Singly-Bordered Block-Diagonal Form for Minimal Problem Solvers

Zuzana Kukelova[1,2], Martin Bujnak[3], Jan Heller[1], and Tomas Pajdla[1]

[1]Czech Technical University in Prague, 166 27 Praha 6, Technická 2, Czech Republic
[2]Microsoft Research Ltd, 21 Station Road, Cambridge CB1 2FB, UK
[3]Capturing Reality s.r.o., Bratislava, Slovakia

Abstract. The Gröbner basis method for solving systems of polynomial equations became very popular in the computer vision community as it helps to find fast and numerically stable solutions to difficult problems. In this paper, we present a method that potentially significantly speeds up Gröbner basis solvers. We show that the elimination template matrices used in these solvers are usually quite sparse and that by permuting the rows and columns they can be transformed into matrices with nice block-diagonal structure known as the singly-bordered block-diagonal (SBBD) form. The diagonal blocks of the SBBD matrices constitute independent subproblems and can therefore be solved, i.e. eliminated or factored, independently. The computational time can be further reduced on a parallel computer by distributing these blocks to different processors for parallel computation. The speedup is visible also for serial processing since we perform $O(n^3)$ Gauss-Jordan eliminations on smaller (usually two, approximately $\frac{n}{2} \times \frac{n}{2}$ and one $\frac{n}{3} \times \frac{n}{3}$) matrices. We propose to compute the SBBD form of the elimination template in the preprocessing offline phase using hypergraph partitioning. The final online Gröbner basis solver works directly with the permuted block-diagonal matrices and can be efficiently parallelized. We demonstrate the usefulness of the presented method by speeding up solvers of several important minimal computer vision problems.

1 Introduction

The Gröbner basis method for solving systems of polynomial equations was recently used to solve many important computer vision problems [2, 7, 19, 3, 24, 25]. This method became popular for creating efficient specific solvers to minimal problems and even an automatic generator for creating source codes of such minimal Gröbner basis solvers was proposed in [18].

Minimal solvers, such as the 5-point relative pose solver [22, 25] or the P4Pf absolute pose solver [2], are often used inside a RANSAC [12] loop and are parts of large systems like SfM pipelines or recognition systems. Maximizing the efficiency of these solvers is therefore highly important.

Gröbner basis solvers usually consist of two separate steps. In the first step Gauss-Jordan (G-J) elimination, QR, or LU decomposition of one or several matrices created using the so-called elimination templates [18] is performed. In the second step the solutions are extracted from the eigenvalues and eigenvectors of a multiplication (action) matrix [23] .

Recently, several papers addressed the numerical stability and the speed of Gröbner basis solvers [5, 8, 6, 18, 21, 4]. In [5, 8, 6], it has been shown that the numerical stability of Gröbner basis solvers can be improved by reordering columns in the elimination templates using QR or LU decompositions or by "basis extension". Several methods for reducing the size of the elimination templates in order to speed up the Gröbner basis solvers were presented in [21, 18].

Most recently, two methods that speed up the second step of Gröbner basis solvers, i.e. the eigenvalue computations, were proposed in [4]. The first method is based on a modified matrix FGLM algorithm for transforming Gröbner bases and results in a single-variable polynomial which roots are efficiently computed only on a certain feasible interval using Sturm-sequences. The second method is based on fast calculation of the characteristic polynomial of an action matrix, again solved using Sturm-sequences. Both methods are in fact equivalent and can be used to significantly speed up the second step of Gröbner basis solvers.

In this paper, we present a method that can significantly speed up the first step of Gröbner basis solvers, i.e. G-J elimination, QR, or LU decomposition of matrices from the elimination templates. We observe that these elimination template matrices are usually quite sparse and by permuting the rows and the columns they can be transformed into matrices with an agreeable block-diagonal structure. The diagonal blocks of such permuted matrices constitute independent subproblems and as such can be solved, i.e. eliminated or factored, independently. The computational time can then be reduced on a parallel computer by distributing these blocks to different processors and by performing the computation in parallel. This speedup is also noticeable on single-threaded computers because $O(n^3)$ G-J eliminations on smaller (for the presented problems on two, approximately $\frac{n}{2} \times \frac{n}{2}$ and one $\frac{n}{3} \times \frac{n}{3}$) matrices is performed.

To obtain a reasonable speed-up, each block in the permuted matrix should contain comparable number of entries, i.e. some balance criterion should be maintained, and it should be as independent as possible from other blocks. For this purpose, we permute sparse rectangular matrices from the elimination templates into a singly-bordered block-diagonal (SBBD) form with minimum border size while maintaining a given balance criterion on the diagonal blocks.

The problem of permuting sparse matrices into SBBD form is usually formulated as hypergraph partitioning [1]. In this paper, we use state-of-the-art hypergraph partitioning tool PaToH [9] that uses multilevel hypergraph partitioning approaches based on Kernighan-Lin and Fiduccia-Mattheyses (KLFM) algorithms [11].

The use of row and column permutations to speed up computations is well-known in linear algebra and has been previously exploited in computer vision applications. For instance, in the bundle adjustment problem, one may transform the involved sparse matrices into arrowhead or block tridiagonal matrices [26, 20].

However, such transformations are dependent on individual problem instances and therefore are not used widely in bundle adjustment.

On the other hand, in the proposed method the permutation of an elimination template matrix into SBBD form can be computed offline once for each type of minimal problem and applied to all instances. The final online Gröbner basis solvers work directly with the permuted block-diagonal matrices and can eliminate the diagonal blocks independently and perform the computations in parallel. Moreover, the proposed method can be used along with the methods from [4] that speed up the second step of Gröbner basis solvers.

We demonstrate the usefulness of the presented approach by speeding up solvers of several important minimal computer vision problems.

Next, we briefly describe the Gröbner basis method for solving systems of polynomial equations and the process of generating Gröbner basis solvers.

2 Gröbner basis method

Gröbner basis method for solving systems of polynomial equations became very popular in the computer vision community since it helps to find fast and numerically stable solutions to difficult problems.

Let

$$f_1(\mathbf{x}) = 0, \dots, f_m(\mathbf{x}) = 0 \tag{1}$$

be a system of m polynomial equations in n unknowns $\mathbf{x} = (x_1, \dots, x_n)$ that we want to solve and let this system have a finite number of solutions. The polynomials $f_1, ..., f_m$ define an *ideal* $I = \{\Sigma_{i=1}^{m} f_i\, h_i \,|\, h_i \in \mathbb{C}\,[x_1, ..., x_n]\}$, which is a set of all polynomials that can be generated as polynomial combinations of the initial polynomials $f_1, ..., f_m$. In general, an ideal can be generated by many different sets of generators which all share the same set of solutions. There is a special set of generators though, the reduced Gröbner basis w.r.t. the lexicographic ordering, which generates the ideal I but is easy (often trivial) to solve at the same time [10].

Unfortunately, computing the Gröbner basis w.r.t. the lexicographic ordering for larger systems of polynomial equations, and therefore for the most computer vision problems, is often not feasible.

Therefore, Gröbner basis solvers usually construct a Gröbner basis G under a different ordering, e.g. the graded reverse lexicographic (grevlex) ordering, which is often easier to compute. This Gröbner basis G is then used to construct a special multiplication matrix $\mathtt{M}_p$ [23], also called the "action matrix". Let A be the quotient ring $A = \mathbb{C}\,[x_1, ..., x_n]\,/I$ [10] and $B = \left\{\mathbf{x}^\alpha | \overline{\mathbf{x}^\alpha}^G = \mathbf{x}^\alpha\right\}$ its monomial basis, where $\mathbf{x}^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} ... x_n^{\alpha_n}$ and $\overline{\mathbf{x}^\alpha}^G$ is the remainder of $\mathbf{x}^\alpha$ after the division by a Gröbner basis G. Then the action matrix $\mathtt{M}_p$ is the matrix of a linear operator $T_p\colon A \rightarrow A$ performing multiplication by a suitably chosen polynomial p in A w.r.t. the basis B.

The action matrix $\mathtt{M}_p$ can be viewed as a generalization of the companion matrix used in solving one polynomial equation in one unknown [10], since the solutions to our system of polynomial equations (1) can be obtained from the eigenvalues and eigenvectors of this action matrix.

3 Gröbner basis solvers

Many polynomial problems in computer vision share the convenient property that the monomials which appear in the set of initial polynomials (1) are, up to the concrete coefficients arising from non-degenerate image measurements, always identical. Thanks to this property, it is not necessary to use general algorithms [10] for computing Gröbner bases for solving these problems. Usually, specific Gröbner basis solvers that can efficiently solve all non-degenerate instances of a given problem are used in computer vision.

The process of creating these specific Gröbner basis solvers consist of two different phases. In the first "offline" phase, so-called "elimination templates" are found. These templates say which input polynomials should be multiplied with which monomials and then eliminated to obtain all polynomials from the grevlex Gröbner basis or at least all polynomials necessary for constructing an action matrix. For a one concrete problem, this phase is performed only once.

The second "online" phase consists of two steps. In the first step, the precomputed elimination templates are filled with specific coefficients arising from image measurements and eliminated using G-J elimination to construct the action matrix. Then, eigenvalues and eigenvectors of this action matrix are used to find solutions to the initial polynomial equations.

It was shown in [4], that the second step of the online solver can be sped up by replacing the eigenvalue computations with the computations of the characteristic polynomial of the action matrix and by efficiently finding roots of this polynomial using Sturm-sequences.

In this paper, we will show that at the cost of some preprocessing performed in the offline phase we can significantly speed up also the first step of online Gröbner basis solvers, i.e., G-J elimination, QR, or LU decomposition of matrices from the elimination templates.

Elimination templates are created in the offline phase by multiplying initial polynomials with monomials. Since we are multiplying polynomials by monomials only, the new generated polynomials have the same number of monomials as the polynomials from which they are created.

In the matrix representation of polynomials that is used in the elimination template, the rows of the matrix correspond to the individual polynomials and the columns to the monomials. This means that we are effectively only shifting the coefficients in this matrix when generating a new polynomial by multiplying some initial polynomial f_i by a monomial. A new row that corresponds to the new polynomial contains the same entries as the row corresponding to f_i, but in different columns. Therefore, the elimination template matrices are usually quite sparse. We will show that in these situations we can permute the rows and the columns of these matrices in the preprocessing offline phase and in this way create matrices that have a nice block-diagonal structure known as the singly-bordered block diagonal (SBBD) form. The diagonal blocks of such SBBD matrices can then be eliminated or factored using LU or QR decomposition independently. The computational time can be significantly reduced by distributing these blocks to different processors and by performing the computations in parallel. Moreover, there is speedup from approximately n^3 to $(k+1)\cdot\left(\frac{n}{k}\right)^3$, even for serial processing of an SBBD matrix with k well balanced blocks.

Since the permutation matrices that transform the elimination template matrix to the SBBD form are computed in the offline phase, the time spent computing these permutation matrices doesn't influence the speed of the final online solver. Moreover, the computational cost of finding the permutation matrices (for the presented solvers less than 0.1s) is comparable or even lower than the computational time of the remaining steps of the offline phase.

4 Sparse Matrix Partitioning

In this section, we describe the singly-bordered block-diagonal form and the way how to transform a given matrix to this form.

4.1 Singly-Bordered Block-Diagonal Form

Our goal is to permute the rows and the columns of an $m \times n$ sparse matrix $\mathtt{A}$ into a k-way singly-bordered block-diagonal form

$$\mathtt{A}_{SB} = \mathtt{P}_r \mathtt{A} \mathtt{P}_c^\top = \begin{bmatrix} \mathtt{A}_{11} & & & \mathtt{B}_1 \\ & \mathtt{A}_{22} & & \mathtt{B}_2 \\ & & \ddots & \vdots \\ & & \mathtt{A}_{kk} & \mathtt{B}_k \end{bmatrix}, \tag{2}$$

where $\mathtt{P}_r$ and $\mathtt{P}_c$ are, respectively, the row and the column permutation matrices to be determined, k is the pre-defined number of blocks, $\mathtt{A}_{11}, \ldots, \mathtt{A}_{kk}$ are rectangular matrices and $\mathtt{B} = \left(\mathtt{B}_1^\top \ldots \mathtt{B}_k^\top\right)^\top$ is $m \times n_c$ border submatrix. Columns of $\mathtt{B}$ are sometimes called the coupling columns.

In our case, we want to permute matrix $\mathtt{A}$ into an SBBD form $\mathtt{A}_{SB}$ (2) such that the number of coupling columns n_c is minimized while a given balanced criterion is satisfied, i.e. each block $\mathtt{A}_{jj}$, $j = 1, \ldots, k$ of the permuted matrix $\mathtt{A}_{SB}$ contains comparable number of entries.

Using hypergraph model for sparse matrices, the problem of permuting a sparse matrix to SBBD form (2) reduces to the well-known hypergraph partitioning problem [1].

Next, we describe hypergraphs and hypergraph partitioning to a level pertinent to this work.

4.2 Hypergraph partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (or hyperedges) $\mathcal{N}$, where every net $n_i \in \mathcal{N}$ is a subset of vertices, i.e. $n_i \subseteq \mathcal{V}$.

Definition 1. *Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$ is a k-way partition of $\mathcal{H}$, if the following holds:*

1. *$\mathcal{V}_j \neq \emptyset$, $\mathcal{V}_j \subseteq \mathcal{V}$, for $1 \leq j \leq k$, i.e. each part $\mathcal{V}_j$ is nonempty subset of $\mathcal{V}$,*
2. *$\mathcal{V}_i \cap \mathcal{V}_j = \emptyset$ for all $1 \leq i \leq j \leq k$, i.e. parts are pairwise disjoint,*
3. *$\bigcup_{j=1}^{k} \mathcal{V}_j = \mathcal{V}$, i.e. the union of all k parts is equal to $\mathcal{V}$.*

As in the case of graphs, we can associate weights with hypergraph vertices. Let us denote $w(v)$ the weight associated with a vertex v. Now, we can define the weight of a set of vertices $\mathcal{S}$ as

$$W(\mathcal{S}) = \sum_{v \in \mathcal{S}} w(v) . \tag{3}$$

The partition $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$ is said to be *balanced* for a given $\epsilon \geq 0$, if each part $\mathcal{V}_j$ satisfies the balance criterion

$$\frac{W(\mathcal{V}_j)}{W_{avg}} \leq 1 + \epsilon, \; j = 1, \ldots, k, \tag{4}$$

where $W(\mathcal{V}_j)$ is the weight (3) of a part $\mathcal{V}_j$ and $W_{avg} = \frac{W(\mathcal{V})}{k}$.

Let $\mathcal{N}_S$ be the set of all nets (hyperedges) that connect more than one part of a partition Π in $\mathcal{H}$. This means that all nets $n_j \in \mathcal{N}_S$ have at least one vertex in more than one part $\mathcal{V}_i \in \Pi$. Such nets $n_j \in \mathcal{N}_S$ are called *cuts* or *external nets*.

Nets that connect only one part $\mathcal{V}_i \in \Pi$, i.e. they have all vertices in this part, are called *internal nets* of part $\mathcal{V}_i$. Let us denote the set of all internal nets of a part $\mathcal{V}_i$ as $\mathcal{N}_i$, $i = 1, \ldots, k$. Then, the k-way partition $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$ that is defined on the vertex set $\mathcal{V}$ can also be considered as a $(k+1)$-way partition $\Pi = \{\mathcal{N}_1, \ldots, \mathcal{N}_k; \mathcal{N}_S\}$ on the net set $\mathcal{N}$. The set $\mathcal{N}_S$ can be considered as a net separator whose removal gives k disconnected vetrex parts $\mathcal{V}_1, \ldots, \mathcal{V}_k$ as well as k disconnected net parts $\mathcal{N}_1, \ldots, \mathcal{N}_k$.

The goal of partitioning is to minimize a cost function called *cutsize* defined over the external nets $\mathcal{N}_S$. Let $c(n)$ denote the cost associated with the net $n \in \mathcal{N}$. Then a cost function can be defined as

$$\chi(\Pi) = \sum_{n_j \in \mathcal{N}_S} c(n_j) . \tag{5}$$

In our problem of transforming a matrix to an SBBD form all nets have unit costs, i.e., $c(n) = 1$ for all $n \in \mathcal{N}$ and therefore the cost function has very simple form

$$\chi(\Pi) = \sum_{n_j \in \mathcal{N}_S} 1 = |\mathcal{N}_S| . \tag{6}$$

The hypergraph partitioning problem can be defined as a problem of dividing vertex set $\mathcal{V}$ of a hypergraph $\mathcal{H}$ into k parts, i.e. a problem of finding a k-way

$$\mathtt{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccccc} n_1 & n_2 & n_3 & n_4 & n_5 & n_6 & n_7 & n_8 \end{array} \\ \left[\begin{array}{cccccccc} \bullet & & \bullet & \bullet & & & & \bullet \\ & \bullet & & & \bullet & & \bullet & \\ & & & \bullet & & \bullet & \bullet & \\ & \bullet & & \bullet & & & & \\ & & & \bullet & & & & \bullet \\ \bullet & & \bullet & & & & \bullet & \end{array}\right] \end{array}$$

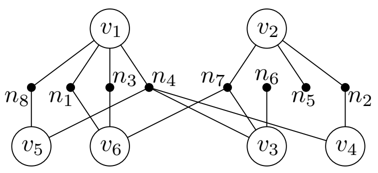

Fig. 1. A 6×8 matrix $\mathtt{A}$ (Left) and its column-net hypergraph representation (Right); symbol "$\bullet$" represents a nonzero entry of the matrix $\mathtt{A}$.

partition $\Pi = \{V_1, \ldots, V_k\}$, such that the cost (5) is minimized while the balance criterion (4) is fulfilled for a given ϵ:

$$\begin{array}{rl} \text{Given} & \mathcal{H} = (\mathcal{V}, \mathcal{N}), \epsilon, \\ \text{find} & \Pi = \arg\min \sum_{n_j \in \mathcal{N}_S} c(n_j), \\ & \Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_k\} = \{\mathcal{N}_1, \ldots, \mathcal{N}_k; \mathcal{N}_S\} \\ \text{subject to} & \frac{W(,\mathcal{V}_j)}{W_{avg}} \leq 1 + \epsilon,\ j = 1, \ldots, k. \end{array}$$

Unfortunately, the hypergraph partitioning problem is known to be NP-hard [13].

4.3 Matrix partitioning

An $m \times n$ matrix $\mathtt{A} = (a_{ij})$ can be represented as a hypergraph $\mathcal{H}_{\mathtt{A}} = (\mathcal{V}, \mathcal{N})$. In the column-net hypergraph model, $\mathcal{H}_{\mathtt{A}}$ contains m vertices and n nets (hyperedges), i.e. there exists one vertex $v_i \in \mathcal{V}$ for each row i of $\mathtt{A}$ and one net $n_j \in \mathcal{N}$ for each column j of $\mathtt{A}$. In this model, the net $n_j \subseteq V$ contains vertices corresponding to the rows that have nonzero entry in column j, i.e. $v_i \in n_j$ if and only if $a_{ij} \neq 0$. This means that the degree of the vertex v_i is equal to the number of nonzero entries in row i of $\mathtt{A}$ and the size of the net n_j is equal to the number of nonzero entries in column j of $\mathtt{A}$.

Figure 1 (Left) shows an example of a 6×8 matrix $\mathtt{A}$ and its column-net hypergraph representation (Right).

Using the hypergraph representation of the matrix $\mathtt{A}$, the problem of transforming $\mathtt{A}$ to SBBD form $\mathtt{A}_{SB}$ (2) can be cast as a hypergraph partitioning problem in which the cost (6) is equal to the number of coupling columns n_c in $\mathtt{A}_{SB}$. This claim can be formalized as theorem [1]:

Theorem 1. *Let $\mathcal{H}_{\mathtt{A}} = (\mathcal{V}, \mathcal{N})$ be the hypergraph representation of the matrix $\mathtt{A}$. A k-way partition $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_k\} = \{\mathcal{N}_1, \ldots, \mathcal{N}_k; \mathcal{N}_S\}$ of $\mathcal{H}_{\mathtt{A}}$ gives a permutation of $\mathtt{A}$ to SBBD form $\mathtt{A}_{SB}$ (2), where vertices in $\mathcal{V}_i$ represent the rows and the internal nets in $\mathcal{N}_i$ the columns of the i^{th} diagonal block of $\mathtt{A}_{SB}$, and external nets in $\mathcal{N}_S$ represent the coupling columns of $\mathtt{A}_{SB}$. Therefore,*

- *minimizing the cutsize (6) minimizes the number of coupling columns, and*
- *if the criterion (4) is satisfied, then the diagonal submatrices are balanced.*

$$\mathtt{A} = \begin{array}{c|cccccccc|} & \mathcal{N}_1 & \mathcal{N}_2 & \mathcal{N}_1 & \mathcal{N}_S & \mathcal{N}_2 & \mathcal{N}_2 & \mathcal{N}_S & \mathcal{N}_1 \\ \mathcal{V}_1 & \bullet & & \bullet & \bullet & & & & \bullet \\ \mathcal{V}_2 & & \bullet & & & \bullet & & \bullet & \\ \mathcal{V}_2 & & & & \bullet & & \bullet & \bullet & \\ \mathcal{V}_2 & & \bullet & & \bullet & & & & \\ \mathcal{V}_1 & & & & \bullet & & & & \bullet \\ \mathcal{V}_1 & \bullet & & \bullet & & & & \bullet & \end{array} \qquad \mathtt{A}_{SB} = \begin{array}{c|ccc|ccc|cc|} & n_1 & n_3 & n_8 & n_2 & n_5 & n_6 & n_4 & n_7 \\ v_1 & \bullet & \bullet & \bullet & & & & \bullet & \\ v_5 & & & \bullet & & & & \bullet & \\ v_6 & \bullet & \bullet & & & & & & \bullet \\ \hline v_2 & & & & \bullet & \bullet & & & \bullet \\ v_3 & & & & & & \bullet & \bullet & \bullet \\ v_4 & & & & \bullet & & & \bullet & \end{array}$$

Fig. 2. (Left) 2-way partitioning $\Pi = \{\mathcal{V}_1, \mathcal{V}_2\} = \{\mathcal{N}_1, \mathcal{N}_2; \mathcal{N}_S\}$ of $\mathcal{H}_{\mathtt{A}}$. (Right) SBBD form $\mathtt{A}_{SB}$ of $\mathtt{A}$ induced by Π.

Figure 2 (Left) shows a 2-way partitioning $\Pi = \{\mathcal{V}_1, \mathcal{V}_2\} = \{\mathcal{N}_1, \mathcal{N}_2; \mathcal{N}_S\}$ of the matrix $\mathcal{H}_{\mathtt{A}}$ from Figure 1 and its SBBD form $\mathtt{A}_{SB}$ induced by Π (Right).

5 Experiments

To demonstrate the usefulness of the presented approach, we used this method to speed up solvers of three important minimal relative and absolute pose problems. Even though these problems have been previously solved using the Gröbner basis method [18], the large elimination templates connected with these problems made the respective solvers relatively slow.

For each of these Gröbner basis solvers, we have precomputed the permutation matrices that transform the elimination template matrix into an SBBD form. We have formulated the problem of permuting the sparse matrix into an SBBD form as the hypergraph partitioning problem (see Section 4.3) and we have used the state-of-the-art hypergraph partitioning tool PaToH [9] to solve this problem. From PaToH, we have received the permutation matrices that we have used to permute the rows and the columns of the matrix from the elimination template.

Since PaToH uses heuristics for solving the hypergraph partitioning and each time returns a slightly different result, we have run this partitioning tool several times for every elimination template matrix. We have obtained reasonable and stable partitions from PaToH for all tested minimal solvers. In all PaToH runs, for the same elimination template matrix, PaToH returned very similar results with a similar number of coupling columns (+/- 5). We have selected the "best partitioning" as the partitioning with the smallest number of coupling columns (border) among all runs. For each problem, we have computed the permutation matrices only once in the preprocessing offline phase.

The difference between the new online Gröbner basis solvers and the original Gröbner basis solvers is that the new solvers work directly with the permuted block-diagonal matrices and therefore perform smaller G-J eliminations and moreover can be parallelized.

Since the numerical stability of the new solvers is similar to the numerical stability of the state-of-the-art solvers, in this section, we compare only the speed of the solvers.

Further, because all of the solvers are algebraically equivalent, we have evaluated them on synthetic noise free data only. In our experiments and performance tests, we executed each algorithm 10K times on synthetically generated data. All scenes in these experiments were generated using 3D points randomly distributed in a 3D cube. Each 3D point was projected by cameras with random yet feasible orientations and positions and with random or fixed focal lengths. In the case of radial distortion problems, radial distortion generated by one-parameter division model was added to all image points to generate noiseless distorted points.

Next, we describe three minimal problems, the existing Gröbner basis solvers as well as the new SBBD solutions we based on these solvers, and the gained speed-up.

5.1 9-point relative pose different radial distortion problem

Omnidirectional cameras and wide angle lenses are often used in computer vision applications. In fact, not only wide angle lenses but virtually all consumer camera lenses suffer from some amount of radial distortion. Ignoring this type of distortion, even for standard cameras, may lead to significant errors in 3D reconstruction, metric image measurements, or in camera calibration.

The first problem that we have studied is the problem of simultaneous estimation of the fundamental matrix and two radial distortion parameters for two uncalibrated cameras with different radial distortions from nine image point correspondences. This problem is useful in applications where images of one scene have been taken by two different cameras, for example by a standard digital camera and by a camera with a wide angle lens or by an omnidirectional camera.

This 9-point distortion problem can be after some manipulation of equations formulated as a system of four equations in four unknowns. Several Gröbner basis solutions to this problem exist [19, 18]. The solution [18] which results in the smallest elimination template, performs G-J elimination of a 179×203 matrix and extracts 24 solutions from the eigenvalues and eigenvectors of a 24×24 action matrix.

The large 179×203 elimination template matrix makes the 9-point distortion solver [18] relatively slow and not very useful in real applications. However, we will show that this elimination template matrix can be transformed to a matrix in SBBD form and therefore the computational time of G-J elimination can be significantly reduced.

SBBD form For the 9-point distortion solver, as well as for the two remaining studied minimal solvers, we have obtained the best results, i.e. the smallest number of the coupling columns and a well balanced blocks in SBBD matrix (2), for two blocks.

First, we have removed the last 24 columns of the 179×203 elimination template matrix $\mathtt{M}$ from the state-of-the-art solver [18]. These columns correspond to the basis B of the quotient ring $A = \mathbb{C}[x_1, ..., x_n]/I$ and should not be permuted and eliminated. Then, we have used the square 179×179 matrix as the

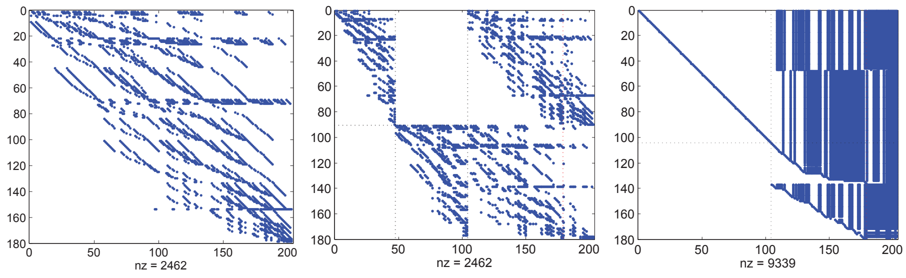

Fig. 3. 9-point radial distortion problem: (Left) the input 179×203 elimination template matrix. (Center) The SBBD form of this matrix found by PaToh for $k = 2$. The black dash-dot lines separate the independent blocks and the coupling columns with the red dash-dot line separating the last 24 basis columns. (Right) A matrix obtained after two independent G-J eliminations of the two blocks of $\mathtt{A}_{SB}$.

input to the hypergraph partitioning tool PaToH [9]. We set the weights to 1 and the PaToH imbalance ratio to 3%. These are the deafult values for PaToH and worked well for all studied solvers.

Figure 3 shows the input 179×203 elimination template matrix (Left) and its SBBD form found by PaToh for $k = 2$ (Center). In this case, the size of the first block $\mathtt{A}_{11}$ in $\mathtt{A}_{SB}$ (2) is 90×47, the size of the second block $\mathtt{A}_{22}$ is 89×57 and the number of the coupling columns n_c together with the 24 basis columns is 99. The diagonal block matrices $\mathtt{A}_{11}$ and $\mathtt{A}_{22}$ are independent and can be therefore eliminated separately.

Figure 3 (Right) shows the matrix $\mathtt{M}_r$ that we have obtained after two separate G-J eliminations of the rows of $\mathtt{A}_{SB}$ that correspond to $\mathtt{A}_{11}$ and $\mathtt{A}_{22}$. In this case, we have permuted the eliminated rows such that the identity matrix is at the top left corner.

After performing these two separate eliminations, it is sufficient to perform G-J elimination of the bottom right 75×99 submatrix of $\mathtt{M}_r$. It is not necessary to eliminate all rows of $\mathtt{M}_r$ above this bottom submatrix. To create the action matrix, we only need four from the first 104 rows and therefore it is sufficient to eliminate only these four rows from the top 104 rows of $\mathtt{M}_r$.

5.2 P4P+f problem

The second problem is the problem of estimating the absolute pose of a camera with unknown focal length from four 2D-3D point correspondences. This problem results in five equations in four unknowns and has ten solutions [2]. The P4P+f problem is very important in real Structure-from-Motion pipelines and therefore the efficiency of its solver is crucial.

As the basis of our method we have used the state-of-the art P4P+f solver downloaded from the webpage [16]. This solver performs G-J elimination of an 78×88 elimination template matrix and extracts solutions from a 10×10 action matrix.

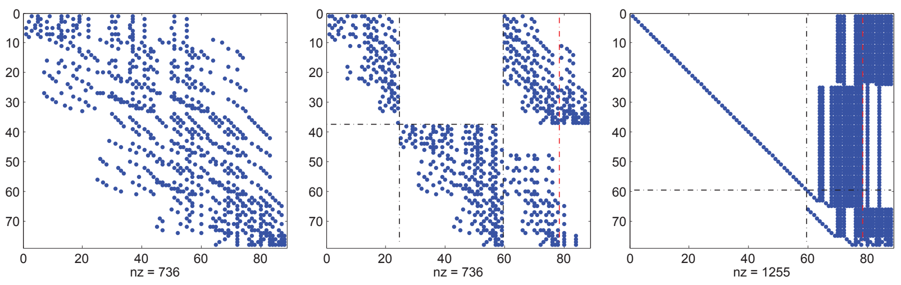

Fig. 4. P4P+f problem: (Left) the input 78×88 elimination template matrix. (Center) The SBBD form of this matrix found by PaToh for $k = 2$. The black dash-dot lines separate the independent blocks and the coupling columns and the red dash-dot line separate the last 10 basis columns. (Right) A matrix obtained after two independent G-J eliminations of the two blocks of $\mathtt{A}_{SB}$.

SBBD form For the P4P+f solver, we have precomputed in the offline phase the 2-block SBBD form $\mathtt{A}_{SB}$ (2) of its 78×88 elimination template matrix. As for the 9-point distortion problem, we have fixed the last 10 columns that correspond to the basis B of the quotient ring A and run PaToH on a square 78×78 matrix.

Figure 4 shows the input 78×88 elimination template matrix of the P4P+f solver (Left) and its SBBD form $\mathtt{A}_{SB}$ found by PaToh for $k = 2$ (Center). In this case, we have obtained nice blocks of size 37×24 and 41×35, and a relatively small number of coupling columns $n_c = 19$. This means that together with 10 basis columns we have obtained a border of size 29.

Figure 4 (Right) shows the matrix $\mathtt{M}_r$ which we have obtained after two separate G-J eliminations of the first 37 and the last 41 rows of the SBBD matrix $\mathtt{A}_{SB}$. We have again permuted the eliminated rows such that the identity matrix is in the top left corner.

After performing the two independent eliminations, it is sufficient to perform G-J elimination of the bottom right 19×29 submatrix of $\mathtt{M}_r$. Again, it is not necessary to eliminate all rows of $\mathtt{M}_r$ above this submatrix. To create the action matrix, it is sufficient to eliminate only four rows from the top 59 rows of $\mathtt{M}_r$.

5.3 P4P+f+r problem

The last problem that we have solved is the problem of estimating the absolute pose of a camera with unknown focal length and unknown radial distortion from four 2D-3D point correspondences. As shown in [15], the consideration of radial distortion in absolute pose solvers may bring a significant improvement in many real world applications. The general formulation of this problem [15] results in five equations in five unknowns and in a quite large and impractical solver (a 1134 x 720 matrix) with 24 solutions. The final solver runs in about 70ms.

A more practical solution to the P4P+f+r problem was proposed in [3]. By decomposing the problem into a nonplanar and planar cases, much simpler and

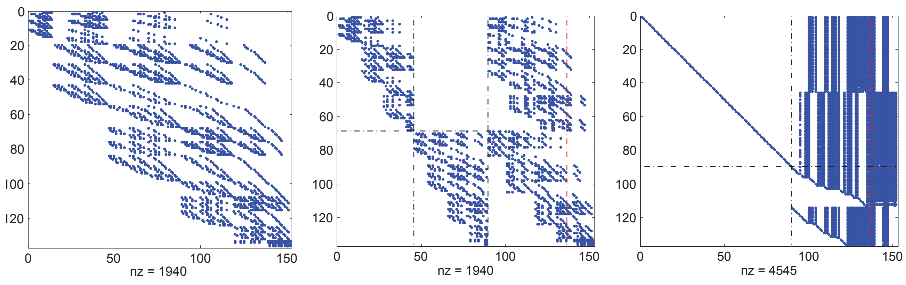

Fig. 5. P4P+f+r problem: (Left) the input 136×152 elimination template matrix. (Center) The SBBD form of this matrix found using PaToh for $k = 2$. The black dash-dot lines separate the independent blocks and the coupling columns, the red dash-dot line separates the last 16 basis columns. (Right) A matrix obtained after two independent G-J eliminations of the two blocks of $\mathtt{A}_{SB}$.

efficient solvers were obtained. The planar solver is quite simple and performs G-J elimination of a 12×18 matrix. The solution to the non-planar case requires to perform G-J elimination of a 136×152 matrix and the eigenvalue computation of a 16×16 matrix. The non-planar solver from [3] was used as the input of our new method.

SBBD form The non-planar P4P+f+r solver [3] has 16 solutions. We have first reordered the columns of the 136×152 elimination template matrix of this solver, such that the columns corresponding to the 16 dimensional basis B of the quotient ring A were at the end of this matrix. Then, we have fixed these last 16 columns and executed the hypergraph partitioning tool PaToH [9] on the square 136×136 matrix. Again, we have set the weights to 1, the number of required blocks to 2, and the imbalance ratio to 3%.

The input 136×152 elimination template matrix for the P4P+f+r solver can be seen in Figure 5 (Left) and its SBBD form $\mathtt{A}_{SB}$ found by PaToH for $k = 2$ in Figure 5 (Center). In this SBBD matrix $\mathtt{A}_{SB}$, the size of the first block $\mathtt{A}_{11}$ is 68×45, the size of the second block $\mathtt{A}_{22}$ is 68×44, and the number of the coupling columns n_c together with the 16 fixed basis columns is 63.

Figure 5 (Right) shows the matrix $\mathtt{M}_r$ that has been obtained after two separate G-J eliminations of the first 68 and the last 68 rows of the SBBD matrix $\mathtt{A}_{SB}$ and after the permutation of the eliminated rows such that the identity matrix is at the top left corner. Again, these two blocks can be eliminated independently.

After performing these two independent eliminations it is sufficient to perform G-J elimination of the bottom right 47×63 submatrix of $\mathtt{M}_r$. In this case, we need the eight of the first 89 rows to create the action matrix. Therefore, in the final step it it sufficient only to eliminate these eight rows from the top 89 rows of $\mathtt{M}_r$.

9pt orig	9pt SBBD	P4P+f orig	P4P+f SBBD	P4P+f+r orig	P4P+f+r SBBD
932.9μs	186.8μs	58.7μs	18.6μs	320.8μs	106.2μs

Table 1. Speed comparison of G-J eliminations of the original and the SBBD elimination template matrices for the three considered minimal problems.

9pt orig	9pt SBBD	P4P+f orig	P4P+f SBBD	P4P+f+r orig	P4P+f+r SBBD
362.8μs	180.6μs	23.1μs	12.1μs	116.2μs	68.3μs

Table 2. Speed comparison of sparse G-J eliminations of the original and the SBBD elimination template matrices for the three considered minimal problems.

5.4 Speedup

In our experiments we focused on the achieved speedup in the first step of the considered Gröbner basis solvers, i.e. the G-J elimination of the elimination template matrices. We reimplemented all state-of-the art solvers in C++ and used the same math libraries in all tests. In the second step of all Gröbner basis solvers, we used standard eigenvalue method [18] to find the solutions to the problem.

The second step of the Gröbner basis solvers can be sped up by replacing the eigenvalue computations with the characteristic polynomial method and Sturm sequences presented in [4]. However, the characteristic polynomial method [4] is independent from the method presented in this paper, i.e. the method from [18] speeds up a different part of Gröbner basis solvers, and therefore we didn't consider it in our experiments. The new SBBD method and the characteristic polynomial method [4] can be used concurrently to obtain final efficient solvers.

All tests were performed on an Intel i7-4700MQ 2.4Ghz based laptop.

Table 1 shows the speed comparison of G-J eliminations of the original and the SBBD elimination template matrices for the three considered minimal problems. In this case we were not exploiting the sparsity of matrices in G-J elimination. We can see that for the 9pt radial distortion solver we have achieved almost 5$\times$ speed up, and for the P4P+f solver and the P4P+f+r solver more than 3$\times$ speed ups.

Table 2 shows the same speed comparison, however, in this case for sparse G-J eliminations. We can see that the speed ups are slightly smaller. This is caused by the fact that the original elimination template matrices have sparser structure than the submatrices used in the SBBD solvers.

Note that in the case of the 9pt SBBD solver there is almost no difference in running times between sparse and general G-J eliminations. This is caused by high fill-in of matrices that appear in this SBBD solver.

6 Conclusion

In this paper, we have shown that the elimination template matrices used in popular Gröbner basis solvers are usually quite sparse and that by permuting their rows and columns can be transformed into matrices with a nice block-diagonal structure known as the singly-bordered block-diagonal (SBBD) form. The permutation of an elimination template matrix into the SBBD form can be computed in the preprocessing offline phase using hypergraph partitioning. Therefore, the time for finding the permutation matrices doesn't influence the speed of the final online Gröbner basis solver.

The final online Gröbner basis solver works directly with the permuted SBBD matrices. The computational cost of the first step of these Gröbner basis solvers is significantly reduced since we perform $O(n^3)$ G-J eliminations on smaller (for the presented solvers usually two approximately $\frac{n}{2} \times \frac{n}{2}$ and one $\frac{n}{3} \times \frac{n}{3}$) matrices. Moreover, two of these three G-J eliminations can be performed independently and therefore parallelized. We have demonstrate the usefulness of the presented method by speeding up several important minimal computer vision solvers.

Acknowledgement This work has been supported by the EC under project PRoViDE FP7-SPACE-2012-312377.

References

1. C. Aykanat, A. Pinar, U. V. Catalyurek, Permuting Sparse Rectangular Matrices into Block-Diagonal Form. *SIAM Journal on Scientific Computing, 12/2002.*
2. M. Bujnak, Z. Kukelova, T. Pajdla, A general solution to the P4P problem for camera with unknown focal length. *CVPR 2008.*
3. M. Bujnak, Z. Kukelova, and T. Pajdla. New efficient solution to the absolute pose problem for camera with unknown focal length and radial distortion. *ACCV 2010.*
4. M. Bujnak, Z. Kukelova, T. Pajdla, Making minimal solvers fast. *CVPR 2008.*
5. M. Byröd, K. Josephson, and K. Åström. Improving numerical accuracy of Gröbner basis polynomial equation solver. *ICCV 2007.*
6. M. Byröd, K. Josephson, and K. Åström. A Column-Pivoting Based Strategy for Monomial Ordering in Numerical Gröbner Basis Calculations. *ECCV 2008.*
7. M. Byröd, M. Brown, and K. Åström. Minimal Solutions for Panoramic Stitching with Radial Distortion. *BMVC 2009.*
8. M. Byröd. Numerical Methods for Geometric Vision: From Minimal to Large Scale Problems. *PhD Thesis, Centre for Mathematical Sciences, Lund University* , 2010.
9. U.V. Catalyurek, C. Aykanat. PaToH: Partitioning Tool for Hypergraphs, Version 3.1, 2011.
10. D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry, Second edition*, volume 185. Springer Verlag, Berlin - Heidelberg - New York, 2005.
11. C. M. Fiduccia and R. M Mattheyses. A linear-time heuristic for improving network partitions. *19th ACM/IEEE Design Automation Conference, 1982.*
12. M. A. Fischler and R. C. Bolles. Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. ACM*, 24(6):381–395, (1981).

13. M. Garey, D. Johnson, and L. Stockmeyer. Some Simplified NP-Complete Graph Problems, *Theoretical Computer Science, vol. 1, 1976.*
14. D. Hook, P. McAree, Using Sturm Sequences To Bracket Real Roots of Polynomial Equations, Graphic Gems I, Academic Press, 416-423, 1990.
15. K. Josephson and M. Byröd. Pose estimation with radial distortion and unknown focal length. *In CVPR09*, 2009.
16. http://cmp.felk.cvut.cz/minimal/
17. K. Kuehnle and E. Mayr. Exponential space computation of Groebner bases. *In Proceedings of ISSAC.* ACM, 1996.
18. Z. Kukelova and M. Bujnak and T. Pajdla. Automatic Generator of Minimal Problem Solvers. *ECCV 2008.*
19. Z. Kukelova, M. Byröd, K. Josephson, T. Pajdla, K. Åström. Fast and robust numerical solutions to minimal problems for cameras with radial distortion. Computer Vision and Image Understanding, Vol. 114, No. 2, pp 234-244, (2010).
20. A. Kushal and S. Agarwal. Visibility Based Preconditioning for Bundle Adjustment *CVPR 2012.*
21. O. Naroditsky and K. Daniilidis, Optimizing Polynomial Solvers for Minimal Geometry Problems, *ICCV 2011.*
22. D. Nister. An efficient solution to the five-point relative pose. *IEEE PAMI*, 26(6):756–770, 2004.
23. H. J. Stetter. *Numerical Polynomial Algebra*, SIAM, 2004.
24. H. Stewenius, D. Nister, F. Kahl, and F. Schaffalitzky. A minimal solution for relative pose with unknown focal length. *CVPR 2005.*
25. H. Stewenius, C. Engels, and D. Nister. Recent developments on direct relative orientation. *ISPRS J. of Photogrammetry and Remote Sensing*, 60:284–294, 2006.
26. B. Triggs , P. Mclauchlan , R. Hartley , A. Fitzgibbon Bundle adjustment a modern synthesis *Vision Algorithms: Theory and Practice* LNCS 1883, pp 298–375, 2000.