Large-Scale Bundle Adjustment by Parameter Vector Partition

Shanmin Pang, Jianrue Xue, Le Wang, and Nanning Zheng

Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University

Abstract. We propose an efficient parallel bundle adjustment (BA) algorithm to refine 3D reconstruction of the large-scale structure from motion (SfM) problem, which uses image collections from Internet. Different from the latest BA techniques that improve efficiency by optimizing the reprojection error function with Conjugate Gradient (CG) methods, we employ the parameter vector partition strategy. More specifically, we partition the whole BA parameter vector into a set of individual subvectors via normalized cut (Ncut). Correspondingly, the solution of the BA problem can be obtained by minimizing subproblems on these subvector spaces. Our approach is approximately parallel, and there is no need to solve the large-scale linear equation of the BA problem. Experiments carried out on a low-end computer with 4GB RAM demonstrate the efficiency and accuracy of the proposed algorithm.

1 Introduction

The large-scale structure from motion (SfM) technique [1], [2], [3], [4], [5] which uses image collections from Internet has become a popular topic in recent years, and attracted more and more attention to the bundle adjustment (BA) technique. BA, which aims to refine a visual reconstruction to produce jointly optimal 3D structure and camera parameter estimates [6], is used as the last step of each SfM algorithm. Even though rapid progress [7], [8], [9], [10], [11] has been made in this field, the efficiency of BA algorithms is still an open problem due to the very large number of parameters involved.

Much effort has been spent on traditional BA problems (i.e., source images of SfM come from video sequences and the size of BA is usually small). Shum et al. [12] introduce an efficient way to reduce the number of parameters by using two virtual key frames to represent a sequence, and it results in a significant speedup of the BA algorithm. However, the convergence of the proposed algorithm is still a pending issue. Instead of iteratively adjusting all the structure and motion parameters, Steedly and Essa [13] propose an incremental BA algorithm that only optimizes the parameters of change when adding a new frame. Though the algorithm converges, and is faster than the original BA, it cannot work in the case that the data are highly interdependent. The technique in [14] does not solve the normal equations directly, instead it permutes the Hessian matrix of the reprojection error function by spectral partitioning such that the large problem can be partitioned into several smaller and well-conditioned subproblems. Its limitation, however, is that at each iteration, a partition is needed, which might increase the complexity of the algorithm. The method in [10] executes BA in an out-of-core manner, which decouples the original problem into several submaps, so that the problem can be solved in parallel. However, an expensive merging step is needed to obtain the final complete solution.

Recently, several methods have been presented to address large-scale BA problems. In [15], an inexact Newton method which pairs with relatively simple preconditioners is employed to get an approximation solution of the normal equations at each iteration. The approach in [16] applies the Conjugate Gradient Least Square (CGLS) algorithm to BA, which avoids formulating the Hessian matrix of the reprojection error function, thus saving memory and computing time. Another work is proposed by Wu et al. [17], in which they address BA on a multicore computer to increase efficiency. Although these methods can solve large-scale optimization problems in theory, their computing cost of the whole optimization is still huge in practice. There is still a computing power gap between the computational requirement of BA algorithms and that can be provided by a normal computer. As claimed in [15], [17], the authors perform experiments on a workstation with dual Quad-core CPUS clocked at 2.27Ghz with 48GB RAM, and another same situation happens in [16]. The latest method [11] also cannot fit into a 8GB RAM memory when using the BAL datasets [15], which further demonstrates that the state-of-the-art BA algorithms still cannot run on a low-end computer due to the high consummation of memory.

In this paper, we propose a parameter vector partition bundle adjustment (VPBA) algorithm by exploiting sparsity of large-scale BA problems. Specifically, we first use the normalized cut (Ncut) algorithm [18] to partition the whole parameter vector into a set of individual sub-vectors, then iteratively solve BA subproblems on these sub-vector spaces to converge to the optimal solution of the original BA problem. Our work is similar to methods in [10] and [14]. However, The proposed VPBA algorithm is different from the method in [10] in that VPBA does not need any merging step, which is an important and necessary step in [10]. This feature makes VPBA more suitable for large-scale BA problems. Moreover, instead of partitioning the original BA problem into several subproblems at each iteration in [14], our algorithm partitions parameter vector only once, and is therefore more simple and efficient.

In summary, the proposed VPBA algorithm has the following characteristics: 1) It does not need to compute the Hessian matrix of the reprojection error function. More importantly, the approach avoids solving large-scale linear systems, which is often a heavy load for large-scale BA problems. Therefore, a significant amount of memory and computation time can be saved. 2) The partition strategy of VPBA makes the algorithm approximately parallel, and each BA subproblem can be easily accomplished on a low-end computer. 3) The experimental results (Section 4) show that the proposed algorithm is reliable, accurate and fast in practice, though we currently cannot provide a theoretical proof of the convergence of VPBA.

The rest of the paper is organized as follows: In Section 2, a brief introduction to the BA problem is provided. The proposed VPBA algorithm is presented in Section 3 and evaluated in Section 4. Finally, a conclusion is given in Section 5.

2 Revisit Bundle Adjustment

Assume that n 3D points are observed in m views, and let x_{ij} be the projected measurement of the ith 3D point on image j. BA minimizes the reprojection error with respect to all 3D points x_i ($i \in 1, \ldots, n$) and camera parameters c_j ($j \in 1, \ldots, m$), specifically

$$\min_{S} \|f(S)\|^2 = \min_{c_j, x_i} \sum_{i=1}^{n} \sum_{j=1}^{m} \|h(c_j, x_i) - x_{ij}\|^2, \tag{1}$$

where $\|f(S)\|^2$ is the sum of squares of reprojection error, and $h(c_j, x_i)$ is the predicted projection of 3D point i on image j. For simplicity, we let $S = (c_1, \ldots, c_m, x_1, \ldots, x_n)^T$ denote all unknown parameters.

The Gauss-Newton algorithm is a standard algorithm for Eq. (1). Usually, f is approximated by a small $\|\delta_S\|$, i.e,

$$f(S + \delta_S) \approx f(S) + J\delta_S, \tag{2}$$

where J is the Jacobian matrix of f. At each iteration, minimizing $\|f(S + \delta_S)\|$ leads to the following *normal equations*:

$$(J^T J)\delta_S = -J^T f, \tag{3}$$

where $J^T J$ is an approximation to the Hessian matrix of $\|f\|^2$. However, it is difficult to meet with the requirement of a suitable step control policy to guarantee convergence of the Gauss-Newton algorithm, especially when J is rank-deficient, or nearly so. The Levenberg-Marquardt (LM) algorithm avoids this by adding a damping term λI ($\lambda > 0$) to $J^T J$, where λ is referred to as the damping term. This leads to solve the following damped system:

$$(J^T J + \lambda I)\delta_S = -J^T f. \tag{4}$$

LM is still inefficient in solving Eq. (4) when it is large-scale.

To reduce the size of the large linear system, one well known method, *Schur complement trick*, is widely adopted. Specifically, we can partition the Jacobian matrix into a camera part J_C and a point part J_P as $J = [J_C, J_P]$ by exploiting the structure of the BA parameter space. Thus $J^T J$ has the form:

$$\begin{bmatrix} J_C^T \\ J_P^T \end{bmatrix} [J_C, J_P] = \begin{bmatrix} J_C^T J_C & J_C^T J_P \\ J_P^T J_C & J_P^T J_P \end{bmatrix} = \begin{bmatrix} U & W \\ W^T & V \end{bmatrix}, \tag{5}$$

where $U \in \mathbb{R}^{mc \times mc}$ is a block diagonal matrix with m blocks of size $c \times c$, and c is the number of the parameters of a single camera; $V \in \mathbb{R}^{np \times np}$ is a block diagonal matrix with n blocks of size $p \times p$, and p is the number of the parameters of a single 3D point. Applying Gaussian elimination to Eq. (4) yields a simplified system

$$(U^* - WV^{*-1}W^T)\delta_{S_C} = -J_C^T f + WV^{*-1}J_P^T f, \tag{6}$$

where * denotes the augmentation of the diagonal elements of U and V. After we get δ_{S_C} with Eq. (6), we can then get δ_{S_P} by

$$V^*\delta_{S_P} = -J_P^T f - W^T\delta_{S_C}. \tag{7}$$

The *Schur complement trick* reduces the size of the linear system from $(mc + np) \times (mc+np)$ to $(mc) \times (mc)$. In practical applications, m is often much smaller than n, so huge amount of memory and computations can be saved.

In the case that there are several hundred cameras, Eq. (6) can be efficiently handled by many efficient strategies. One of most popular algorithms employing the *Schur complement trick* is sparse BA (SBA) [19]. SBA solves Eq. (6) via the Cholesky factorization method, and achieves a high performance. As reported in [15], SBA is successful for small problems. However, for large-scale problems ($m = 10^3 \sim 10^4$), SBA may still fail because the cost of cholesky factorization is prohibitively expensive. In order to solve this challenging problem, Conjugate Gradient (CG) methods [15], [16], [17] are used to solve Eq. (4) at the cost of obtaining an approximate solution of Eq. (4).

However, all these aforementioned BA algorithms still have to deal with huge matrix operations and the needs of solving large-scale linear systems, which are not trivial.

3 The Vector Partition BA Algorithm

In this section, we present the proposed VPBA algorithm. The latest BA algorithms proposed in [15], [16] and [17] try to improve the efficiency of solving Eq. (4). Being distinct from these approaches, the VPBA contrives to partition the whole parameter vector into a set of individual sub-vectors, and decomposes the original optimization problem Eq. (1) into a set of individual subproblems. After partitioning, each subproblem can be solved by the LM algorithm on a low-end computer. The final solution of Eq. (1) is a straightforward combination of solutions of these subproblems.

3.1 Exploiting Sparsity

BA becomes a large-scale optimization problem when the size of parameter vector S is large. Fortunately, the large-scale BA problem has useful properties of structure and sparseness. This motivates us to design an efficient BA algorithm by exploiting the structure and sparseness of the large-scale problem.

By investigating the reprojection error function Eq. (1), we find that each individual component only depends on two composite parameters c_j and x_i.

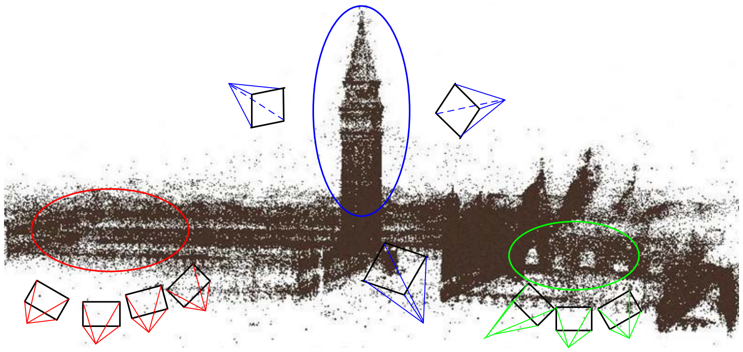

Fig. 1. Illustration of sparsity of BA. 3D points in blue ellipse are mainly reconstructed by cameras marked in blue, and these 3D points have fewer projections on images marked in red and green. Cameras marked in red and green photograph different parts of the scene, and thus no 3D points to connect images in red and green together. **Remarks**: 1) The Venice model with 1778 cameras is optimized by our VPBA algorithm, and this initial 3D model is released by Agarwal et al [15]. 2) Only schematic positions of a few cameras are illustrated.

This means that the reprojection error function is a *partial separable* function [20]. This structural property forms a solid basis for our parameter vector partition, and inspires us to consider the sparseness of the camera parameter space and 3D point parameter space separately.

Firstly, we consider the sparseness of the camera parameter space. Each camera only photographs a very small portion of landmarks due to its limited view scope. For example, for the scene containing 4.5 millon points and 13682 cameras in the BAL datasets [15], a camera covers at most 20,000 3D points, and most of cameras can only cover several hundred or thousand points. This means that if we fix j in Eq. (1), except for a few cameras, components of Eq. (1) related to camera c_j account for a very small part of the overall parameter vector.

Secondly, the sparseness also exists in the 3D points. For a specific scene, only extremely few points are simultaneously visible in the hundreds of cameras, and most of points are only observed by dozens of cameras. In other words, when we fix i in Eq. (1), the number of components of Eq. (1) depended on point x_i is relatively small, too.

For a single image, only a small fraction of the image collection can be matched with a large number of feature points. With the sparseness of the camera parameter space, we can conclude that the number of 3D points reconstructed by these matched images is small compared with the size of the whole 3D point set. Meanwhile, we can infer these reconstructed 3D points are mainly visible in these images, and have fewer projections on other images by the sparseness of 3D point parameter space. Fig. 1 illustrates this observation.

Additionally, the sparseness of the parameter space gives rise to another apparent fact: some cameras photograph different parts of a large scene, and they share no common content. This leads to a situation that we cannot reconstruct any 3D points from these images. In other words, there are no 3D points to connect these images together. This is also illustrated in Fig. 1.

Based on the aforementioned structural properties, we obtain a parameter vector partition strategy as follows: 1) partition the camera parameter vector into individual camera groups, 2) partition the 3D point parameter vector into point groups according to the partitioned camera groups. Through this partition, a large-scale BA problem can be decomposed into subproblems accordingly. This partition strategy makes the VPBA algorithm avoid huge matrix operations, such as computing and inverting of the full Hessian matrix, and thus results in speedup, and memory saving.

3.2 Partition Parameter Vector

In this section, we present the parameter vector partition in detail. To do this, let us first define a partition of vector $S \in \mathbb{R}^n$ as follows:

Definiton 1 (Partition) *For the parameter vector S composed by variables a_1, $a_2, \cdots, a_m$, where $a_i \in \mathbb{R}^{i_k}$ and $\sum_{i=1}^{m} i_k = n$. namely, $S = (a_1^T, a_2^T, \cdots, a_m^T)^T$. if S is partitioned into $S = (S_1, S_2, \cdots, S_q)^T$, where $S_j = (a_{j_1}^T, a_{j_2}^T, \cdots, a_{j_{l_j}}^T)$ and $j_t(t = 1, 2, \cdots, l_j) \in \{1, 2, \cdots, m\}$, then we say these sub-vectors S_i form a partition $\{S_1, \cdots, S_q\}$ of the original vector S. That is, $\bigcup_{j=1}^{q} S_j = S$ and $S_i \cap S_j = \emptyset$, if $i \neq j$. Moreover, $\bar{S}_j$, which is the complement vector of S_j, satisfies $\bar{S}_j \cup S_j = S$ and $\bar{S}_j \cap S_j = \emptyset$, $\forall j \in \{1, \cdots, q\}$.*

According to this definition, we can decompose any minimization problem $\min_{S \in \mathbb{R}^n} g(S)$ as

$$g(S) = g(S_j, \bar{S}_j) + g(\bar{S}_j), \tag{8}$$

where $g(\bar{S}_j)$ only depends on the parameters in $\bar{S}_j$.[1] After getting $g(\bar{S}_j)$, $g(S_j, \bar{S}_j)$ can be computed by $g(S_j, \bar{S}_j) = g(S) - g(\bar{S}_j)$.

Now, we state our method to solve the large-scale minimization problem $\min_{S \in \mathbb{R}^n} g(S)$: first, partition the vector $S \in \mathbb{R}^n$ into sub-vectors $\{S_1, \cdots, S_q\}$, and then decompose the original problem into subproblems defined on these sub-vector spaces according to Eq. (8), and finally, solve the original problem by iteratively minimizing these q subproblems (we describe details in Subsection 3.3).

Thus, the first step of VPBA becomes clear: to partition the whole parameter vector S into a partition P. Considering the convergence speed and the size of subproblems, the partition P should meet two basic requirements: 1) the size of sub-vectors should be small enough so as to be solved with a normal computer; 2) the number of coupled parameters should be as small as possible.

[1] We give a simple example to make Eq.(8) more readable: suppose $g(S) = (x_1 - x_2)^2 + (x_1 - x_3)^2 + (x_2 - x_3)^2$, and $S = (x_1, x_2, x_3)^T$; if we let $S_1 = x_1$ and $\bar{S}_1 = (x_2, x_3)^T$, then $g(S_1, \bar{S}_1) = (x_1 - x_2)^2 + (x_1 - x_3)^2$ and $g(\bar{S}_1) = (x_2 - x_3)^2$.

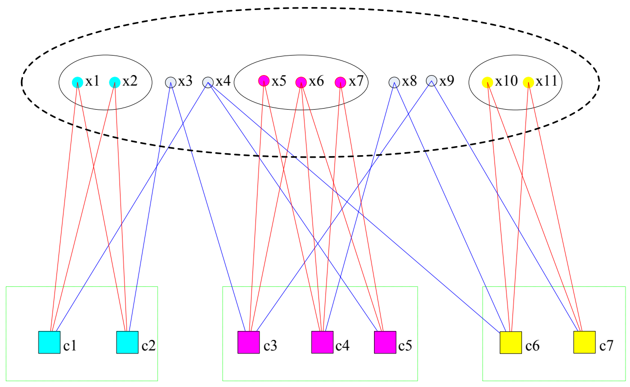

Fig. 2. Schematic illustration of decomposing all camera and 3D point parameters. This example assumes that 7 cameras observe 11 points. According to image similarities, we first use Ncut to partition 7 cameras into 3 groups: C_1, C_2, C_3, where $C_1 = \{c_1, c_2\}$, $C_2 = \{c_3, c_4, c_5\}$, $C_3 = \{c_6, c_7\}$. Correspondingly, points are partitioned into 4 groups: $X_1 = \{x_1, x_2\}$, $X_2 = \{x_5, x_6, x_7\}$, $X_3 = \{x_{10}, x_{11}\}$, $X_4 = \{x_3, x_4, x_8, x_9\}$ (see text in detail). Thus, $S_i = \{C_i, X_i\}(i = 1, 2, 3)$ and $S_4 = X_4$ constitute a partition.

In order to trade-off between these two requirements, we first divide the camera parameters by Ncut. Other grouping algorithms, such as K-means and Mean shift, may also be easily adopted in our framework. Specifically, we build an undirected weighted graph for the image collection, where each node denotes a single image, and each edge denotes the connection of each pair of images. The weighted matrix W is built in this way: w_{ij} stands for the number of 3D points that image i and image j share. This implies that if image i and image j are very similar, w_{ij} is large, and vice versa. Once W is constructed, the Ncut algorithm can partition the full image set into groups. In order to satisfy requirement (1), we adopt two-way Ncut repeatedly until every camera group contains a small number of cameras.

Next, we partition 3D point parameters into groups. Given the K camera groups (we denote them as $C_1, \cdots, C_K$) that we have already obtained by Ncut, images within a same group are with strong similarities, and jointly represent a segment of the scene. Different groups share few or no connection, and represent different segments of the scene. With this observation, we can divide the 3D point sets into two classes: intra-points, and inter-points. Intra-points are those observed by cameras within a same group C_l $(1 \leq l \leq K)$. Inter-points are points which do not meet with this condition (i.e., they are observed by cameras from at least two groups). In this way, all these 3D points are divided into $K + 1$ groups: $X_1, \cdots, X_{K+1}$, where X_l $(1 \leq l \leq K)$ is made up by intra-points, and X_{K+1} is made up by inter-points, respectively.

Finally, we do a merging step to partition the whole parameter vector into $K + 1$ sub-vectors: parameters of both the camera group C_l and 3D point group

X_l are merged to span a parameter sub-vector S_l $(1 \le l \le K)$. Thus, according to Definition 1, $P = \{S_1, \cdots, S_{K+1}\}$ constitutes a partition, where $S_{K+1} = X_{K+1}$. The complete process of parameter vector partition is illustrated in Fig. 2.

Corresponding to partition P, the original BA problem defined on vector space S is decomposed into $K+1$ subproblems defined on vector spaces $S_1, .., S_{K+1}$. These subproblems are interacted with each other only by coupled parameters. As we will see in Subsection 3.3, all inter-points (i.e., S_{K+1}) and cameras related to them make up coupled parameters. Obviously, coupled parameters are not too many based on the features of Ncut, and this can be further verified with BAL datasets [15] in Subsection 4.1.

Now, the parameter vector partition algorithm can be summarized as follows:

1. Represent the full image set as a graph and set up the weighted matrix W, then use Ncut to partition all cameras into K groups: $C_1, \cdots, C_K$.
2. Use camera groups to cut 3D points and get $K+1$ groups of points: $X_1, \cdots,$ X_{K+1}, where points in X_l $(1 \le l \le K)$ are only observed by cameras in C_l. Points which do not meet this condition form X_{K+1}.
3. Let $S_l = \{C_l, X_l\}$ $(1 \le l \le K)$ and $S_{K+1} = X_{K+1}$, then $\{S_1, \cdots, S_{K+1}\}$ is a partition of the parameter vector.

It should be noted that, in order to meet requirement 1), Ncut sometimes produces a few groups with too small size. However, this is not a problem, since we can simply merge these small groups into a group. This step is necessary, because it can reduce the number of coupled parameters, and also can keep a balance between the largest groups and the smallest ones, which is important to the parallelization of the VPBA algorithm (see Subsection 3.4).

3.3 Iterate to Convergence

Given a partition, the corresponding minimization functions have defined expressions. Let us denote f_l as a minimization function corresponding to vector space S_l. According to Eq. (8), f_l $(1 \le l \le K)$ is only dependent on set C_l, X_l, and $X_{K+1,l}$, where $X_{K+1,l}$ is a a subset of X_{K+1}, and points in it have image projections on cameras in C_l. Clearly,

$$\begin{aligned} f_l = &\sum_{x_i \in X_l} \sum_{c_j \in C_l} \|h(c_j, x_i) - x_{ij}\|^2 \\ &+ \sum_{x_i \in X_{K+1,l}} \sum_{c_j \in C_l} \|h(c_j, x_i) - x_{ij}\|^2. \end{aligned} \tag{9}$$

We fix points in set $X_{K+1,l}$ (i.e., $X_{K+1,l}$ is the constant parameter) and optimize every element of S_l when minimizing Eq. (9). This demonstrates that f_l has nothing to do with another minimization function f_q $(l \ne q)$, so that they can be minimized in parallel.

Note that compared with the original BA problem in Eq. (1), Eq. (9) only adds some fixed points. This indicates Eq. (9) and Eq. (1) have nearly the same

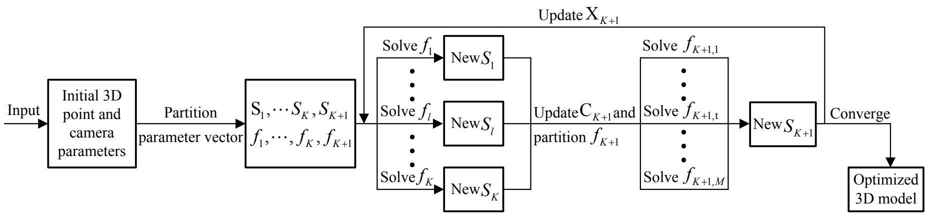

Fig. 3. The framework of the VPBA algorithm. Note that $f_1, \cdots, f_K$ are independent, so we solve them in parallel. After updating S_{K+1}, we handle f_{K+1} on M processors since it is a separable function. Moreover, the number of parameters of f_{K+1} (see Table 1) is not much, so solving it takes a little time at each iteration.

structure, but Eq. (9) has much fewer parameters to be optimized, so we can use previous BA methods with a slight modification. In the experiment, we choose the SBA algorithm to solve them, since as reported in [15], SBA has the best performance in solving these small BA problems.

Similarly, according to Eq. (8), f_{K+1} depends on inter-points (i.e., X_{K+1}) and cameras (we call C_{K+1}) which observe inter-points. Its parameters and constant parameters are X_{K+1} and C_{K+1}, respectively. Likewise, we state f_{K+1} as follows:

$$f_{K+1} = \sum_{x_i \in X_{K+1}} \sum_{c_j \in C_{K+1}} \|h(c_j, x_i) - x_{ij}\|^2. \tag{10}$$

When minimizing Eq. (10), we need to fix camera parameters and optimize inter-points. Given the camera parameters, each point can be optimized independently (i.e., f_{K+1} is *a separable function*). This means solving f_{K+1} is much easier than f_l, and its solution can be obtained only by solving $|X_{K+1}|$ linear systems with size $2|C_{K+1}| \times 3$ [12].

Finally, we can solve the original BA problem by iteratively solving subproblems as follows:

1. Fix $X_{K+1,l}$ and solve one step of f_l $(1 \leq l \leq K)$ simultaneously using any monotonically descent and convergent algorithm (for example, SBA [19]), get new $S_1, \cdots, S_K$.
2. Use the result of step 1 to update C_{K+1} into f_{K+1}, then fix C_{K+1} and solve one step of f_{K+1}, get new X_{K+1} and in turn update f_l.
3. Repeat steps 1 and 2 until convergence.

The algorithm alternates coupled parameters X_{K+1} and C_{K+1} between steps 1 and 2, and updates uncoupled parameters $X_1, \cdots, X_K$ only at step 1. Since algorithms employed for solving subproblems are monotonically descent, hence the total reprojection error is decreased at each iteration. Furthermore, VPBA can be implemented almost in parallel, and experimental results show its efficiency. Fig. 3 illustrates the entire implementation process of the VPBA algorithm.

3.4 Parallel Measures

A difficult issue in the implementation of VPBA is that, though we start $f_1, \cdots, f_K$ at the same time, they may end at different time due to their different size. In order to shorten waiting time, we reduce the exchanging frequency of coupled parameters by two measures. First, at step 1 of the VPBA algorithm, rather than just running one step of f_l, we iterate it several times, and different subproblems have different iterations (i.e., for larger subproblems, we set fewer iteration numbers, and vice versa.). This idea ensures the cost time of these subproblems is roughly the same.

Second, given the updated camera parameters C_{K+1}, we minimize f_{K+1} with respect to X_{K+1} at step 2 of the VPBA algorithm. f_{K+1} is separable and has a relatively small number of parameters (see Table 1) to be optimized. We can handle f_{K+1} on multiple processors, thus it only takes a little time in each loop. These two simple measures make the VPBA algorithm approximately parallel.

4 Experiments and Results

In this section, we evaluate the VPBA algorithm using BAL datasets released by Agarwal et al. [15]. The BAL datasets contain five categories of datasets: Dubrovnik, Final, Ladybug, Trafalgar Square and Venice. Each of them has dozens of 3D models that are reconstructed with different numbers of cameras. We choose 24 large models to evaluate the performance of the VPBA algorithm for large-scale BA problems.

The models initially contain a relatively large number of outliers. Similar to methods in [1], [16], [17], we remove outliers as follows: 1) remove 3D points that are in the back of (or close to) camera planes; 2) reject points with a large reprojection error; 3) filter out cameras whose calibration information is obvious wrong, such as focal length is negative.

4.1 Comparison with SBA

In this section, we design experiments to demonstrate whether the parameter vector partition strategy of VPBA is effective. For this purpose, we implement both VPBA and SBA using a low-end computer with 4GB RAM. Since the SBA algorithm does not use our parameter vector partition strategy, thus we can compare the performance of VPBA with SBA using the same datasets.

In the implementation of the VPBA algorithm, we stop partitioning the parameter vector until the largest camera group has fewer than 1000 cameras for each BA problem. Table 1 lists the maximum number of cameras including all the subproblems f_l $(1 \leq l \leq K)$. Inter-points and cameras related to them constitute all coupled parameters, and points usually account for most of these coupled parameters. We also list the number of inter-points for each BA problem in Table 1. It clearly shows that inter-points account for a very small fraction $(0.02 \sim 0.10)$ of all 3D points in the Venice and Ladybug datasets. It should be noted, for the Final dataset, the fraction of inter-points becomes a little larger, and is about twenty percent of all the 3D points. We think this is probably caused by two factors: 1) the connectivity graph of cameras is more complex than that in the Venice and Ladybug datasets, and grouping these cameras leads to more coupled parameters ; 2) the size of these models is much larger than that in the Venice and Ladybug datasets. To make it feasible running on a PC, we need more groups of cameras which inevitably gives rise to more coupled parameters.

For each initial model in the Ladybug and Venice datasets, VPBA and SBA are stopped by the same criteria: maximum iterations (50) or the relative

Table 1. Comparison results of VPBA and SBA. The first column corresponds to the name and index in the original datasets: "L" for "Ladybug", "V" for "Venice" and "F" for "Final". m and n denote the number of cameras and 3D points of the original problem, respectively. K is the number of camera groups and m_s stands for the maximum number of cameras including all the subproblems f_l $(1 \leq l \leq K)$. n_u is the amount of coupled 3D points. We evaluate our algorithm in terms of time (in minutes) and the final mean squared reprojection error (in pixels). The last column denotes the speed up ratio of VPBA over SBA. '–' means SBA cannot fit into memory on our platform (4GB RAM).

							VPBA		SBA		
name	m	m_s	K	n	n_u	n_u/n	error	time	error	time	r_t
L-17	969	392	3	121,633	3,856	0.03	0.86	16	0.92	157	9.8
L-19	1064	419	3	121,633	4,681	0.04	0.83	25	0.85	318	12.7
L-22	1,197	499	3	121,633	2,896	0.02	0.78	35	0.81	450	12.9
L-24	1,266	503	3	127,787	2,953	0.02	0.72	35	0.76	557	15.9
L-25	1,340	501	3	129,306	3,033	0.02	1.01	38	0.98	724	19.1
L-26	1,469	520	3	140,029	4,388	0.03	0.83	38	0.81	853	22.5
L-27	1,586	545	4	145,006	4,512	0.03	0.70	43	0.72	1061	24.7
L-29	1,690	570	4	149,121	4,593	0.03	0.72	47	–	–	–
L-31	1,712	573	4	149,707	4,598	0.03	0.77	54	–	–	–
V-04	423	271	2	272,523	16,352	0.06	2.51	25	2.47	63	2.5
V-05	740	412	2	475,217	17,234	0.04	2.02	62	1.99	198	3.2
V-09	1,179	424	3	699,114	23,752	0.03	1.97	66	1.95	573	8.7
V-11	1,281	552	3	743,047	20,405	0.03	1.81	90	1.81	1121	12.5
V-12	1,343	514	3	766,029	26,482	0.03	1.74	68	1.75	1279	18.8
V-16	1,483	430	4	796,053	26,310	0.03	1.97	62	–	–	–
V-18	1,537	444	4	802,756	28,153	0.04	1.98	67	–	–	–
V-26	1,689	439	4	840,442	72,779	0.09	1.94	53	–	–	–
V-29	1,770	486	4	849,761	86,463	0.10	1.92	54	–	–	–
F-03	869	494	2	418,517	96,572	0.23	1.59	73	1.56	231	3.2
F-04	961	546	2	161,069	30,909	0.19	1.68	48	1.70	270	5.6
F-05	1,936	599	4	561,238	150,127	0.27	1.97	119	–	–	–
F-06	3,017	875	6	252,466	41,268	0.16	1.81	293	–	–	–
F-07	4,557	922	6	1,280,289	260,998	0.20	1.53	364	–	–	–
F-08	13,608	923	**17**	3,773,337	684,261	0.18	1.78	378	–	–	–

reduction in the magnitude of the reprojection error (10^{-12}). VPBA works well on these two datasets: it's much faster than SBA (3 times $\backsim$ 24 times speedup), and can reach the comparable reprojection error (Table 1).

For models F-03 $\sim$ F-08, we run the VPBA algorithm for a maximum 100 iterations (50 iterations for SBA), and can reach the comparable reprojection error with SBA. The slow convergence on these models is probably caused by more coupled parameters than those in the Venice and Ladybug datasets. However, note that BA problems has cubic complexity, it's worth increasing the number of iterations because the size of each subproblem is much smaller than the original problem. More importantly, even with 100 iterations, our algorithm is much faster than SBA, and it can solve large-scale BA problems on a low-end computer.

4.2 Comparison with the State-of-the-Art BA Algorithms

In this section, we compare VPBA with the state-of-the-art algorithms presented in [15] in terms of speed. It's difficult for us to compare our VPBA algorithm with them directly, as they need to perform on a workstation with large memory and powerful CPU. However, we can compare VPBA with the latest algorithms indirectly. Specifically, Agarwal et al. [15] propose four new algorithms: explicit-jacobi, normal-jacobi, implicit-ssor, implicit-jacobi, and they also report the

Table 2. Compare our algorithm with four latest algorithms proposed in [15] in terms of speed. The first column denotes the test datasets. From the second column to the last column, each one shows the speed up ratio over SBA. The comparison result of VPBA with SBA is obtained in a same computing platform. The other four algorithms are compared with SBA with another computing platform, and their results are reported in [15]. Since authors of [15] do not publish their source code, so we use SBA as a basis to perform the comparison. This indirect way shows that the VPBA algorithm is much faster than SBA, hence outperforms these four latest algorithms.

name	VPBA	explicit-jacobi	implicit-jacobi	implicit-ssor	normal-jacobi
L-17	**9.8**	1.6	5.8	0.5	0.9
L-19	**12.7**	0.7	6.1	0.2	0.3
L-22	**12.9**	5.1	10.4	3.4	4.1
L-24	**16.9**	3.1	10.0	1.0	1.5
L-25	**19.1**	1.5	11.2	0.7	0.7
L-26	**22.5**	4.3	11.6	0.7	0.7
L-27	**24.7**	4.9	7.8	2.3	2.1
V-04	**2.5**	1.1	0.6	0.7	0.1
V-05	**3.2**	0.9	2.2	0.5	0.9
V-09	**8.7**	1.0	0.6	0.1	0.3
V-11	**12.5**	0.8	1.1	1.1	0.5
V-12	**18.8**	0.8	1.1	1.1	0.5
F-03	**3.2**	1.4	0.5	1.4	2
F-04	5.6	2.6	**11.9**	6.9	1.4

comparison results of these four algorithms with SBA. These comparison results can enable us to compare VPBA with these four algorithms indirectly. Table 2 lists speed up ratios of these four algorithms over SBA on platform reported in [15]. It clearly shows that explicit-jacobi, normal-jacobi, and implicit-ssor have no significant advantage than SBA. However, VPBA is much faster than SBA, so we can conclude that the VPBA algorithm is faster than these three algorithms. In addition, VPBA can compare with implicit-jacobi, which is the best of the four algorithms in [15]. All of these indicate that our VPBA algorithm is fast.

5 Conclusions

We have presented a new VPBA algorithm to large-scale BA problems that avoids huge matrix operations by decomposing the original optimization problem into subproblems. We first partition the large-scale parameter vector into a set of sub-vectors according to the features of BA problems, then define subproblems on these sub-vectors, and finally solve them iteratively. The structure of subproblems is similar with the original problem, but have much fewer number of cameras and points than the original problem, so each of them can be more efficiently solved. A key contribution of our work is that we can accomplish large-scale BA problems on a low-end computer. We demonstrate the performance of the VPBA algorithm in our experiments, and the results are promising, though we currently can not provide theoretical proof of its convergence.

Our future work includes three aspects. First, the VPBA algorithm has not reached parallel completely, so how to parallelize the algorithm is a task to investigate. Second, like other BA algorithms, the proposed algorithm converges fast during the first few steps, but slows down after dozens of steps. We will further study how to make the algorithm perform faster. Third, we will prove the convergence of the VPBA algorithm and apply this result to other large-scale optimization problems in computer vision.

References

1. Snavely, N., Seitz, S., Szeliski, R.: Modeling the world from internet photo collections. International Journal of Computer Vision 80, 189–210 (2008)
2. Agarwal, S., Snavely, N., Simon, I., Seitz, S., Szeliski, R.: Building rome in a day. In: IEEE 12th International Conference on Computer Vision, pp. 72–79 (2009)
3. Snavely, N., Seitz, S., Szeliski, R.: Skeletal graphs for efficient structure from motion. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2008)
4. Farenzena, M., Fusiello, A., Gherardi, R.: Structure-and-motion pipeline on a hierarchical cluster tree. In: IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops), pp. 1489–1496 (2009)
5. Crandall, D., Owens, A., Snavely, N., Huttenlocher, D.: Discrete-continuous optimization for large-scale structure from motion. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 3001–3008 (2011)

6. Triggs, B., McLauchlan, P., Hartley, R., Fitzgibbon, A.: Bundle adjustment–a modern synthesis. In: Vision Algorithms: Theory and practice, pp. 153–177 (2000)
7. Jeong, Y., Nister, D., Steedly, D., Szeliski, R., Kweon, I.: Pushing the envelope of modern methods for bundle adjustment. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1474–1481 (2010)
8. 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, 1178–1193 (2009)
9. Lourakis, M., Argyros, A.: Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment? In: IEEE 10th International Conference on Computer Vision, pp. 1526–1531 (2005)
10. Ni, K., Steedly, D., Dellaert, F.: Out-of-core bundle adjustment for large-scale 3d reconstruction. In: IEEE 11th International Conference on Computer Vision, pp. 1–8 (2007)
11. Jian, Y., Balcan, D., Dellaert, F.: Generalized subgraph preconditioners for large-scale bundle adjustment. In: IEEE 13th International Conference on Computer Vision, pp. 1–8 (2011)
12. Shum, H., Ke, Q., Zhang, Z.: Efficient bundle adjustment with virtual key frames: A hierarchical approach to multi-frame structure from motion. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, pp. II:538–II:543 (1999)
13. Steedly, D., Essa, I.: Propagation of innovative information in non-linear least-squares structure from motion. In: IEEE 8th International Conference on Computer Vision, pp. 223–229 (2001)
14. Steedly, D., Essa, I., Dellaert, F.: Spectral partitioning for structure from motion. In: IEEE 9th International Conference on Computer Vision, pp. 996–103 (2003)
15. Agarwal, S., Snavely, N., Seitz, S., Szeliski, R.: Bundle adjustment in the large. In: European Conference on Computer Vision, pp. 29–42 (2010)
16. Byröd, M., Åström, K.: Conjugate gradient bundle adjustment. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part II. LNCS, vol. 6312, pp. 114–127. Springer, Heidelberg (2010)
17. Wu, C., Agarwal, S., Curless, B., Seitz, S.: Multicore bundle adjustment. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 3057–3064 (2011)
18. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 22, 888–905 (2000)
19. Lourakis, M., Argyros, A.: Sba: A software package for generic sparse bundle adjustment. ACM Transactions on Mathematical Software (TOMS) 36, 1–30 (2009)
20. Nocedal, J., Wright, S.: Numerical optimization. Springer (2006)