Decomposable Bundle Adjustment using a Junction Tree

P. Piniés  L. M. Paz
Instituto de Investigación en Ingeniería de Aragón,
Universidad de Zaragoza (Spain)
Email: {ppinies, linapaz}@unizar.es

Sebastian Haner  Anders Heyden
Department of Mathematics
Lund Institute of Technology (LTH)
Email: heyden@maths.lth.se, seb.haner@gmail.com

Abstract—The Sparse Bundle Adjustment (SBA) algorithm is a widely used method to solve multi-view reconstruction problems in vision. The critical cost of SBA depends on the fill in of the reduced camera matrix whose pattern is known as the Secondary structure of the problem. In centered object applications where a large number of images are taken in a small area the camera matrix obtained when points are eliminated is dense. On the contrary, visual mapping systems where long trajectories are traversed yields sparse matrices. In this paper, we propose a Decomposable Bundle Adjustment (DBA) method which naturally adapts to the fill in pattern of the camera matrix improving the performance on visual mapping systems. The proposed algorithm is able to decompose the normal equations into small subsystems which are ordered in a junction tree structure. To solve the original system, local factorizations of the small dense matrices are passed between clusters in the tree. The DBA algorithm has been tested for simulated and real data experiments for different environment configurations showing good performance.

I. INTRODUCTION

Bundle adjustment (BA) is a non-linear least squares technique used for the refinement of cameras and 3D structure parameters from a set of images. The standard reference is the work presented in [1]. Efficient solutions can be achieved by exploiting the sparseness of the problem, known as primary structure [2], where constraints just exist between points and cameras. The key idea is to eliminate point elements from the system and solve for the reduced camera matrix instead. For dense camera matrices the computational cost of BA is \( O(n_c^3) \) where \( n_c \) is the number of cameras. A well optimized Sparse Bundle Adjustment (SBA) implementation is presented in [3]. In [4] the secondary structure of the system (connection between cameras) is exploited for visual mapping applications where long trajectories are traversed. In this case, the camera matrix becomes a sparse system [5] which is solved using a sparse Cholesky solver (CHOLMOD [6]) along with an engineering approach to index sparse matrices properly. The final solution obtained is a MAP estimate of the camera poses and the environment structure.

In this paper, we propose a Decomposable Bundle Adjustment (DBA) method to efficiently solve the reduced camera matrix obtaining at the same time the probability marginals of the variables involved. This algorithm decomposes the original system into smaller dense submatrices which are ordered in a tree. To decompose the normal equations in an ordered and correct way we use a Junction Tree structure [7]. The result is an exact Bundle Adjustment algorithm with the following main advantages: 1- We do not have to choose between dense (e.g. LAPACK Cholesky) or sparse (e.g. CHOLMOD) linear solvers depending on the secondary structure of the problem. Since the camera matrices in the tree are dense we can always use dense solvers which are very well optimized. 2- When the optimization ends, the small Hessian in each node of the tree represent the marginal information matrix of the camera and point elements in that cluster. 3- Matrices present in different branches of the tree can be treated independently which means that parallel or distributed solutions can be implemented to efficiently solve Bundle Adjustment problems.

Using a Junction Tree structure to decompose and solve a linearized system of equations is close related to the work presented in [8] and the recent and quite interesting Bayes Tree structure proposed in [9]. The main differences with these techniques are twofold: First, these methods are based on factorizing the measurement jacobian using QR matrix decompositions and, as a consequence, they do not obtain clique marginals of the clusters but a MAP estimate. In our case the local information matrices of the cameras are decomposed using Schur complement operations, in order to calculate the belief propagation messages, which allow us to easily recover the covariance marginals for each cluster. These marginals are important to check the decisions made during data association. Second, instead of using a general ordering algorithm to solve the sparse system we take advantage of the special structure of the complete Information matrix in pure visual scenarios. In visual applications the number of elements in the structure (points) drastically outnumber the camera poses. The junction tree is built according to the reduced camera matrix, which is the costly system to solve in BA, whereas the points and observations are added afterwards. This way each tree cluster replicates in a smaller scale the same typical Primary structure of BA. As a consequence solving for the points at each tree node requires linear time.

The paper is organized as follows. In section II we explain the concept of decomposable systems. Then, in section III we show that the normal equations solved in each iteration of BA are decomposable. In section IV we present the proposed DBA algorithm with an explanation of its main protocol to perform computations along the junction tree. The results on
simulated experiments and real data are presented in section V. The potential of the DBA algorithm to be parallelizable is treated in section VI. Finally we draw the conclusions and future work in section VII.

II. DECOMPOSABLE SYSTEMS

This subsection is mainly based on [10]. A linear system $Ux = u$ is called decomposable if there is no direct relation between some of its variables. For example the following system

$$
\begin{bmatrix}
U_1 & U_{12} & 0 \\
U_{21} & U_2 & U_{23} \\
0 & U_{32} & U_3
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
=
\begin{bmatrix}
u_1 \\
u_2 \\
u_3
\end{bmatrix}
$$

is decomposable since there is no direct relation between variables $x_1$ and $x_3$ ($U_{13} = 0$ and $U_{31} = 0$). The advantage of a decomposable system is that it can be split into smaller subsystems. Equation 1 can be decomposed as $U'x' = u'$ and $U''x'' = u''$ given by:

$$
\begin{bmatrix}
U_1 & U_{12} \\
U_{21} & U_2 \\
U_{23} & U_3
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
=
\begin{bmatrix}
u_1 \\
u_2 \\
u_3
\end{bmatrix}
$$

such that $U_2 = U'_2 + U''_2$ and $u_2 = u'_2 + u''_2$. Let us express this decomposition as $U = U' \oplus U''$ and $u = u' \oplus u''$ where $\oplus$ stands for the generalized sum of matrix or vector elements according to their index.

In order to obtain a solution for the original system eqs.(2,3) cannot be solved independently since both systems are linked by $x_2$. Instead the following protocol can be applied:

1) Perform Gaussian Elimination in eq.(2) to eliminate the influence of $x_1$ on the common element $x_2$ obtaining:

$$
\begin{bmatrix}
U_1 & U_{12} \\
0 & U''_{2m}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
=
\begin{bmatrix}
u_1 \\
u''_2
\end{bmatrix}
$$

where

$$
u''_2 = u'_2 - U_{21}^{-1} u_1$$

$U''_{2m}$ is known as the Schur complement of $U_1$.

2) Add this marginal to eq.(3), i.e., $U' \oplus U''_{2m}$ and $u' \oplus u''_m$:

$$
\begin{bmatrix}
U'' + U'_{2m} & U_{23} \\
U_{32} & U_3
\end{bmatrix}
\begin{bmatrix}
x_2 \\
x_3
\end{bmatrix}
=
\begin{bmatrix}
u'_2 + u''_m \\
u_3
\end{bmatrix}
$$

3) Solve for $x_2$ and $x_3$. To solve for $x_1$ just apply Back Substitution in eq.(4):

$$
U_1 x_1 = u_1 - U_{12} x_2
$$

It can be shown that this procedure is equivalent to eliminate $x_1$ in eq.(1), solve for $x_2$ and $x_3$ and then apply back substitution to obtain $x_1$.

III. DECOMPOSABLE STRUCTURE OF BUNDLE ADJUSTMENT

Given a set of measured image feature locations $z_{ij}$, the goal of bundle adjustment is to find 3D point positions $p_j$ and camera parameters $C_i$ that minimize the L2 norm of the reprojection error $r(x)$ [11]. Using a first order approximation for the residuals yields a linearized least squares problem,

$$
\min_x \|r(x + \delta x)\|^2 \approx \|r(x) + J \delta x\|^2
$$

where $x$ comprises the set of cameras and points and $J$ is the jacobian of the residual. At each iteration, the step $\delta x$ is calculated by solving the normal equations

$$
(J^T J + \lambda I) \delta x = -J^T r
$$

The damping parameter $\lambda$ is commonly added to ensure a decreasing step when Newton directions are rejected [12]. The jacobian $J$ can be partitioned into a camera and point parts $[J_C, J_P]$. For an observation $z_{ij}$, the corresponding element of $J_C$ is given by $J_{C_{ij}} = \partial r(C_i, p_j)/\partial C_i$ whereas for $J_P$ we have $J_{P_{ij}} = \partial r(C_i, p_j)/\partial p_j$.

Figure 1 left shows a simple example that will help us study the special structure of BA. Since each row in $J$ only relates a camera and point pair, the information matrix and vector in eq.(10) inherit a special fill-in structure called **Primary structure** represented by the sparse block diagonal camera and point matrices $U, V$. Matrix $W$ represents the pairwise relation of cameras and points.

The Sparse Bundle Adjustment (SBA) algorithm [2] takes advantage of the **Primary structure** to efficiently handle the normal equations. The key idea is to eliminate the points obtaining a smaller camera system $U^m$, solve for the cameras and then apply back substitution to solve for the points. When $U^m$ is sparse, the camera system is commonly solved using CHOLMOD routines [6] in addition to reordering strategies like COLAMD while in the dense case, Cholesky factorization is implemented using LAPACK routines.

After Gaussian Elimination the reduced camera matrix $U^m$ is given by:

$$
\begin{bmatrix}
c_{11} & c_{12} & c_{13} & 0 & \cdots & \cdots \\
c_{12} & c_{22} & c_{23} & c_{24} & 0 & \cdots \\
c_{13} & c_{23} & c_{33} & c_{34} & \cdots & \cdots \\
c_{14} & c_{24} & c_{34} & c_{44} & \cdots & \cdots \\
c & \cdots & \cdots & \cdots & \cdots & \cdots
\end{bmatrix}
$$

Figure 1 right represents the new camera graph $G_C(V_C, E_C)$ obtained. The notation used for the elements in eq.(11) is chosen as a mnemotechnic rule to easily calculate the matrix $U^m$ from graph $G$ in Fig. 1 left. A diagonal element $C_{ij}$ stands for the information gained about camera $i$ due to its relation with point $j$. Off-diagonal elements $C_{kij}$ represent the indirect relation between cameras $C_i$ and $C_k$ that appears

$$
\begin{bmatrix}
U_{ij} \\
W_{ij}
\end{bmatrix}
\begin{bmatrix}
J_{C_{ij}}^T J_{C_{ij}} & J_{C_{ij}}^T J_{P_{ij}} & \sum_{i \neq j \in \text{edge}} J_{P_{ij}}^T J_{P_{ij}}
\end{bmatrix}
$$
Fig. 1. Bundle Adjustment example. (Left) The relation between cameras and points is given by the undirected graph \( G(V, E) \) where \( V \) is the set of vertices representing cameras and points and \( E \) is the edge set of \( G \). Observe that edge \((C_i, P_j) \in E\) if there exists an observation \( z_{ij} \) of point \( j \) from camera \( i \). (Right) Camera graph \( G_C(V_C, E_C) \) obtained when points are eliminated. Observe that cameras get connected if they observe a common point.

Matrices \( H \) and \( U^m \) depicted in the example can be interpreted as the corresponding adjacency matrices of the graphs.

\[
\begin{align*}
\text{Algorithm 1} \ [x, T] &= \text{DBA}(x_0, z) \\
1: & \text{Initialize } \lambda; \\
2: & x = x_0; \ k = 0; \ \text{stop} = 0; \\
3: & T = \text{buildJT}(x, z); \\
4: & r = \text{calculateJacobiansAndResiduals}(T); \\
5: & \textbf{while} \ (\text{not stop}) \ \text{and} \ (k < \text{maxIter}) \ \textbf{do} \\
6: & k = k + 1; \\
7: & \text{collectEvidence}(T \rightarrow N_{\text{root}}, \lambda); \\
8: & [\delta_k, r_{\text{new}}] = \text{distributeEvidence}(T \rightarrow N_{\text{root}}, \lambda); \\
9: & \textbf{if} \ [\delta_k] \leq \epsilon_1 (||x|| + \epsilon_1) \ \textbf{then} \\
10: & \text{stop} = 1; \\
11: & \textbf{else} \\
12: & \text{if} \ [r_{\text{new}}]^2 < ||r||^2 \ \textbf{then} \\
13: & \text{stop} = (||r|| - ||r_{\text{new}}|| < \epsilon_2 ||r||); \\
14: & x = x + \delta_k; \\
15: & \text{updateJT}(T, x); \\
16: & r = \text{calculateJacobiansAndResiduals}(T); \\
17: & \text{Decrease } \lambda; \\
18: & \textbf{else} \\
19: & \text{Increase } \lambda; \\
20: & \textbf{end if} \\
21: & \textbf{end if} \\
22: & \textbf{end while}
\end{align*}
\]

when a common observed point \( P_j \) is eliminated. These elements are calculated as follows:

\[
\begin{align*}
C_{ij} & = U_{ij} - W_{ij}V_j^{-1}W_j^T \quad (12) \\
C_{ik} & = -W_{ij}V_j^{-1}W_j^T \quad (13)
\end{align*}
\]

Equation (11) reflects that for visual mapping scenarios BA produces camera matrices \( U^m \) that can be decomposable. In fact, for long camera trajectories the camera matrix becomes increasingly sparse which is known as the Secondary structure of the problem.

IV. DECOMPOSABLE BUNDLE ADJUSTMENT USING A JUNCTION TREE

We propose a decomposable algorithm for BA based on the operations explained in section II. To deal with complex systems we use a Junction Tree structure that allows us to automatically decompose and solve the original system in the correct order.

Given a graph \( G(V, E) \) a junction tree for \( G \) is an undirected graph \( T(N, \mathcal{E}_T) \) with the following properties:

- Each vertex \( N_i \in \mathcal{N} \) is a subset of \( V \). These vertices are called clusters.
- For each edge \((V_i, V_j) \in \mathcal{E}_T \) there is some cluster \( N_k \) containing both \( V_i \) and \( V_j \).
- For any two clusters \( N_i \) and \( N_j \), all clusters on the unique path joining them contain the intersection \( N_i \cap N_j \). For each edge \((N_i, N_j) \in \mathcal{E}_T \) we associate a separator \( S_{ij} = N_i \cap N_j \).

Our proposed method is shown in Algorithm 1. Before the iteration loop starts the function buildJT builds a junction tree that decomposes the original system by distributing cameras, points and measurements to each cluster. Then, for each iteration, a two-way message passing protocol [13] is implemented to solve the normal equations using the basic operations explained in section II. In the first pass, the collectEvidence function propagates from the leaves up to the root the information of common elements between a cluster and its parent using Gaussian Elimination. In the second pass, the distributeEvidence function starts solving for variables at the root and then goes down to the leaves performing back substitution to solve for the remaining cluster variables. In the following subsections we analyze in more detail these functions.

A. Building a Junction Tree

Using \( z \) and the mnemotechnic rule explained in section III we build the adjacency matrix of cameras \( G_C \). Then a junction tree for \( G_C \) is built applying the following steps:

1. Choose an ordering for \( V_C \) and eliminate vertices. We assign each eliminated vertex and its neighbors to a cluster \( N_i \). After a node is eliminated its neighbor nodes in \( G_C \) get connected.
2. Build a graph with the maximal clusters, i.e., those that are not contained in other clusters.
3. Weight each edge of the cluster graph \((N_i, N_j) \) with the number of common elements between \( N_i \) and \( N_j \). The junction tree is given by the maximum weight spanning tree of the cluster graph.
Different junction trees can be obtained for the same graph depending on the elimination order and the maximum spanning tree chosen. In this paper we use COLAMD [6] for the elimination ordering. Figure 2 left shows an example of the construction of a junction tree. Note that a camera can belong to different clusters.

Once the cameras have been distributed we uniquely assign each point \( P_j \) and its corresponding measurements \( z_{ij} \) to a cluster of the tree. The cluster must contain all the cameras from which the point is observed. If more than one cluster fulfills the condition the point is assigned to the node with less elements. Figure 2 right shows the final junction tree obtained for our example.

B. Collect Evidence

An implementation of the function \texttt{collectEvidence} is shown in algorithm 2. For each cluster in the tree this algorithm recursively collects information from its children. To facilitate the explanation we will make use of Fig. 3 left. In the example \( N_2 \) is the root of the tree with children \( N_1 \) and \( N_3 \). Line 4 of the algorithm eliminates points from the current cluster system obtaining the reduced camera matrix \( U_{N_2}^m \). The camera marginals of the example can be calculated from Fig. 2 right using the mnemotechnic rule:

\[
U_{N_1}^m = C_{214} C_{241} C_{2451} \\
C_{214} C_{241} C_{2451} \\
C_{2514} C_{241} C_{2451}
\]

\[
U_{N_2}^m = 0 0 0 \\
0 C_{313} C_{3413} \\
0 C_{3413} C_{413}
\]

\[
U_{N_5}^m = C_{112} C_{112} C_{1312} \\
C_{112} C_{1312} 0 \\
C_{1312} 0 C_{312}
\]

Observe that the original camera matrix in eq.(11) has been decomposed as \( U^m = U_{N_1}^{m_1} \oplus U_{N_2}^{m_2} \oplus U_{N_3}^{m_3} \).

Suppose that we are in cluster \( N_1 \). In line 15 we obtain the common cameras \( s = (C_2, C_4) \) with its parent \( N_2 \). In line 18 we apply Gaussian Elimination to obtain the Schur complement \( U_{N_5}^m (C_2, C_4) \) of \( C_5 \). The same procedure is carried out for cluster \( N_3 \) obtaining the marginal \( U_{N_5}^m (C_2, C_4) \) of the common elements with \( N_2 \). Finally line 9 adds these matrices to the parent camera system, \( U_{N_5}^m = U_{N_5}^m \oplus U_{N_1}^m (C_2, C_4) \oplus U_{N_3}^m (C_2, C_4) \). Notice that we are basically following steps 1 and 2 explained in section II to solve a decomposed system.

C. Distribute Evidence

The \texttt{distributeEvidence} function is shown in algorithm 3. Since the root contains all the information required, line 3 directly solves for the correction \( \delta C_{N_2} \) of cameras in \( N_2 \). For the rest of the clusters line 9 sends back the solution of the common cameras to its children whereas line 10 performs a Back Substitution operation to solve for the remaining camera elements in the cluster. For example, \( N_1 \) receives the correction for cameras \( (C_2, C_4) \) from its parent \( N_2 \) and then calculates the correction for \( C_5 \) using Back Substitution. For all clusters in the tree line 13 performs Back Substitution to calculate point corrections. Finally, in line 17 the function is recursively called to traverse the whole tree down to the leaves.

V. RESULTS

The experimental setup is based on a MATLAB comparison between the DBA and a standard SBA implementation. For the SBA we use the MATLAB built-in CHOLMOD library to solve sparse camera matrices. In order to speed up the execution and provide the same resource conditions common operations of both algorithms (i.e. computing residuals, Jacobians, and the inverse of block diagonal matrices) have been implemented in C code using MEX functions. Both SBA and DBA run in a PC Pentium Core Quad at 2.6 GHz provided with 4Gb of RAM.
For the third experiment, cameras are located randomly. The first experiment consists on a set of cameras that are observed from at least three separated camera positions. This is a valid supposition since in real applications objects have a point detected is limited between 10 and 40 mts. This is a valid supposition since in real applications objects have different levels of description at different distances. Also, to avoid geometry configuration problems, we only use those points that are observed from at least three separated camera positions. The first experiment consists on a set of cameras located in a zig-zag path. The second experiment is designed such that cameras follow an in-outward path. For the third experiment, cameras are located randomly. For the random simulation we fixed the maximum number of points in a bounded area while in the rest of the experiments the point cloud grows as long as we add more cameras as occurs in exploration trajectories.

We have designed three synthetic experiments. Cameras are configured on a regular 3D environment where points are uniformly distributed. A bound on the depth at which a point is detected is limited between 10 and 40 mts. This is a valid supposition since in real applications objects have different levels of description at different distances. Also, to avoid geometry configuration problems, we only use those points that are observed from at least three separated camera positions. The first experiment consists on a set of cameras located in a zig-zag path. The second experiment is designed such that cameras follow an in-outward path. For the third experiment, cameras are located randomly. For the random simulation we fixed the maximum number of points in a bounded area while in the rest of the experiments the point cloud grows as long as we add more cameras as occurs in exploration trajectories.

Figure 4, shows the running time per iteration for DBA and SBA. In order to analyze the scalability, both algorithms are run several times for different number of cameras from 300 to 1500. Table I summarizes the mean cost per iteration at the maximum number of cameras along with the percentage of time reduction. The number of points and processed observations are also listed. For both algorithms the computation time per iteration considers the time required to carry out Schur Complement and Back substitution operations, as well as the time required to solve for the complete camera reduced system. For DBA we additionally consider the time of adding all the camera marginals sent by children as well as the time required to solve for the complete camera reduced system. For DBA we additionally consider the time of adding all the camera marginals sent by children as well as the time required to solve for the complete camera reduced system.

<table>
<thead>
<tr>
<th>Experiments</th>
<th>nᵖ</th>
<th>nₛ</th>
<th>tSB</th>
<th>tDBA</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zig-Zag</td>
<td>79516</td>
<td>570900</td>
<td>2.83s</td>
<td>2.41s</td>
<td>17.12</td>
</tr>
<tr>
<td>Outward</td>
<td>72134</td>
<td>570900</td>
<td>3.47s</td>
<td>2.41s</td>
<td>30.62</td>
</tr>
<tr>
<td>Random</td>
<td>69764</td>
<td>798798</td>
<td>6.83s</td>
<td>4.02s</td>
<td>41.16</td>
</tr>
</tbody>
</table>

Algorithm 2: `collectEvidence(T → Nᵢ, λ)`

```plaintext
1: uᵢ = Jᵢ, rᵢ; vᵢ = Jᵢ, rᵢ;
2: Uᵢ = Jᵢ, Jᵢ; Vᵢ = Jᵢ, Jᵢ; Wᵢ = Jᵢ, Jᵢ;
3: Vᵢ = Vᵢ + λI;
4: [uᵢ, uᵢ] = gaussElim(vᵢ, uᵢ, Vᵢ, Vᵢ, Wᵢ);
5: for j = children(T → Nᵢ) do
6:   collectEvidence(T → Nᵢ, λ);
7:   {s: elements in Sᵢ, separator} = Uᵢ;
8:   Uᵢ(s, s) = Uᵢ(s, s) + Uᵢ(s, s);
9:   uᵢ = uᵢ + uᵢ(s);
10: end for
11: if T → Nᵢ ⊈ T → Nᵢ then
12:   k = parent(T → Nᵢ);
13:   Sᵢ = Sᵢ ∪ Sᵢ;
14:   Uᵢ = Uᵢ + Uᵢ;
15:   δᵢ = backSubs(uᵢ, Uᵢ, Vᵢ, Wᵢ);
16:   δᵢ = δᵢ + δᵢ;
17: end if
```

Algorithm 3: `distributeEvidence(T → Nᵢ, λ)`

```plaintext
1: if T → Nᵢ = T → Nᵢ then
2:   Uᵢ = Uᵢ + λI;
3:   δᵢ = δᵢ + δᵢ;
4: else
5:   k = parent(T → Nᵢ);
6:   δᵢ = backSubs(uᵢ, Uᵢ, Vᵢ, Wᵢ);
7:   δᵢ = δᵢ + δᵢ;
8:   for j = children(T → Nᵢ) do
9:     distributeEvidence(T → Nᵢ, λ);
10: end for
```

A. Running Time Evaluation on Synthetic Experiments

<table>
<thead>
<tr>
<th>Experiments</th>
<th>nᵖ</th>
<th>nₛ</th>
<th>tSB</th>
<th>tDBA</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zig-Zag</td>
<td>79516</td>
<td>570900</td>
<td>2.83s</td>
<td>2.41s</td>
<td>17.12</td>
</tr>
<tr>
<td>Outward</td>
<td>72134</td>
<td>570900</td>
<td>3.47s</td>
<td>2.41s</td>
<td>30.62</td>
</tr>
<tr>
<td>Random</td>
<td>69764</td>
<td>798798</td>
<td>6.83s</td>
<td>4.02s</td>
<td>41.16</td>
</tr>
</tbody>
</table>
camera graph such that the secondary structure gets denser leading to a quadratic time growth. This occurs because we increase $n_C$ in a fixed size area. In average, DBA takes 49.8% less of the time required by SBA. We have verified in all cases that the calculations of residuals, jacobians and forming the Hessian submatrices take the same time for both SBA and DBA algorithms.

B. Evaluation on Real Experiments

We have carried out a comparison of DBA and the standard SBA for four real image collections. Three of the datasets make part of the Photo Tourism Microsoft project [14]. Table II summarizes the setup configuration for each collection. The first dataset corresponds to a tree-lined avenue traversed by a car with a mounted Ladybug sensor providing six images at each acquisition instant. The second dataset is an ordered sequence of a public square gathered at our downtown. The third collection is a set of tourist images taken at Trafalgar Square. Finally, the fourth dataset corresponds to tourist pictures of a portion of Saint Mark Square in Venice. For all datasets, a modified version of Bundler software [14] with partial optimization has been used to obtain the initial seed. The experiments present different fill-in patterns for the reduced camera matrix as shown in Fig. 5, left column.

For ladybug, public square and Trafalgar datasets, we can identify separate regions in the secondary structure matrix mainly due to a distributed location of the cameras in the experiments. In these cases, there is an important computational time reduction for DBA, specially in the Ladybug and public square experiments where DBA requires approx. 50% less time than SBA. The Venice experiment is an example of little gain using DBA since all cameras are looking to a common part of the scene. For object centered applications this indicates that DBA and SBA achieve almost the same performance. Observe that the junction tree built for the Venice experiment gets automatically adapted to the dense relation between cameras since only 4 consecutive clusters are generated, see Fig. 5 bottom row.

VI. PARALLELIZATION

The most costly operations of a BA method are the calculations needed to build the camera and point jacobians, the construction of the information vectors and matrices and the solution of the linear system of equations. An important feature of the proposed algorithm is its potential capability to parallelize or distribute in several machines most of these operations.

Since all nodes of the tree have an independent set of observations the calculateJacobiansAndResiduals function can be run in parallel for all nodes simultaneously. For the collect and distribute functions we can take advantage of the junction tree structure. The key insight is that calculations performed in different branches of the tree are independent and therefore can be parallelized. For example, line 7 in the collect algorithm 2 can be run simultaneously for all children of the current node. This can be easily understood using Fig. 3 left. Observe that the matrices send to the root $N_2$ by clusters $N_1$ and $N_3$ can be calculated simultaneously. Similarly, line 17 in the distribute function in algorithm 3 can also be called in parallel. Table II shows for each of the real experiments the number of potentially parallelizable branches $n_{PB}$. Observe that for the first three experiments a Core Quad processor could take advantage of its four cores since $n_{PB} > 4$. However, it would be interesting to reduce the depth of the junction tree $JT_{depth}$ in order to make the branches of the tree more balanced and increase $n_{PB}$.

VII. CONCLUSIONS

In this paper we have analyzed the decomposable structure of Bundle Adjustment which depends on the sparseness of the reduced camera matrix. To take advantage of this property, we have proposed a new Decomposable Bundle Adjustment (DBA) algorithm that automatically splits the original normal equations into smaller systems using a junction tree. A good capability of DBA is that the tree structure gets adapted to the Secondary structure of the problem revealing its natural sparsity. To solve the decomposed system a two-way passing algorithm based on local Gaussian Elimination and Back Substitution operations is implemented. An interesting side effect of the message passing protocol is that each clique tree obtains its corresponding marginal information matrix that can be used to check or carry out data association decisions. The performance of the proposed algorithm has been empirically evaluated using simulated and real experiments. In these experiments the DBA algorithm obtains good results compared to a standard SBA implementation for a wide range of camera configurations.

A very appealing property of DBA is that matrices present in different branches of the tree can be treated independently. This makes DBA specially suitable for parallel and distributed implementations. For future research we are interested in developing algorithms for multiple core systems using this technique as well as analyzing good ordering strategies to build junction trees that increase the number of
Fig. 5. Real experiments: Ladybug, Public Square, Trafalgar and Venice. The secondary structure matrix (Left column). Junction Tree built (middle column). Obtained BA reconstruction (right column).

<table>
<thead>
<tr>
<th>Experiments</th>
<th>$n_C$</th>
<th>$n_P$</th>
<th>$n_z$</th>
<th>$t_{SBA}$</th>
<th>$t_{DBA}$</th>
<th>$%$</th>
<th>$n_V$</th>
<th>$JT_{depth}$</th>
<th>$n_{PB}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LADYBUG</td>
<td>783</td>
<td>83581</td>
<td>325140</td>
<td>27.57s</td>
<td>11.73s</td>
<td>57.45</td>
<td>56</td>
<td>112</td>
<td>16</td>
</tr>
<tr>
<td>PUBLIC SQUARE</td>
<td>519</td>
<td>32086</td>
<td>286162</td>
<td>4.45s</td>
<td>2.38s</td>
<td>46.52</td>
<td>22</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>TRAFALGAR</td>
<td>256</td>
<td>65127</td>
<td>225688</td>
<td>1.97s</td>
<td>1.49s</td>
<td>24.37</td>
<td>86</td>
<td>95</td>
<td>19</td>
</tr>
<tr>
<td>VENICE</td>
<td>87</td>
<td>110844</td>
<td>554826</td>
<td>5.59s</td>
<td>5.04s</td>
<td>9.86</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

TABLE II
Real experiments setup. Detail of fields: Number of cameras $n_C$, number of points $n_P$, number of observations $n_z$, number of junction tree clusters $n_V$, Junction Tree depth $JT_{depth}$, number of parallelizable branches $n_{PB}$.
parallel branches. In addition, we are interested in developing an incremental version of this algorithm following the nice ideas presented in [9].

REFERENCES


