

Bundle Adjustment in the Large

Sameer Agarwal^{1*}, Noah Snavely², Steven M. Seitz³, and Richard Szeliski⁴

¹ Google Inc.

² Cornell University

³ Google Inc. & University of Washington

⁴ Microsoft Research

Abstract. We present the design and implementation of a new inexact Newton type algorithm for solving large-scale bundle adjustment problems with tens of thousands of images. We explore the use of Conjugate Gradients for calculating the Newton step and its performance as a function of some simple and computationally efficient preconditioners. We show that the common Schur complement trick is not limited to factorization-based methods and that it can be interpreted as a form of preconditioning. Using photos from a street-side dataset and several community photo collections, we generate a variety of bundle adjustment problems and use them to evaluate the performance of six different bundle adjustment algorithms. Our experiments show that truncated Newton methods, when paired with relatively simple preconditioners, offer state of the art performance for large-scale bundle adjustment. The code, test problems and detailed performance data are available at <http://grail.cs.washington.edu/projects/bal>.

Key words: Structure from Motion, Bundle Adjustment, Preconditioned Conjugate Gradients

1 Introduction

Recent work in Structure from Motion (SfM) has demonstrated the possibility of reconstructing geometry from large-scale community photo collections [1–3]. Bundle adjustment, the joint non-linear refinement of camera and point parameters, is a key component of most SfM systems, and one which can consume a significant amount of time for large problems. As the number of photos in such collections continues to grow into the hundreds of thousands or even millions, the scalability of bundle adjustment algorithms has become a critical issue.

The basic mathematics of the bundle adjustment problem are well understood [4], and there is also a freely available high-quality implementation – SBA [5]. SBA is based on a dense Cholesky factorization of the reduced camera matrix. It has space complexity that is quadratic and time complexity that is cubic in the number of photos. While this works well for problems with a few hundred photos, for problems involving tens of thousands of photos, it is prohibitively expensive.

* Part of this work was done while the author was at University of Washington.

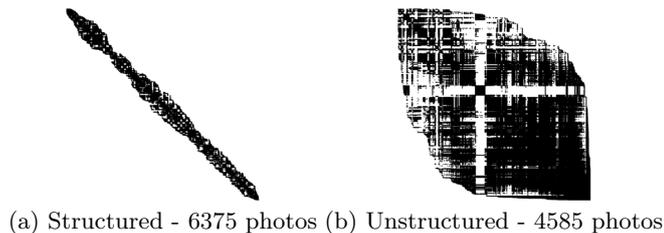


Fig. 1. Connectivity graphs for a structured dataset (captured from a moving truck) and a community photo collection (consisting of photos matching the search term “Dubrovnik” downloaded from Flickr). For each dataset, we show an adjacency matrix representation of the connectivity graph, where black indicates a connection between two photos.

With the exception of a few efforts [6–8, 1, 9], the development of large-scale bundle adjustment algorithms has not received significant attention in the computer vision community. We believe this is because until now, the most common sources of large SfM problems have been video and structured survey datasets such as street-level and aerial imagery. For these datasets, the connectivity graph—i.e., the graph in which each photo is a node, and two photos are connected if they are looking at the same part of the scene—is extremely sparse, and has a mostly band-diagonal structure with a large diameter. For instance, in the case of data acquired using a camera mounted on a vehicle driving down a street, there is little to no overlap between photos taken even a few seconds apart. Figure 1(a) shows one such graph. Thus, techniques that reduce the size of the bundle adjustment problem by focusing on the most recently modified part of the reconstruction are quite effective [7, 6].

Connectivity graphs of community photo collections are much less structured and have a significantly smaller diameter, as they tend to represent popular landmarks rather than a long, extended sequence of views. Figure 1(b) shows the graph for a set of photos of the city of Dubrovnik downloaded from Flickr. Compared to the structured dataset in Figure 1(a) which is 98% sparse with a mostly band diagonal structure, the graph for Dubrovnik is only 84% sparse, with a significantly more complex structure. This means that even though the dataset in Figure 1(a) has almost 1800 more photos than the dataset in Figure 1(b), the former requires 40x less time to find a sparse factorization of its reduced camera matrix than the latter.

In this paper, we present the design and implementation of a new inexact Newton type bundle adjustment algorithm, which uses substantially less time and memory than standard Schur complement based methods, without compromising on the quality of the solution. We explore the use of the Conjugate Gradients algorithm for calculating the Newton step and its performance as a function of some simple and computationally efficient preconditioners. We also show that the use of the Schur complement is not limited to factorization-based

methods, how it can be used as part of the Conjugate Gradients (CG) method without incurring the computational cost of actually calculating and storing it in memory, and how this use is equivalent to the choice of a particular preconditioner.

We present extensive experimental results on structured and unstructured datasets with a wide variety of problem complexity, and present recommendations based on these experiments. The code, test problems and detailed performance results from this paper are available at <http://grail.cs.washington.edu/projects/bal>.

The rest of the paper is organized as follow. We begin in Section 2 with a brief overview of the general nonlinear least squares problem, the Levenberg Marquardt (LM) algorithm, and the Schur complement trick. In Section 3, we introduce the inexact step LM algorithm, with a discussion of various methods for preconditioning the Conjugate Gradients (CG) algorithm in Section 4. Section 5 reports the results of our experiments and we conclude in Section 6 with a discussion.

2 Bundle Adjustment

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. This optimization problem is usually formulated as a non-linear least squares problem, where the error is the squared L_2 norm of the difference between the observed feature location and the projection of the corresponding 3D point on the image plane of the camera. However, we are not limited to using the L_2 norm; even when robust loss functions like Huber's norm are used, the problem can be cast as a re-weighted non-linear least squares problem [10]. Thus in what follows, we will use the term bundle adjustment to mean a particular class of non-linear least squares problems.

2.1 Levenberg Marquardt Algorithm

The Levenberg-Marquardt (LM) algorithm [11] is the most popular algorithm for solving non-linear least squares problems, and the algorithm of choice for bundle adjustment. In this section, we begin with a quick review of LM, and then describe the Schur complement trick that substantially reduces the computational complexity of LM applied to bundle adjustment. Several excellent references exist for the reader interested in more details of LM [11–14].

Let $x \in \mathbb{R}^n$ be an n -dimensional vector of variables, and $F(x) = [f_1(x), \dots, f_m(x)]^\top$ be a m -dimensional function of x . We are interested in solving the following optimization problem,

$$\min_x \frac{1}{2} \|F(x)\|^2 . \quad (1)$$

The Jacobian $J(x)$ of $F(x)$ is an $m \times n$ matrix, where $J_{ij}(x) = \partial_j f_i(x)$ and the gradient vector $g(x) = \nabla \frac{1}{2} \|F(x)\|^2 = J(x)^\top F(x)$.

The general strategy when solving non-linear optimization problems is to solve a sequence of approximations to the original problem [11]. At each iteration, the approximation is solved to determine a correction Δx to the vector x . For non-linear least squares, an approximation can be constructed by using the linearization $F(x + \Delta x) \approx F(x) + J(x)\Delta x$, which leads to the following linear least squares problem:

$$\min_{\Delta x} \frac{1}{2} \|J(x)\Delta x + F(x)\|^2 \quad (2)$$

Unfortunately, naïvely solving a sequence of these problems and updating $x \leftarrow x + \Delta x$ leads to an algorithm that may not converge. To get a convergent algorithm, we need to control the size of the step Δx . One way to do this is to introduce a regularization term:

$$\min_{\Delta x} \frac{1}{2} \|J(x)\Delta x + F(x)\|^2 + \mu \|D(x)\Delta x\|^2 . \quad (3)$$

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 μ is a non-negative parameter that controls the strength of regularization. It is straightforward to show that the step size $\|\Delta x\|$ is inversely related to μ . LM updates the value of μ at each step based on how well the Jacobian $J(x)$ approximates $F(x)$. The quality of this fit is measured by the ratio of the actual decrease in the objective function to the decrease in the value of the linearized model $L(\Delta x) = \frac{1}{2} \|J(x)\Delta x + F(x)\|^2$. This kind of reasoning is the basis of Trust-region methods, of which LM is an early example [11].

The dominant computational cost in each iteration of the LM algorithm is the solution of the linear least squares problem (3). For general, small to medium scale least squares problems, the recommended method for solving (3) is using the the QR factorization [13]. However, the bundle adjustment problem has a very special structure, and a more efficient scheme for solving (4) can be constructed.

2.2 The Schur Complement Trick

We begin by introducing the regularized Hessian matrix $H_\mu(x) = J(x)^\top J(x) + \mu D(x)^\top D(x)$. It is easy to show that for $\mu D(x) > 0$, H_μ is a symmetric positive definite matrix and the solution to (3) can be obtained by solving the *normal equations*:

$$H_\mu(x)\Delta x = -g(x) . \quad (4)$$

Now, suppose that the SfM problem consists of p cameras and q points and the variable vector x has the block structure $x = [y_1, \dots, y_p, z_1, \dots, z_q]$. Where, y and z correspond to camera and point parameters, respectively. Further, let the camera blocks be of size c and the point blocks be of size s (for most problems $c = 6-9$ and $s = 3$).

In most cases, a key characteristic of the bundle adjustment problem is that there is no term f_i that includes two or more camera or point blocks. In other

words, each term $f_i(x)$ in the objective function can be re-written as $f_i(x) = f_i(y_{(i)}, z_{(i)})$, where, $y_{(i)}$ and $z_{(i)}$ are the camera and point blocks that occur in the i^{th} term. This in turn implies that the matrix H_μ is of the form

$$H_\mu = \begin{bmatrix} B & E \\ E^\top & C \end{bmatrix}, \quad (5)$$

where, $B \in \mathbb{R}^{pc \times pc}$ is a block diagonal matrix with p blocks of size $c \times c$ and $C \in \mathbb{R}^{qs \times qs}$ is a block diagonal matrix with q blocks of size $s \times s$. $E \in \mathbb{R}^{pc \times qs}$ is a general block sparse matrix, with a block of size $c \times s$ for each observation. Let us now block partition $\Delta x = [\Delta y, \Delta z]$ and $-g = [v, w]$ to restate (4) as the block structured linear system

$$\begin{bmatrix} B & E \\ E^\top & C \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} v \\ w \end{bmatrix}, \quad (6)$$

and apply Gaussian elimination to it. As we noted above, C is a block diagonal matrix, with small diagonal blocks of size $s \times s$. Thus, calculating the inverse of C by inverting each of these blocks is an extremely cheap, $O(q)$ algorithm. This allows us to eliminate Δz by observing that $\Delta z = C^{-1}(w - E^\top \Delta y)$, giving us

$$[B - EC^{-1}E^\top] \Delta y = v - EC^{-1}w. \quad (7)$$

The matrix

$$S = B - EC^{-1}E^\top, \quad (8)$$

is the Schur complement of C in H_μ . It is also known as the *reduced camera matrix*, because the only variables participating in (7) are the ones corresponding to the cameras. $S \in \mathbb{R}^{pc \times pc}$ is a block structured symmetric positive definite matrix, with blocks of size $c \times c$. The block S_{ij} corresponding to the pair of images i and j is non-zero if and only if the two images observe at least one common point.

Now, (6) can be solved by first forming S , solving for Δy , and then back-substituting Δy to obtain the value of Δz . Thus, the solution of what was an $n \times n$, $n = pc + qs$ linear system is reduced to the inversion of the block diagonal matrix C , a few matrix-matrix and matrix-vector multiplies, and the solution of block sparse $pc \times pc$ linear system (7). For almost all problems, the number of cameras is much smaller than the number of points, $p \ll q$, thus solving (7) is significantly cheaper than solving (6). This is the *Schur complement trick* [15].

This still leaves open the question of solving (7). The method of choice for solving symmetric positive definite systems exactly is via the Cholesky factorization [16] and depending upon the structure of the matrix, there are, in general, two options. The first is direct factorization, where we store and factor S as a dense matrix [16]. This method has $O(p^2)$ space complexity and $O(p^3)$ time complexity and is only practical for problems with up to a few hundred cameras. But, S is typically a fairly sparse matrix, as most images only see a small fraction of the scene. This leads us to the second option: sparse direct methods. These methods store S as a sparse matrix, use row and column re-ordering

algorithms to maximize the sparsity of the Cholesky decomposition, and focus their compute effort on the non-zero part of the factorization [17]. Sparse direct methods, depending on the exact sparsity structure of the Schur complement, allow bundle adjustment algorithms to significantly scale up over those based on dense factorization.

This however is not enough for community photo collections, where the size and sparsity structure of S (e.g. Figure 1) is such that even constructing it is a significant expense, and factoring it leads to near dense Cholesky factors. Hence we would like to find alternatives that do not depend on the construction, storage, and factorization of S and yet give good performance on large problems.

3 A Truncated Newton Solver

The factorization methods described above are based on computing an exact solution of (3). But it is not clear if an exact solution of (3) is necessary at each step of the LM algorithm to solve (1). In fact, we have already seen evidence that this may not be the case, as (3) is itself a regularized version of (2). Indeed, it is possible to construct non-linear optimization algorithms in which the linearized problem is solved approximately. These algorithms are known as inexact Newton or truncated Newton methods [11].

An inexact Newton method requires two ingredients. First, a cheap method for approximately solving systems of linear equations. Typically an iterative linear solver like the Conjugate Gradients method is used for this purpose [11]. Second, a termination rule for the iterative solver. A typical termination rule is of the form

$$\|H_\mu(x)\Delta x + g(x)\| \leq \eta_k \|g(x)\|. \quad (9)$$

Here, k indicates the LM iteration number and $0 < \eta_k < 1$ is known as the forcing sequence. Wright & Holt [18] prove that a truncated LM algorithm that uses an inexact Newton step based on (9) converges for any sequence $\eta_k \leq \eta_0 < 1$ and the rate of convergence depends on the choice of the forcing sequence η_k .

4 Preconditioned Conjugate Gradients

The convergence rate of CG for solving (4) depends on the distribution of eigenvalues of H_μ [19]. A useful upper bound is $\sqrt{\kappa(H_\mu)}$, where, $\kappa(H_\mu)$ is the condition number of the matrix H_μ . For most bundle adjustment problems, $\kappa(H_\mu)$ is high and a direct application of CG to (4) results in extremely poor performance.

The solution to this problem is to replace (4) with a *preconditioned* system. Given a linear system, $Ax = b$ and a preconditioner M the preconditioned system is given by $M^{-1}Ax = M^{-1}b$. The resulting algorithm is known as Preconditioned Conjugate Gradients algorithm (PCG) and its worst case complexity now depends on the condition number of the *preconditioned* matrix $\kappa(M^{-1}A)$.

The key computational cost in each iteration of PCG is the evaluation of the matrix vector product $\beta = A\alpha$ and solution of the linear system $M\phi = \psi$

for arbitrary vectors α and ψ . Thus, for each iteration of PCG to be efficient, M should be cheaply invertible and for the number of iterations of PCG to be small, the condition number $\kappa(M^{-1}A)$ should be as small as possible. The ideal preconditioner would be one for which $\kappa(M^{-1}A) = 1$. $M = A$ achieves this, but it is not a practical choice, as applying this preconditioner would require solving a linear system equivalent to the unpreconditioned problem. So how does one choose an effective preconditioner that is cheap to invert and results in a significant reduction of the condition number of the preconditioned matrix?

The simplest of all preconditioners is the diagonal or Jacobi preconditioner, *i.e.*, $M = \text{diag}(A)$, which for block structured matrices like H_μ can be generalized to the block Jacobi preconditioner. H_μ also has the special property that its diagonal blocks B and C are themselves block diagonal matrices. This property makes the block Jacobi preconditioner

$$M_J = \begin{bmatrix} B & 0 \\ 0 & C \end{bmatrix}. \quad (10)$$

the optimal block diagonal preconditioner for H_μ [20].

Another option is to apply PCG to the reduced camera matrix S instead of H_μ . One reason to do this is that S is a much smaller matrix than H_μ , but more importantly, it can be shown that $\kappa(S) \leq \kappa(H_\mu)$. There are two obvious choices for block diagonal preconditioners for S . The matrix B [21] and the block diagonal $\mathcal{D}(S)$ of S , *i.e.* the block Jacobi preconditioner for S .

Consider now, the generalized Symmetric Successive Over-relaxation (SSOR) preconditioner for H_μ ,

$$M_\omega(P) = \begin{bmatrix} P & \omega E \\ 0 & C \end{bmatrix} \begin{bmatrix} P^{-1} & 0 \\ 0 & C^{-1} \end{bmatrix} \begin{bmatrix} P \\ \omega E^\top & C \end{bmatrix}, \quad (11)$$

where P is some easily invertible matrix and $0 \leq \omega < 2$ is a scalar parameter.

Observe that for $\omega = 0$, $M_0(B) = M_J$ is the block Jacobi preconditioner. More interestingly, for $\omega = 1$, it can be shown that using $M_1(P)$ as a preconditioner for H_μ is exactly equivalent to using the matrix P as a preconditioner for the reduced camera matrix S [19]. This means that for $P = I$ using $M_1(I)$ as a preconditioner for H_μ is the same as running pure CG on S and we can run PCG on S with preconditioners B and $\mathcal{D}(S)$ by using $M_1(B)$ and $M_1(\mathcal{D}(S))$ as preconditioners for H_μ . Thus, the Schur complement which started out its life as a way of specifying the order in which the variables should be eliminated from H_μ when solving (4) exactly, returns to the scene as a generalized SSOR preconditioner when solving the same linear system iteratively.

As discussed earlier, the cost of forming and storing the Schur complement S can be prohibitive for large problems. Indeed, for an inexact Newton solver that uses PCG on S , almost all of its time is spent in constructing S ; the time spent inside the PCG algorithm is negligible in comparison.

Because PCG only needs access to S via its product with a vector, one way to evaluate Sx is to use (11) for $\omega = 1$. However we can do even better. Observe

that,

$$x_1 = E^\top x, \quad x_2 = C^{-1}x_1, \quad x_3 = Ex_2, \quad x_4 = Bx, \quad Sx = x_4 - x_3. \quad (12)$$

Thus, we can run PCG on S with the same computational effort per iteration as PCG on H_μ , while reaping the benefits of a more powerful preconditioner. Even if we decide to use the block Jacobi preconditioner $\mathcal{D}(S)$, it can be constructed at a cost that is linear in the number of observations $O(m)$ and memory cost that is linear in the number of cameras - $O(p)$. Both of these are substantially less than the cost of computing and storing the full matrix S .

Equation (12) is closely related to *Domain Decomposition methods* for solving large linear systems that arise in structural engineering and partial differential equations. In the language of Domain Decomposition, each point in the SFM problem is a domain, and the cameras form the interface between these domains. The iterative solution of the Schur complement then falls within the sub-category of techniques known as Iterative Sub-structuring [19, 22].

5 Experimental Evaluation

5.1 Algorithms

We compared the performance of six bundle adjustment algorithms: explicit-direct, explicit-sparse, normal-jacobi, explicit-jacobi, implicit-jacobi and implicit-ssor. The first two methods are exact step LM algorithms, and the remaining four are inexact step LM algorithms. `explicit-direct`, `explicit-sparse` and `explicit-jacobi` *explicitly* construct the Schur complement matrix S and solve (7) using dense factorization, sparse direct factorization, and PCG using the block Jacobi preconditioner $\mathcal{D}(S)$ respectively. `normal-jacobi` uses PCG on H_μ with the block Jacobi preconditioner M_J . `implicit-jacobi` and `implicit-ssor` run PCG on S using the block Jacobi preconditioner $\mathcal{D}(S)$ and B respectively. Unlike `explicit-jacobi` they use (12) to *implicitly* evaluate matrix vector products with S .

Assuming that all the algorithms store H_μ in the same format, the difference between their memory usages depends on how they use the Schur complement S . `implicit-jacobi`, `implicit-ssor` and `normal-jacobi` do not compute or store S , and therefore require the least amount of memory. `explicit-direct` is the most expensive of the three as it uses $O(p^2)$ memory to store and factor S . `explicit-sparse` and `explicit-jacobi` are less expensive as they store S as a sparse matrix, and thus their storage requirements scale with the sparsity of S . `explicit-sparse` requires additional storage to store the Cholesky factorization of S , and the amount of memory required is a function of the sparsity structure of S and not just the number of non-zero entries.

For each solver, LM was run for a maximum of 50 iterations, i.e. (3) was solved 50 times. After each LM iteration the step Δx may or may not be accepted, depending on whether it leads to a better solution. Inside each iteration of LM, PCG was run for a minimum of 10 iterations, and terminated when either $\|H_\mu(x)\Delta x + g(x)\| \leq \eta_k \|g(x)\|$ was satisfied or a 1000 iterations were performed.

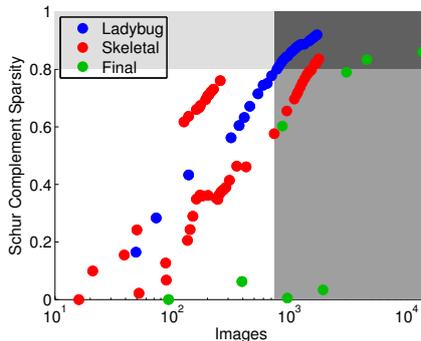


Fig. 2. *Datasets.* This scatter plot shows each of the datasets in our testbed, colored according to type (Ladybug, Skeletal, Final). The x -axis is the number of images in the problem and the y -axis is the sparsity of the Schur complement matrix S . The background of the plot is shaded according to the characteristics of the problem: small and dense (white), then in increasing gray-level, small and sparse, large and dense, and large and sparse.

The forcing sequence η_k was set to a constant $\eta_k = 0.1$. At the beginning of LM, the square root of the diagonal of the matrix $J(x_0)^\top J(x_0)$ is estimated and used as a scaling matrix for the variables. This is a standard method for normalizing all the variables in a problem [23] and is necessary as some parameters, (e.g., radial distortion), are up to 20 orders of magnitude more sensitive than others (e.g., rotation). For the factorization methods, especially CHOLMOD, this improves numerical stability. For the iterative solvers, this is equivalent to applying the Jacobi preconditioner before any of the other preconditioners are used.

All six algorithms were implemented as part of a single C++ code base. We use GotoBLAS2 [24] for dense linear algebra and CHOLMOD [17] for sparse Cholesky factorization. All experiments were performed on a workstation with dual Quad-core CPUs clocked at 2.27Ghz with 48GB RAM running a 64-bit Linux operating system.

5.2 Datasets

We experimented with two sources of data:

1. Images captured at a regular rate using a Ladybug camera mounted on a moving vehicle. Image matching was done by exploiting the temporal order of the images and the GPS information captured at the time of image capture.
2. Images downloaded from Flickr.com and matched by the authors of [3]. We used images from Trafalgar Square and the cities of Dubrovnik, Venice, and Rome.

For Flickr photographs, the matched images were decomposed into a skeletal set (i.e., a sparse core of images) and a set of leaf images [1]. The skeletal set was reconstructed first, then the leaf images were added to it via resectioning

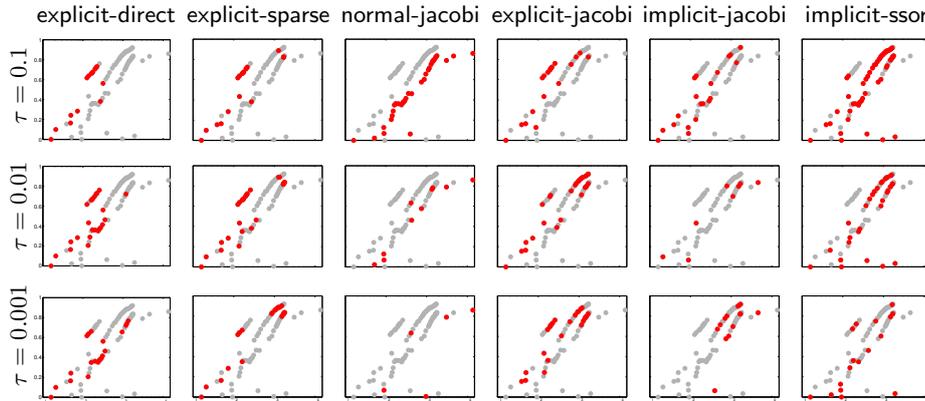


Fig. 3. Performance analysis. Each column in this set of plots corresponds to one of six algorithms, and each row corresponds to one of three tolerances τ . For each solver (column), a point is colored red if the solver was declared a winner for the given tolerance, and gray otherwise. Winning solvers are the ones for which the relative decrease in the RMS error $(r_k - r^*) / (r_0 - r^*) \leq \tau$ in the least amount of time (there can be more than one such solver). The axes of the individual plots are the same as in Figure 2.

followed by triangulation of the remaining 3D points. The skeletal sets and the Ladybug datasets were reconstructed incrementally using a modified version of Bundler [25], which was instrumented to dump intermediate unoptimized reconstructions to disk. This gave rise to the *Skeletal* and the *Ladybug* problems. We refer to the bundle adjustment problems obtained after adding the leaf images to the skeletal set and triangulating the remaining points as the *Final* problems. For each dataset we use a nine parameter camera model (6 for pose, 1 for focal length and 2 for radial distortion).

Figure 2 plots the three types of problems. The x -axis is the number of images on a log-scale and the y -axis is the sparsity of the S matrix. The *Ladybug* (blue) set has small dense problems and large sparse problems with almost band diagonal sparsity. The *Skeletal* (red) set has small dense, and medium to large sparse problems with random sparsity. The *Final* (green) set has large problems with low to high sparsity. Their size and sparsity can pose significant challenges for state of the art algorithms. Complete details on the properties of each problem used in the experiments can be found on the project website.

5.3 Analysis

Detailed statistics on the performance of all algorithms are available on the project website. Here we summarize the broad trends in the data.

We compare solvers across problems by looking at how often they are the first one to improve the RMS error by a certain fraction. Concretely, for each solver and problem, let $r_k = \sqrt{\sum_i^m f_i^2(x_k) / m}$ denote the RMS error at end of iteration

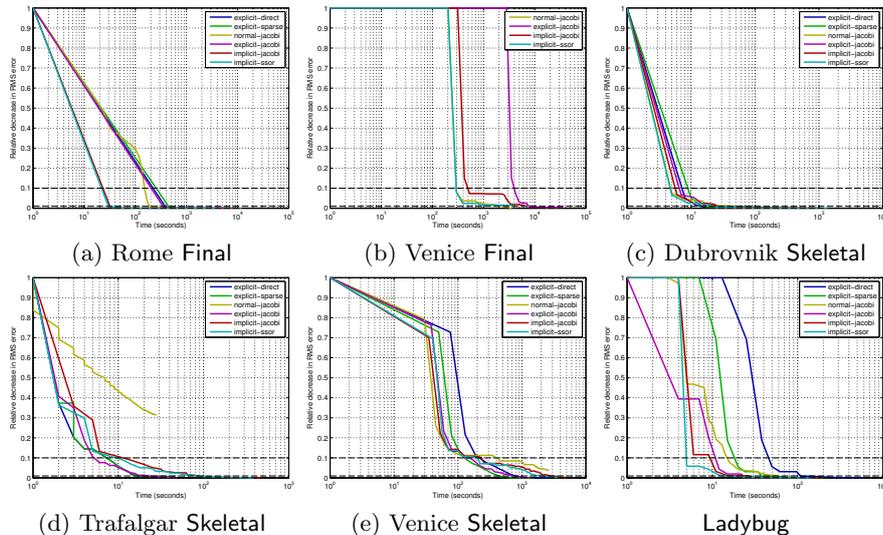


Fig. 4. A sampling of run time plots. In each plot, the x -axis is time on a log scale, and the y -axis is the relative decrease in the RMS error $(r_k - r^*)/(r_0 - r^*)$. The three black dashed horizontal lines in each plot correspond to the three tolerances, *i.e.*, $\tau = 0.1, 0.01$ and 0.001 . Note that `explicit-direct` and `explicit-sparse` are missing from the Venice Final plot as they ran out of memory.

k and let r^* denote the minimum RMS error across all solvers for that problem. Then, for a given tolerance τ , we find the solvers for which $(r_k - r^*)/(r_0 - r^*) \leq \tau$ is satisfied in the least amount of time. We do this for three exponentially tighter tolerances, $\tau = 0.1, 0.01, 0.001$. Figure 3 plots the results. The three rows correspond to the values of τ and the six columns correspond to different bundle adjustment algorithms. As in Figure 2, for each plot, x -axis is the number of images on a log scale, and y -axis is the sparsity of the Schur complement matrix S . In each plot, we plot all the problems in light grey, and then in red, the problems for which that solver at that tolerance level was one of the winners⁵.

From Figure 3, we observe that for problems with up to a few hundred images and all three tolerances, `explicit-direct` offers consistently good performance. State of the art BLAS and LAPACK libraries on multicore systems have excellent performance, and for small to moderate sized matrices, an exact step LM with a dense Cholesky solver is hard to beat. This explains the continuing popularity and success of SBA [5].

For larger problems and high tolerance values $\tau = 0.1$, both `normal-jacobi` and `implicit-ssor` do well, with `implicit-ssor` working on a much wider variety of problems. As the value τ decreases, the performance of `normal-jacobi` rapidly degrades, indicating that the quality of preconditioning is not good enough to produce high quality Newton steps in a short amount of time. On the other

⁵ Since time is measured in seconds, there may be more than one such solver.

hand as τ is decreased, `explicit-jacobi` which is the most expensive of the iterative solvers, becomes a viable candidate with the block Jacobi preconditioning of S starting to show its benefits. `implicit-ssor` beats `explicit-jacobi` when S has low sparsity. This is not surprising, as the cost of computing a nearly dense reduced camera matrix becomes a significant factor, where as `implicit-ssor` is able to avoid this extra computational burden.

A closer examination of the data reveals that despite an overall degradation in performance, `normal-jacobi` continues to work well for the larger problems in the `Final` set. We believe this is because of the structure of the `Skeletal` sets algorithm. After the skeletal set has been reconstructed, the geometric core of the reconstruction is quite rigid and stable. The error in the reconstruction after the leaf images have been added is mostly local and no major global changes that span the entire reconstruction are expected. Therefore, the simple block Jacobi preconditioner captures the structure of H_μ quite well and at a substantially less computational cost than any other preconditioner.

It is also worth observing that for some of the problems, as the value of τ is decreased, factorization-based solvers become more competitive. This is expected, as lower values of τ demand that the LM algorithm take higher quality steps at each iteration. In this regime, the higher cost of the exact step algorithms, at least for the smaller problems, wins over the increased iteration complexity of the inexact step algorithms. Better performance for inexact step algorithms will require more sophisticated forcing sequence η_k and preconditioners.

There were two surprises. First, the discrepancy in the performance of `explicit-jacobi` and `implicit-jacobi`. In exact arithmetic, these two algorithms should return exactly the same answer, but that is not the case in practice. A closer look at the data revealed that for the same linear system, the two methods resulted in different number of iterations and answers, sometimes significantly so, indicating numerical instability in `implicit-jacobi` which merits further investigation. Second, `explicit-sparse` did not emerge as a clear winner in any of the problem categories. Either the problems were too small for the additional setup cost and the more complicated algorithm used in `CHOLMOD` to beat dense Cholesky factorization, or the problems were large enough that the exact factorization algorithms, sparse and dense, were beaten by the inexact step algorithms.

In summary, we observe that for large scale problems, the iterative methods are a significant memory and time win over Cholesky factorization-based methods. Particularly for `Final` problems, this can be the difference between being able to solve the problem or not, as evidenced by the large `Venice` example. But even for medium sized problems involving a few thousand images, the iterative solvers are up to an order of magnitude faster while consuming 3-5 times less memory. For the sparse problems in the `Ladybug` and `Skeletal` datasets, the advantage is usually in terms of memory and simplicity of implementation rather than time, as the cost of exact factorization is offset by its superior quality. However, we must remember that these experiments were performed on state of the art workstations with much more RAM than is commonly available today, which makes the memory usage of the iterative methods even more attractive.

For small to medium problems, we recommend the use of a dense Cholesky-based LM algorithm. For larger problems, the situation is more complicated and there is no one clear answer. Both `implicit-ssor` and `explicit-jacobi` offer competitive solvers, with `implicit-ssor` being preferred for problems with lower sparsity and `explicit-jacobi` for problems with high sparsity. We hope that once the cause of numerical instability in `implicit-jacobi` can be understood and rectified, it will offer a memory efficient solver that bridges the gap between these two solvers and works on large bundle adjustment problems, independent of their sparsity.

6 Discussion

The classical solution to bundle adjustment is based on exploiting the primary sparsity structure of the problem to form a Schur complement and factoring it [26, 4, 10]. With the exception of a few recent attempts [27, 28], it has remained the dominant method for doing bundle adjustment. While suitable for problems with a few hundred images, this method does not scale to larger problems with thousands of images. In this paper, we have shown with the help of an extensive test suite of large scale bundle adjustment problems that a truncated Newton style LM algorithm coupled with a simple preconditioner delivers state of the art performance at a fraction of the time and memory cost of methods based on factoring the Schur complement.

Going forward, the preconditioners considered in this paper are relatively simple but we hope that the identification with domain decomposition methods opens up the possibility of using much more sophisticated preconditioners developed in the structural engineering literature [19, 22]. Numerical stability is another critical issue. As we noted earlier, even though `explicit-jacobi` and `implicit-jacobi` are algebraically equivalent algorithms, they show problem-dependent numerical behavior. A more thorough development that accounts for the numerical stability of evaluating the matrix-vector products using the explicit and the implicit schemes is needed.

Acknowledgements The authors are grateful to Drew Steedly and David Nister for useful discussions and Kristin Branson for reading multiple drafts of the paper.

This work was supported in part by National Science Foundation grant IIS-0811878, SPAWAR, the Office of Naval Research, the University of Washington Animation Research Labs, Google, Intel, and Microsoft.

References

1. Snavely, N., Seitz, S.M., Szeliski, R.: Skeletal graphs for efficient structure from motion. In: CVPR. (2008) 1–8
2. Li, X., Wu, C., Zach, C., Lazebnik, S., Frahm, J.: Modeling and recognition of landmark image collections using iconic scene graphs. ECCV (1) (2008) 427–440

3. Agarwal, S., Snavely, N., Simon, I., Seitz, S.M., Szeliski, R.: Building Rome in a day. In: ICCV. (2009)
4. Triggs, B., McLauchlan, P., R.I. H., Fitzgibbon, A.: Bundle Adjustment - A modern synthesis. In: Vision Algorithms'99. (1999) 298–372
5. Lourakis, M., Argyros, A.A.: SBA: A software package for generic sparse bundle adjustment. TOMS **36** (2009) 2
6. Mouragnon, E., Lhuillier, M., Dhome, M., Dekeyser, F., Sayd, P.: Generic and real-time structure from motion using local bundle adjustment. Image and Vision Computing **27** (2009) 1178–1193
7. Steedly, D., Essa, I.: Propagation of innovative information in non-linear least-squares structure from motion. In: ICCV. (2001) 223–229
8. Ni, K., Steedly, D., Dellaert, F.: Out-of-core bundle adjustment for large-scale 3d reconstruction. In: ICCV. (2007)
9. Steedly, D., Essa, I., Dellaert, F.: Spectral partitioning for structure from motion. In: ICCV. (2003) 996–103
10. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge University Press (2003)
11. Nocedal, J., Wright, S.: Numerical optimization. Springer (2000)
12. More, J.: The Levenberg-Marquardt algorithm: implementation and theory. Lecture Notes in Math. **630** (1977) 105–116
13. Björck, A.: Numerical methods for least squares problems. SIAM (1996)
14. Madsen, K., Nielsen, H., Tingleff, O.: Methods for non-linear least squares problems. (2004)
15. Brown, D.C.: A solution to the general problem of multiple station analytical stereotriangulation. Technical Report 43, Patrick Airforce Base, Florida (1958)
16. Trefethen, L., Bau, D.: Numerical linear algebra. SIAM (1997)
17. Chen, Y., Davis, T., Hager, W., Rajamanickam, S.: Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. TOMS **35** (2008)
18. Wright, S.J., Holt, J.N.: An inexact Levenberg-Marquardt method for large sparse nonlinear least squares. J. Austral. Math. Soc. Ser. B **26** (1985) 387–403
19. Saad, Y.: Iterative methods for sparse linear systems. SIAM (2003)
20. Elsner, L.: A note on optimal block-scaling of matrices. Numer. Math. **44** (1984) 127–128
21. Mandel, J.: On block diagonal and Schur complement preconditioning. Numer. Math. **58** (1990) 79–93
22. Mathew, T.: Domain decomposition methods for the numerical solution of partial differential equations. Springer Verlag (2008)
23. Dennis Jr, J., Gay, D., Welsch, R.: Algorithm 573: NL2SOLan adaptive nonlinear least-squares algorithm [E4]. TOMS **7** (1981) 369–383
24. Goto, K., Van De Geijn, R.: High-performance implementation of the level-3 blas. TOMS **35** (2008) 1–14
25. Snavely, N., Seitz, S.M., Szeliski, R.: Photo Tourism: Exploring photo collections in 3D. TOG **25** (2006) 835–846
26. Engels, C., Stewenius, H., Nister, D.: Bundle adjustment rules. Photogrammetric Computer Vision **2** (2006)
27. Lourakis, M., Argyros, A.: Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment. In: ICCV. (2005) 1526–1531
28. Byröd, M., Åström, K., Lund, S.: Bundle adjustment using conjugate gradients with multiscale preconditioning. (2009)