

Mining Structure Fragments for Smart Bundle Adjustment

Luca Carlone¹

luca.carlone@gatech.edu

Pablo Fernandez Alcantarilla²

pablo.alcantarilla@crl.toshiba.co.uk

Han-Pang Chiu³

han-pang.chiu@sri.com

Zsolt Kira⁴

Zsolt.Kira@gtri.gatech.edu

Frank Dellaert¹

dellaert@cc.gatech.edu

¹ Georgia Institute of Technology,
College of Computing, USA

² Toshiba Research Europe,
Cambridge Research Laboratory, UK

³ SRI International,
Center for Vision Technologies, USA

⁴ Georgia Tech Research Institute,
ATAS Laboratory, USA

Abstract

Bundle Adjustment (BA) can be seen as an inference process over a *factor graph*. From this perspective, the Schur complement trick can be interpreted as an *ordering* choice for elimination. The elimination of a single point in the BA graph induces a *factor* over the set of cameras observing that point. This factor has a very low information content (a point observation enforces a low-rank constraint on the cameras). In this work we show that, when using conjugate gradient solvers, there is a computational advantage in “grouping” factors corresponding to sets of points (*fragments*) that are co-visible by the same set of cameras. Intuitively, we collapse many factors with low information content into a single factor that imposes a high-rank constraint among the cameras. We provide a grounded way to group factors: the selection of points that are co-observed by the same camera patterns is a data mining problem, and standard tools for *frequent pattern mining* can be applied to reveal the structure of BA graphs. We demonstrate the computational advantage of grouping in large BA problems and we show that it enables a consistent reduction of BA time with respect to state-of-the-art solvers (Ceres [1]).

1 Introduction

Efficient bundle adjustment (BA) is an important prerequisite to a number of practical applications, ranging from 3D modeling and photo tourism [2, 7], to hand-eye calibration [3], augmented reality [2], and autonomous navigation [3]. Modern applications require reconstruction of the 3D geometry of a scene from collections of thousands or millions of images [4, 5, 6], hence the scalability of BA becomes a critical issue.

Standard BA is based on successive linearizations: the nonlinear optimization problem is linearized around the current estimate and a local update for points and camera parameters is computed by minimizing a quadratic approximation of the cost. At each step, computing the

local update requires solving a linear system (*normal equations*). Besides the computational costs of linearization, the main bottleneck is solving the linear system at each step.

The *Schur complement trick* is commonly used to solve only for the cameras (the *reduced camera system*). The points are then updated via back-substitution. The reduced camera system has a special sparse block structure (the *secondary structure* [14]), that can be exploited by sparse direct solvers such as SBA [21] to reduce storage complexity and speed-up computation. Direct methods (dense or sparse Cholesky factorization) work well for small problem instances (few hundred photos), while their cost becomes prohibitive for larger problems [9].

Several recent works suggest the use of iterative methods such as Conjugate Gradient (CG) [14], for solving the symmetric positive-definite linear systems that appear in large-scale BA problems [9, 15]. Iterative methods such as CG involve only sparse matrix-vector multiplications, hence requiring less memory than direct methods. However, the number of required CG iterations for convergence depends on how well-conditioned the original problem is. Therefore, *preconditioning* is normally used to reduce the condition number of the original problem, speeding-up convergence of CG [9, 16, 19, 22]. Another technique that results in the reduction of the number of CG iterations is the *truncated Newton method* [9], which trades off accuracy of the solution of the linear system for computational efficiency. The work [9] also provides the key insight that, in the CG method, the Schur complement trick can be applied without the explicit computation of the reduced camera matrix. This *implicit* representation is shown to be convenient (storage and computation-wise) with respect to *explicit* representations, in which a large (but sparse) square matrix has to be formed.

In this paper, rather than proposing strategies to reduce the *number* of CG iterations, we propose an insight that reduces the complexity of *each* CG iteration. We adopt a factor graph perspective, and interpret BA in terms of inference over a factor graph. We show that the elimination of a single point induces a *factor* (i.e., a probabilistic constraint) over the cameras observing the point. The elimination of all points leads to the standard Schur complement, while in our approach we never need to build the reduced cameras system explicitly. Reasoning in terms of factor graphs allows the solver to choose the best representation (i.e., implicit vs explicit) for each factor. A factor produced by the elimination of a single point provides a low-rank constraint on the cameras, and the use of an explicit representation is not efficient for those. However, we show that “grouping” factors corresponding to points that are co-visible by the same set of cameras produces a single *grouped* factor for which the explicit representation can be convenient. This information compression (the *grouping*) can be done in a grounded way: the grouping problem is formally equivalent to well studied problems in data mining (e.g., *frequent items mining* in basket case analysis [23, 24]). We demonstrate the proposed approach in the Bundle Adjustment in the Large benchmarking datasets [9], showing that the grouping entails a computational advantage in the inner CG iterations. The implementation of our approach is also shown to produce consistent advantages over state-of-the-art implementations available online (Ceres, [10]).

2 A Factor Graph View of Bundle Adjustment

In this section we show that factor graphs provide an intuitive interpretation (and visualization) of BA. We consider the standard setup in which N points $y_j \in \mathbb{R}^3$ are observed in M images. With each image we associate the corresponding camera parameters $x_i \in \mathcal{X}$, where \mathcal{X} is assumed to be a d -dimensional manifold. A frequently used model, e.g., in Bundler [24], uses $x_i = (\rho_i, \kappa_i)$ with the pose $\rho_i \in \text{SE}(3)$ and calibration $\kappa_i \in \mathbb{R}^3$, where we assume zero skew and square pixels, but optimize the focal length and two radial distortion param-

ters. Standard BA consists in the estimation of camera parameters and points positions from a set of pixel measurements $z_{ij} \in \mathbb{R}^2$, representing observations of the projection of point y_j into camera x_i . Assuming Gaussian noise, BA minimizes the negative log-likelihood of the camera parameters $X = \{x_1, \dots, x_M\}$ and points $Y = \{y_1, \dots, y_N\}$ given the measurements Z :

$$\min_{X,Y} -\log L(X,Y;Z) = \min_{X,Y} \sum_{y_j \in Y} \sum_{x_i \in X_j} \frac{1}{2} \|\pi(x_i, y_j) - z_{ij}\|_{R_{ij}}^2 \quad (1)$$

where the function $\pi(x_i, y_j)$ models the projection of point y_j into camera x_i , and $X_j \subseteq X$ is the set of cameras observing y_j . $\|e\|_{R_{ij}}^2 = e^\top R_{ij}^{-1} e$ is the (squared) Mahalanobis distance, where $R_{ij} \in \mathbb{R}^{2 \times 2}$ is the measurement covariance. For sake of simplicity we assume $R_{ij} = I$ (identity matrix), while the treatment can be easily generalized to unstructured covariances.

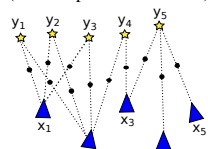
Full BA	Objective Function	Factor Graph (1 factor per measurement)	CG Primitive
	$\sum_{y_j \in Y} \sum_{x_i \in X_j} \frac{1}{2} \ F_{ij} \delta x_i + E_{ij} \delta y_j - \delta z_{ij}\ ^2$		
	Normal Equations		
	$\begin{bmatrix} F^\top F & F^\top E \\ E^\top F & E^\top E \end{bmatrix} \begin{bmatrix} \delta X \\ \delta Y \end{bmatrix} = \begin{bmatrix} F^\top \delta Z \\ E^\top \delta Z \end{bmatrix}$		

Table 1: Standard BA. The table shows the objective function to minimize in a BA iteration and its factor graph visualization, for a toy example with 5 cameras and 5 points. Minimizing the objective resorts to solving the normal equations. When using conjugate gradient as the linear solver, the key operation is the matrix-vector multiplication in the right-most column.

Problem (1) can be conveniently visualized as a *factor graph* [13]. A factor graph is a bipartite graph $\mathcal{G} = \{\mathcal{F}, \Theta\}$, where \mathcal{F} is the set of *factor nodes*, and Θ is the set of *variable nodes*. The set Θ contains the parameters to estimate, which in BA are the cameras X and the points Y . Each factor in \mathcal{F} corresponds to a measurement z_{ij} . An example of a factor graph corresponding to a small BA problem is reported in Table 1, with variable nodes denoted with triangles (cameras) and stars (points), and factor nodes denoted with dots.

A popular approach to minimize (1) is a trust-region method such as Levenberg-Marquardt (LM) [14] or Powell's dog leg [15, 16]. Both of these rely on successive linearizations, and, at each iteration, solve a linear least-squares problem. In detail, at iteration τ , the reprojection error (1) is linearized around the current estimate $\{X^{(\tau)}, Y^{(\tau)}\}$ and a quadratic approximation of (1) is minimized to find the optimal update $\{\delta X^*, \delta Y^*\}$:

$$\min_{\delta X, \delta Y} \sum_{y_j \in Y} \sum_{x_i \in X_j} \frac{1}{2} \|F_{ij} \delta x_i + E_{ij} \delta y_j - \delta z_{ij}\|^2 = \min_{\delta X, \delta Y} \frac{1}{2} \|F \delta X + E \delta Y - \delta Z\|^2. \quad (2)$$

Above, $\delta X = \{\delta x_1, \dots, \delta x_M\} \in \mathbb{R}^{dM}$ is the camera update, $\delta Y = \{\delta y_1, \dots, \delta y_N\} \in \mathbb{R}^{3N}$ is the point update, the Jacobians F_{ij} and E_{ij} have sizes $2 \times d$ and 2×3 , and $\delta z_{ij} \triangleq z_{ij} - \pi(x_i^{(\tau)}, y_j^{(\tau)})$. In the last equality in (2) we rewrote the cost in matrix form, by stacking all residual errors δz_{ij} into a vector $\delta Z \in \mathbb{R}^{2K}$, where K is the total number of available measurements. Similarly, we included the Jacobians in larger (sparse) matrices $F \in \mathbb{R}^{2K \times dM}$ and $E \in \mathbb{R}^{2K \times 3N}$. In trust-region methods the cost (2) is augmented with regularization terms, which damp the correction $\{\delta X^*, \delta Y^*\}$ improving convergence and stability. We omit these terms in our derivation, and we discuss a standard damping policy in the experimental section.

Minimizing the quadratic cost (2) requires solving a set of linear equations (*normal equations*). The linearized cost (2) and the normal equations are reported in Table 1.

3 Schur Complement as Elimination Ordering

Standard BA approaches, rather than solving the large system in Table 1, apply the Schur complement trick, which allows computing the camera update by solving a smaller problem, involving only cameras¹. The *reduced camera system* is described in Table 2.

In this section, we show that Schur complement can be understood as an *ordering* choice, when doing inference over a factor graph. Inference over the *linear* factor graphs (2) is performed by eliminating a variable at a time, according to a given *ordering*. When a variable is eliminated, it is removed from the factor graph and a new factor (connecting the variables

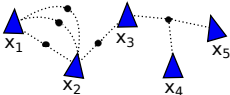
Schur complement	Objective Function	Factor Graph (1 factor per point)	CG Primitive
	$\sum_{y_j \in Y} \frac{1}{2} \ (I - E_j P_j E_j^T) (F_j \delta X_j - \delta Z_j)\ ^2$		Explicit form: $A_x = (F^T F - F^T E P E^T F)^{(*)}$ $A_x v$
	Reduced Camera System $(F^T F - F^T E P E^T F) \delta X =$ $= F^T (I - E P E^T) \delta Z$		Implicit form: $v_1 = F v$ $F^T (v_1 - (E (P (E^T (v_1))))))$

Table 2: Schur complement trick. The table shows the objective to minimize after point elimination and its factor graph visualization, for the same example of Table 1. Minimizing the objective resorts to solving the *reduced camera system*. The matrix-vector multiplication in conjugate gradient can be implemented in *explicit* or *implicit* form. (*) The computation of the matrix A_x has to be done only once, since A_x remains the same across the CG iterations.

linked to the eliminated variable) is added to the graph. For instance, in BA, the elimination of a point y_j induces a factor over the set of cameras X_j observing that point.

In order to show how to eliminate a single point, let us rewrite (2) as:

$$\min_{\delta X, \delta Y} \sum_{y_j \in Y} \frac{1}{2} \|F_j \delta X_j + E_j \delta y_j - \delta z_j\|^2 \quad (3)$$

where δX_j is a vector including the updates for all cameras observing point y_j , namely X_j , and δz_j is the vector stacking all M_j measurements of point y_j (M_j coincides with the number of cameras observing y_j). For each point y_j , we rearranged the matrices F_{ij} and E_{ij} into larger (sparse) matrices $F_j \in \mathbb{R}^{2M_j \times dM_j}$ and $E_j \in \mathbb{R}^{2M_j \times 3}$, that encode the Jacobians with respect to measurements δz_j . Since each point y_j is contained in a single summand of (3), it is easy to see that, for each camera update δX_j , the optimal point update is

$$\delta y_j^* = \arg \min_{\delta y_j} \frac{1}{2} \|[F_j \delta X_j + E_j \delta y_j - \delta Z_j]\|^2 = P_j E_j^T (\delta Z_j - F_j \delta X_j) \quad (4)$$

where $P_j \triangleq (E_j^T E_j)^{-1}$ is the 3×3 covariance on the point update δy_j . The fact that the update (4) only depends on the cameras X_j , can be easily understood from the factor graph of Table 1: each point y_j is only connected to a subset of cameras X_j , hence it is *conditionally independent* on the other variables given X_j . Point δy_j can be algebraically eliminated from (3) by substituting (4) into (3). The resulting objective function is shown in Table 2. The elimination of each point in the original factor graph induces a factor. The factor graph of Table 2 is the result of the elimination of the points in the original factor graph of Table 1.

¹Besides resulting in a smaller system, the very fact of eliminating the points yields a better conditioned system [9, Supplementary Material].

The reduced camera system, computed from the objective in Table 2, is the same produced by applying the Schur complement to the linear system in Table 1; hence, the Schur complement trick can be seen as an ordering choice, in which the points are eliminated first.

Two main methods are employed to solve the linear systems in Table 1 and Table 2: *direct* and *iterative* methods. The first class includes sparse Cholesky or QR factorization [8, 9], while the second class is dominated by conjugate gradient (CG) methods. The key operation in CG (called the *CG primitive*) is a matrix-vector multiplication Av [9], where a vector v is multiplied by the matrix A describing the linear system. For the full BA case, the CG primitive is reported in Table 1. When applying the Schur complement, one can compute the Schur complement matrix $A_x = F^\top F - F^\top E P E^\top F$ and implement the primitive as $A_x v$. The corresponding techniques are usually called *explicit* methods. However, as pointed out in [9], one can perform the CG primitive *implicitly*, without forming A_x ; the implementation of the CG primitive in explicit and implicit methods is reported in Table 2.

4 Speeding-up CG Iterations via Grouping

In this section, we first discuss the computational trade-off between implicit and explicit methods for conjugate gradient. Then we show that we can improve computation by grouping factors corresponding to points that are co-visible by the same pattern of cameras.

The complexity of a CG iteration is dictated by the complexity of the CG primitive, i.e., the matrix-vector multiplication. In order to quantify this complexity, let us consider a BA instance in which N_k points are observed by *all* cameras in the set X_k , with $|X_k| = M_k$.

An explicit method would compute the CG primitive as $A_x v$, see Table 2. Since all points are seen by all cameras, the matrix A_x is dense, and the matrix vector multiplication requires $(dM_k - 1)dM_k$ flops, which entails a complexity $\mathcal{O}(M_k^2)$. Storage is also quadratic in the number of cameras. In [9] it has been pointed out that a more efficient implementation can be obtained using *implicit* Schur complement, see Table 2. Counting the number of flops of each matrix-vector multiplication in this primitive, one gets a complexity that is $\mathcal{O}(K)$, where K is the total number of measurements. Recalling that each point is seen by all cameras, $K = M_k N_k$, and the complexity of the implicit CG primitive becomes $\mathcal{O}(N_k M_k)$.

This simple evaluation of the complexity reveals an interesting trade-off: when a set of cameras observes few common points, implicit methods are convenient, implying complexity $\mathcal{O}(N_k M_k)$. However, when the cameras observe many common points, explicit methods become advantageous: $\mathcal{O}(M_k^2)$ becomes better than $\mathcal{O}(N_k M_k)$ for N_k larger than M_k .

4.1 Factor Grouping

In our complexity analysis, we assumed the N_k points are observed in all cameras. This is not always the case in BA. Therefore, *how can we exploit this insight in general BA instances?*

The factor graph perspective gives an easy way to reason in terms of factors, rather than working on the overall matrix describing the system. For instance, in Table 2 we can see the contribution of each point to the overall cost. Similarly, we can express the primitive $A_x v$ as a sum of contributions of each factor, i.e., $A_x v = \sum_{y_j \in Y} \hat{A}_j v$, where \hat{A}_j is the contribution of factor j to the Schur complement matrix; from the objective function in Table 2 one can verify that the matrix \hat{A}_j is an augmented version of $A_j = F_j^\top F_j - F_j^\top E_j P_j E_j^\top F_j$, with suitable zero blocks for padding: the zero blocks compensate for the fact that the factor only involves cameras X_j , while the vector v has the same size of X .

Now the first question one may ask is: what is the most efficient representation (implicit or explicit) for a *single* factor? A single factor encodes a point observation from M_j cameras. Therefore, the cost of the *explicit* multiplication $\hat{A}_j v$ in CG is $\mathcal{O}(M_j^2)$ ($A_j \in \mathbb{R}^{M_j \times M_j}$ is the only nonzero part of \hat{A}_j). The dense matrix A_j , that encodes the factor, carries a very low information content: A_j is always be rank-deficient, as each camera brings a 2-dimensional measurement, but adds d unknowns. Therefore an explicit method would spend $\mathcal{O}(M_j^2)$ flops in $\hat{A}_j v$, while adding a small piece of information in the overall multiplication $A_x v$. This would suggest to *always* choose an implicit representation, which implies a cost $\mathcal{O}(M_j)$ (each factor includes a single point). In this section, we show that this is not the case.

In BA, it is common that many points are co-observed by a set of cameras. If the same set of cameras X_k observes many points (N_k), instead of using an implicit representation ($\mathcal{O}(N_k M_k)$), we can “group” these points in a single factor and use an explicit representation for such factor, which is convenient for $N_k > M_k$. Grouping is easy: it only requires summing the matrices A_j , produced by each point y_j in the group; furthermore, after grouping, we need to store a single $dM_k \times dM_k$ matrix, independently on the number of co-observed points.


Grouping	Objective Function	Factor Graph (1 factor per group)	CG Primitive
	$\sum_{k \in \text{groups}} \frac{1}{2} \left\ \left(I - (E_k^G)^P (E_k^G)^\top \right) (F_k^G \delta X_k - \delta Z_k^G) \right\ ^2$ <p>where E_k^G, F_k^G include Jacobians w.r.t. all points and cameras in the group k</p>		$\sum_{k \in \text{groups}} (\hat{A}_k v)$ <p>for each group k:</p> <p>$\hat{A}_k v$ is explicit if $N_k > M_k$ $\hat{A}_k v$ is implicit if $N_k \leq M_k$</p>

Table 3: Factor grouping. The objective function includes a summand (factor) for each group of points. The matrix-vector multiplication in CG is computed as the sum of the contributions of each factor. Depending on the number of points (N_k) and cameras (M_k) in group k , the multiplication $\hat{A}_k v$ is computed in explicit or implicit form (as defined in Table 2).

This *information compression* is particularly advantageous in BA in which we find many cameras portraying the same scene; for instance, in photo tourism, many images picture the same monument and instead of adding a less-informative factor for each point of the monument, we add a single grouped factor corresponding to a high-informative constraint on the cameras observing the same scene fragment (Fig. 2).

In summary, our strategy works as follows: we group points that are co-observed by the same cameras and we adopt an explicit representation (i.e., we explicitly compute the matrix \hat{A}_k) for the corresponding grouped factor as soon as $N_k > M_k$. When the latter condition does not hold, we use an implicit representation. This strategy is also summarized in Table 3.

So far we gave the key insight on our approach; in Section 5 we provide computational tools to find groups of points (the *fragments*) for which explicit representation is convenient.

5 Mining Structure Fragments

Selecting the points that are convenient to group resorts to finding large sets of points that are observed by the same pattern of cameras. This problem is similar to other problems in data mining literature. For instance, in *market basket analysis* [10, 11], one looks for items that recur in many *transactions* (i.e., that are frequently bought together). Similarly, in our problem, we look for patterns of cameras that occur in (i.e., observe) many points.

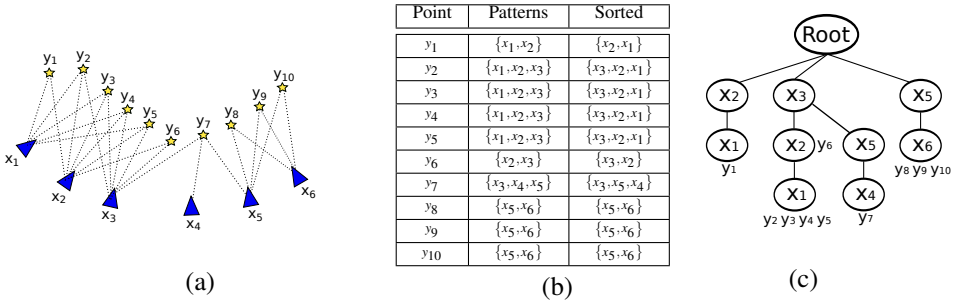


Figure 1: (a) BA example with 6 cameras and 10 points. (b) Transaction database with camera patterns (sorted by support in the last column) for each point. (c) Frequent pattern tree.

More formally, each point can be seen as a *transaction* that involves a set of *items* (the cameras). Therefore, we can define an *item set* X (set of cameras), and the *transaction database* Y (set of points). The *support* of a *pattern* of items is the number of transactions in which the pattern occurs. In BA the *support* of a pattern of cameras $X_k \subseteq X$, is the number of points that are observed by *all* cameras in X_k , hence a set X_k will have large support if it observes a large number of common points. In *frequent pattern mining* [10], given X , Y , and a minimum support $N_{\min} \in \mathbb{N}$, one looks for the *frequent item sets* (of *frequent patterns*), which are all the subsets of X having support larger than N_{\min} . In BA terminology, we look for the sets of cameras, the *frequent camera patterns*, that share enough (i.e., more than N_{\min}) common point observations. *How can we compute the frequent item sets in BA?*

Brute force approaches that check the support for each possible camera pattern are intractable in general. State-of-the-art data mining algorithms [10, 11] use an efficient data structure, the *frequent pattern tree* (FP-tree), to encode the database without storing an exponential number of candidate patterns. An example of FP-tree is given in Fig. 1.

In Fig. 1(a) we consider a small BA example. In Fig. 1(b) we show the transaction database: the first column lists the points (*transactions*), while the second column, for each point, reports the pattern of cameras observing the point. According to [9, 10], we assign an ordering to the cameras: we compute the support of each camera (number of points that the camera observes) and we order the cameras from the ones with large support to the ones with small support (last column in Fig. 1(b)). The FP-tree of Fig. 1(c) can be built from the “sorted” transactions as follows. We arrange each sorted transaction starting from the root: for transaction $\{x_2, x_1\}$ we add a node x_2 to the root, and then we add x_1 as a child. Similarly, we build the branch corresponding to $\{x_3, x_2, x_1\}$. When adding the transaction $\{x_3, x_5, x_4\}$, we create a sub-branch starting from x_3 , since x_3 was already added to the root.

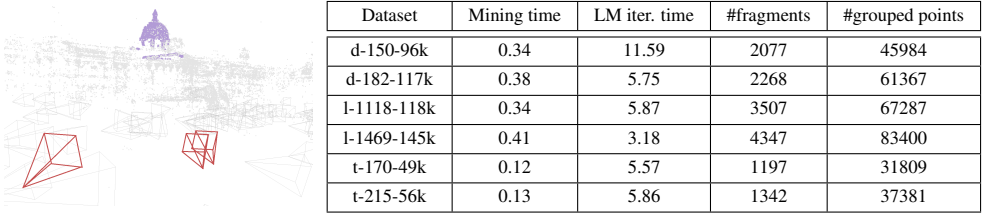
The tree can be interpreted as follows: each node is labeled with a camera, say x_i , and describes a pattern: the pattern corresponding to a node is constructed by adding, as prefix to x_i , all the cameras encountered when traversing the tree from the current node to the root. For instance, in Fig. 1(c), the node labeled with x_4 describes the pattern $\{x_3, x_5, x_4\}$ (traversing the branch, we first add as prefix x_5 and then add x_3). For each node, we also record the points observed by the corresponding pattern of cameras (the y_j listed in Fig. 1(c)). For instance, node x_4 is labeled with point y_7 as the pattern (x_4, x_5, x_3) observes that point.

After the FP-tree has been built we have to select the “fragments”, i.e., the group of points for which an explicit representation is convenient: this is done in the following section.

5.1 Selecting Optimal Scene Fragments

How can we use the FP-tree to find the patterns of M_k cameras observing $N_k > M_k$ points? We find such patterns by traversing the tree from root to leaves. If we encounter N_k points at level M_k , we decide to create a grouped factor if $N_k > M_k$. For instance, let us traverse the branch $\{x_3, x_2, x_1\}$ in Fig. 1(c). When we arrive at the leaf (level 3 of the tree), we find 4 points $\{y_2, y_3, y_4, y_5\}$, hence the factors corresponding to those points are good candidates for grouping (4 points are observed by 3 cameras). Similarly, the leaf of the right-most branch describes 3 points, seen by 2 cameras $\{x_5, x_6\}$, hence grouping is convenient. Conversely, point y_7 is a single point observed by 2 cameras, and an implicit representation is preferable.

Before concluding about the representation for the remaining points (y_1 and y_6), an observation is in order. Assume that two groups of points G_1 and G_2 are observed by the patterns of cameras X_{G_1} and X_{G_2} , and $X_{G_1} \supseteq X_{G_2}$; also assume that we decided to use an explicit representation for points in G_1 . Then, merging G_1 and G_2 always improves the cost C of each CG iteration, because $C(G_1) + C(G_2) = d^2 M_1^2 + C(G_2) > d^2 M_1^2 = C(G_1 \cup G_2)$. In words, the cost of treating the groups separately is the sum of the costs. Assuming an explicit representation for group G_1 implies a total cost $d^2 M_1^2 + C(G_2)$, and this is larger (for any nonzero $C(G_2)$) than $d^2 M_1^2$, which is the cost of considering the two groups together (G_2 involves a subset of cameras in G_1 , hence does not increase the size of the matrix A_{G_1}).



Dataset	Mining time	LM iter. time	#fragments	#grouped points
d-150-96k	0.34	11.59	2077	45984
d-182-117k	0.38	5.75	2268	61367
l-1118-118k	0.34	5.87	3507	67287
l-1469-145k	0.41	3.18	4347	83400
t-170-49k	0.12	5.57	1197	31809
t-215-56k	0.13	5.86	1342	37381

Figure 2: (a) Example of fragment (set of points in violet) seen by 3 cameras (in red). (b) Statistics regarding grouping, including time spent in mining, average time for a single LM iteration, number of fragments, and total number of points in the fragments.

According to this observation, once we committed to group a set of points observed by a pattern of cameras X_k , it is always convenient to add to the group the points that are observed by a subset of X_k . Therefore, in the example of Fig. 1 we can safely group y_1 and y_6 with $\{y_2, y_3, y_4, y_5\}$. The FP-tree further simplifies this operation: by construction, if we start from a node, corresponding to a pattern X_k , and we move towards the root, any point that we encounter is seen by a subset of X_k . Therefore, if we decided to group points corresponding to a leaf, we can safely include in the group all points in the corresponding branch.

A general algorithm works as follows: we traverse each branch of the tree from root to leaf and we find the set of points that is worth grouping ($N_k > M_k$). If we find one, we traverse the branch towards the root and we also add the points encountered along this path. Finally, for the remaining points, we check if it is possible to include them in other groups, otherwise we adopt an implicit representation for them.

6 Experiments

We tested our approach in the Bundle Adjustment in the Large benchmarking datasets [8]. We use acronyms for the datasets, e.g., the dataset *Dubrovnik* with 150 cameras and 95821 points is denoted with d-150-96k. Similar labels are used for *Ladybug*(l) and *Trafalgar*(t).

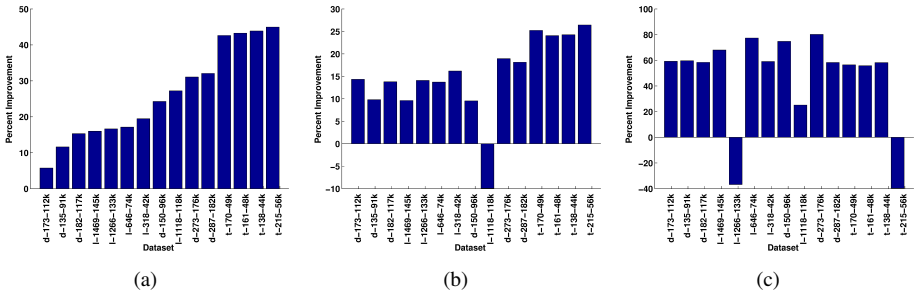


Figure 3: (a) Average improvement per CG iteration of g_{CG} w.r.t. i_{CG} . Time reduction is computed as $(t_{iCG} - t_{gCG})/t_{iCG}$ (%), where t_i is the average time for a CG iteration in technique i . (b) Total time reduction of g_{CG} w.r.t. i_{CG} . Time reduction is computed as $(T_{iCG} - T_{gCG})/T_{iCG}$ (%), where T_i is the time to reduce the objective by 90%. (c) Total time reduction of g_{CG} w.r.t. c_{CG} , using inexact Newton step: $(T_{cCG} - T_{gCG})/T_{cCG}$ (%).

Our method is implemented in C++ and released in [6]. We solve the nonlinear optimization (1) via successive linearization, using the Levenberg-Marquardt (LM) method [10]. LM adds damping terms to the cost (2), which, in a factor graph perspective, can be seen as *priors* on cameras and points. After linearization, we use preconditioned conjugate gradient to solve the reduced camera system. Before running CG, we group points as discussed in Section 5. For fragment mining, we implemented our own algorithm akin to FPmax [4]: this returns the sets of points (the fragments) for which an explicit representation is convenient. The others are represented as implicit factors. We refer to the corresponding method as g_{CG} .

We compare our technique against one using an implicit representation for all factors. The implicit method, called i_{CG} , has been implemented and released in [6]. In order to show improvements w.r.t. the state-of-the-art, we also compare our technique against a state-of-the-art solver available in the *Ceres* optimization suite [4]. This is the *Iterative Schur Solver*, later called c_{CG} . We use a Block-Jacobi preconditioning [4] in all techniques.

Fig. 2 shows an example of fragment, corresponding to a large set of points seen in 3 cameras. The figure also contains a table with statistics on grouping. For a meaningful subset of datasets we report the time required to build the FP-tree and find the groups (“mining time” column), and the average time for a single LM iteration: the time spent in data mining is negligible. The last two columns report the number of groups and the total number of points for which the explicit representation was preferable.

Fig. 3(a) shows the time improvement in the CG iterations, comparing g_{CG} and i_{CG} . Datasets, on the x-axis, are sorted in increasing time improvement. Grouping entails a reduction in the CG iteration time that ranges from 5% to 50%. The advantage varies across the datasets as it depends on the structure of the problem (i.e., existence of many fragments).

Fig. 3(b) shows the reduction in the total optimization time, comparing g_{CG} and i_{CG} . This is computed as in [4]: normalizing the objective function between 1 (initial cost) and 0 (minimum final cost across the compared techniques), we measure the time required to reduce the cost by 90%. The figure shows that grouping reduces the total optimization time by 10-25%; the overall advantage is smaller than the advantage shown in Fig. 3(a), as the total time also includes other costs (e.g., linearization) that are not influenced by the grouping. In some cases, a single LM iteration suffices to reduce the error by 90% in both techniques, but, for numerical differences, i_{CG} performs less CG iterations, hence being faster. This corresponds to the negative bar in Fig. 3(b). As future work, we plan to adopt a

different performance metric that is not affected by those fluctuations.

For a fair comparison against `Ceres`, we implemented an *inexact Newton step* [9] in `gCG`. Fig. 3(c) shows the total optimization time when using inexact Newton step, comparing `gCG` and `cCG`. Grouping leads to a time reduction of 50% (averaged across all datasets), with peaks reaching 80%; only in few cases `cCG` was able to beat `gCG`, due to early termination in `cCG`. We include more comments and results in the supplemental material, attached to this paper, to show that this advantage is consistent across a large variety of tests.

7 Conclusion

Interpreting bundle adjustment as an inference process over a factor graph improves understanding and visualization of the structure of the underlying optimization problem. Moreover, when using conjugate gradient, it allows for the selection of the best *representation* (i.e., implicit vs explicit) for each factor, without committing to a single choice over the entire reduced camera matrix, as is done in previous work. After point elimination, the factor graph modeling BA includes a single factor for each point. For these factors, explicit representation is not efficient, and implicit Schur complement is preferable. However, if we group factors corresponding to many points that are co-visible by the same set of cameras, the explicit representation becomes advantageous in terms of computation. After providing this insight, we show how to apply data mining tools to find the set of points (the *fragments*) that is convenient to group. Experimental results confirm our findings and show that the grouping entails a computational advantage in CG-based linear solvers, outperforming state-of-the-art implementations.

References

- [1] S. Agarwal, K. Mierle, and Others. Ceres solver, <https://code.google.com/p/ceres-solver/>.
- [2] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building Rome in a day. In *Intl. Conf. on Computer Vision (ICCV)*, 2009.
- [3] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. In *European Conf. on Computer Vision (ECCV)*, pages 29–42, 2010.
- [4] M. Byröd and K. Åström. Conjugate gradient bundle adjustment. In *European Conf. on Computer Vision (ECCV)*, pages 114–127, 2010.
- [5] T.A. Davis. *Direct Methods for Sparse Linear Systems*. Fundamentals of Algorithms. Society for Industrial and Applied Mathematics, 2006.
- [6] F. Dellaert et al. GTSAM: Georgia Tech Smoothing And Mapping, <https://borg.cc.gatech.edu>.
- [7] J.M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.H. Jen, E. Dunn, B. Clipp, S. Lazebnik, et al. Building Rome on a cloudless day. In *European Conf. on Computer Vision (ECCV)*, pages 368–381, 2010.
- [8] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, third edition, 1996.

- [9] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *FIMI*, volume 3, pages 110–121, 2003.
- [10] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM, 2000.
- [11] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.
- [12] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.
- [13] J. Heller, M. Havlena, A. Sugimoto, and T. Pajdla. Structure-from-motion based hand-eye calibration using $l - \infty$ minimization. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3497–3503, 2011.
- [14] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, December 1952.
- [15] Y. Jeong, D. Nister, D. Steedly, R. Szeliski, and I.S. Kweon. Pushing the envelope of modern methods for bundle adjustment. *IEEE Trans. Pattern Anal. Machine Intell.*, 34(8):1605–1617, 2012.
- [16] Y.-D. Jian, D. Balcan, and F. Dellaert. Generalized subgraph preconditioners for large-scale bundle adjustment. In *Intl. Conf. on Computer Vision (ICCV)*, 2011.
- [17] K. Konolige. Sparse sparse bundle adjustment. In *British Machine Vision Conf. (BMVC)*, September 2010.
- [18] F.R. Kschischang, B.J. Frey, and H-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47(2), February 2001.
- [19] A. Kushal and S. Agarwal. Visibility based preconditioning for bundle adjustment. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1442–1449, 2012.
- [20] M.I.A. Lourakis and A.A. Argyros. Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment? In *Intl. Conf. on Computer Vision (ICCV)*, volume 2, pages 1526–1531, 2005.
- [21] 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. doi: <http://doi.acm.org/10.1145/1486525.1486527>.
- [22] R.A. Newcombe and A.J. Davison. Live dense reconstruction with a single moving camera. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1498–1505. IEEE, 2010.
- [23] G. Sibley, C. Mei, I. Reid, and P. Newman. Vast scale outdoor navigation using adaptive relative bundle adjustment. *Intl. J. of Robotics Research*, 29(8):958–980, 2010.

- [24] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3D. In *SIGGRAPH*, pages 835–846, 2006.
- [25] N. Snavely, S. M. Seitz, and R. Szeliski. Skeletal graphs for efficient structure from motion. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.
- [26] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from Internet photo collections. *Intl. J. of Computer Vision*, 80(2):189–210, 2008.
- [27] C. Wu, S. Agarwal, B. Curless, and S.M. Seitz. Multicore bundle adjustment. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3057–3064, 2011.