Variants of Bundle Adjustment

Introduction:

Bundle Adjustment starts with optimizing the standard least squares problem between points in images and predicted projection of 3D world points using given cameras. Both the 3D world points and camera projection matrices are unknowns. Let x denote all the unknowns in the least squares solution. The standard Levenberg-Marquardt solves it by computing update δx at each update step using

$$H_{\mu}(x)\delta x = -g(x)$$

Schur's complement trick uses the sparse structure of this Hessian H_{μ} where the update now splits as solving the equation

$$[B - EC^{-1}E^{T}]\delta y = (v - EC^{-1}w)$$

This linear equation is typically solved by using Cholesky Factorization. Due to poor conditional number of H_{μ}, pre-conditioners are introduced in addition to using inexact type methods like Conjugate Gradients methods to solve these equations. This CG method with preconditioners is known as Preconditioned Conjugate Gradients (PCG). We experiment with several of these methods and determine which of these is better for the given problem. These observations are compared against those in the original paper.

Experiments:

We evaluate Bundle Adjustment in six different settings.

1) Explicit-direct: Exact LM algorithm is used to update and solve the Schur's complement simplified system. Dense Cholesky Factorization is used as technique. Schur's complement is computed explicitly.
2) Explicit-sparse: Exact LM algorithm is used to update and solve the Schur's complement simplified system. Sparse Cholesky Factorization is used to explore the sparsity of S.. Schur's complement is computed explicitly.
3) Normal-Jacobi: Firstly, block jacobi matrix M_J is used as preconditioner for H_{μ}. The PCG algorithm is now done on the original equation (not simplified by Schur). This is inexact type methods not involving computation of S.
4) Explicit-jacobi: PCG method is now applied on Schur's reduced equation with $D(S)$ as the preconditioner. Explicitly computes the Schur's complement S.
5) Implicit-jacobi: PCG is applied without directly computing Schur's complement by using tricks to evaluate matrix-vector product Sv. $D(S)$ is used as the preconditioner.

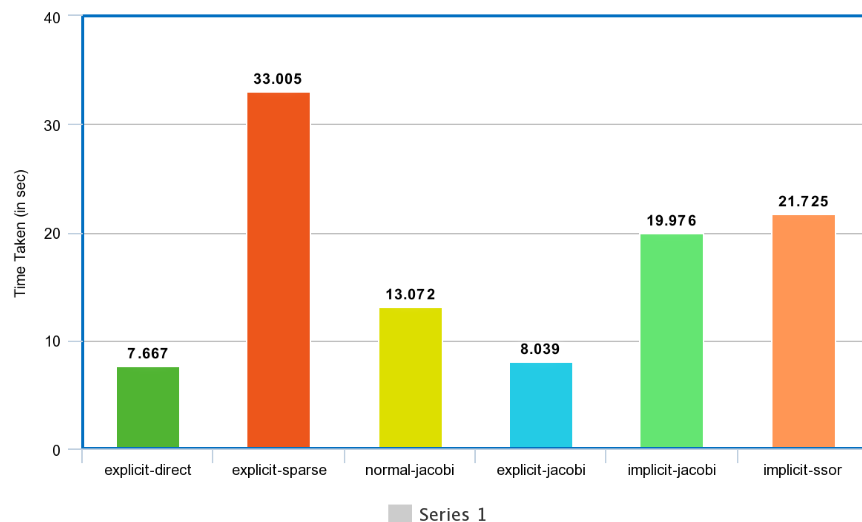
6) Implicit-ssor: PCG is applied without directly computing Schur's complement by using tricks to evaluate matrix-vector product Sv. Block matrix B is used as preconditioner on Schur's complement.

We evaluate these six different alternatives on LadyBug dataset provided by the same authors. We take 3 datasets in LadyBug which vary in size and report the performance details on these datasets. The initialization of camera parameters and 3D points is provided by solving Sfm problem on these datasets. This is done through a toolkit called 'Bundler'.

Smaller Dataset:

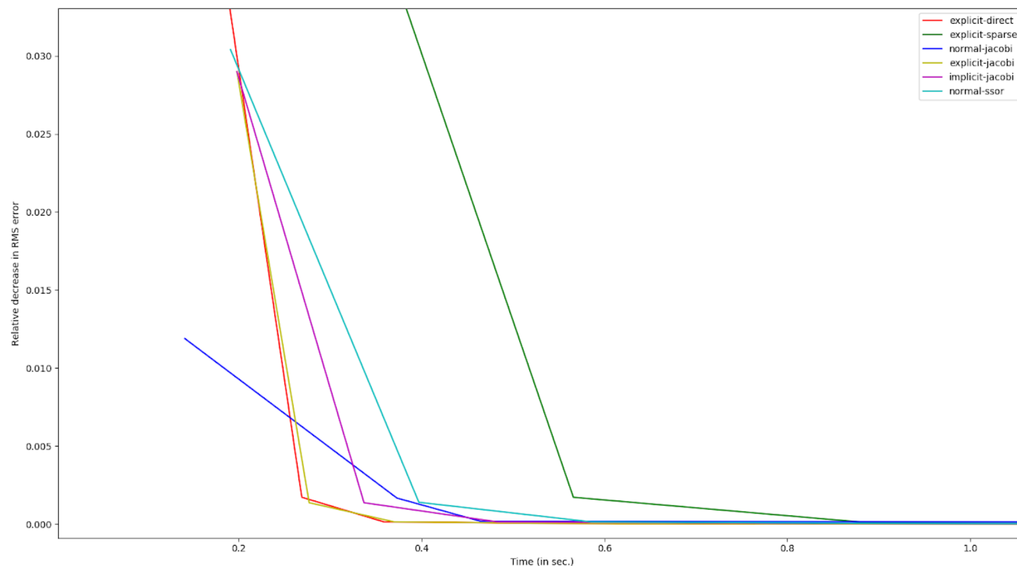
The dataset consists of 49 images and 7779 points in 3D world with a total of 31843 observations. The results of running these six algorithms of BA are summarized in the table and graphs below.

	Time taken (in sec.) for 100 iterations
Explicit-direct	7.667
Explicit-sparse	33.005
Normal-jacobi	13.072
Explicit-jacobi	8.039
Implicit-jacobi	19.976
Implicit-ssor	21.725



The above images show the time taken for 100 iterations of each of the variations. We observe that the exact method (Explicit-direct) takes the least amount of time. This is mostly because the exact nature of the algorithm coupled with the small problem size allowed the optimization to be solved accurately and hence with less error.

We also observe that the sparse version (Explicit-sparse) takes much longer. This might be due to the overheads involved in sparse methods which may exceed the actual runtime. The other explicit method (Explicit-jacobi) also performs well because explicitly computing S gives better accuracy without trading with time in the smaller dataset cas

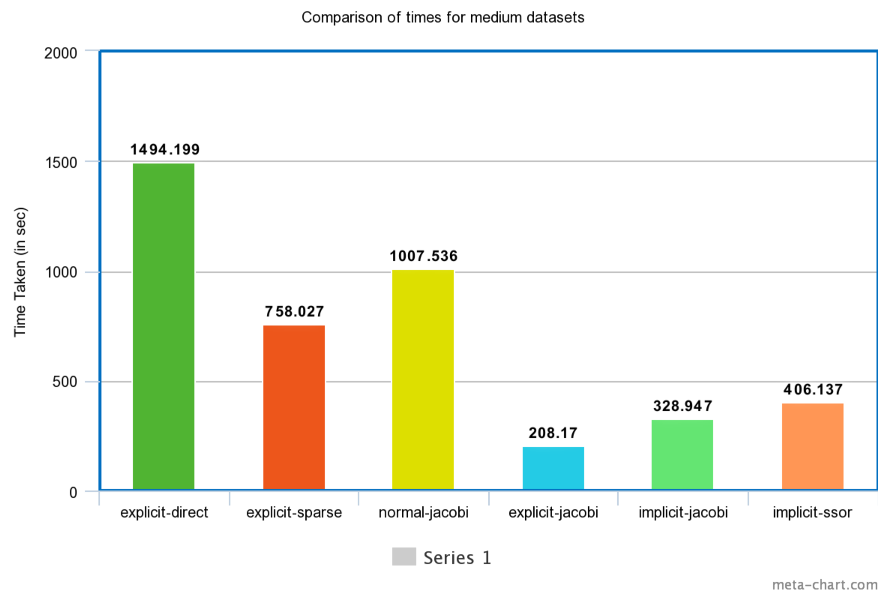


The above graph shows the relative decrease in RMS error at each iteration with time. This can be used to define tolerance levels on error and determine which algorithm performs best within that limit. We notice that at lower tolerances, explicit-direct and explicit-jacobi reach minimum the fastest and at very high tolerance, normal-jacobi performs better.

Medium Dataset:

The dataset consists of 810 images and 88814 points in 3D world with a total of 393775 observations. The results of running these six algorithms of BA are summarized in the table and graphs below.

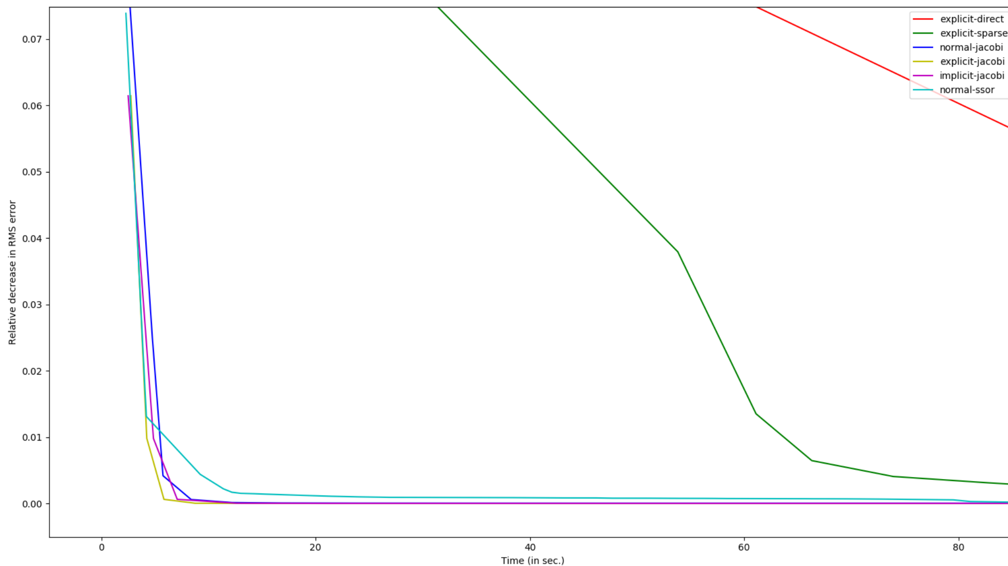
	Time taken (in sec.) for 100 iterations
Explicit-direct	1494.199
Explicit-sparse	758.027
Normal-jacobi	1007.536
Explicit-jacobi	208.17
Implicit-jacobi	328.947
Implicit-ssor	406.137



The above images show the time taken for 100 iterations of each of the variations. We first observe that the explicit-direct and explicit-sparse take a large amount of time because of the overhead in Cholesky factorization of larger matrices. This is due to the exact method of computation which is very slow.

We notice that explicit-jacobi performs the best in this case. This is because the overhead of computing S is very less in this problem due to the inherent sparsity in it. This makes it much more viable to directly compute S and then applying PCG on it instead of indirectly using it. In this same scenario, if alternatively sparsity of S is not high, then implicit-ssor become much more viable.

These observations on medium dataset are consistent with the observations in the paper.

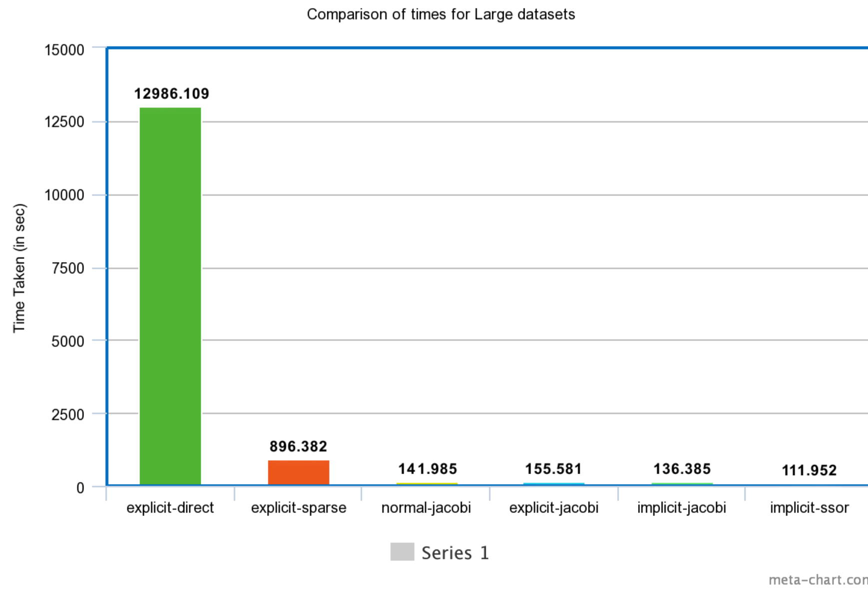


The above graph shows the relative decrease in RMS error at each iteration with time. This can be used to define tolerance levels on error and determine which algorithm performs best within that limit. We observe that at all tolerances below 0.01, explicit-jacobi performs the best. This is also consistent with the conclusion made above.

Large Dataset

The dataset consists of 1723 images and 156502 points in 3D world with a total of 678198 observations. The results of running these six algorithms of BA are summarized in the table and graphs below.

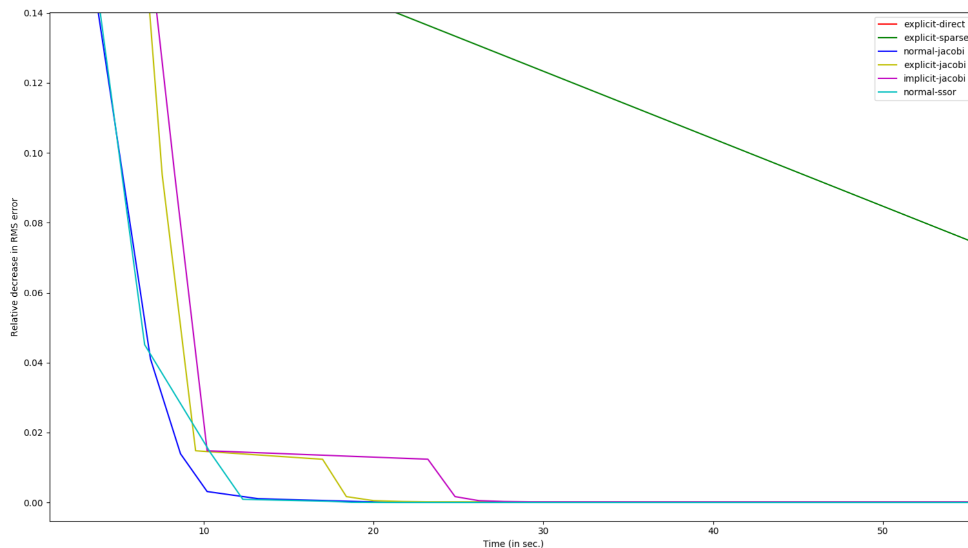
	Time taken (in sec.) for 100 iterations
Explicit-direct	12986.109
Explicit-sparse	896.382
Normal-jacobi	141.985
Explicit-jacobi	155.581
Implicit-jacobi	136.385
Implicit-ssor	111.952



The above images show the time taken for 100 iterations of each of the variations. First, we observe that the explicit-direct takes very large amount of time. This is expected because applying direct methods to large problems in bundle adjustment requires to operate on very large matrices, hence creating a significant overhead which increases with problem size. Even using sparse cholesky version does not give performance comparable to other indirect computations.

We observe that implicit-ssor performs the best among all the methods. This happens when the sparsity in S is slightly less and hence creates an overhead for explicit-jacobi. Also, the block jacobi of B acts as better preconditioner instead of block jacobi of S. Even these results are consistent with the observations in the paper which states that implicit-ssor performs the best for large problems.

The below graph shows the relative decrease in RMS error at each iteration with time. This can be used to define tolerance levels on error and determine which algorithm performs best within that limit.



We notice that at tolerances like 0.001 and 0.0001, implicit-ssor performs the best. At tolerances of 0.01, normal-jacobi performs good. These results are consistent again with the previous observations.

Conclusion

We took three representative datasets covering small, medium and large sizes. We use Ceres [1][2] solver for solving these bundle adjustment problems. We notice that for small problems explicit-direct performs the best. For medium problems, explicit-jacobi performs the best and for large problems implicit-ssor performs the best. These results are consistent with the actual conclusion in the paper.

[1] "Bundle Adjustment in the Large - GRAIL - University of Washington." http://grail.cs.washington.edu/projects/bal/.
[2] "Ceres Solver." http://ceres-solver.org/.