

Sparse Sparse Bundle Adjustment

Kurt Konolige
<http://pub1.willowgarage.com/~konolige>

Willow Garage,
 Menlo Park, CA

Sparse Bundle Adjustment (SBA) is the standard method for optimizing a structure-from-motion problem in computer vision. With the success of Photosynth and similar systems for stitching together large collections of images [5], attention has turned to the problem of making SBA more efficient.

There are two typical usage patterns for SBA. In one, a set of images are taken of an object, so most of the images have features in common and the secondary structure is dense. For example, in the Venice dataset of tourist photos¹, the density of \mathbf{H} is 40%, that is, non-zeros account for 40% of the matrix entries.

On the other end of the spectrum are datasets from visual mapping, where usually a single camera moves around an area, and the images are registered to produce an extended map. For example, in the Intel Seattle indoor dataset², the camera motion is mostly along corridors, and the density is only 1.4%. The difference in the sparsity pattern is shown in Figure 1. Note that the mapping pattern consists of a fat diagonal band, with some parallel bands for overlapping trajectories. (Guilbert et al. call these “sparse systems” [2]). In this paper, we are interested in fast SBA methods for the mapping case, where it is possible to exploit the sparse secondary structure (camera to camera relations) of the problem.

Nonlinear optimization in SBA typically proceeds by iteration: form a linear subproblem around the current solution, solve it, and repeat until convergence. For large problems, the computational bottleneck is usually the solution of the linear subproblem, which can grow as the cube of the number of cameras. The fill-in of the linear problem is directly tied to the camera-point structure of the problem: if each camera only sees features in a small neighborhood of other cameras, the number of non-zero elements grows only linearly or nearly linearly with the number of cameras.

The linear system is formed from the projection of points \mathbf{p} onto cameras \mathbf{c} . The linear equation can be written as:

$$\begin{bmatrix} \mathbf{J}_c^T \mathbf{J}_c & \mathbf{J}_c^T \mathbf{J}_p \\ \mathbf{J}_p^T \mathbf{J}_c & \mathbf{J}_p^T \mathbf{J}_p \end{bmatrix} \begin{bmatrix} \Delta \mathbf{c} \\ \Delta \mathbf{p} \end{bmatrix} = \begin{bmatrix} -\mathbf{J}_c^T \mathbf{e}_c \\ -\mathbf{J}_p^T \mathbf{e}_p \end{bmatrix} \quad (1)$$

where \mathbf{J}_c is the Jacobian with respect to camera variables, and \mathbf{J}_p with respect to point variables. Because all the error functions involve one camera and one point, the Jacobian products $\mathbf{J}_c^T \mathbf{J}_c$ and $\mathbf{J}_p^T \mathbf{J}_p$ are block-diagonal. After some manipulation, the reduced system is

$$[\mathbf{H}_{cc} - \mathbf{H}_{cp} \mathbf{H}_{pp}^{-1} \mathbf{H}_{pc}] \Delta \mathbf{c} = -(\mathbf{J}_c^T \mathbf{e}_c - \mathbf{J}_p^T \mathbf{H}_{cp} \mathbf{H}_{pp}^{-1} \mathbf{e}_p) \quad (2)$$

where \mathbf{H}_{xy} refers to the Jacobian products in Equation 1. Note the matrix inversion is simple because of the block-diagonal structure of \mathbf{H}_{pp} .

Solving this equation produces an increment $\Delta \mathbf{c}$ that adjusts the camera variables, and then is used to update the point variables according to

$$\Delta \mathbf{p} = -\mathbf{H}_{pp}^{-1} (\mathbf{J}_p^T \mathbf{e}_p + \mathbf{H}_{pc} \Delta \mathbf{c}) \quad (3)$$

In forming the left-hand side of Equation 2, the main computational bottleneck is computing the product $\mathbf{H}_{cp} \mathbf{H}_{pp}^{-1} \mathbf{H}_{pc}$. We are interested in large systems, where the number of camera variables $\|\mathbf{c}\|$ can be 10k or more (the largest real-world dataset we have used is about 3k poses, but we can generate synthetic datasets of any order). The number of system variables is $6 \cdot \|\mathbf{c}\|$, and the reduced system matrix of Equation 2 has size $36 \cdot \|\mathbf{c}\|^2$, or over 10^9 elements. Manipulating such large matrices is expensive. To do it efficiently, we have to take advantage of its sparse structure. The sparsity pattern of the reduced SBA system is referred to as *secondary structure*.

The most compute-intensive part of creating \mathbf{H} involves an outer product over the projections in each point track (for details of the whole algorithm, see Engels et al. [1]). In our version of this algorithm, we create a sparse structure for accessing arbitrary 6×6 blocks i, j of \mathbf{H} . We use a `C++ std::map` container for each column of \mathbf{H} . The `map` is keyed by row

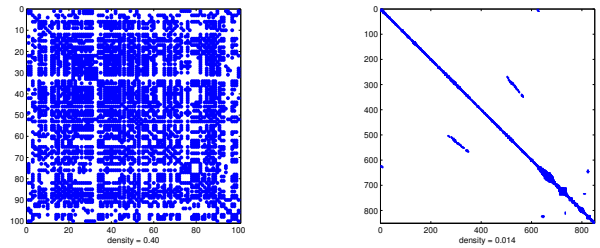


Figure 1: \mathbf{H} matrix non-zero patterns for the Venice dataset (left) and the Intel indoor dataset (right). Only the first 100 frames (out of 871) of the Venice dataset are diagrammed, because it would be too dense to show the full structure.

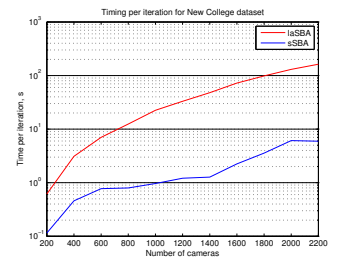
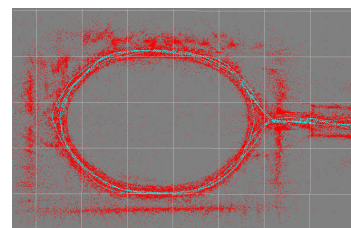


Figure 2: An overhead view showing part of the New College mapping dataset [4], with 2.2k views (cyan) and 290k points (red). Grid lines are at 10 m. At the end, sSBA computes each iteration in 6 seconds. Graph shows timings compared to Lourakis and Argyros [3]

index, and its value is the 6×6 block. Lookup of an arbitrary row element within a `map` is order $\log n$ in the number of elements in the map, while column lookup is constant time (simple array access). We call this system *Sparse SBA*, or *sSBA*.

We validated sSBA using both synthetic and real-world datasets. Figure 2 shows results for a large mapping problem, with over 2k cameras and 290k points. The table below shows timings for other datasets.

Name	# cams	projs/cam	\mathbf{H} fill	setup	solve	total
Venice	871	3259	40%	12.15	6.09	18.24
sSBA				79.58	5.30	84.88
laSBA						
Samantha Erbe	184	705	17%	0.13	0.017	0.15
sSBA				1.0	0.8	1.1
laSBA						
Samantha Bra	320	1222	10%	0.40	0.028	0.43
sSBA				4.47	0.38	4.8
laSBA						
Intel	851	133	1.4%	0.079	0.023	0.10
sSBA				1.91	6.10	8.01
laSBA						

- [1] Chris Engels, Henrik Stewénus, and David Nister. Bundle adjustment rules. *Photogrammetric Computer Vision*, September 2006.
- [2] Nicolas Guilbert, Adrien Bartoli, and Anders Heyden. Affine approximation for direct batch recovery of euclidian structure and motion from sparse data. *Int. J. Comput. Vision*, 69(3):317–333, 2006.
- [3] M.I. A. Lourakis and A.A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.
- [4] M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman. The new college vision and laser data set. *International Journal for Robotics Research (IJRR)*, 28(5):595–599, May 2009. ISSN 0278-3649.
- [5] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Skeletal sets for efficient structure from motion. In *Conference on Computer Vision and Pattern Recognition*, 2008.

¹This dataset provided courtesy of Noah Snavely.

²This dataset provided courtesy of Peter Henry.