Conjugate Gradient Bundle Adjustment

Martin Byröd and Kalle Åström*

Centre for Mathematical Sciences, Lund University, Lund, Sweden
{byrod,kalle}@maths.lth.se

Abstract. Bundle adjustment for multi-view reconstruction is traditionally done using the Levenberg-Marquardt algorithm with a direct linear solver, which is computationally very expensive. An alternative to this approach is to apply the conjugate gradients algorithm in the inner loop. This is appealing since the main computational step of the CG algorithm involves only a simple matrix-vector multiplication with the Jacobian. In this work we improve on the latest published approaches to bundle adjustment with conjugate gradients by making full use of the least squares nature of the problem. We employ an easy-to-compute QR factorization based block preconditioner and show how a certain property of the preconditioned system allows us to reduce the work per iteration to roughly half of the standard CG algorithm.

1 Introduction

Modern structure from motion (SfM) systems, which compute cameras and 3D structure from images, rely heavily on bundle adjustment. Bundle adjustment refers to the iterative refinement of camera and 3D point parameters based on minimization of the sum of squared reprojection errors and hence belong to the class of non-linear least squares problems. Bundle adjustment is important both as a final step to polish off a rough reconstruction obtained by other means as well as a way of avoiding accumulation of errors during an incremental reconstruction procedure.

A recent trend in SfM applications is to move from small and medium size setups to large scale problems (typically in the order 10^3-10^4 cameras or more), *cf.* [1,2,3,4]. Bundle adjustment in general has $\mathcal{O}(N^3)$ complexity, where N is the number of variables in the problem [5]. In the large scale-range of the spectrum, bundle adjustment hence starts to become a major computational bottleneck.

The standard algorithm for bundle adjustment is Levenberg-Marquardt with Cholesky factorization to solve the normal equations [6,7]. An interesting alternative to this is the method of conjugate gradients (CG), which has recently been applied in the context of bundle adjustment [8,1]. The conjugate gradient

* The research leading to these results has received funding from the swedish strategic research project Elliit, the European Research Council project Globalvision, the Swedish research council project Polynomial Equations and the Swedish Strategic Foundation projects ENGROSS and Wearable Visual Systems.

K. Daniilidis, P. Maragos, N. Paragios (Eds.): ECCV 2010, Part II, LNCS 6312, pp. 114–127, 2010.

algorithm can be applied both as a non-linear optimization algorithm replacing Levenberg-Marquardt or as an iterative linear solver for the normal equations, where the latter approach seems to be the right choice for non-linear least squares.

In [8], which is of a speculative nature, the graph structure of the problem was used to derive multiscale preconditioners for bundle adjustment and conjugate gradients. While the authors show some promising preliminary results, they were not able to overcome some fundamental limitations yielding the preconditioners themselves expensive to construct and apply. In this paper we take a more straightforward approach and make use of the inherent sparsity structure of the problem to design a light-weight matrix based preconditioner. Doing bundle adjustment with conjugate gradients and block diagonal preconditioning was mentioned in the work of Agarwal *et al* on large scale structure from motion [1]. Compared to the work of Agarwal *et al*, where essentially the standard conjugate gradient algorithm was applied to the normal equations, the main novelty of this work is to make explicit use of the least squares nature of the problem for maximum efficiency and precision. Here we make use of the least squares property in several ways. Our main contributions are:

- We apply the CGLS algorithm (instead of the standard CG algorithm), which allows us to avoid forming $J^T J$, where J is the Jacobian, thus saving time and space and improving precision.
- A QR factorization based block-preconditioner, which can be computed in roughly the same time it takes to compute the Jacobian.
- We note that the preconditioned system has "property A" in the sense of Young [9], allowing us to cut the work per iteration in roughly half.
- An experimental study which sheads some new light on when iterative solvers for the normal equations may be successfully used.

2 Problem Formulation

We consider a setup with m cameras $C = (C_1, \ldots, C_m)$ observing n points $U = (U_1, \ldots, U_n)$ in 3D space. An index set $\mathcal{I}$ keeps track of which points are seen in which views by $(i, j) \in \mathcal{I}$ *iff* point j is seen in image i. If all points are visible in all views then there are mn projections. This is not the case in general and we denote the number of image points $n_r = |\mathcal{I}|$. The observation model $f(C_i, U_j)$ yields the 2D image coordinates of the point U_j projected into the view C_i. The input data is a set of observations $\hat{f}_{ij}$ such that

$$\hat{f}_{ij} = f(C_i, U_j) + \eta_{ij}, \tag{1}$$

where η_{ij} is measurement noise drawn from a suitable distribution. The unknown parameters $x = (C, U)$ are now estimated given the set of observations by adjusting them to produce a low re-projection error as realized in the following non-linear least squares problem

$$x^* = \underset{x}{\operatorname{argmin}} \sum_{(i,j)\in\mathcal{I}} \|\hat{f}_{ij} - f(C_i, U_j)\|^2. \tag{2}$$

The standard algorithm for dealing with non-linear least squares problem is the Gauss-Newton algorithm. Rewriting (2), our task is to solve the following optimization problem

$$x^* = \underset{x}{\operatorname{argmin}} \|r(x)\|^2, \tag{3}$$

where r is the vector of individual residuals $r_{ij} = f(C_i, U_j) - \hat{f}_{ij}$.

A first order expansion inside the norm in the non-linear sum of squares expression yields a linear least squares problem

$$\min_x \|r(x+\delta x)\|^2 \approx \|r(x) + J(x)\delta x\|^2, \tag{4}$$

where solving for δx in the usual least squares sense yields the equation for the update step:

$$J(x)^T J(x)\delta x = -J(x)^T r(x). \tag{5}$$

If the system matrix $J^T J$ does not have full rank, or if there are significant non-linearities then it is common to add a *damping* term λI to $J^T J$ and solve the damped system

$$(J^T J + \lambda I)\delta x = -J^T r. \tag{6}$$

We use the strategy by Nielsen [10] to update λ based on how well the decrease in error agrees with the decrease predicted by the linear model.

In the case of bundle adjustment, it is possible to partition the Jacobian into a camera part J_C and a point part J_P as $J = [J_C \ J_P]$, which gives

$$J^T J = \begin{bmatrix} J_C^T J_C & J_C^T J_P \\ J_P^T J_C & J_P^T J_P \end{bmatrix} = \begin{bmatrix} U & W \\ W^T & V \end{bmatrix}, \tag{7}$$

where U and V are block diagonal. One can now apply block wise Gaussian elimination producing the simplified system

$$(U - WV^{-1}W^T)\delta x_C = b_C - WV^{-1}b_P \tag{8}$$

and then substituting the obtained value of δx_C into

$$V\delta x_P = b_P - W^T \delta x_C \tag{9}$$

and solving for δx_P. This procedure is known as Schur complementation and reduces the computational load from solving a $(6m+3n)\times(6m+3n)$ system to solving a $6m \times 6m$ system followed by a quick substitution and block diagonal solve. In applications m is usually much smaller than n so this typically means substantial savings. For systems with up to a couple of hundred cameras, the most expensive step actually often lies in forming $WV^{-1}W^T$, since W is often quite dense. However, for larger problems the cost of solving the Schur system will dominate the computations. With the method of conjugate gradients we can avoid both Schur complementation and Cholesky factorization, thus avoiding the two dominant steps in terms if time and memory requirements. The price for this can be slower convergence especially near the optimum.

3 The Linear and Non-linear Conjugate Gradient Algorithms

The conjugate gradient algorithm is an iterative method for solving a symmetric positive definite system of linear equations

$$Ax = b, \tag{10}$$

introduced by Hestenes and Stiefel [11,12]. In its basic form it requires only multiplication of the matrix A with a vector, *i.e* no matrix-matrix multiplications and no matrix factorizations.

Conjugate Gradient Algorithm(x^0, A, b)

// An initial solution x^0 (possibly zero) has to be provided
$s^0 = b - Ax^0, p^0 = s^0, k = 0$
while $|s^k| >$ threshold
$\alpha^k = \frac{{s^k}^T s^k}{{p^k}^T A p^k}$
$x^{k+1} = x^k + \alpha^k p^k$
$s^{k+1} = s^k - \alpha^k A p^k$
$\beta^k = \frac{{s^{k+1}}^T s^{k+1}}{{s^k}^T s^k}$
$p^{k+1} = s^{k+1} + \beta^k p^k$
$k = k + 1$

The basic way to apply the conjugate gradient algorithm to the bundle adjustment problem is to form the normal equations $J^T J \delta x = -J^T r$ and set $A = J^T J, b = -J^T r$.

The linear CG method corresponds to minimization of the quadratic form $q(x) = \frac{1}{2} x^T A x - b^T x$. Fletcher and Reeves generalized the procedure to non-quadratic functions yielding the non-linear conjugate gradients algorithm [13]. Here, only the function $f(x)$ and its gradient $\nabla f(x)$ are available.

4 Conjugate Gradients for Least Squares

A naive implementation of the conjugate gradient algorithm for the normal equations would require forming $A = J^T J$ which is a relatively expensive operation. However, we can rewrite the updating formulas for α^k and s^{k+1} as

$$\alpha^k = \frac{{s^k}^T s^k}{(Jp^k)^T (Jp^k)}, \tag{11}$$

$$s^{k+1} = s^k - \alpha^k J^T (Jp^k), \tag{12}$$

implying that we only need to compute the two matrix-vector multiplications $w^k = Jp^k$ and $J^T w^k$ in each iteration. The resulting algorithm is known as CGLS [14]. The conjugate gradient method belongs to the wider family of Krylov

subspace optimizing algorithms. An alternative to CGLS is the LSQR algorithm by Paige and Saunders [15], which is based on Lanczos bidiagonalization. Mathematically CGLS and LSQR are equivalent, but LSQR has in some cases been observed to be slightly more stable numerically. However, in our bundle adjustment experiments these two algorithms have produced virtually identical results. Since LSQR requires somewhat more storage and computation than CGLS we have stuck with the latter.

5 Inexact Gauss-Newton Methods

As previously mentioned, there are two levels where we can apply conjugate gradients. Either we use linear conjugate gradients to solve the normal equations $J^T J dx = -J^T r$ and thus obtain the Gauss-Newton step or we apply non-linear conjugate gradients to directly solve the non-linear optimization problem.

Since $c(x) = r^T(x)r(x)$, we get $\nabla c(x) = -J^T(x)r(x)$ and we see that computing ∇c implies computing the Jacobian J of r. Once we have computed J (and r) we might as well run a few more iterations keeping these fixed. But, since the Gauss-Newton step is anyway an approximation to the true optimum, there is no need to solve the normal equations very exactly and it is likely to be a good idea to abort the linear conjugate gradient method early, going for an approximate solution. This leads to the topic of inexact Newton methods (see *e.g* [16] for more details). In these methods a sequence of stopping criteria are used to abort the inner iterative solver for the update step early. The logical termination quantity here is the relative magnitude of the residual of the normal equations $|s^k|$ (not to be confused with the residual of the least squares system r). A common choice is to terminate the inner CG iteration when

$$\frac{|s^k|}{|\nabla c(x_j)|} < \eta_j,$$

where the sequence $\eta_j \in (0, 1)$ is called a *forcing sequence*. There is a large body of research on how to select such a forcing sequence. We have however found the rule of thumb to select the constant $\eta_j = 0.1$ to provide a resonable trade off between convergence and number of CG iterations.

6 Preconditioning

The success of the conjugate gradient algorithm depends largely on the conditioning of the matrix A. Whenever the condition number $\kappa(A)$ is large convergence will be slow. In the case of least squares, $A = J^T J$ and thus $\kappa(A) = \kappa(J)^2$, so we will almost inevitably face a large condition number[1]. In these cases one

[1] Note that even if we avoid forming $A = J^T J$ explicitly, A is still implicitly the system matrix and hence it is the condition number $\kappa(A)$ we need to worry about.

can apply *preconditioning*, which in the case of the conjugate gradient method means pre-multiplying from left and right with a matrix E to form

$$E^T A E \hat{x} = E^T b,$$

where E is a non-singular matrix. The idea is to select E so that $\hat{A} = E^T A E$ has a smaller condition number than A. Finally, x can be computed from $\hat{x}$ with $x = E\hat{x}$. Often E is chosen so that EE^T approximates A^{-1} in some sense. Explicitly forming $\hat{A}$ is expensive and usually avoided by inserting $M = EE^T$ in the right places in the conjugate gradient method obtaining the *preconditioned conjugate gradient method*. Two useful preconditioners can be obtained by writing $A = L + L^T - D$, where D and L are the diagonal and lower triangular parts of A. Setting $M = D^{-1}$ is known as Jacobi preconditioning and $M = L^{-T} D L^{-1}$ yields Gauss-Seidel preconditioning.

6.1 Block QR Preconditioning

The Jacobi and Gauss-Seidel preconditioners alone do not make use of the special structure of the bundle adjustment Jacobian. Assume for a moment that we have the QR factorization of J, $J = QR$ and set $E = R^{-1}$. This yields the preconditioned normal equations

$$R^{-T} J^T J R^{-1} \delta\hat{x} = -R^{-T} J^T r,$$

which by inserting $J = QR$ reduce to

$$\delta\hat{x} = -R^{-T} J^T r$$

and $\delta\hat{x}$ is found in a single iteration step (δx is then be obtained by $\delta x = R^{-1}\delta\hat{x}$). Applying R^{-1} is done very quickly through back-substitution. The problem here is of course that computing $J = QR$ is exactly the sort of expensive operation we are seeking to avoid. However, we can do something which is similar in spirit. Consider again the partitioning $J = [J_C,\ J_P]$. Using this, we can do a block wise QR factorization in the following way:

$$J_C = Q_C R_C,\ J_P = Q_P R_P.$$

Due to the special block structure of J_C and J_P respectively we have

$$R_C = R(J_C) = \begin{bmatrix} R(\tilde{A}_1) & & & \\ & R(\tilde{A}_2) & & \\ & & \ddots & \\ & & & R(\tilde{A}_n) \end{bmatrix}$$

and

$$R_P = R(J_P) = \begin{bmatrix} R(B_1) & & & \\ & R(B_2) & & \\ & & \ddots & \\ & & & R(B_n) \end{bmatrix},$$

where

$$\tilde{A}_k = \begin{bmatrix} A_{k1} \\ A_{k2} \\ \vdots \\ A_{kn} \end{bmatrix}$$

and

$$B_k = \begin{bmatrix} B_{1k} \\ B_{2k} \\ \vdots \\ A_{mk} \end{bmatrix}$$

and where

$$A_{ij} = \partial_{C_i} r_{ij}, \; B_{ij} = \partial_{U_j} r_{ij}.$$

In other words, we can perform QR factorization independently on the block columns of J_C and J_P, making this operation very efficient (linear in the number of variables) and easy to parallelize. The preconditioner we propose to use thus becomes

$$E = \begin{bmatrix} R(J_C)^{-1} & \\ & R(J_P)^{-1} \end{bmatrix}.$$

Similar preconditioners were used by Golub *et al* in [17] in the context of satellite positioning. One can easily see that the QR preconditioner is in fact analytically equivalent to block-Jacobi preconditioning. Two important advantages are however that (i) we do not need to form $J^T J$ (as is the case with block-Jacobi) and (ii) that QR factorization of a matrix A is generally considered numerically superior to forming $A^T A$ followed by Cholesky factorization.

6.2 Property A

A further important aspect of the bundle adjustment Jacobian is that the preconditioned system matrix $\hat{J}^T \hat{J}$ has "property A" as defined by Young in [9].

Definition 1. *The matrix A has "property A"* iff *it can be written*

$$A = \begin{bmatrix} D_1 & F \\ F^T & D_2 \end{bmatrix}, \tag{13}$$

where D_1 and D_2 are diagonal.

The benefit is that for any matrix posessing "property A", the work that has to be carried out in the conjugate gradient method can roughly be cut in half as showed by Reid in [18]. This property can easily be seen to hold for $\hat{J}^T \hat{J}$:

$$\hat{J}^T \hat{J} = \begin{bmatrix} R(J_C) & \\ & R(J_P) \end{bmatrix}^{-T} \begin{bmatrix} J_C^T J_C & J_C^T J_P \\ J_P^T J_C & J_P^T J_P \end{bmatrix} \begin{bmatrix} R(J_C) & \\ & R(J_P) \end{bmatrix}^{-1} = \begin{bmatrix} Q_C^T Q_C & Q_C^T Q_P \\ Q_P^T Q_C & Q_P^T Q_P \end{bmatrix},$$

where $Q_C^T Q_C$ and $Q_P^T Q_P$ are both identity matrices and $Q_P^T Q_C = (Q_C^T Q_P)^T$. Partition the variables into camera and point variables and set $s^k = \begin{bmatrix} s_C^k \\ s_P^k \end{bmatrix}$. Applying Reid's results to our problem yields the following: By initializing so that $\delta x_C = 0$ and $\delta x_P = -J_P^T r$, we will have $s_C^{2m} = s_P^{2m+1} = 0$. We can make use of this fact in the following way (where for clarity, we have dropped the subscript j from the outer iteration):

INNER CG LOOP USING "PROPERTY A"(J, r)

$\eta = 0.1$
$\delta x_C^0 = 0,\ \delta x_P^0 = -J_P^T r,\ \hat{r}^0 = -r - J\delta x^0,\ p^0 = s^0 = J^T \hat{r}^0,$
$\gamma^0 = {s^0}^T s^0,\ q^0 = Jp^0,\ k = 0$
while $\|s^k\| > \eta \|s^0\|$

$\alpha^k = \frac{\gamma^k}{{q^k}^T q^k}$
$\delta x^{k+1} = \delta x^k + \alpha^k p^k$
$\begin{cases} s_C^{k+1} = -\alpha^k J_C^T q^k,\ s_P^{k+1} = 0\ k \text{ odd} \\ s_P^{k+1} = -\alpha^k J_P^T q^k,\ s_C^{k+1} = 0\ k \text{ even} \end{cases}$
$\gamma^{k+1} = {s^{k+1}}^T s^{k+1}$
$\beta^k = \frac{\gamma^{k+1}}{\gamma^k}$
$p^{k+1} = s^{k+1} + \beta^k p^k$
$\begin{cases} q^{k+1} = \beta^k q^k + J_C s_C^{k+1}\ k \text{ odd} \\ q^{k+1} = \beta^k q^k + J_P s_P^{k+1}\ k \text{ even} \end{cases}$

One further interesting aspect of matrices with "Property A" is that one can show that for these matrices, block-Jacobi preconditioning is always superior to Gauss-Seidel and SSOR preconditioners [14] (pages 286-287).

7 Experiments

For evaluation we compare three different algorithms on synthetic and real data. Standard bundle adjustment was performed using the Levenberg-Marquardt algorithm and sparse Cholesky factorization of the Schur complement to solve the normal equations. Cholesky factorization was performed using the Cholmod library with reverse Cuthill-McKee ordering. We henceforth denote this algorithm DBA for direct bundle adjustment. Secondly, we study a straight forward adaptation of the conjugate gradient algorithm to bundle adjustment by using $J^T J$ as the system matrix and the block diagonal of $J^T J$ as a preconditioner. We simply refer to this algorithm as CG. Finally, we denote the conjugate gradient method taylored to bundle adjustment as proposed in this paper CGBA for conjugate gradient bundle adjustment.

In all cases we apply adaptive damping to the normal equations as suggested in [10]. In the case of CGBA, we never form $J^T J$ and we instead apply damping by using the damped Jacobian

$$J_\lambda = \begin{bmatrix} J \\ \lambda I \end{bmatrix},$$

which can be factorized in the same manner as J for preconditioning.

For clarity, we focus on calibrated cameras only in this work. Including additional parameters such as focal length and distortion parameters presents no problem and fits into the same general framework without modification.

7.1 Synthetic Data: When Is the CG Algorithm a Good Choice?

A common statement is that standard bundle adjustment is good for small to medium size problems and that Conjugate Gradients should probably be the way to go for large and sparse problems. This is not quite true as we will show with a couple of synthetic experiments. In some cases CG based bundle adjustment can actually be a better choice for quite small problems. On the other hand it might suffer from hopelessly slow convergence on some large very sparse setups. Theoretically, the linear CG algorithm converges in a number of iterations proportional to roughly the square root of the condition number and a large condition number hence yields slow convergence. Empirically, this happens in particular for sparsely connected structures where unknowns in the camera-structure graph are far apart. Intuitively such setups are much less "stiff" and can undergo relatively large deformations with only little effect on the reprojection errors.

To capture this intuition we have simulated two qualitatively very different scenarios. In the first setup, points are randomly located inside a sphere of radius one centered at the origin. Cameras are positioned uniformly around the sphere at around two length units from the origin pointing roughly towards the origin. There are 10 times as many points as cameras and each camera sees 100 randomly selected points. Due to this, each camera shares features with a large percentage of the other cameras. In the second experiments, points are arranged along a circular wall with cameras on the inside of the wall pointing outwards. There are four points for each camera and due to the configuration of the cameras, each camera only shares features with a small number of other cameras. For each scenario we have generated a series of configurations with increasingly many cameras and points. One example from each problem type can be seen in Figure 1. For each problem instance we ran both standard bundle adjustment with Cholesky factorization and the Conjugate Gradient based bundle adjustment procedure proposed in this paper until complete convergence and recorded the total time. Both solvers produced the same final error up to machine precision. Since the focus of this experiment was on iterative versus direct solvers, we omitted the comparision CG method. The results of this experiment are perhaps somewhat surprising. For the sphere problem, CGBA is orders of magnitude faster for all but the smallest problems, where the time is roughly equal. In fact, the empirical time complexity is almost linear for CGBA whereas DBA displays the familiar cubic growth. For the circular wall scenario the situation is reversed. While CGBA here turns out to be painfully slow for the larger examples, DBA seems perfectly suited to the problem and requires not much more than linear time in the number of cameras. Note here that the Schur

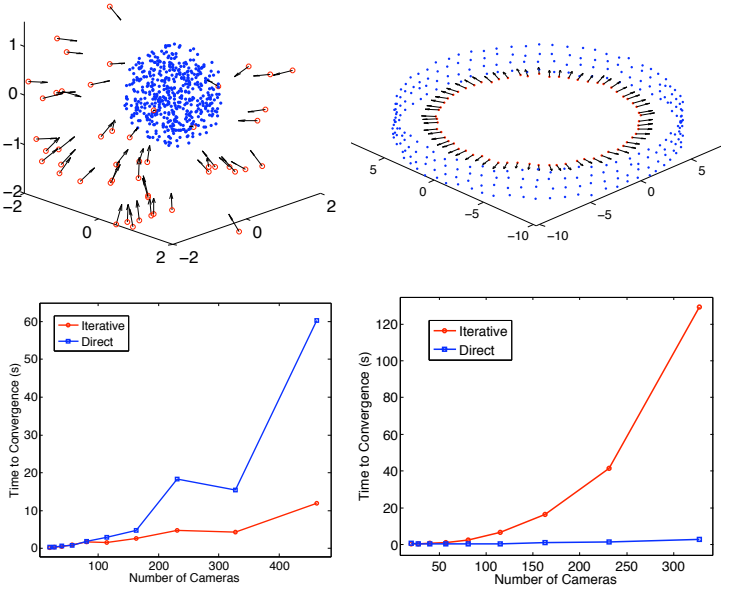

Fig. 1. Top-left: An instance of the sphere problem with 50 cameras and 500 3D points. Top-right: Points arranged along a circular wall, with 64 cameras viewing the wall from the inside. Bottom-left: Time to convergence vs. number of cameras for the sphere problem. This configuration is ideally suited to CG based bundle adjustment which displays approximately linear complexity. Bottom-right: Time vs. problem size for the circular wall. The CG based solver takes very long to converge, whereas the direct solver shows an almost linear increase in complexity, far from the theoretical $\mathcal{O}(N^3)$ worst case behaviour.

complement in the sphere setup is almost completely dense whereas in the wall case it is extremely sparse. The radically different results on these data sets can probably understood like this. Since the CG algorithm in essence is a first order method "with acceleration", information has to flow from variable to variable. In the sphere case, the distance between cameras in the camera graph is very small with lots of connections in the whole graph. This means that information gets propagated very quickly. In the wall problem though, cameras on opposite sides of the circular configuration are very far apart in the camera graph which yields a large number of CG iterations. For the direct approach "stiffness" of the graph does not matter much. Instead fill-in during Cholesky factorization is the dominant issue. In the wall problem, the Schur complement will have a narrow banded structure and is thus possible to factorize with minimal fill-in.

8 Community Photo Collections

In addition to the synthetic experiments, we have compared the algorithms on four real world data sets based on images of four different locations downloaded from the internet: The St. Peters church in Rome, Trafalgar square in London,

the old town of Dubrovnik and the San Marco square in Venice. The unoptimized models were produced using the systems described in [19,20,1].

The models produced by these systems initially contained a relatively large number of outliers, 3D points with extremely short baselines and very distant cameras with a small field of view. Each of these elements can have a very large impact on the convergence of bundle adjustment (both for iterative and direct solvers). To ensure an informative comparison, such sources of large residuals and ill conditioning were removed from the models. This meant that approximately 10% of the cameras, 3D points and reprojections were removed from the models.

In addition, we used the available calibration information to calibrate all cameras before bundle adjustment. In general this gave good results but for a very small subset of cameras ($< 0.1\%$) the calibration information was clearly incorrect and these cameras were removed as well from the models.

For each data set we ran bundle adjustment for 50 iterations and measured the total time and final RMS reprojection error in pixels. All experiments were done on a standard PC equipped with 32GB of RAM to be able to process large data sets. For the CG based solvers, we used a constant $\eta = 0.1$ forcing sequence and set the maxium number of linear iterations to 100. The results can be found in Table 1. Basically, we observed the same general pattern for all four data sets. Due to the light weight nature of the CG algorithms, these showed very fast convergence (measured in seconds) in the beginning. At a certain point close to the optimum however, convergence slowed down drastically and in none of the cases did either of the CG methods run to complete convergence. This is likely to correspond to the bound by the condition number of the Jacobian (which we were not able to compute due to the sizes of these problems). In other words, the CG algorithms have problems with the eigenmodes corresponding to the smallest singular values of the Jacobian. This situation makes it hard to give a fair comparison between direct BA and BA based on an iterative linear solver. The choice has to depend on the application and desired accuracy. In all cases,

Table 1. Performance statistics for the different algorithms on the four community photo data sets

Data set	m	n	n_r	Algorithm	Total Time	Final Error (Pixels)
St. Peter	263	129502	423432	DBA	113s	2.18148
				CGBA	441s	2.23135
				CG	629s	2.23073
Trafalgar Square	2897	298457	1330801	DBA	68m	1.726962
				CGBA	18m	1.73639
				CG	38m	1.75926
Dubrovnik	4564	1307827	8988557	DBA	307m	1.015706
				CGBA	130m	1.015808
				CG	236m	1.015812
Venice	13666	3977377	28078869	DBA	N/A	N/A
				CGBA	230m	1.05777
				CG	N/A	N/A

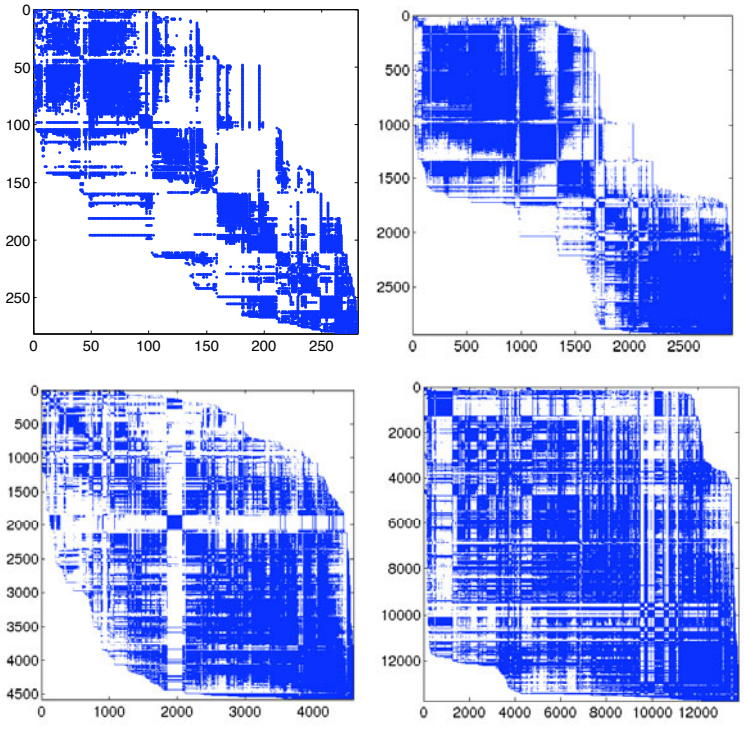

Fig. 2. Sparsity plots for the reverse Cuthill-McKee reordered Schur complements. Top-left: St. Peter, top-right: Trafalgar, bottom-left: Dubrovnik, bottom-right: Venice.

CGBA was about two times faster than CG as expected and in general produced slightly more accurate results.

For the Venice data set, we were not able to compute the Cholesky factorization of the Schur complement since we ran out of memory. Similarly, there was not enough memory in the case of CG to store both J and J^TJ. While Cholesky factorization in this case is not likely to be feasible even with considerably more memory, a more clever implementation would probably not require both J and J^TJ and could possibly allow CG to run on this instance as well. However, as can be seen from the other three examples, the relative performance of CG and CGBA is pretty constant so this missing piece of information should not be too serious.

As observed in the previous section, problem structure largely determines the convergence rate of the CG based solvers. In Figure 2, sparsity plots for the Schur complement in each of the four data sets is shown. To reveal the structure of the problem we applied reverse Cuthill-McKee reordering (this reordering was also applied before Cholesky factorization in DBA), which aims at minimizing the bandwidth of the matrix. As can be seen, this succeeds quite well in the case of St. Peter and Trafalgar. In particular in the Trafalgar case, two almost independent sets are discovered. As discussed in the previous section, this is

a disadvantage for the iterative solvers since information does not propagate as easily in these cases. In the case of Dubrovnik and in particular Venice, the graph is highly connected, which is beneficial for the CG solvers, but problematic for direct factorization.

9 Conclusions

In its current state, conjugate gradient based bundle adjustment (on most problems) is not in a state where it can compete with standard bundle adjustment when it comes to absolute accuracy. However, when good accuracy is enough, these solvers can provide a powerful alternative and sometimes the only alternative when the problem size makes Cholesky factorization infeasible. A typical application would be intermediate bundle adjustment during large scale incremental SfM reconstructions. We have presented a new conjugate gradient based bundle adjustment algorithm (CGBA) which by making use of "Property A" of the preconditioned system and by avoiding $J^T J$ is about twice as fast as "naive" bundle adjustment with conjugate gradients and more precise. An interesting path for future work would be to try and combine the largely orthogonal strengths of the direct versus iterative approaches. One such idea would be to solve a simplified (skeletal) system using a direct solver and use that as a preconditioner for the complete system.

References

1. Agarwal, S., Snavely, N., Simon, I., Seitz, S.M., Szeliski, R.: Building rome in a day. In: Proc. 12th Int. Conf. on Computer Vision, Kyoto, Japan (2009)
2. Snavely, N., Seitz, S.M., Szeliski, R.: Modeling the world from Internet photo collections. Int. Journal of Computer Vision 80, 189–210 (2008)
3. Mordohai, A.F.: Towards urban 3d reconstruction from video (2006)
4. Cornelis, N., Leibe, B., Cornelis, K., Gool, L.V.: 3d urban scene modeling integrating recognition and reconstruction. Int. Journal of Computer Vision 78, 121–141 (2008)
5. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, Cambridge (2004)
6. Triggs, W., McLauchlan, P., Hartley, R., Fitzgibbon, A.: Bundle adjustment: A modern synthesis. In: Triggs, B., Zisserman, A., Szeliski, R. (eds.) ICCV-WS 1999. LNCS, vol. 1883, p. 298. Springer, Heidelberg (2000)
7. Lourakis, M.I.A., Argyros, A.A.: Sba: A software package for generic sparse bundle adjustment. ACM Trans. Math. Softw. 36, 1–30 (2009)
8. Byröd, M., Åström, K.: Bundle adjustment using conjugate gradients with multiscale preconditioning. In: Proc. British Machine Vision Conference, London, United Kingdom (2009)
9. Young, D.M.: Iterative solution of large linear systems. Academic Press, New York (1971)
10. Nielsen, H.B.: Damping parameter in marquardt's method. Technical Report IMM-REP-1999-05, Technical University of Denmark (1999)

11. Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. Journal of Research of the National Bureau of Standards 49, 409–436 (1952)
12. Golub, G.H., van Loan, C.F.: Matrix Computations, 3rd edn. The Johns Hopkins University Press, Baltimore (1996)
13. Fletcher, R., Reeves, C.M.: Function minimization by conjugate gradients. The Computer Journal 7, 149–154 (1964)
14. Björck, Å.: Numerical methods for least squares problems. SIAM, Philadelphia (1996)
15. Paige, C.C., Saunders, M.A.: Lsqr: An algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Softw. 8, 43–71 (1982)
16. Nocedal, J., Wright, S.J.: Numerical optimization, 2nd edn. Springer, Berlin (2006)
17. Golub, G.H., Manneback, P., Toint, P.L.: A comparison between some direct and iterative methods for certian large scale godetic least squares problems. SIAM J. Sci. Stat. Comput. 7, 799–816 (1986)
18. Reid, J.K.: The use of conjugate gradients for systems of linear equations possessing "property a". SIAM Journal on Numerical Analysis 9, 325–332 (1972)
19. Snavely, N., Seitz, S.M., Szeliski, R.: Modeling the world from internet photo collections. International Journal of Computer Vision 80, 189–210 (2007)
20. Snavely, N., Seitz, S.M., Szeliski, R.: Skeletal sets for efficient structure from motion. In: Proc. Conf. Computer Vision and Pattern Recognition, Anchorage, USA (2008)