Visibility Based Preconditioning for Bundle Adjustment

Avanish Kushal
University of Washington, Seattle
kushalav@cs.washington.edu

Sameer Agarwal
Google Inc.
sameeragarwal@google.com

Abstract

We present Visibility Based Preconditioning (VBP) a new technique for efficiently solving the linear least squares problems that arise in bundle adjustment [23]. Using the camera-point visibility structure of the scene, we describe the construction of two preconditioners. These preconditioners when combined with an inexact step Levenberg-Marquardt algorithm[24] offer state of the art performance on the BAL data set [1], with 3-5x reduction in execution time over currently available methods while delivering comparable or better solution quality.

1. Introduction

Recent work in Structure from Motion (SfM) has enabled three dimensional reconstruction from large unstructured community photo-collections [2, 8, 20] and reconstructions with thousands of images are now routinely computed. Given the feature matches between images, the bottleneck in an SfM system is the bundle adjustment process – the joint non-linear refinement of camera parameters and the 3D scene to minimize the reprojection error [23]. This has sparked a renewed interest in scalable large bundle adjustment algorithms [1, 3, 7, 10, 11, 14, 16, 20–22, 25, 26].

All the successful bundle adjustment methods that we are aware of formulate the bundle adjustment problem as a non-linear least squares problem and use some variant of the Levenberg-Marquardt algorithm to solve it [17]. Levenberg-Marquardt operates by repeatedly linearizing the objective function into a linear least squares problem and solving its *normal equations*. Thus, reducing the cost of bundle adjustment comes down to reducing the number of times the *normal equations* are solved, and reducing the cost of each individual solve.

Traditionally, bundle adjustment algorithms have exploited the primary sparsity structure of the bundle adjustment problem (also known as the Schur complement trick) and a sparse or dense Cholesky factorization of the resulting Schur complement matrix [12, 23]. But as the size and complexity of the SfM problems has increased, interest has shifted to Conjugate Gradients (CG) based methods [1, 3, 10, 25, 26].

CG based methods have a fraction of the memory usage of factorization based methods and can even be run matrix-free, where the Jacobian is never stored in memory [25]. There is however a catch. The rate of convergence of CG depends on the condition number of the linear system being solved and bundle adjustment problems are notoriously ill-conditioned. This ill-conditioning occurs because of gauge ambiguity and wide variability in the sensitivity of the objective function to the different parameters, e.g., small changes in the translation of a camera affect the objective less than small changes in the radial distortion parameters.

The way around this problem is to use a *preconditioner*, an invertible matrix that improves the condition number of the linear system [18]. A good preconditioner has the competing goals of reducing the condition number as much as possible while still being efficiently computable. Constructing such preconditioners is the subject of this paper.

We present *Visibility Based Preconditioning*, a new technique for constructing efficient, high quality preconditioners for bundle adjustment problems. Based on the idea that the number of 3D points visible to a pair of cameras is an indicator of the strength of their coupling, we present two preconditioners, cluster-jacobi and cluster-tridiagonal. The former is a block-diagonal preconditioner and the latter a block-tridiagonal preconditioner. When coupled with an inexact Levenberg-Marquardt algorithm [24], these preconditioners give state of the art performance on the BAL dataset [1].

The rest of the paper is organized as follows. Section 2 presents a brief overview of the Bundle Adjustment problem and recent work on the use of preconditioned iterative methods for solving it. Section 3 describes the construction of two new preconditioners. Section 4 compares these new preconditioners to the state of the art using problems from the BAL dataset. We conclude with a discussion in section 5.

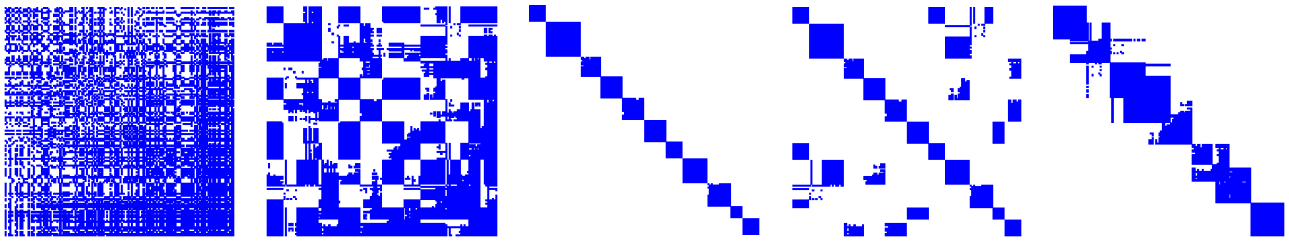

Figure 1. Visibility based preconditioning. S is the sparsity pattern of the Schur complement matrix for the ladybug-138 [1]. S_1 is the same matrix, with its rows and columns permuted using the permutation matrix P_1 induced by the clustering. cluster-jacobi preconditioner is the block diagonal of S_1. We show two views of the cluster-tridiagonal preconditioner, as a *degree 2* tree in S_1 and as a block tridiagonal matrix, after permuting it by P_2

2. Bundle Adjustment

We present a necessarily brief overview of the bundle adjustment problem and methods to solve it. Please see Triggs et al. [23] for a comprehensive survey.

Given a set of measured image feature locations and correspondences, the goal of bundle adjustment is to find 3D point positions and camera parameters that minimize the reprojection error [23]. More formally, let x be the parameter vector and $f(x) = [f_1(x), \ldots, f_k(x)]$ be the vector of reprojection errors for the 3D reconstruction. Then the bundle adjustment problem is formulated as the non-linear least squares problem:

$$x^* = \arg\min_x \sum_{i=1}^{k} \|f_i(x)\|^2. \tag{1}$$

Let $J(x)$ be the Jacobian of $f(x)$, then in each iteration LM solves a linear least squares problem of the form.

$$\delta^* = \arg\min_\delta \left\| \begin{bmatrix} J \\ \sqrt{\lambda} D \end{bmatrix} \delta + \begin{bmatrix} f \\ 0 \end{bmatrix} \right\|^2 \tag{2}$$

and updates $x \leftarrow x + \delta^*$ if $\|f(x+\delta^*)\| < \|f(x)\|$. Here, $D(x)$ is a non-negative diagonal matrix, typically the square root of the diagonal of the matrix $J(x)^\top J(x)$ and λ is a non-negative parameter that controls the strength of regularization [17].

Before going further, lets make some notational simplifications. We will assume that the matrix $\sqrt{\lambda}D$ has been concatenated at the bottom of the matrix J and similarly a vector of zeros has been added to the bottom of the vector f and the rest of our discussion will be in terms of J and f, i.e. the linear least squares problem.

$$\min_\delta \|J(x)\delta + f(x)\|^2. \tag{3}$$

Further, let $g(x) = -J(x)^\top f(x)$ and for notational convenience let us drop the dependence on x. Then it is easy to see that solving (3) is equivalent to solving the *normal equations*

$$J^\top J \delta = g \tag{4}$$

The solution of (4) in each iteration of the LM algorithm is the dominant computational cost in bundle adjustment.

Let the parameter vector be organized as $x = [x_c; x_p]$, where x_c is the camera parameter vector and x_p the point parameter vector. Similarly for δ, g, and J, we use subscripts c and p to denote the camera part and the point part respectively . Let $U = J_c^T J_c$, $V = J_p^T J_p$ and $W = J_c^T J_p$, then (4) can be re-written as the block structured linear system

$$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta_c \\ \delta_p \end{bmatrix} = \begin{bmatrix} g_c \\ g_p \end{bmatrix}. \tag{5}$$

It is worth noting that for most bundle adjustment problems V is a block diagonal matrix, with blocks of size 3×3 and thus trivial to invert. This observation lies at the heart of the *Schur complement trick* used to solve this linear system efficiently, where by applying Gaussian elimination to the point parameters, we obtain a linear system consisting of just the camera parameters:

$$S\delta_c = r \tag{6}$$

where $r = g_c - WV^{-1}g_p$ and $S = U - WV^{-1}W^T$ is the Schur complement or the reduced camera matrix. Given the solution to (6), δ_p, the point parameters vector can be obtained by back-substitution:

$$\delta_p = V^{-1}(g_p - W^\top \delta_c) \tag{7}$$

Since S is symmetric positive-definite, Cholesky factorization can be used to exactly solve (6) [13]. But Cholesky factorization libraries, even ones like CHOLMOD [4] which exploit the sparsity structure of S, are space and time intensive that makes them prohibitively expensive for large problems. Thus, the recent focus on Conjugate Gradients based methods for solving (4) and (6) [1, 3, 10, 25, 26]

In the rest of the paper we will focus on preconditioners for the iterative solution of (4). In particular, we will look at solving (6) using Preconditioned Conjugate Gradients. Note that CG can be run on the Schur complement S without actually computing and storing it in memory [1, 25], by exploiting the relation

$$Sx = J_c^\top \left[J_c x - J_p \left[V^{-1} \left[J_p^\top \left[J_c x \right] \right] \right] \right]. \qquad (8)$$

2.1. Related Work

Jeong et al. proposed using the band block diagonals of the Schur complement matrix S as preconditioners. They observed that amongst the various banded preconditioners, the block Jacobi preconditioner was a cheap and robust choice[10]. Byröd & Åström avoided the computation of H and S, and instead ran CG on J directly with an incomplete QR factorization based preconditioner [3]. Their construction is equivalent to running CG on H with a block Jacobi preconditioner. Agarwal et al. proposed the use of the block diagonal of $J_c^\top J_c$ and the block diagonal of S as preconditioners for S without storing S in memory explicitly [1]. Wu et al. [25] extended this work to a multicore Jacobian free bundle adjustment method. It ran CG on S by using (8) with the block diagonal of $J_c^\top J_c$ as the preconditioner.

More recently, there has been work towards designing preconditioners based on the combinatorial structure of the bundle adjustment problem [5, 26]. Inspired by the work in combinatorial preconditioning, the authors have proposed using low-stretch spanning tree approximations to H as preconditioners for (4).

3. Visibility Based Preconditioning

Recall that we are interested in the efficient iterative solution of the symmetric positive definite linear system $S\delta_c = r$. The convergence rate of CG on this linear system, depends on the condition number $\kappa(S)$ of S. If we use a preconditioner matrix M, the condition number changes to $\kappa(SM^{-1})$. The computational cost of using M is the cost of computing M and evaluating the product $M^{-1}y$ for arbitrary vectors y. Thus, there are two competing factors to consider: How much of S's structure is captured by M so that the condition number $\kappa(SM^{-1})$ is low, and the computational cost of constructing and using M. It is usually the case that the more information M has about S, the more expensive it is use. For example, Incomplete Cholesky factorization based preconditioners have much better convergence behavior than the Jacobi preconditioner, but are also much more expensive.

In designing new preconditoners for S, our point of departure is the block Jacobi preconditioner for S [1, 10]. It is a simple approximation to S, that ignores all pairwise camera interactions. A better approximation would be one that includes off-diagonal block from S in the form of its band block diagonals [10]. This has two problems. One, the ordering of the cameras in S dictates which off diagonal blocks are included, and two, band diagonals of positive definite matrices are not guaranteed to be positive definite (unless the matrix being preconditioned is diagonally dominant). This makes their use in CG problematic.

So the task ahead of us is to construct a symmetric positive definite matrix M, that accounts for the significant camera-camera interactions in S. This of course begs the question, how do we measure the interaction/coupling between a pair of cameras? A number of heuristic choices are possible. Some that use the numerical values of the entries of S, others that only pay attention to its sparsity structure. We propose the use of scene visibility as a predictor of the coupling between cameras, i.e., In an SfM reconstruction, the strength of coupling depends positively on the number of points visible in both the cameras. Scene visibility has previously been used to speed up image matching and bundle adjustment [2, 8, 20]. We will now exploit this structure for preconditioning.

A significant advantage of using scene visibility over other measures is that it is independent of the actual numerical values of the camera and point parameters and in turn the numerical entries of S. It can be computed and used to determine the sparsity structure of the preconditioner before the start of the bundle adjustment algorithm. At the same time it is not just the 0-1 structure of S. It is a soft measure that treats two cameras with two points in common different from two cameras with two hundred points in common. Our experimental results will show that despite ignoring the magnitude of the entries in S, scene visibility leads to excellent preconditioning performance.

3.1. Clustering

SfM problems, especially the ones coming from community photo collections have highly non-uniform visibility - popular points of interest like entrances to landmarks, historically interesting locations, etc. have a very high concentration of images. On the other hand as we move from one popular viewpoint to another, since the physical world is opaque, it is very common that there is little to no interaction between the images. Thus most of the interactions occur within dense clusters of cameras and show up as dense sub blocks in S. Identifying and accounting for these clusters in M can be expected to lead to a good approximation to S, therefore the first step in our algorithm is to cluster the cameras.

If the scene contains n_p points, then each camera i can be described by a binary visibility vector $v_i \in \{0,1\}^{n_p}$, where the k^{th} entry is 1 if the point k is visible in camera i and 0 otherwise. Given these visibility vectors, we can now define a similarity measure between a pair of camera as the dot products of their corresponding normalized visi-

bility vectors which can then be used to cluster the cameras using a variety of clustering algorithms. We use the Canonical Views algorithm of Simon et al. [19]. The Canonical Views algorithm has been shown to be effective algorithm in identifying image clusters in 3d reconstructions. It greedily computes a set C of *canonical views* that maximizes the objective function

$$\sum_{i \in I} \max_{j \in C} \frac{v_i^\top v_j}{\|v_i\| \|v_j\|} - \alpha |C| \tag{9}$$

Here, the first term maximizes coverage while the second term tries to minimize the number of canonical views selected. We use a fixed value of $\alpha = 2.2$ in all our experiments. Given the canonical views, the clusters can be obtained by assigning each camera to the canonical view that it is most similar to. The average sizes of the clusters returned were mostly independent of the problem.

3.2. Cluster-Jacobi

Given a clustering, the rows and columns of S can be permuted so that all the cameras from cluster 1 appear before all the cameras from cluster 2 and so on. Let's denote the matrix that performs this permutation by P_1 and let $S_1 = P_1 S P_1^\top$. Figure 1 shows the sparsity structure of a Schur complement matrix before and after the clustering induced permutation. Cameras within a cluster are expected to interact strongly with each other – this is reflected in the near dense super-blocks (one for each cluster) along the diagonal of S_1. Comparing this to S we can see that we have moved a significant amount of the mass in S closer to the diagonal of S_1. We can now treat S_1 as a block matrix with blocks corresponding to the clusters. We will refer to the block diagonal of this matrix, as shown in Figure 1, as the cluster-jacobi preconditioner, in analogy with the block Jacobi preconditioner. The block Jacobi preconditioner is a strict specialization of cluster-jacobi with each cluster containing exactly one camera. Note that since S is positive definite, S_1, which is a symmetric permutation of S is also positive definite. Thus cluster-jacobi, which is the block diagonal of a positive definite matrix is also a positive definite matrix.

3.3. Cluster-Tridiagonal

The cluster-jacobi preconditioner only accounts for intra-clusters interactions. As can be seen in Figure 1, inter-cluster interaction can be quite significant, showing up as dense off-diagonal blocks in S_1. Of course, accounting for all of them is not feasible, because then we would end up factorizing and inverting S. Instead, we want to add only those off-diagonal blocks which capture significant interactions between clusters and yet, still lead to an easily factorable matrix M.

One class of matrices whose Cholesky factorization does not introduce any fill-in is the set of block band diagonal matrices. We consider the simplest of these - the block-tridiagonal matrices. Simply taking the block-tridiagonal of S_1 is not a good idea, as the permutation P_1 will determine which off diagonal blocks will be included. Thus, our aim is to find a second permutation P_2 such that $S_2 = P_2 S_1 P_2^\top$ has as much mass as possible in its block-super and sub diagonals, i.e., as many of the inter cluster interactions as possible are accounted for. Finding such permutations is studied in the literature on *Bandwidth Minimization* and is known to be NP-Hard [9]. We propose a greedy heuristic suitable for bundle adjustment.

Define an undirected weighted graph $G(\mathcal{V}, \mathcal{E})$, whose vertices are the clusters, i.e, $\mathcal{V} = \{\mathcal{C}_1, \cdots, \mathcal{C}_k\}$. The weight on the edge connecting two vertices is the number of 3D points visible in the two camera clusters it connects.

Now, it is easy to see, that if S_2 were a block-tridiagonal matrix, the graph G would need to be a *degree* 2 tree – a tree where all its vertices have degree at most 2. A degree-2 tree is nothing more than a set of vertices on a line, and the corresponding permutation matrix P_2 can be obtained by traversing this line from one terminal vertex to another.

Thus to get a block-tridiagonal approximation of S_2, we first compute G_T, a *degree* 2 tree approximation of G. However, such an approximation, doesn't always span the graph. For example a star graph, i.e. a graph with n-nodes where one node is connected to the other $n-1$ nodes does not have any spanning trees of degree less than $n-1$. To get around this problem, we propose to approximate the graph G with a disjoint collection of degree-2 trees (or a degree-2 forest) and P_2 is then computed by concatenating the orderings implied by the trees – the trees themselves can be ordered arbitrarily.

To compute the degree-2 forest we use a constrained variant of Kruskal's MST algorithm. We start with a graph $G_T(\mathcal{V}, \{\})$, that has the same vertices as G and an empty edge set. We then iterate over the edges of G in decreasing order of weight, adding them to G_T if doing so does not create a cycle in G_T and the degree of all the vertices in G_T remains bounded by two. This $O(|E|)$ algorithm results in a disjoint collection of *degree* 2 trees that span the graph G.

Now S_2, like S_1 is a block matrix, where the blocks correspond to the clusters and the preconditioner cluster-tridiagonal contains the block diagonal, and the first super and sub-diagonal of S_2. Observe that an off-diagonal block corresponding to two clusters is included in the preconditioner if and only if the two clusters are connected by an edge in G_T. Figure 1 shows the *degree 2* tree approximation, and the result of applying P_2 to it.

However, cluster-tridiagonal in its current form is not guaranteed to be positive definite. As we mentioned earlier, band diagonals of positive definite matrices can be indefi-

nite. There are two ways around this problem. The first is to find a *modified* Cholesky factorization $M = LL^\top - R$, where a correction R is applied to M to make it positive definite. Methods for determining a good R require complicated book keeping and are not simple to implement [18]. For block tridiagonal matrices, there is however a simpler and more efficient static strategy – increase the diagonal dominance of the matrix M by scaling down the off-diagonal blocks of M. The following lemma (stated in scalar form, but easily generalizable to block matrices), whose proof can be found in the supplemental material, describes an *optimal* static scaling strategy.

Lemma 1. *Let A be a positive semidefinite symmetric matrix, then the tridiagonal matrix $M(\nu)$*

$$m_{ij}(\nu) = \begin{cases} a_{ij} & i = j \\ \nu a_{ij} & |i-j| = 1 \\ 0 & \textit{otherwise} \end{cases} \tag{10}$$

is positive semidefinite for $\nu = 0.5$ and for every $\epsilon > 0$, there exists a positive semidefinite matrix A, such that $M(0.5+\epsilon)$ is indefinite.

Lemma 1, only states that the cluster-tridiagonal matrix, $M(1)$ may be indefinite, but this may not always be the case. Thus we first construct the preconditioner as the unscaled version. If its Cholesky factorization exists, we are done. If we encounter a non-positive pivot, we apply the scaling suggested by Lemma 1, i.e use $M(0.5)$ and recompute the Cholesky factorization. This re-computation effort is a negligible fraction of the setup time, and helps us avoid unnecessary loss of mass in the preconditioner.

3.4. Implementation

Given the permutation matrices P_1 for cluster-jacobi and P_1P_2 for cluster-tridiagonal, we identify which of the blocks of the original matrix S make it into each of the preconditioners. Only these entries of the Schur complement are computed. The resulting preconditioner matrix M is factorized using CHOLMOD [4]. CHOLMOD's optimized triangular solve is also used to implement the matrix-vector product $M^{-1}y$ in each iteration of CG.

4. Experiments

We compare the performance of seven linear solvers/preconditioners: cluster-tridiagonal, cluster-jacobi, implicit-jacobi, implicit-ssor, normal-jacobi, gsp-3, explicit-sparse. implicit-jacobi and implicit-ssor run CG on S using (8) with the block diagonal of $J_c^\top J_c$ and S as preconditioners respectively. normal-jacobi uses the block Jacobi preconditioner for H. gsp-3 is a combinatorial preconditioner for H that uses a low stretch maximum weight spanning tree [26]. Finally, explicit-sparse is a direct factorization based method that computes S and factorizes it using CHOLMOD [4]. All the preconditioners with the exception of gsp-3 were implemented as part of the same C++ library. Only a MATLAB implementation of gsp-3 was available from the authors the time of writing.

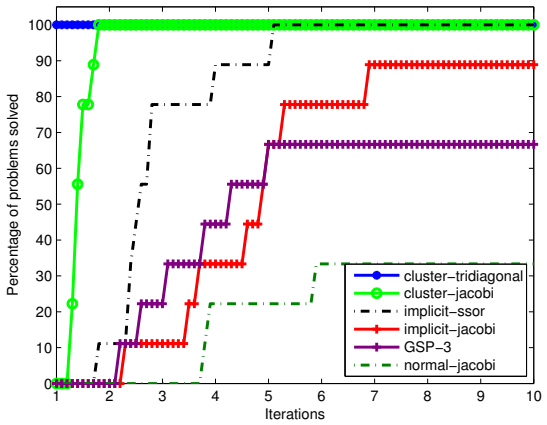

Figure 2. Performance profiles for the six preconditioners on small linear problems from the BAL dataset. Performance is based on the number of iterations.

4.1. Performance Profiles

We use *Performance Profiles* to report and compare the performance of the various solvers [6]. We briefly describe the method here.

Let $\mathcal{P}$ be a set of problems, $\mathcal{S}$ be a set of solvers and let $0 < \tau < 1$ be a user specified tolerance. For each problem $p \in \mathcal{P}$, run all the solvers $s \in \mathcal{S}$ till some convergence criterion (time, iteration or error tolerance) is satisfied.

Let $f(p,s)$ denote the minimum function value achieved on problem p using solver s. Let $f^*(p) = \min_s f(p,s)$, be the minimum value across all solvers for problem p. Define

$$f_\tau(p) = f^*(p) + \tau(f_0(p) - f^*(p)), \tag{11}$$

where $f_0(p)$ is the initial value of p.

Now to characterize the time performance of the solvers in $\mathcal{S}$, let $t(p,s)$ denote the time it takes solver s to reach $f_\tau(p)$, where $t(p,s) = \infty$ if the solver is unable to reach $f_\tau(p)$. Then the performance profile of a solver s over the problem set P is the curve

$$\rho(s,\alpha) = 100 \times \frac{|\{p : t(p,s) < \alpha \min_s t(p,s)\}|}{|\mathcal{P}|} \tag{12}$$

$\rho(s,\alpha)$ is a non-decreasing function of α. It measures the percentage of problems that are solved to a relative tolerance τ by solver s in time bounded by $\alpha \min_s t(p,s)$. A perfect solver (one that solves all the problems the fastest) will have a performance profile $\rho(s,\alpha) = 100, \forall \alpha \geq 1$.

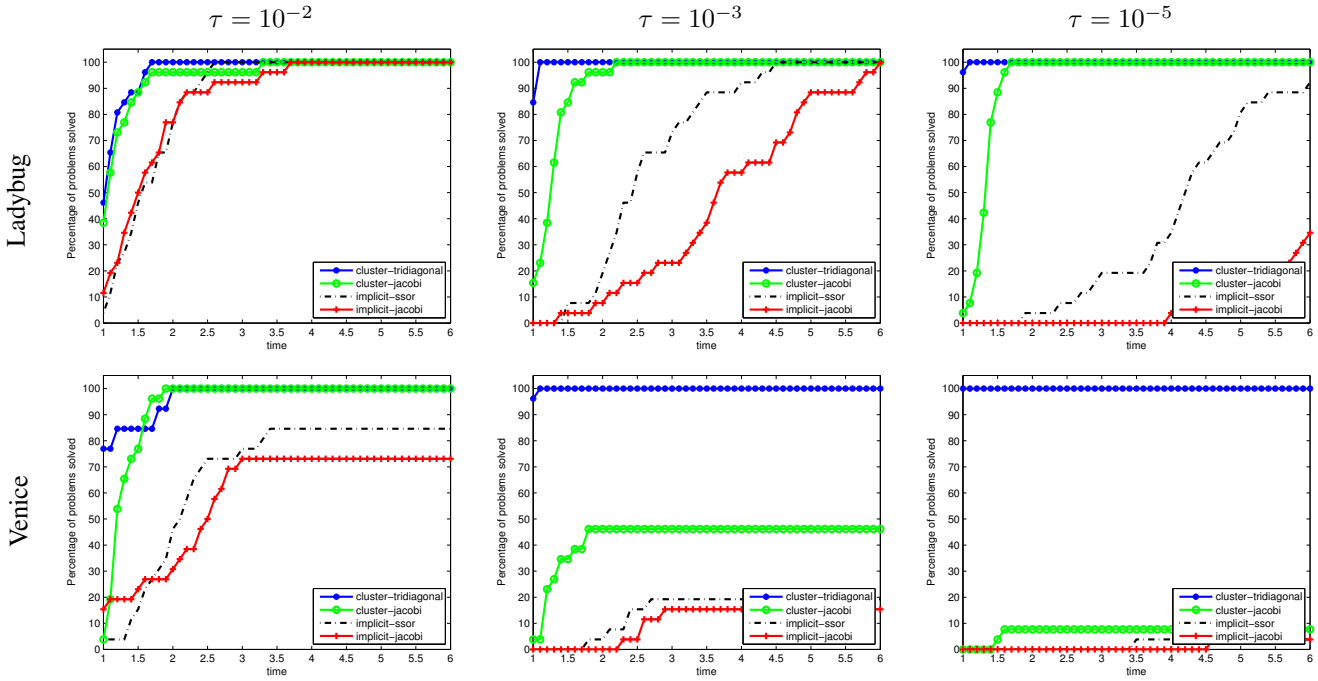

Figure 3. Performance profiles for the four Schur complement based preconditioners on linear problems from the BAL dataset. Convergence is based on time.

4.2. Linear Problems

We begin by comparing the performance of the six iterative solvers on linear least squares problems. Since there is no high-performance implementation of gsp-3 available, comparisons based on execution time would not be fair. Further, the MATLAB implementation made available by the authors does not scale to large problems. Therefore, we selected five small bundle adjustment problems from the ladybug dataset and four small problems from the Venice dataset[1]. For each problem we generated a linear least squares problem at the initial estimate of the solution. Each of the six solvers were run with relative residual tolerance of 10^{-6} or a 1000 iterations, whichever came first. Figure 2 shows the performance profiles for $\tau = 10^{-5}$. Since time is not a factor in this experiment, the setup time for all the preconditioners is ignored and the iteration time (time taken per iteration) for each preconditioner is assumed to be the same. Notice that the cluster-jacobi and cluster-tridiagonal preconditioners perform extremely well, with cluster-tridiagonal the better of the two – infact it is the perfect solver. The next best solver implicit-ssor takes upto five times as many iterations to solve the same problems. The other three solvers including gsp-3 were not even able to solve all of the problems. normal-jacobi the simplest of the six preconditioners is the worst performer. This is by no means a comprehensive experimental suite, but it does indicate that Schur complement based methods perform better than Hessian based methods. These performance profiles are also correlated with the condition number of the corresponding preconditioned matrices[2].

A more strenuous and fair test of linear solver performance will be one that accounts for the total time taken to solve a problem. Thus, in our second experiment we took 26 problems each from the ladybug and Venice datasets with more than 400 cameras each and ran the Schur-based preconditioners, cluster-tridiagonal, cluster-jacobi, implicit-jacobi and implicit-ssor on them. Each solver was allowed a time budget of 300 seconds with a convergence tolerance of 10^{-6}. We account for the time needed to compute the entries of the Schur complement for the cluster-tridiagonal, cluster-jacobi and implicit-ssor as well as the time needed to factor the resulting matrix. The time for computing the clustering and the tridiagonal permutation is only spent once per bundle adjustment problem and amortized across all linear solves and therefore not accounted for. Figure 3 shows the profile curves for each of these two datasets for $\tau = 10^{-2}$, $\tau = 10^{-3}$ and $\tau = 10^{-5}$. Note that the cluster-diagonal and cluster-tridiagonal solvers dominate all the other solvers, especially for $\tau = 10^{-3}$, $\tau = 10^{-5}$. Only for extremely loose tolerance of $\tau = 10^{-2}$ is the performance of the other solvers even comparable where the setup time of the preconditioner is a factor. As

[1] Details of problem selection are provided in the supplementary material

[2] A table of condition numbers is included in the supplementary material

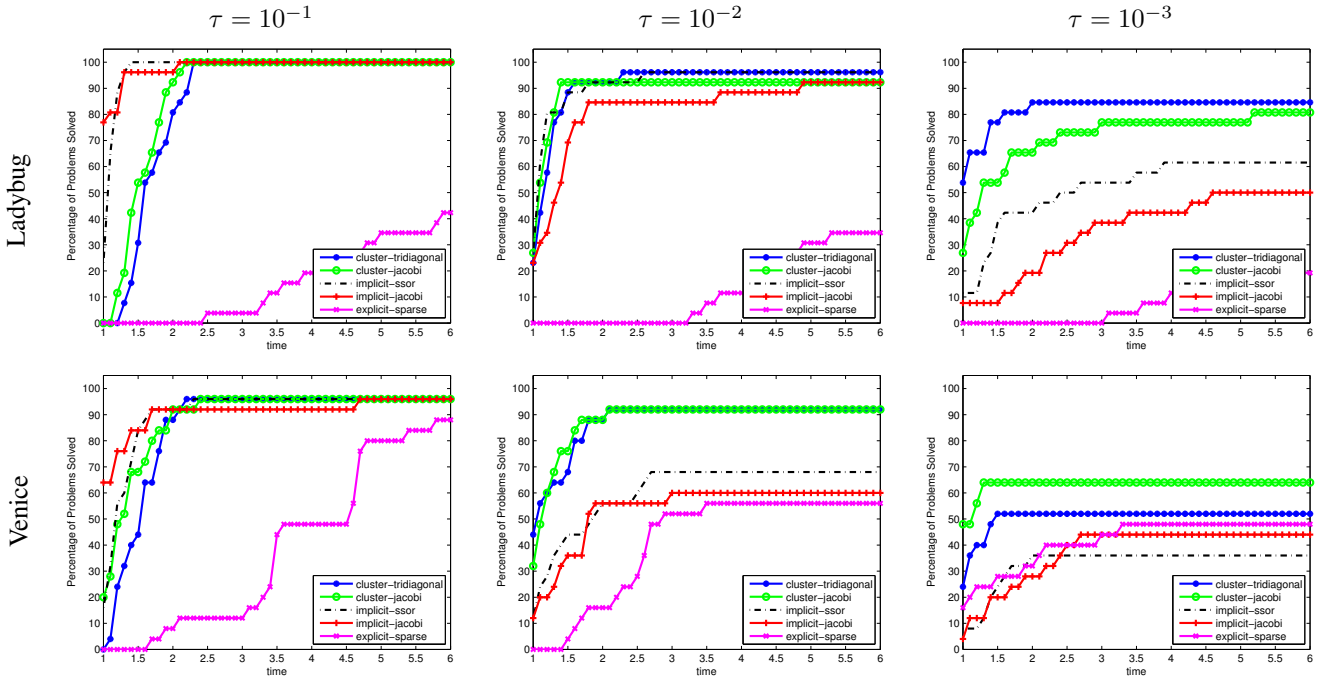

Figure 4. The Performance Profiles for the full bundle adjustment problems. Convergence is based on time.

we decrease τ, cluster-tridiagonal beats all other solvers, especially for the Venice problems, where we expect the clustering structure and some inter-clustering interaction to be present. The extra mass in cluster-tridiagonal means the time spent computing it is well worth it. It is also worth noting, that the ladybug is not a community photo collection dataset, rather it was collected by mounting cameras on a moving vehicle, and thus isn't likely to have significant cluster structure. This would explain the similar performance of cluster-jacobi and cluster-tridiagonal on this dataset.

4.3. Bundle Adjustment

We now look at the performance for these iterative solvers and explicit-sparse on the bundle adjustment problems. All the iterative solver based bundle adjustment algorithms were run inside an inexact LM loop [24], with the forcing sequence set to a constant $\eta_k = 0.1$ and the termination rule suggested by Nash & Sofer [15]. We use the same set of bundle adjustment problems as the previous experiment. As the ladybug problems are smaller they were run with a time budget of 600 seconds; while the Venice problems are denser and were run with a time budget of 1200 seconds. No other convergence tolerances were used. The solver time now accounts for time spent in computing the visibility, clustering and *degree-2* tree structures. Figure 4 show the performance profiles for $\tau = 10^{-1}, 10^{-2}$ and 10^{-3}.

As has been observed by others [1], explicit-sparse is dominated by the iterative solvers. Only for $\tau = 10^{-3}$ in the Venice dataset is explicit-sparse doing better than implicit-jacobi, the simplest of the four preconditioners. Also worth noting is that the visibility based preconditioners find better solutions faster for the vast majority of the problems. For $\tau = 10^{-1}$, the high setup cost of the more sophisticated preconditioner dominates and the simpler preconditioners are able to get to the solution faster. However once the tolerance is tightened both implicit-ssor and implicit-jacobi are unable to compete with the visibility based preconditioners either in solution quality or in time. For the same percentage of problems solved for $\tau = 10^{-2}$ the visibility based preconditioners are up to two times faster than the next best solver, and for $\tau = 10^{-3}$ this gap widens to up to 5 times.

Perhaps the most surprising fact is the similar performance of cluster-jacobi and cluster-tridiagonal and that the former at times, performs a bit better. We believe the reason for this is the rather large constant value of the forcing sequence $\eta_k = 0.1$, which only rarely requires the full power of the more sophisticated preconditioner to achieve that convergence threshold. Thus the extra time needed to construct the cluster-tridiagonal preconditioner is not always worth it. Having said that, even with their higher setup times and complexity both the visibility based preconditioners are a clear win over the existing state of the art and we recommend their use.

5. Conclusion

We have presented *Visibility Based Preconditioning*, a new technique for preconditioning the linear least squares problems arising in large scale Bundle adjustment problems. Using the visibility information in the scene, we cluster the cameras into tightly interacting clusters. These clusters form the basis of our block diagonal and block tridiagonal preconditioners. When combined with an inexact step LM algorithm, these preconditioners offer equal or better solution quality compared to the best available methods at 3-5x less execution time on problems from the BAL dataset.

The theory of preconditioning despite its long history is still in its infancy. For certain classes of matrices, e.g., Symmetric Diagonally Dominant matrices and matrices arising from Finite Element Methods, Support Graph Theory provides bounds on the condition number, but for general PSD matrices very few techniques exist. And when they do, e.g. the Cauchy-Bunyakowski-Schwarz inequality, they require explicit knowledge of the entries of the matrix. Even for well known preconditioners like Incomplete Cholesky the theory only deals with existence and breakdown of the preconditioner and not its performance. Thus, a theoretical analysis of visibility based preconditioning remains an interesting open problem.

Our experiments are limited to the BAL dataset. The sparsity patterns present in the BAL dataset are only a subset of the sparsity patterns encountered in real world SfM problems. A notable exception for example is the presence of camera blocks with long range interactions, e.g., aerial views of a scene that would correspond to near dense rows in the Schur Complement S. Visibility based clustering is not the right approach here, and better approximations can be obtained by treating such views separately.

And finally, while the preconditioners excel at solving linear least squares problems to high precision, when used with an inexact step LM algorithm, which uses fairly high tolerance convergence criterion, the extra work of setting up the block tridiagonal preconditioner is at times not worth the gain. In future work, we plan to explore better forcing sequences (η_k) and ways of reducing the setup time of the block tridiagonal preconditioner.

References

[1] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. In *ECCV*, pages 29–42, 2010.

[2] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building Rome in a day. In *ICCV*, 2009.

[3] M. Byröd, K. Åström, and S. Lund. Bundle adjustment using conjugate gradients with multiscale preconditioning. In *BMVC*, 2009.

[4] Y. Chen, T. Davis, W. Hager, and S. Rajamanickam. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *TOMS*, 35(3), 2008.

[5] F. Dellaert, J. Carlson, V. Ila, K. Ni, and C. E. Thorpe. Subgraph-preconditioned conjugate gradients for large scale slam. In *IROS*, pages 2566–2571, 2010.

[6] E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Math. Prog.*, 91(2):201–213, 2002.

[7] C. Engels, H. Stewenius, and D. Nister. Bundle adjustment rules. *Photogrammetric Computer Vision*, 2, 2006.

[8] J. Frahm et al. Building rome on a cloudless day. *ECCV*, pages 368–381, 2010.

[9] N. E. Gibbs, W. G. Poole, Jr., and P. K. Stockmeyer. A comparison of several bandwidth and profile reduction algorithms. *ACM Trans. Math. Softw.*, 2:322–330, December 1976.

[10] Y. Jeong, D. Nistér, D. Steedly, R. Szeliski, and I.-S. Kweon. Pushing the envelope of modern methods for bundle adjustment. In *CVPR*, pages 1474–1481, 2010.

[11] X. Li, C. Wu, C. Zach, S. Lazebnik, and J. Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. *ECCV (1)*, pages 427–440, 2008.

[12] M. Lourakis and A. Argyros. Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment. In *ICCV*, pages 1526–1531, 2005.

[13] M. A. Lourakis and A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *TOMS*, 36(1):1–30, 2009.

[14] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Generic and real-time structure from motion using local bundle adjustment. *Image and Vision Computing*, 27(8):1178–1193, 2009.

[15] S. Nash and A. Sofer. Assessing a search direction within a truncated-newton method. *Operations Research Letters*, 9(4):219–221, 1990.

[16] K. Ni, D. Steedly, and F. Dellaert. Out-of-core bundle adjustment for large-scale 3d reconstruction. In *ICCV*, 2007.

[17] J. Nocedal and S. Wright. *Numerical optimization*. Springer, 2000.

[18] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[19] I. Simon, N. Snavely, and S. M. Seitz. Scene summarization for online image collections. In *ICCV*, pages 1–8, 2007.

[20] N. Snavely, S. M. Seitz, and R. Szeliski. Skeletal graphs for efficient structure from motion. In *CVPR*, pages 1–8, 2008.

[21] D. Steedly and I. Essa. Propagation of innovative information in non-linear least-squares structure from motion. In *ICCV*, pages 223–229, 2001.

[22] D. Steedly, I. Essa, and F. Dellaert. Spectral partitioning for structure from motion. In *ICCV*, pages 996–103, 2003.

[23] B. Triggs, P. McLauchlan, H. R.I, and A. Fitzgibbon. Bundle Adjustment - A modern synthesis. In *Vision Algorithms'99*, pages 298–372, 1999.

[24] S. J. Wright and J. N. Holt. An inexact Levenberg-Marquardt method for large sparse nonlinear least squares. *J. Austral. Math. Soc. Ser. B*, 26(4):387–403, 1985.

[25] C. Wu, S. Agarwal, B. Curless, and S. Seitz. Multicore bundle adjustment. In *CVPR*, pages 3057–3064, 2011.

[26] D. C. B. Yong-Dian Jian and F. Dellaert. Generalized subgraph preconditioners for large-scale bundle adjustment. In *ICCV*, 2011.