve the reduced camera system by Cholesky factorization. However, this strategy does not scale well because the memory requirement of factorization methods grows quadratically with the number of variables in the worst case.

Several recent works suggest using iterative methods such as the conjugate gradient (CG) method to solve the linear systems arising in bundle adjustment, as its memory requirement grows only linearly with the number of variables. The convergence speed of the CG method depends on how *well conditioned* the original problem is [21]. Hence having a good preconditioner is crucial to make CG converge faster, yet most of the previous approaches [2,7,8,14] apply only standard preconditioning techniques, neglecting to exploit SfM-specific constraints.

In robotics, Dellaert et al. [12] proposed the Subgraph-Preconditioned Conjugate Gradients method (SPCG), which aims to combine the advantages of direct and iterative methods to solve 2-D Simultaneous Localization and Mapping (SLAM) problems. The main idea is to pick a subset of measurements that can be solved efficiently by direct methods, and use it to build a preconditioner for the CG method. They show that SPCG is superior to using either direct or iterative methods alone. However, for the bundle adjustment problem, whose graph structure is bipartite and highly unbalanced, SPCG may over-estimate the uncertainty of the variables and hence lead to unsatisfactory preconditioners.

In this paper, we propose the Generalized Subgraph Preconditioners (GSP) that adapt SPCG to solve large-scale bundle adjustment efficiently [15]. While SPCG simply picks a subgraph of the *Jacobian* factor graph, GSP operates on the *Hessian* factor graph which is more general and leads to more effective preconditioners. From this perspective, the problem of designing a good subgraph preconditioner is reduced to picking a subset of the Hessian factors that (1) can be solved efficiently by direct methods, and also (2) make the linear systems well-conditioned.

An important open question in [12] is how to pick a good subgraph. To this end, we introduce the ideas developed in the field of combinatorial preconditioners to build good subgraph preconditioners [6]. The insight is that a good subgraph should not only be sparse but also have small structural distortion (*stretch*) with respect to the original graph. Yet finding the optimal subgraph that satisfies the above criteria is computationally intractable for large graphs. Instead we propose a greedy algorithm to construct a family of subgraphs by incrementally adding edges to reduce stretch without inducing large cliques in the factorization phase.

This paper has three contributions: we (1) adapt the ideas of SPCG to the bundle adjustment problem, (2) propose GSP which generalizes SPCG and leads to more effective subgraph preconditioners, and (3) develop a greedy algorithm based on the ideas in combinatorial preconditioners to construct a family of subgraph preconditioners. We use the proposed method to solve large-scale datasets and have promising results.

2 Bundle Adjustment

2.1 Formulation

Here we review the bundle adjustment, whose goal is to jointly estimate the optimal camera parameters and 3-D structure by minimizing the total reprojection error. We define $\mathbf{X} = \{x_i\}_{i=1}^{M}$ as the camera parameters, $\mathbf{L} = \{l_j\}_{j=1}^{N}$ as the 3-D points, and $\mathbf{Z} = \{z_k\}_{k=1}^{K}$ as the measurements of the 3-D point l_{kj} in camera x_{ki}. We also define a function $h_k(x_{ki}, l_{kj})$ that projects a 3-D point to an image (see Figure 1). The goal of bundle adjustment is to find the optimal cameras $\mathbf{X}$ and 3-D points $\mathbf{L}$ that minimizes the total reprojection error

$$\sum_{k=1}^{K} \|h_k(x_{ki}, l_{kj}) - z_k\|^2. \tag{1}$$

Equation (1) is nonlinear and has no closed-form solution, but suppose we know some initial estimates of the cameras parameters and 3-D points, we can apply the first-order Taylor expansion to linearize Equation (1) as

$$\sum_{k=1}^{K} [h(x_{ki}, l_{kj}) + \frac{\partial h(x_{ki}, l_{kj})}{\partial x_{ki}} \delta x_{ki} + \frac{\partial h(x_{ki}, l_{kj})}{\partial l_{kj}} \delta l_{kj} - z_k]. \tag{2}$$

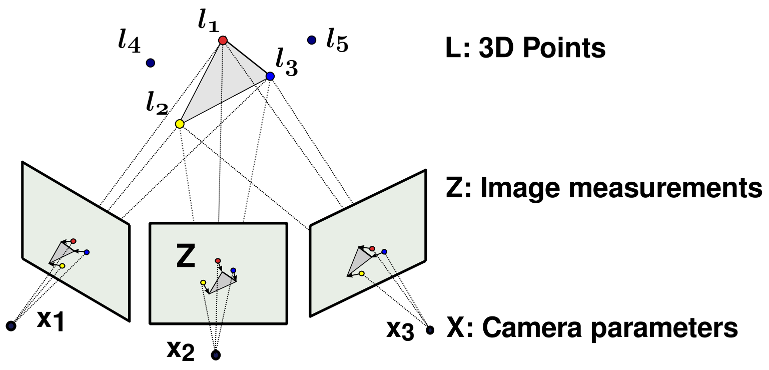


Fig. 1. The bundle adjustment problem

By setting the first-order derivative of the measurements in Equation (2) to zero, we can build a linear system

$$\mathbf{A}\boldsymbol{\theta} = \mathbf{b}, \tag{3}$$

where $\mathbf{A}$ is a sparse rectangular matrix containing the Jacobian of the measurements with respect to the cameras and 3-D points, $\boldsymbol{\theta}$ is a vector that concatenates all δx_i and δl_j, and $\mathbf{b}$ is a vector that concatenates the negative measurement errors. Then we solve Equation (3) and use its solution to update the current estimates. This process is repeated until convergence. We can see that solving bundle adjustment is equivalent to solving a sequence of linear systems. An alternative to the second step is to form and solve the *normal* equation

$$(\mathbf{A}^{\mathbf{T}}\mathbf{A})\boldsymbol{\theta} = \mathbf{A}^{\mathbf{T}}\mathbf{b}, \tag{4}$$

where $\mathbf{A}^{\mathbf{T}}\mathbf{A} \approx \mathbf{H}$ is a first-order approximation to the Hessian of the total reprojection error in Equation (1). Unfortunately, this method may not converge to the local minimum if the initial estimate is close to a saddle point. To to resolve this problem, one can solve a regularized linear system

$$(\mathbf{A}^{\mathbf{T}}\mathbf{A} + \lambda\mathbf{D})\boldsymbol{\theta} = \mathbf{A}^{\mathbf{T}}\mathbf{b}, \tag{5}$$

where λ is a non-negative scalar, and $\mathbf{D}$ can be an identity matrix or the diagonal of $\mathbf{A}^{\mathbf{T}}\mathbf{A}$. In bundle adjustment, the Levenberg-Marquardt algorithm is used to update the value of λ according to quality of the solution. Note that the least-square linear system corresponding to the normal equation (5) is

$$\begin{bmatrix} \mathbf{A} \\ \sqrt{\lambda\mathbf{D}} \end{bmatrix} \boldsymbol{\theta} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}. \tag{6}$$

2.2 Jacobian Factor Graph Representation

The bundle adjustment problem can also be considered as an inference problem on a factor graph. In particular, the sparse Jacobian matrix $\mathbf{A}$ in Equation (3) can be regarded

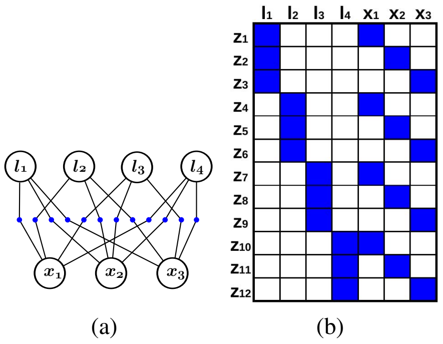


Fig. 2. A toy bundle adjustment problem with three cameras and four 3-D points. All of the 3-D points are observed by all of the cameras. (a) The Jacobian factor graph. The vertices denote the camera and the 3-D point variables. The blue dots are the factors, and each factor indicates the squared error term of a projection measurement. (b) The symbolic representation of the Jacobian matrix $\mathbf{A}$. Each row denotes one Jacobian factor, and each column indicates one variable.

as a *Jacobian* factor graph, where the vertices are the cameras and the 3-D points, and each factor denotes the squared error term (block row) of a measurement. Figure 2 illustrates the idea with a simple example. Suppose we define the likelihood of a factor as an exponential function of the negative squared error

$$P(z_k|x_{ki}, l_{kj}) \propto \exp\{-\frac{\|h(x_{ki}, l_{kj}) - z_k\|^2}{2\sigma^2}\}, \tag{7}$$

we can see that the maximum likelihood estimator of the factor graph is the minimizer of Equation (1), i.e.

$$\underset{\mathbf{X},\mathbf{L}}{\operatorname{argmax}} \prod_{k=1}^{K} P(z_k|x_{ki}, l_{kj}) = \underset{\mathbf{X},\mathbf{L}}{\operatorname{argmin}} \sum_{k=1}^{K} \|h(x_{ki}, l_{kj}) - z_k\|^2 \tag{8}$$

This connection provides a foundation to the subgraph preconditioners.

2.3 Direct Methods

There are two ways to solve linear systems and the first one is called direct methods. They work by factorizing the matrix to the product of an upper triangular matrix $\mathbf{R}$ and its transpose, followed by a backward and forward substitution step. For instance, we can use QR factorization to solve the linear least-square problem in Equation (3), and use Cholesky factorization to solve the normal equation in Equation (4) [25]. On factor graph, direct methods can be explained as a sequence of variable eliminations. Each time we eliminate a variable (vertex), we will instantiate a new factor connecting to all of its neighbors. After eliminating all of the variables, we will get an upper triangular matrix $\mathbf{R}$ as a result. The process is illustrated in Figure 3. The variable elimination ordering is very important to the efficiency of direct methods. Using a good ordering

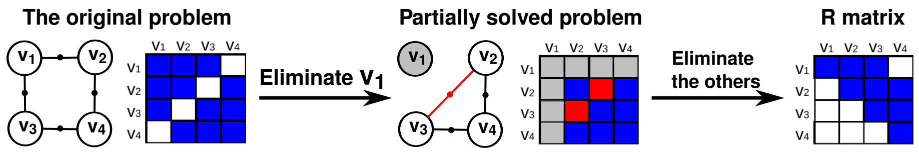


Fig. 3. An illustration of how direct methods factorize an $\mathbf{H}$ matrix into $\mathbf{R}^t\mathbf{R}$. Suppose we have a variable elimination ordering. On the factor graph, each time we eliminate a vertex (variable), we will introduce a new factor (red) connecting to all of its neighbors. After eliminating all of the vertices, we will get the factorized matrix $\mathbf{R}$.

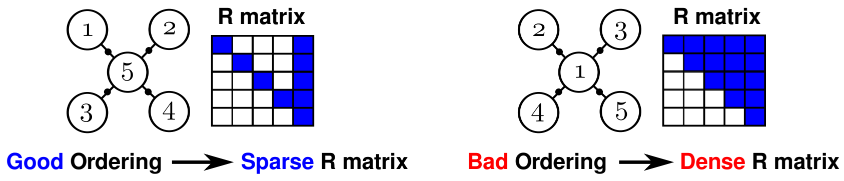


Fig. 4. An illustration of how the elimination ordering affects the sparsity of $\mathbf{R}$ matrix. Suppose we eliminate the leaf vertices first, the $\mathbf{R}$ matrix will be very sparse. Yet if we eliminate the center vertex first, it will introduce a clique over the remaining vertices, and hence the $\mathbf{R}$ matrix becomes very dense, which negatively affects the performance.

will result in a sparse $\mathbf{R}$ matrix and make the forward and backward substitutions more efficient. Figure 4 shows how the ordering affects the sparsity of the factorized matrix.

Using direct methods to solve bundle adjustment has been well-studied in the literature [16,17,26]. The common practice is to eliminate all 3-D points first, and use Cholesky factorization to solve the reduced camera system. Yet as shown in [2,8,14], this strategy only works well for small problems, but does not scale satisfactorily because (1) the cost of forming and storing the reduced camera systems is prohibitive for large problems, and (2) building the reduced camera system could destroy the sparse problem structure and hence make it harder to solve. Therefore direct methods cannot be directly applied to solve large-scale bundle adjustment without using hierarchical or incremental techniques [19,22].

2.4 Iterative Methods

The second way to solve linear systems is called iterative methods. They are better than direct methods for large problems because they involve only simple operations and require less memory, but they may suffer from slow convergence if the original problem is *ill-conditioned*.

The conjugate gradient (CG) method is the most efficient variant of iterative methods, but the convergence speed still depends on the condition number of the linear system, which is defined as the ratio of extreme eigenvalues of the matrix $\mathbf{A}^T\mathbf{A}$.

Several preconditioning techniques have been applied to make bundle adjustment well-conditioned. Agarwal et al. [2] examined the performance of several standard preconditioners and implementation strategies on large-scale datasets. Byröd and Åström

[7,8] proposed to use multi-scale and the block Jacobi preconditioners respectively. Jeong et al. [14] suggested using the band-diagonal of the reduced camera system as a preconditioner. Yet these methods are very generic: We show that by exploiting the problem structure of bundle adjustment we can obtain better preconditioners.

3 Combining the Best of Direct and Iterative Methods

3.1 Variable Reparameterization and Preconditioning

Re-parameterizing the variables can result in faster convergence for iterative methods. In the robot mapping and localization problem, Olson et al. [20] showed that if the robot poses are parameterized in the global coordinate system, it takes a long time to propagate the loop closure constraints through the graph, but suppose the robot poses are incrementally parameterized along the odometry chain, so that the new variables denote the difference between two consecutive poses, they show that it makes the stochastic gradient descent method converge faster. Generally speaking, this re-reparameterization can be considered as a linear transformation $\mathbf{R}$ between two domains.

Similarly, the preconditioned conjugate gradient method [21] also uses a preconditioner $\mathbf{R}$ to linearly re-parameterize the problem such that the condition number becomes smaller and it can converge faster. This point of view indicates that linear variable re-parametrization is essentially a preconditioning process.

3.2 Subgraph-Preconditioned Conjugate Gradient Method

Dellaert et al. [12] proposed the Subgraph-Preconditioned Conjugate Gradient (SPCG) method, which aims to combine the advantages of direct and iterative methods to solve 2-D Simultaneous Localization and Mapping (SLAM) problems. The main idea is to identify a sub-problem (subgraph) that can be solved efficiently by direct methods (e.g., a subgraph with small tree-width) and use it to build a preconditioner for the conjugate gradient method. They show that this technique is a better alternative to using either direct or iterative methods alone. Figure 5 illustrates the key steps of the algorithm.

Here we show how SPCG works in detail. Suppose we want to solve a linear system (Jacobian factor graph) as in Equation (6). We pick a subset of the rows (factors), and denote it as $(\mathbf{A}_1, \mathbf{b}_1)$, and denote the remaining rows as $(\mathbf{A}_2, \mathbf{b}_2)$. We can re-arrange the linear system in Equation (6) as

$$\begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix} \boldsymbol{\theta} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} . \tag{9}$$

After applying QR factorization to $\mathbf{A}_1$, we have $\mathbf{A}_1 = \mathbf{Q}_1 \mathbf{R}_1$. By left-multiplying the upper part with $\mathbf{Q}_1^T$, we get

$$\begin{bmatrix} \mathbf{R}_1 \\ \mathbf{A}_2 \end{bmatrix} \boldsymbol{\theta} = \begin{bmatrix} \mathbf{Q}_1^T \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} . \tag{10}$$

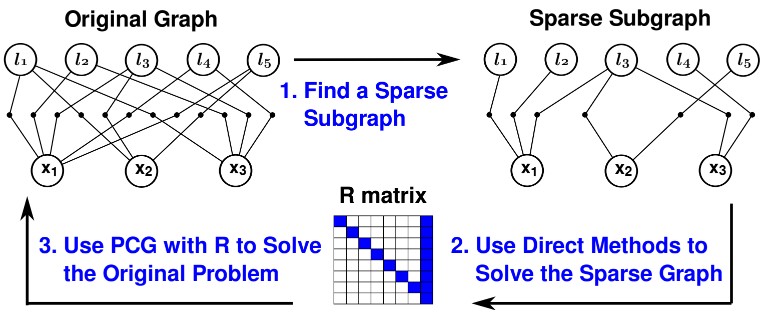

Original Graph
Sparse Subgraph
1. Find a Sparse Subgraph
R matrix
2. Use Direct Methods to Solve the Sparse Graph
3. Use PCG with R to Solve the Original Problem

Fig. 5. An illustration of the SPCG method. Suppose on the left is the original factor graph. SPCG has three main steps: (1) Pick a sparse subgraph out of the original one. (2) Use direct methods to factorize this sparse subgraph. This step is efficient because a good variable elimination ordering for a sparse graph is always available. (3) Use the $\mathbf{R}$ matrix of the subgraph as the preconditioner in the preconditioned conjugate gradient method to solve the original problem.

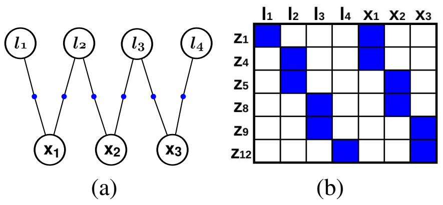


Fig. 6. An example that illustrates the SPCG technique. (a) The Jacobian factor graph that corresponds to a subset of the measurements (sub-problem) in Figure 2. (b) The symbolic matrix representation of the subgraph.

Suppose $\mathbf{c}_1 = \mathbf{Q}_1^T \mathbf{b}_1$ and $\bar{\boldsymbol{\theta}} = \mathbf{R}_1^{-1}\mathbf{c}_1$ is the optimal solution by considering only the upper part of Equation (9). Then by re-parameterizing $\mathbf{y} = \mathbf{R}_1(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})$, we have

$$\begin{bmatrix} \mathbf{I} \\ \mathbf{A}_2\mathbf{R}_1^{-1} \end{bmatrix} \mathbf{y} = \begin{bmatrix} \mathbf{0} \\ \mathbf{c}_2 \end{bmatrix}, \tag{11}$$

where $\mathbf{c}_2 = \mathbf{b}_2 - \mathbf{A}_2\mathbf{R}_1^{-1}\mathbf{c}_1$. Equation (11) couples the solution of the subgraph part ($\mathbf{R}_1^{-1}$) to precondition the remaining part. The intuition behind the re-parameterization is that we penalize the deviation of $\mathbf{y}$ from the subgraph solution $\bar{\boldsymbol{\theta}}$. Finally Equation (11) is solved by using the least-squares variant of the conjugate gradient method [4].

Figure 6 illustrates the SPCG technique with an example. Suppose we pick a spanning tree of the original graph as in Figures 6(a) and 6(b). We can use direct methods to factorize the spanning tree efficiently. Then we use the factorized matrix to precondition (re-parameterize) the original problem.

In addition, we also visualize the solutions obtained from the subgraph and the solutions from the original graph in Figure 7. We can see that although the solution of the subgraph is blurry and hence inferior to that of the original graph, we can use it to build a preconditioner to solve the original graph efficiently.

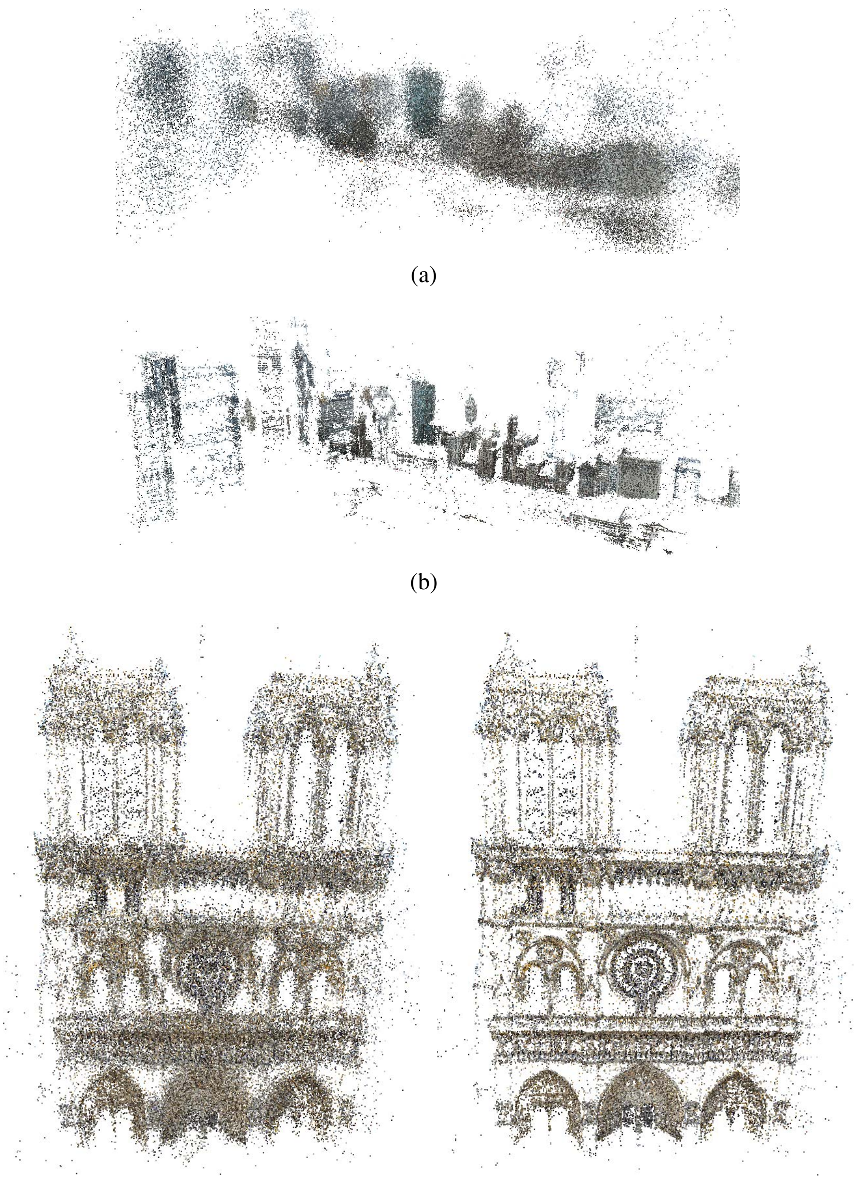


Fig. 7. The solutions obtained from solving (a) (c) the subgraph and (b) (d) the original graph on the *Chicago-2* dataset (from Grant Schindler) and the *NotreDame* datasets [23] respectively. Note that the solutions of the subgraphs are more blurry than (inferior to) those of the original graphs, but they could serve as good preconditioners to solve the original graph.

4 Generalized Subgraph Preconditioners

Although SPCG works well for 2-D pose SLAM problems, its performance is actually worse than the Jacobi preconditioner, a simple and empirically effective preconditioner [2,8,14], in our experiments on large-scale bundle adjustment. This indicates that we need a different representation to design subgraph preconditioners.

To this end, we propose the Generalized Subgraph Preconditioners (GSP), which generalize SPCG and are more suitable for large-scale bundle adjustment. While SPCG works on the *Jacobian* factor graph where each measurement corresponds to a Jacobian factor, GSP works on the *Hessian* factor graph where each measurement contribute three factors to the graph. We will show that this finer-grained graph possesses greater representation power than the Jacobian factor graph.

Compared to conventional matrix preconditioning machinery, GSP not only provides an expressive language to design subgraph preconditioners, but also explains the standard Jacobi preconditioner naturally.

4.1 Hessian Factor Graph Representation

To gain insight into the performance properties of both Jacobi and SPCG preconditioners, we investigate the structure of the *Hessian* matrix $\mathbf{H} \approx \mathbf{A}^{\mathrm{T}}\mathbf{A}$ appearing in the normal equation (4). The Hessian matrix can also be represented as a graph, more specifically a Gaussian Markov Random Field (GMRF). Every principal sub-matrix of $\mathbf{H}$ corresponds to the information matrix of the conditional distribution given the other variables [11,18]. In this sense, solving the GMRF is analogous to solving Equation (4).

Yet a GMRF is usually represented as an undirected graph which is not expressive enough for designing subgraph preconditioners. It prompts us to resort to a finer-grained Hessian factor graph representation. The main difference is that we create two unary and one binary factors out of each measurement, and accumulate all of them in the *Hessian*

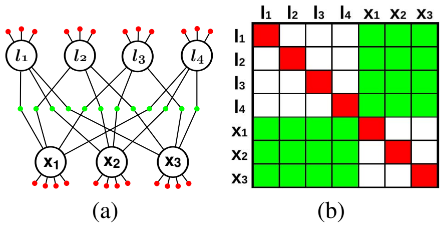


Fig. 8. The Hessian representation of the bundle adjustment problem in Figure 2. (a) The Hessian factor graph. The red dots denote unary factors while the green dots denote binary factors. This representation resembles to the Gaussian Markov Random Field representation [11,18]. (b) The symbolic representation of the Hessian matrix $\mathbf{H} \approx \mathbf{A}^{\mathrm{T}}\mathbf{A}$. Both rows and columns indicate variables. A diagonal (red) block indicates the certainty of a variable given the other variables are known. An off-diagonal block indicates whether two variables are correlated given that the other variables are known. Each non-zero off-diagonal (green) block corresponds to a Jacobian factor in Figure 2(a) or a binary Hessian factor in (a).

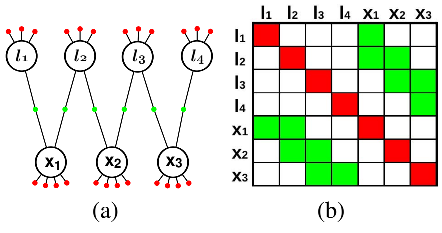
factor graph. The number of unary factors attached to a variable is equal to the number of the associated measurements, with one binary factor per measurement.

As an example, consider the measurement between x_0 and l_0 in Figure 2(a) and assume A_{x_0} and A_{l_0} are the corresponding block entries in the first row of the Jacobian matrix in Figure 2(b). Since the Hessian matrix is the sum of outer product of the block rows of the Jacobian matrix, we can see that this measurement actually corresponds to three terms in the Hessian matrix: $A_{x_0}^T A_{x_0}$, $A_{l_0}^T A_{l_0}$ and $A_{x_0}^T A_{l_0}$. Notice that the first two are unary factors of x_0 and l_0, and the third is a binary factor between them. They encode the information contributed by this measurement to the conditional Gaussian densities. Repeating this process for all measurements, we can build the Hessian factor graph representation illustrated in Figure 8(a).

From this perspective, the problem of designing a good subgraph preconditioner is reduced to picking a subset of Hessian factors from the graph that (1) can be solved efficiently by direct methods, and also (2) make the linear systems well-conditioned. Once a subgraph is selected, we can use sparse direct methods to factorize the linear system (i.e., $\mathbf{H_1} = \mathbf{R_1^T R_1}$) and use $\mathbf{R}_1$ as the preconditioner in the conjugate gradient method. The detail of how to to pick a subgraph will be discussed in Section 5.

GSP is more expressive than SPCG because we can always build a Hessian factor graph from a subset of measurements, but not vice versa. For instance, suppose we want to construct a Hessian factor subgraph as in Figure 9 by picking a subset of measurements. One can see that no subset of Jacobian factors in Figure 2(a) corresponds to this Hessian factor graph. Hence the GSP is indeed a generalization of SPCG.

The difference between GSP and SPCG is critical for large-scale bundle adjustment, whose graph structure is bipartite and highly unbalanced. The amount of information that SPCG brings in for each variable corresponds to the associated measurements in the subgraph. In bundle adjustment, if SPCG picks a spanning tree as the subgraph, then it can only collect at most two out of potentially thousands of unary factors for the camera vertices. This results in over-estimating the uncertainty of the variables and hence leads to unsatisfactory preconditioners. This idea is illustrated in Figure 10. Adding more measurements to the subgraph might help, but it also makes it harder for direct methods to to solve the subgraphs. In contrast, GSP provides the flexibility to keep part or all of the unary factors (information) for each variable, and hence overcomes this problem.



	l_1	l_2	l_3	l_4	x_1	x_2	x_3
l_1	🟥				🟩		
l_2		🟥			🟩	🟩	
l_3			🟥			🟩	🟩
l_4				🟥			🟩
x_1	🟩	🟩			🟥		
x_2		🟩	🟩			🟥	
x_3			🟩	🟩			🟥

Fig. 9. A subgraph that GSP can generate but SPCG cannot

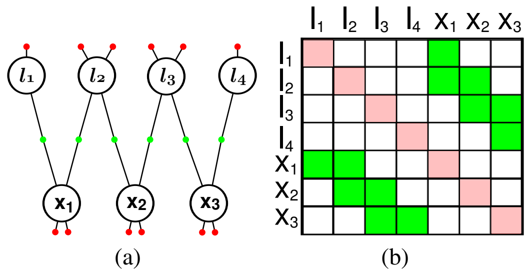


Fig. 10. The Hessian representation of the sub-problem in Figure 6. (a) The Hessian factor graph with the corresponding unary and binary factors. (b) The symbolic matrix of the sub-problem. The non-zero off-diagonal blocks are identical to those in Figure 8(b), but the diagonal entries are smaller than those in Figure 8(b). It leads to over-estimating the uncertainty of the variables, especially for the camera variables. This is problematic for large-scale bundle adjustment where the graph is bipartite and unbalanced.

4.2 The Jacobi Preconditioner

The Jacobi preconditioner is a generic technique and it has been shown empirically effective for large bundle adjustment [2,8,14]. Here we show that the Jacobi preconditioner has a simple explanation within the GSP framework. The Jacobi preconditioner works by taking only the diagonal entries of the Hessian matrix, and discarding all off-diagonal entries [21]. A simple generalization is the block Jacobi preconditioner which treats each camera and each 3-D point as an entity, and it corresponds to picking the block diagonal of the Hessian matrix. The block Jacobi preconditioners can be solved efficiently because all blocks are independent.

In the GSP machinery, the block Jacobi preconditioner corresponds to picking all of the unary factors and discarding all of the binary factors of in the Hessian factor graph. The idea is illustrated in Figure 11. Note that hereafter when we refer to the Jacobi preconditioner, we actually mean the block Jacobi preconditioner.

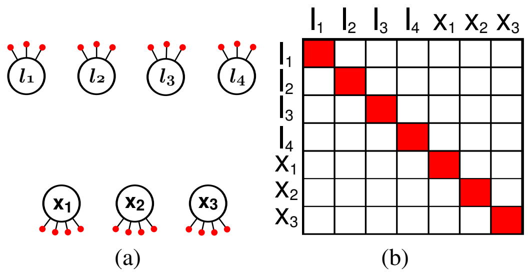


Fig. 11. Block Jacobi preconditioner of the toy problem

5 The GSP-n Preconditioners

5.1 Matrix Preconditioners

Conventional matrix preconditioning techniques focus more on the efficiency of solving the preconditioners rather than on directly minimizing the condition number of the preconditioned system [21]. For example, the Jacobi preconditioner offers good computational efficiency by discarding the conditional correlation between variables. The incomplete Cholesky preconditioner controls the computational cost by limiting the amount of fill-in and discarding negligible entries during the factorization process. Although these techniques work to some extent in practice, deriving theoretical bounds on their condition numbers is generally non-trivial, and their actual meaning is also hard to interpret graphically or probabilistically.

5.2 Combinatorial Preconditioners

Recently, combinatorial (graph) preconditioners have been studied to analyze and construct effective preconditioners for the conjugate gradient method. Promising results have been reported on solving linear systems with symmetric and diagonally dominant matrices [6,24]. The main idea is to find ultra-sparsifiers such that the original graph and the approximating graph have similar conductance – a measure of how fast information travels between different parts of the graph. Insisting on sparse approximating graphs produces preconditioners that can be solved efficiently by direct methods, while maintaining the graph conductance effectively reduces the condition number of the preconditioned systems, therefore the number of CG iterations.

If the subgraph is restricted to be a spanning tree, Boman and Hendrickson [6] recognized that the condition number of the preconditioned system is upper bounded by the *stretch* of the original graph with respect to the spanning tree. More specifically, suppose $G = (V, E, w)$ is the graph of the original system where V, E and w denote the vertices, edges and the weights of the edges respectively. If T is a spanning tree of G, then for every edge $e = (u, v) \in E$, there is a unique path in T connecting u and v. The stretch of e with respect to T is defined as

$$\mathbf{st}(T, e) = \sum_{f \in P(T,e)} \frac{w(e)}{w(f)}, \quad \text{for } e \in E \tag{12}$$

where $P(T, e)$ denotes the edges on the unique path between u and v in T. The stretch of G with respect to T is defined as the sum of the stretches of all the edges in G:

$$\mathbf{st}(T, G) = \sum_{e \in E} \mathbf{st}(T, e). \tag{13}$$

Intuitively speaking, the higher the stretch of a tree, the more time it takes for information to percolate, negatively affecting convergence.

If we relax the restriction and consider a general subgraph, a common practice is to use a *low-stretch* spanning tree as a skeleton and augment it with additional edges to further reduce the stretch. However, when additional edges are added to the subgraph, not only may the subgraph take longer to build, but also the preconditioners will become more expensive to apply in the conjugate gradient method. Clearly, there is a trade-off between the quality of the preconditioner and the time required to build and apply it.

5.3 The GSP-n Preconditioners

Finding the optimal subgraph is computationally intractable for large problems. Instead we propose a greedy algorithm to construct a family of subgraphs with adjustable complexity. On top of these subgraphs, we use GSP to build subgraph preconditioners. The resulting preconditioners are called the GSP-n preconditioners, where n is a parameter that controls the complexity of the subgraph.

The bundle adjustment graph is a bipartite graph $G = (X, L, E)$, where X denote the camera and L denote the 3-D points vertices on the two sides of G. Each edge in E denotes a measurement that connects the corresponding camera and point vertices.

The goal is to find a subset E_S of E, such that (1) the resulting subgraph G_S has low stretch with respect to G, and (2) the maximum size of the induced cliques does not exceed the predefined parameter n. By the maximum size of the induced cliques we actually mean the clique number in the factorization phase, which can indirectly affect the computational complexity. A straightforward strategy would be to use a low-stretch spanning tree of G as the subgraph, but this strategy is sub-optimal because it does not exploit the bipartite and unbalanced nature of G.

Here we introduce some notation to facilitate the explanation. We denote $X(l)$ as the set of cameras associated with a 3-D point l, and $E(l)$ as the corresponding set of edges (measurements). Note that by picking t edges from $E(l)$ into the subgraph, we will induce a clique of size t between the corresponding cameras after eliminating the 3-D point l in the factorization phase. Moreover, if the edges and the elimination ordering are not chosen appropriately, even larger cliques will appear in the factorization phase.

Here we describe a greedy algorithm to construct a family of subgraphs. First, we build a camera graph G_X where the vertices consist of all cameras and the edge weight between two cameras is defined as the number of 3-D points that are observed by both of them. Then we find a low-stretch spanning tree T_X in G_X. The tree T_X aims to preserve the structural information of G, and provides a reference to augment additional edges.

Second, we show how to augment additional edges to the subgraph. Suppose initially the edge set E_S is empty. For each point l, we sort $X(l)$ according to their average distance to the other cameras in $X(l)$ with respect to T_X. Then we pick the edges of $E(l)$ into the subgraph according to this ordering. An edge is added into E_S if it does not induce a camera clique of size greater than n. To this end, we also maintain an array (initially set to 0, whose length is the number of cameras) which holds the size of the maximum clique that a camera belongs to. The array is updated whenever an edge is added. Repeating this process for all 3-D points results in edge set E_S.

Finally we construct the GSP-n preconditioner by using all of the unary factors in the original graph and the binary factors corresponding to the edge set E_S. Note that there are two interesting special cases of the GSP-n preconditioners: GSP-0 corresponds to the Jacobi preconditioner while GSP-∞ corresponds to using the original graph to construct the subgraph preconditioner.

5.4 The Symmetry and Positive Semidefiniteness of GSP-n

Being symmetric and positive semi-definite (spsd) is a necessary condition for being a valid preconditioner in the conjugate gradient method. Here we show that any GSP-n preconditioner is spsd. First, we know that any $\mathbf{H} \approx \mathbf{A^T A}$ matrix is always spsd, and hence GSP-n is also symmetric by construction. Second, discarding off-diagonal block pairs $A_{x_0}^T A_{l_0}$, $A_{l_0}^T A_{x_0}$ in the Hessian while leaving the block-diagonal unchanged corresponds to replacing a binary factor by two unary factors in the Jacobian factor graph. The replaced binary factor corresponds to $\mathbf{A}$'s block-row with nonzero blocks A_{x_0} and A_{l_0}, while each new unary factor contains exactly one of these blocks. The inner product of the new factor matrix with itself is spsd, which guarantees the validity of GSP preconditioners. Note that discarding symmetrical off-diagonal entries of an *arbitrary* spsd matrix may not produce a spsd matrix. In the scalar case, Boman et al. [5] proved that matrices with this property must admit a factorization $\mathbf{A^T A}$, with $\mathbf{A}$ having a factor width ≤ 2.

6 Results

6.1 Configurations

Here we compare the sparse factorization method (DBA) and the conjugate gradient (CG) method with three preconditioners: (1) the block Jacobi preconditioner (JACOBI), (2) the subgraph preconditioner (SPCG), and (3) the generalized subgraph preconditioner (GSP-n). The number attached to "GSP-n " indicates the maximum clique size allowed in the greedy algorithm.

We use the Levenberg-Marquardt method as the nonlinear solver. The stopping criteria are (1) the number of iterations exceeds 20, (2) the average reprojection error is less than 0.8 pixel, or (3) the relative decrease of the error is less than 10^{-2}.

For the linear solvers, DBA uses the *cholmod* package [9] with an approximate mininum degree ordering. For the solvers using the CG method, we solve Equation (6) by using the least-squares variant of CG [4] without forming the normal equation (see Algorithm 1). The stopping criteria for the CG method are (1) the number of iterations exceeds 2000, (2) the relative decrease of residual is less than 10^{-2}.

For JACOBI, we accumulate all unary factors for each variable (i.e., the diagonal blocks of $\mathbf{A^T A}$) and solve them independently. For SPCG, we use the Sparse QR factorization package [10]. For GSP-n, we use the *cholmod* package [9] with an ordering in which the 3-D points are eliminated first and the cameras are eliminated according to the topological ordering of the camera low-stretch spanning tree. We use Alon et al.'s algorithm to find a low-stretch spanning tree in the camera graph [3]. Note that for SPCG and GSP-n, the topology of the subgraph is determined at the beginning, and never changed during the optimization.

We run the experiments on the *bal* datasets released by Agarwal et al. [2]. Since *bal* contains many datasets and some of them cannot fit into the memory of a regular PC, we select ten proper datasets from *bal* which have 100K to 500K points (see Table. 2). We run all of the experiments on a Core2 Duo PC with 8G RAM.

Algorithm 1. Preconditioned Conjugate Gradient Least-Squares Method

```
Input: Let $A$ be the Jacobian matrix, $R^T R$ be the factorized preconditioner, $x_0$ be an
       initial estimate, $\epsilon$ be the tolerance, and $t$ be the maximum number of iterations.
$r_0 = b - Ax_0, p_0 = s_0 = R^{-T}(A^T r_0), \gamma_0 = \|s_0\|_2^2$
for $k$ = 0 to $t$ do
    if $\gamma_k < \epsilon$ then break
    $t_k = R^{-1} p_k$
    $q_k = At_k$
    $\alpha_k = \gamma_k / \|q_k\|_2^2$
    $x_{k+1} = x_k + \alpha_k t_k$
    $r_{k+1} = r_k - \alpha_k q_k$
    $s_{k+1} = R^{-T}(A^T r_{k+1})$
    $\gamma_{k+1} = \|s_{k+1}\|_2^2$
    $\beta_k = \gamma_{k+1} / \gamma_k$
    $p_{k+1} = s_{k+1} + \beta_k p_k$
end
```

6.2 The Performance of GSP-n

We first investigate the performance of GSP-n for different values of n, and show the timing results in Figure 12. Notice that GSP-n is equivalent to JACOBI when $n = 0$. We exclude the linearization time and focus on comparing the linear solvers. The results show that GSP-n converges faster than JACOBI by 10-30% in most cases.

We also observe that as n increases, the overall time decreases at first, but increases if n is set too high. To better understand the behavior of GSP-n, we break down the timing results of one dataset and show the major components in Table 1. We can see that as n increases, the subgraph becomes denser and harder to solve, but the time spent on building the subgraph preconditioner is not significant when n is small. Here the important parts are (1) the time to apply the preconditioner per CG iteration, and (2) the number of total CG iterations. The former increases because the preconditioner

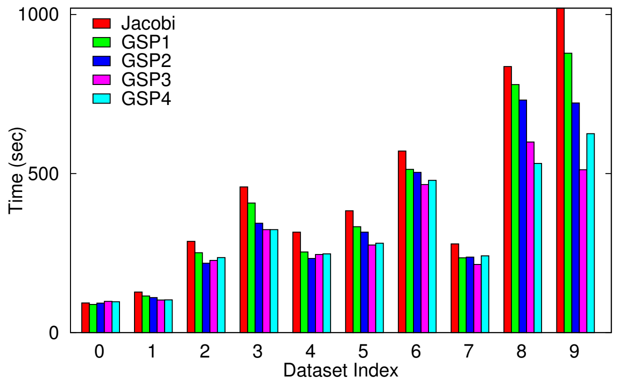


Fig. 12. Timing results of JACOBI and GSP-n on *bal*

Table 1. Timing results of GSP-n on the "F-05" dataset. We only show the components relevant to the linear solvers. The columns indicate (1) the maximum clique size in GSP-n, (2) the percentage of edges used in the subgraph, (3) the time of building the subgraph, (4) the time per CG iteration, and (5) the number of total CG iterations, and (6) the total time.

n	edges (%)	build (s)	time/iter (s)	#iters	total (s)
0	0.0	27.2	0.48	1438	732.6
1	19.8	33.4	0.53	1130	648.8
2	26.6	48.7	0.56	866	550.5
3	32.5	69.1	0.62	631	**473.7**
4	39.0	101.5	0.78	526	512.8

Table 2. Timing results (secs) of the four methods on ten *bal* datasets. The second column corresponds to the name and index in the original *bal*: "D" for "Dubrovnik", "L" for "Ladybug", "V" for "Venice" and "F" for "Final".

Set	Source	Cameras	Points	Measurements	DBA	JACOBI	SPCG	GSP-3
0	V-01	89	110,973	562,976	**42**	84	401	89
1	F-01	394	100,368	534,408	**79**	113	256	96
2	V-02	245	198,739	1,091,386	**155**	245	415	196
3	D-15	356	226,730	1,255,268	**187**	397	804	285
4	V-03	427	310,384	1,699,145	313	273	695	**212**
5	L-30	1,723	156,502	678,718	578	312	718	**223**
6	V-04	744	543,562	3,058,863	886	506	913	**407**
7	F-03	961	187,103	1,692,975	1148	252	741	**191**
8	F-02	871	527,480	2,785,977	1939	776	1154	**564**
9	F-05	3,068	310,854	1,653,812	3504	894	2035	**473**

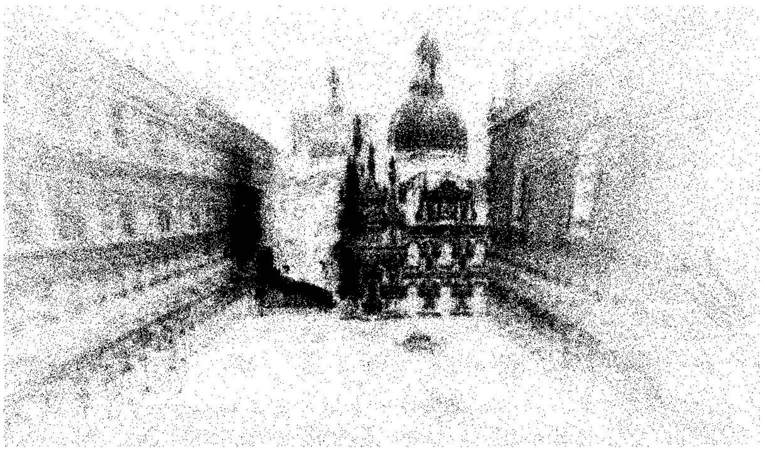
becomes denser and hence more computation is involved in the back substitution. The latter decreases because the linear systems become better conditioned. We can see that their product dominate timing and clearly there is a trade-off between these two factors.

6.3 Timing Results

Here we compare the timing results of four linear solvers on the *bal* datasets. We use $n = 3$ to build subgraphs for both SPCG and GSP-n. The timing results in Table 2 are sorted according to the DBA time, which reflects the intrinsic difficulty of the datasets. The results confirm that sparse direct methods are efficient for small datasets, but iterative methods are better alternatives for large datasets. Comparing JACOBI and GSP, the results show that by adding extra factors to the subgraph, GSP provide better

Table 3. The condition numbers of the SPCG, JACOBI, and GSP-3 on three *bal* datasets

Set	Original	SPCG	JACOBI	GSP-3
D-15	5.58e+21	1.87e+06	5.94e+04	4.36e+03
V-02	6.54e+21	6.46e+09	6.35e+05	1.38e+05
F-01	3.68e+11	1.92e+08	7.54e+06	8.71e+05


(a)

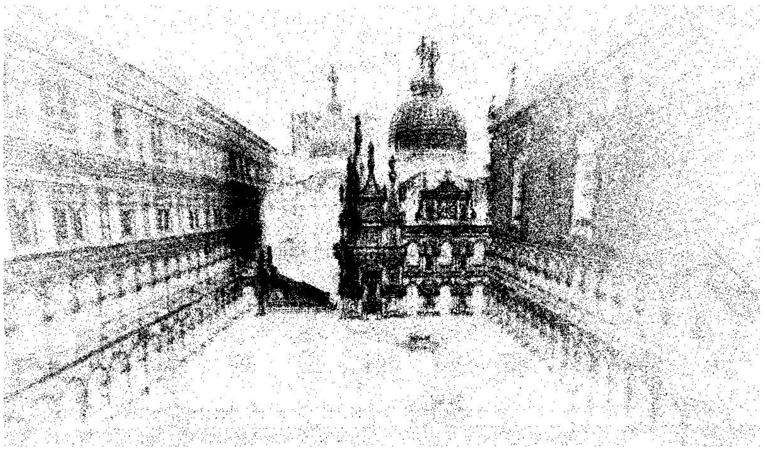

(b)

Fig. 13. Visualization of the "F-03" datasets. The solutions obtained from solving (a) the subgraph and (b) the original graph. Similar to Figure 7, the solution to subgraph serves as a good preconditioner to solve the original problem.

preconditioners than JACOBI in most of the cases. Comparing SPCG and GSP, the results show that being able to add more unary factors to the graph is crucial to improve the convergence speed of the CG method. An example of the result is shown in Figure 13.

6.4 The Condition Numbers

The condition number is a common measure to estimate the convergence speed of the conjugate gradient method [21]. Here we compare the condition numbers of the linear systems preconditioned by the SPCG, JACOBI and GSP-3 preconditioners on several medium *bal* datasets. The results are shown in Table 3. We can see that the original condition numbers are huge, which indicate the slow convergence of using a plain CG solver. The SPCG precondtitioner works to some extent, but is not as good as JACOBI and GSP-3. The condition numbers of GSP-3 are 5-10 times smaller than JACOBI.

7 Conclusions and Future Work

While direct methods are efficient for small datasets and iterative methods are more appropriate if the memory requirement is of concern, a subgraph-based preconditioning method combines their advantages and provides a better alternative for solving large-scale bundle adjustment. One such method is SPCG, which to the best of our knowledge has not been applied to the bundle adjustment problem until now. Although for large datasets SPCG is significantly better than direct methods and the plain CG method, its behavior is sub-optimal: as the bundle adjustment graph is bipartite and unbalanced, SPCG over-estimates the uncertainty of the variables. In contrast, GSP avoids this problem, and is more expressive and suitable for bundle adjustment. Well-known preconditioners like Jacobi fit naturally in the GSP context. To exploit the graphical structure of the problem, we develop an efficient algorithm rooted in combinatorial preconditioning, to construct a family of subgraph preconditioners. When applied to large datasets, the GSP-n preconditioners display promising performance.

For future work, first we would like to develop a more expressive factor representation to explain and understand the other matrix preconditioners such as the Incomplete Factorization and the Symmetric and Successive Over-Relaxation preconditioners. The second is to develop a better algorithm to construct the subgraph preconditioners, and provide theoretical guarantees for their performance.

References

1. Agarwal, S., Snavely, N., Simon, I., Seitz, S., Szeliski, R.: Building rome in a day. In: IEEE 12th International Conference on Computer Vision, pp. 72–79 (2009)
2. Agarwal, S., Snavely, N., Seitz, S.M., Szeliski, R.: Bundle Adjustment in the Large. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010. LNCS, vol. 6312, pp. 29–42. Springer, Heidelberg (2010)

3. Alon, N., Karp, R., Peleg, D., West, D.: A graph-theoretic game and its application to the k-server problem. SIAM Journal on Computing 24(1), 78–100 (1995)
4. Björck, A.: Numerical Methods for Least Squares Problems. SIAM Publications (1996)
5. Boman, E., Chen, D., Parekh, O., Toledo, S.: On factor width and symmetric h-matrices. Linear Algebra and its Applications 405, 239–248 (2005)
6. Boman, E., Hendrickson, B.: Support theory for preconditioning. SIAM Journal on Matrix Analysis and Applications 25(3), 694–717 (2003)
7. Byröd, M., Åström, K.: Bundle adjustment using conjugate gradients with multiscale preconditioning. In: British Machine Vision Conference (2009)
8. Byröd, M., Åström, K.: Conjugate Gradient Bundle Adjustment. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010. LNCS, vol. 6312, pp. 114–127. Springer, Heidelberg (2010)
9. Chen, Y., Davis, T., Hager, W., Rajamanickam, S.: Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. ACM Transactions on Mathematical Software 35(3), 1–14 (2009)
10. Davis, T.: Algorithm 915, SuiteSparseQR: multifrontal multithreaded rank-revealing sparse QR factorization. ACM Transactions on Mathematical Software 38(1) (2011)
11. Dellaert, F., Kaess, M.: Square root sam: Simultaneous localization and mapping via square root information smoothing. International Journal of Robotics Research 25(12), 1181–1203 (2006)
12. Dellaert, F., Carlson, J., Ila, V., Ni, K., Thorpe, C.E.: Subgraph-preconditioned conjugate gradient for large scale slam. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2010)
13. Frahm, J.-M., Fite-Georgel, P., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y.-H., Dunn, E., Clipp, B., Lazebnik, S., Pollefeys, M.: Building Rome on a Cloudless Day. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010. LNCS, vol. 6314, pp. 368–381. Springer, Heidelberg (2010)
14. Jeong, Y., Nister, D., Steedly, D., Szeliski, R., Kweon, I.: Pushing the envelope of modern methods for bundle adjustment. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1474–1481 (2010)
15. Jian, Y.D., Balcan, D.C., Dellaert, F.: Generalized subgraph preconditioners for large-scale bundle adjustment. In: IEEE 13th International Conference on Computer Vision (2011)
16. Konolige, K., Garage, W.: Sparse sparse bundle adjustment. In: Proc. of the British Machine Vision Conference (2010)
17. Lourakis, M., Argyros, A.: SBA: A software package for generic sparse bundle adjustment. ACM Transactions on Mathematical Software 36(1), 1–30 (2009)
18. MacKay, D.: Information theory, inference, and learning algorithms. Cambridge Univ. Press (2003)
19. Ni, K., Steedly, D., Dellaert, F.: Out-of-core bundle adjustment for large-scale 3D reconstruction. In: IEEE 11th International Conference on Computer Vision (2007)
20. Olson, E., Leonard, J., Teller, S.: Fast iterative alignment of pose graphs with poor initial estimates. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 2262–2269 (2006)
21. Saad, Y.: Iterative methods for sparse linear systems. Society for Industrial Mathematics (2003)
22. Snavely, N., Seitz, S.M., Szeliski, R.S.: Skeletal graphs for efficient structure from motion. In: IEEE Conference on Computer Vision and Pattern Recognition (2008)

23. Snavely, N., Seitz, S., Szeliski, R.: Modeling the world from internet photo collections. International Journal of Computer Vision 80(2), 189–210 (2008)
24. Spielman, D.A.: Algorithms, graph theory, and linear equations. In: International Congress of Mathematicians (2010)
25. Trefethen, L., Bau, D.: Numerical linear algebra, vol. 50. Society for Industrial Mathematics (1997)
26. Triggs, B., McLauchlan, P.F., Hartley, R.I., Fitzgibbon, A.W.: Bundle Adjustment – A Modern Synthesis. In: Triggs, B., Zisserman, A., Szeliski, R. (eds.) ICCV-WS 1999. LNCS, vol. 1883, pp. 298–372. Springer, Heidelberg (2000)