数理解析研究所講究録
1040 巻 1998 年 125-134

Computing the Combinatorial Canonical Form of a Layered Mixed Matrix

京大数理研　室田一雄 (Kazuo Murota, Kyoto U)
ミュンヘン工科大　マーク・シャーブロート (Mark Scharbrodt, TU–München)

1. Introduction. A matrix $A = \begin{pmatrix} Q \\ T \end{pmatrix}$ is said to be *layered mixed* (or an *LM–matrix*), if the set of nonzero entries of T is algebraically independent over the field to which the entries of Q belong. Its *Combinatorial Canonical Form* (*CCF* for short) is a (combinatorially unique) finest block-triangular representation of the matrix under the admissible transformation of the form

$$P_r \begin{pmatrix} S & O \\ O & I \end{pmatrix} \begin{pmatrix} Q \\ T \end{pmatrix} P_c,$$

where S is a nonsingular matrix, and P_r and P_c are permutation matrices. In the CCF, each diagonal block is a full rank square matrix. For a singular or rectangular matrix the CCF includes also full rank horizontal and/or vertical tails. Note that the CCF reduces to the well-known Dulmage–Mendelsohn decomposition [BR91, DER86, DM59, DR78, EGLPS87, Gu76, Ho76, PF90] if the Q-part is empty.

Example 1. If t_i $(i = 1,2,3,4)$ denote independent parameters, A below is a 4×5 LM–matrix, and its CCF is given by $\tilde{A}$, which consists of a 1×2 horizontal tail and a 3×3 square nonsingular block, with an empty vertical tail:

$$A = \begin{pmatrix} Q \\ T \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 2 & 1 & 1 & 0 \\ t_1 & 0 & 0 & 0 & t_2 \\ 0 & t_3 & 0 & 0 & t_4 \end{pmatrix}, \quad \tilde{A} = \begin{pmatrix} 1 & 1 & 0 & 2 & 0 \\ & & 1 & -1 & 0 \\ & & t_1 & 0 & t_2 \\ & & 0 & t_3 & t_4 \end{pmatrix}.$$

The concept of mixed matrices was introduced by Murota–Iri [MI85] as a tool for describing discrete physical/engineering systems (see [Mu96] for exposition), and subsequently, the CCF of LM-matrices was established by Murota–Iri–Nakamura [MIN87] and Murota [Mu87]. An efficient algorithm for computing the CCF was designed with matroid theoretical methods (submodular flow model). This algorithm, to be described in Section 2, operates in two phases; the first phase detects a maximal independent assignment in an auxiliary network, and the second phase finds the decomposition.

In the present paper, we deal with practical computing of the CCF. Since engineering applications usually are large scale, it is important to identify typical characteristics in practical situations in order to significantly speed up the algorithm on top of its theoretical efficiency. In that line, based on the original algorithm, we will present practically faster versions which use simple but very effective procedures in order to improve solving the underlying independent assignment subproblem. Also, we discuss implementation strategies. We implemented the algorithm in the *Mathematica* language which is suitable especially for symbolic computation. The code is available via anonymous ftp (see Appendix A for details).

2. The original CCF–algorithm. The CCF of a layered mixed matrix can be computed by first identifying a maximum independent assignment in an associated bipartite graph, and then applying the Min–Cut–decomposition to the resulting auxiliary network. This is based on the fact that the rank of a layered mixed matrix can be characterized by the minimum value of a certain submodular function that can be represented as the cut function of an independent assignment problem. In what follows, we describe the algorithm of [Mu93], while referring the reader to [Mu87][Mu93] for theoretical backgrounds.

For a layered mixed matrix A, we define $R_T = \mathrm{Row}(T)$ and $C = \mathrm{Col}(A)$. Furthermore, let C_Q denote a disjoint copy of C, where the copy of $j \in C$ will be denoted as $j_Q \in C$. The network for the underlying independent assignment problem is a directed graph $G = (V, B)$ with vertex set $V = R_T \cup C_Q \cup C$ and arc set $B = B_T \cup B_C \cup B^+ \cup M$, where $B_T = \{(i,j) \mid i \in R_T,\ j \in C,\ T_{ij} \neq 0\}$, $B_C = \{(j_Q, j) \mid j \in C\}$ and B^+ and M are arcs which are dynamically defined with respect to the current independent assignment; B^+ allows to perform exchanges in the base of matrix Q, whereas M is built by the reversals of the arcs matched in the assignment. We denote by ∂M the set of end vertices of M.

In addition, the algorithm works on two matrices P and S and a vector *base*. At the beginning of the algorithm, P is set to Q and finally, after executing pivotings, P can be permuted to a block triangular matrix according to the CCF decomposition. The other matrix S gives the matrix S in the admissible transformation. The variable *base* is a vector of size m_Q, which represents a mapping $R_Q \to C \cup \{0\}$. Then, the algorithm can be stated as follows, where Step 1 to Step 3 compute the independent assignment and Step 4 aims at processing the decomposition:

[Algorithm for the CCF of a layered mixed matrix A]

```
Step 1: $M := \emptyset$; $base[i] := 0\ (i \in R_Q)$; $P[i,j] := Q_{ij}\ (i \in R_Q,\ j \in C)$;
        $S :=$ unit matrix of order $m_Q$.

Step 2: $I := \{i \in C \mid i_Q \in \partial M \cap C_Q\}$;
        $J := \{j \in C - I \mid \forall i:\ base[i] = 0 \Rightarrow P[i,j] = 0\}$;
        $S_T^+ := R_T - \partial M$; $S_Q^+ := \{j_Q \in C_Q \mid j \in C - (I \cup J)\}$; $S^+ := S_T^+ \cup S_Q^+$;
        $S^- := C - \partial M$;
        $B^+ := \{(i_Q, j_Q) \mid h \in R_Q,\ j \in J,\ P[h,j] \neq 0,\ i = base[h]\}$.

Step 3: If there does not exists in $G$ a directed path from $S^+$ to $S^-$ (including the
        case where $S^+ = \emptyset$ or $S^- = \emptyset$) then go to Step 4; otherwise execute
        the following:
        { Let $L\ (\subseteq B)$ be (the set of arcs on) a shortest path from $S^+$ to $S^-$
          ("shortest" in the number of arcs);
          $M := (M - L) \cup \{(j,i) \mid (i,j) \in L \cap B_T\} \cup \{(j, j_Q) \mid (j_Q, j) \in L \cap B_C\}$;
          If the initial vertex ($\in S^+$) of the path $L$ belongs to $S_Q^+$, then do
          the following:
          { Let $j_Q\ (\in S_Q^+ \subseteq C_Q)$ be the initial vertex;
            Find $h$ such that $base[h] = 0$ and $P[h,j] \neq 0$;
                                        [$j \in C$ corresponds to $j_Q \in C_Q$]
            $base[h] := j$; $w := 1/P[h,j]$;
            $P[k,l] := P[k,l] - w \times P[k,j] \times P[h,l]\ (h \neq k \in R_Q, l \in C)$;
            $S[k,l] := S[k,l] - w \times P[k,j] \times S[h,l]\ (h \neq k \in R_Q, l \in R_Q)$ };
          For all $(i_Q, j_Q) \in L \cap B^+$ (in the order from $S^+$ to $S^-$ along $L$)
          do the following:
          { Find $h$ such that $i = base[h]$;      [$j \in C$ corresponds to $j_Q \in C_Q$]
            $base[h] := j$; $w := 1/P[h,j]$;
            $P[k,l] := P[k,l] - w \times P[k,j] \times P[h,l]\ (h \neq k \in R_Q,\ l \in C)$;
            $S[k,l] := S[k,l] - w \times P[k,j] \times S[h,l]\ (h \neq k \in R_Q,\ l \in R_Q)$ };
          Go to Step 2 }.

Step 4: Let $V_\infty\ (\subseteq V)$ be the set of vertices reachable from $S^+$ by a directed path in
        $G$;
        Let $V_0\ (\subseteq V)$ be the set of vertices reachable to $S^-$ by a directed path in $G$;
        $C_0 := C \cap V_0$; $C_\infty := C \cap V_\infty$;
        Let $G'$ denote the graph obtained from $G$ by deleting the vertices $V_0 \cup V_\infty$
        (and arcs incident thereto);
        Decompose $G'$ into strongly connected components $\{V_\lambda \mid \lambda \in \Lambda\}$ ($V_\lambda \subseteq V$);
        Let $\{C_k \mid k = 1, \ldots, b\}$ be the subcollection of $\{C \cap V_\lambda \mid \lambda \in \Lambda\}$ consisting
        of all the nonempty sets $C \cap V_\lambda$, where $C_k$'s are indexed in such a way that
```

for $l < k$ there does not exist a direct path in G' from C_k to C_l;
$R_0 := (R_T \cap V_0) \cup \{h \in R_Q \mid base[h] \in C_0\}$;
$R_\infty := (R_T \cap V_\infty) \cup \{h \in R_Q \mid base[h] \in C_\infty \cup \{0\}\}$;
$R_k := (R_T \cap V_k) \cup \{h \in R_Q \mid base[h] \in C_k\}$ $(k = 1, \ldots, b)$;
$\overline{A} := P_r \begin{pmatrix} P \\ T \end{pmatrix} P_c$, where the permutation matrices P_r and P_c are determined so that the rows and the columns of $\overline{A}$ are ordered as $(R_0; R_1, \ldots, R_b; R_\infty)$ and $(C_0; C_1, \ldots, C_b; C_\infty)$, respectively.

The above algorithm can be best understood by means of matroid-theoretic concepts. Let $\mathbf{M}(Q)$ denote the matroid defined on C by Q, namely, $I \subseteq C$ is independent in $\mathbf{M}(Q)$ if $\text{rank}Q[R_Q, I] = |I|$. Similarly, we associate matroids $\mathbf{M}(T)$ and $\mathbf{M}(A)$ with T and A, respectively. Then it is known that $\mathbf{M}(A)$ is the union of $\mathbf{M}(Q)$ and $\mathbf{M}(T)$, i.e., $\mathbf{M}(A) = \mathbf{M}(Q) \vee \mathbf{M}(T)$. Throughout the algorithm, I is an independent set in $\mathbf{M}(Q)$, whereas $J \cup I$ is the closure of I in $\mathbf{M}(Q)$. On the other hand, $(\partial M \cap C) - I$ is an independent set in $\mathbf{M}(T)$. Since $\partial M \cap C$ is independent in $\mathbf{M}(Q) \vee \mathbf{M}(T) = \mathbf{M}(A)$, it holds that $\text{rank}[A, \partial M \cap C] = |M|$. At each execution of Step 3 the size of $|M|$ increases by one, and at the termination of the algorithm, we have the relation: rank $A = |M|$. The updates of P in Step 3 are the usual pivoting operations on P.

For the 4×5 LM–matrix A in Example 1, we label $\text{Col}(A) = C = \{x_1, x_2, x_3, x_4, x_5\}$ and $\text{Row}(T) = R_T = \{f_1, f_2\}$. The copy of C is denoted by $C_Q = \{x_{1Q}, x_{2Q}, x_{3Q}, x_{4Q}, x_{5Q}\}$. Figure 1 (a) shows the corresponding network. In (b), the final network after executing Step 1 to Step 3 is given (bold arcs indicate arcs in M). Since x_4 remains an exit vertex, there is a rank deficiency and the CCF, $\tilde{A}$, has a nonempty horizontal tail.

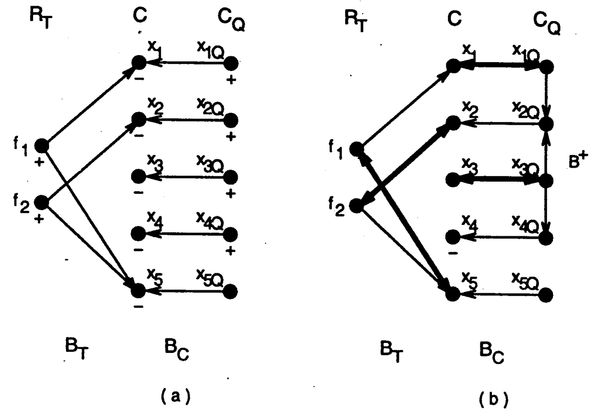

FIG. 1. *Applying the CCF algorithm to the matrix of Example 1*

3. The improved CCF–Algorithm. This section presents the improved algorithm for computing the CCF. The basic algorithm is retained, but a practically faster algorithm is employed in order to improve solving the independent assignment subproblem.

We will present this algorithm in two improving steps. In Section 3.1, a **revised** version of the basic algorithm is introduced which incorporates two precalculation phases called **Step A** and **Step B**, where Step A computes an assignment in the subgraph induced by B_T and where Step B works on B_C. For a second improvement Step B is refined so that it runs with higher efficiency (see Section 3.2).

3.1. A revised CCF–algorithm. Recall that the basic components for computing an independent assignment are path searching and matching update. Due to extensive updating processes the latter tends to be time critical especially, as up until now it has been performed independently of structural properties of the augmenting paths. The clue to a faster algorithm lies in carrying out the update processes more carefully than the original algorithm does. This is reasonable, since there are three types of augmenting paths, namely

(1) those which only contain arcs $e \in B_T$,
(2) those with arcs $e \in B_T \cup B_Q$ (which initiate pivoting),
(3) those with arcs $e \in B_T \cup B_Q \cup B^+$ (which initiate pivoting and exchanges in the base set and in B^+).

The strategy is to separate augmentings for the generic matrix T (paths of the form (1)) from those ones for the elimination (pivoting) operations on Q (paths of the form (2) and (3)) and to apply sophisticated routines in the respective augmenting processes. For the algorithm it is also appropriate to start with a large assignment instead of the empty one. Then certain augmenting steps can be unemployed until the final phase of the algorithm where an optimal assignment has to be reached. We can apply this technique, since the structure of the underlying network makes it easy to compute a large initial assignment.

To be more concrete, the algorithm proposed here consists of the two preprocedures, Step A and Step B, followed by the procedures of the original algorithm. In Step A, a maximal (not necessarily maximum) matching M_{B_T} in the subgraph induced on the vertex set $R_T \cup C$ is computed using a simple greedy strategy. Each arc $e \in M_{B_T}$ gives a shortest path of length one whose augmenting will involve no pivoting. Step A adds the reversal of the arcs $e \in M_{B_T}$ to the assignment M and updates the sets S^+ and S^-. The task of Step B is to compute an initial set I of independent columns whose size is possibly large. For I, it chooses a maximal (not necessarily maximum) set of diagonal arcs $B'_C \subseteq B_C$ such that each arc $(j_Q, j) \in B_C$ connects an entrance with an exit vertex and such that the set of corresponding matrix columns enjoys linear independency. In the revised algorithm, procedure Step B traverses the set B_C of diagonal arcs one by one. Exactly those arcs $(j_Q, j) \in B_C$ with $j_Q \in S^+$ and $j \in S^-$ at the time of the traversal are selected for inclusion in I. Before visiting the next arc, the corresponding pivotings are executed, so that the list of dependent columns as well as the set of entrance vertices can be updated accordingly.

It is important to note that arcs of B^+, expressing exchangeability among columns of I and J, are not needed in Step A and Step B and that their computation is therefore omitted in either procedure. In the following step, however, B^+ will be computed for the first time, since exchanges in the base set may become unavoidable for augmenting the current assignment to optimality. Subsequently, the entire network will be submitted to the standard processes of path searching and matching update (Step 2 and Step 3) of the original algorithm, before the Min–Cut–decomposition into strongly connected components is called.

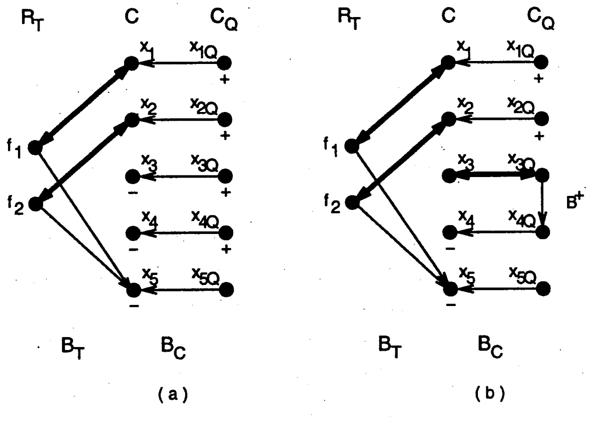

FIG. 2. *Step A and Step B of the revised algorithm*

Example 3. We illustrate the revised algorithm on the matrix given in Example 1. In Figure 2 (a), we see the matching computed by Step A of the algorithm (bold lines). The assignment after executing Step B which covers arcs of B_C is displayed in (b). There, the only arc contained in B^+ is shown as well. The initial assignment computed by Step A and Step B is not optimal yet. The network will be submitted to the original algorithm which augments the assignment along the path $x_{1Q} \to x_1 \to f_1 \to x_5$, which yields the final optimal assignment as depicted in Figure 1 (b).

The revised algorithm for computing the CCF may now be stated as follows:

[Revised algorithm for the CCF of a layered mixed matrix A]

```
Step 1: $M := \emptyset$; $I := \emptyset$; $base[i] := 0$ $(i \in R_Q)$; $P[i,j] := Q_{ij}$ $(i \in R_Q,\ j \in C)$
        $S :=$ unit matrix of order $m_Q$;
        $J := \{j \in C \mid P[i,j] = 0$ for all $i \in R_Q\}$.

Step A: Let $\overline{M}$ be the set of arcs of a maximal matching in the subgraph induced
        by the vertex set $R_T \cup C$;
        $M := \{(j,i) \mid (i,j) \in \overline{M}\}$;
        $S^- := C - \partial M$; $S_T^+ := R_T - \partial M$; $S_Q^+ := \{j_Q \in C_Q \mid j \in C - J\}$; $S^+ := S_T^+ \cup S_Q^+$.

Step B: For all $(j_Q, j) \in B_C$ do the following:
        {If $j_Q \in S_Q^+$ and $j \in S^-$ do the following:
          { $M := M \cup (j, j_Q)$; $S_Q^+ := S_Q^+ - \{j_Q\}$; $S^- := S^- - \{j\}$; $I := I \cup \{j\}$;
            Find $h$ such that $base[h] = 0$ and $P[h,j] \neq 0$;
            $base[h] := j$; $w := 1/P[h,j]$;
            $P[k,l] := P[k,l] - w \times P[k,j] \times P[h,l]$ $(h \neq k \in R_Q, l \in C)$;
            $S[k,l] := S[k,l] - w \times P[k,j] \times S[h,l]$ $(h \neq k \in R_Q, l \in R_Q)$
            $J := \{j \in C - I \mid \forall i:\ base[i] = 0 \Rightarrow P[i,j] = 0\}$
            $S_Q^+ := \{j_Q \in C_Q \mid j \in C - (I \cup J)\}$ }};
        $S^+ := S_T^+ \cup S_Q^+$;
        $B^+ := \{(i_Q, j_Q) \mid h \in R_Q,\ j \in J,\ P[h,j] \neq 0,\ i = base[h]\}$.

Step 2': Go to Step 3 of the original algorithm.
```

Improvements through the revised algorithm are based on the following facts: Firstly, all shortest paths which have a length of 1 are detected in a single run. As for the original algorithm, these paths are detected in almost the same way, but Step A and Step B are more straightforward, since they visit all arcs of B_T (resp. B_C) only once. Secondly, as already mentioned, a considerable amount of computation time is saved by postponing the computation and update of B^+ to the final augmenting phases, rather than repeatedly computing B^+ already at the beginning. Thirdly, the separation of augmentings for B_T from those for B_C makes it easier for each case, to decide which update steps become necessary during the respective augmentings.

3.2. Improving Step B. While it is easy to compute an initial assignment in B_T in Step A, Step B is a rather slow procedure. For each augmenting in B_C, the corresponding row elimination operations have to be carried out and accordingly the new dependence/independence structure has to be identified, i.e., J has to be computed. As a consequence the amount of computation as well as the computation time for J increases drastically with the matrix size.

One possible strategy for speeding up Step B is in a combinatorial relaxation for computing an initial set I. That approach ignores the concrete values of the matrix entries in favour of the combinatorial structure when it chooses an initial set of (yet not matched) diagonal arcs for the set I. This procedure can work without J.

A starting set I is computed by a bipartite matching algorithm as follows. Let M be the initial matching computed by Step A on the network G of the independent assignment subproblem. We define $R_Q = \text{Row}(Q)$ and, as usual, $C = \text{Col}(A)$, and consider a directed graph $G_Q = (V_Q, B_Q)$ with the vertex set $V_Q = (C - \partial M) \cup R_Q$ and the arc set $B_Q = \{(i,j) \mid i \in R_Q, j \in C - \partial M, Q_{ij} \neq 0\}$. We then compute a maximum matching $\overline{M}$ in G_Q. The set of columns of Q (vertices in $C - \partial M$) covered by $\overline{M}$ is a good candidate for the base set I, though there remains the possibility of accidental numerical cancellation that causes linear dependency in I. In the latter case, the dependent columns will be excluded from I.

To be concrete, each arc (i,j) included in $\overline{M}$, where $i \in R_Q$ and $j \in C - \partial M$, is used as a pivoting position, if the value of the entry is distinct from zero at the time of pivoting (if the entry equals zero, the column is not independent). Hence the base set I is composed of all the columns $j \in C$ which are covered by the matching and whose corresponding pivoting element does not vanish during previous row eliminations. We will refer to this algorithm, which uses a combinatorial relaxation technique, as the **relaxation** algorithm. For notational convenience, we do not distinguish vertices in $V_Q - R_Q$ and columns of Q.

[Step B for the relaxation algorithm]

```
Step B: Let $\overline{M}$ be a maximum bipartite matching on $G_Q$;
        For all $j \in V_Q - R_Q$ do the following:
        { If $j \in \partial\overline{M}$, find $i$ such that $(i,j) \in \overline{M}$;
          If $P[i,j] \neq 0$, then do the following:
          {$base[i] := j$; $I := I \cup \{i\}$; $w := 1/P[i,j]$;
            $P[k,l] := P[k,l] - w \times P[k,j] \times P[i,l]$ $(i \neq k \in R_Q, l \in C)$;
            $S[k,l] := S[k,l] - w \times P[k,j] \times S[i,l]$ $(i \neq k \in R_Q, l \in R_Q)$ }};
        $J := \{j \in C - I \mid \forall i:\ base[i] = 0 \Rightarrow P[i,j] = 0\}$;
        $S_Q^+ := \{j_Q \in C_Q \mid j \in C - (I \cup J)\}$;
        $S^+ := S_T^+ \cup S_Q^+$;
        $B^+ := \{(i_Q, j_Q) \mid h \in R_Q,\ j \in J,\ P[h,j] \neq 0,\ i = base[h]\}$.
```

We will finally introduce a third version of Step B, which is simple and fast, and installed for our new algorithm. It gains its good performance by the fact that we can dispense with the graph G_Q as well as the bipartite matching algorithm. This algorithm works as follows, with row eliminations on the matrix Q. For each row i of Q (after elimination), the first entry Q_{ij} with $Q_{ij} \neq 0$ is chosen for pivoting, where j is not matched already. The base is then enlarged by column j and the row eliminations are carried out, using Q_{ij} as the pivoting element. This quickly gives an initial assignment in B_C, which is, as the computational experiments given in the next section will show, already close to an optimal one.

[Step B for the new algorithm]

```
Step B: For all $i \in R_Q$ do the following:
        { Find $j \in C$ such $j \notin \partial M$ and $P[i,j] \neq 0$;
          If such $j$ exists, do the following:
          {$base[i] := j$; $I := I \cup \{i\}$; $w := 1/P[i,j]$;
            $P[k,l] := P[k,l] - w \times P[k,j] \times P[i,l]$ $(i \neq k \in R_Q, l \in C)$;
            $S[k,l] := S[k,l] - w \times P[k,j] \times S[i,l]$ $(i \neq k \in R_Q, l \in R_Q)$ }};
        $J := \{j \in C - I \mid \forall i:\ base[i] = 0 \Rightarrow P[i,j] = 0\}$;
        $S_Q^+ := \{j_Q \in C_Q \mid j \in C - (I \cup J)\}$;
        $S^+ := S_T^+ \cup S_Q^+$;
        $B^+ := \{(i_Q, j_Q) \mid h \in R_Q,\ j \in J,\ P[h,j] \neq 0,\ i = base[h]\}$.
```

This version has also the advantage that the pivoting elements can be determined quickly, since for each row the first suitable entry is chosen for pivoting.

4. Computational Experiments. In the literature we could find only a few examples where results on the combinatorial canonical form have been reported. Murota [Mu87] considered matrices coming from chemical models and small matrices for the analysis of electronic networks. The latter includes a problem (Problem na18) with 6 resistors and 3 voltage controlled sources, where the coefficient matrix A of the system of equations to be solved was a layered mixed matrix of order 18. Emms [E94] also presented some results for the chemical model (reactor separator model EV–6). The system of linear/non–linear equations to be solved involves 120 equations in 120 unknowns and also a singular matrix (Problem p119) was formed out of EV–6 by deleting row 107 and column 109 from the corresponding mixed matrix. For our test series we additionally ran the algorithms on a collection of matrices taken from the Harwell-Boeing databases (Problems IMPCOL and WEST)[DGL89, DGL92]. In these examples, we regarded all integer coefficients whose modulus is less than or equal to 10 as constant numbers and the others as indeterminates.

Our computational experiments were carried out on a SUN SPARCstation 10 (125MHz) using *Mathematica*, Version 2.2 for SPARC. We used *Mathematica* for its ability in symbolic computation and in order to provide an elegant code which is easy to follow for interested readers. Note, however, that the overhead in computation time is quite enormous. Moreover, computation time is slightly influenced by *Mathematica*'s internal data handling, where for instance the time needed for operating on the network (such as traversing) is relatively high compared to the time needed for pivoting.

TABLE 1
Structure of the problem instances

Problem	#Cols	#Rows in Q	#Rows in T	#Entries in Q	#Entries in T
na18	18	9	9	29	18
EV–6	205	119	86	257	264
p119	202	117	85	253	255
IMPCOL A	228	171	57	338	276
IMPCOL B	89	58	31	137	194
IMPCOL C	154	136	18	399	35
IMPCOL D	483	425	58	1255	116
IMPCOL E	364	223	141	566	1015
WEST0067	86	31	55	94	238
WEST0132	211	93	118	203	368
WEST0156	229	135	94	264	244

Table 1 summarizes properties of the input matrices. The size of the matrices ranges from 18 to 483 and the number of nonzero coefficients from 47 to 1581. Table 2 describes the CCF for those matrices. From left to right it displays the rank of each matrix instance, the size of the horizontal and vertical tail, the total number of nonsingular square blocks, the number of those blocks of size one and finally the size of the largest square block.

In our tests, we ran the original, the revised, the relaxation and the new algorithm on the above test matrices. In order to evaluate the behavior of the algorithms, we investigated three criteria: the number of **pivoting** operations, the number of **base exchanges** (both Table 3) and the **computation time** (Table 4).

Since the original and the revised algorithm augment the assignment along the same paths, the resulting numbers of pivotings and base exchanges are identical. On the other hand, one can observe that the improved versions of Step B need less pivotings as they make use of the combinatorial structure of Q.

Table 4 displays the computation time consumed by the algorithms. While already the revised algorithm constantly outperforms the original one, a significant speed up is obtained under the faster procedure for Step B, both in the relaxed and the new algorithm. Especially for large instances, one can observe a significant gain. We can also see that usually the new algorithm processes faster than the relaxation algorithm, except for IMPCOL D and IMPCOL E, where the number of rows in T is very small compared to the number of rows

TABLE 2
CCF for test matrices

Problem	Rank	htail?	vtail?	Nonsingular Blocks		
				#total	#size 1	size of largest
na18	18	no	no	9	7	8
EV–6	205	no	no	151	146	17
p119	201	27 × 26	14 × 15	118	115	17
IMPCOL A	228	no	no	184	173	27
IMPCOL B	89	no	no	45	44	45
IMPCOL C	154	no	no	151	150	4
IMPCOL D	483	no	no	475	472	5
IMPCOL E	364	no	no	259	255	70
WEST0067	86	no	no	2	1	85
WEST0132	211	no	no	97	96	115
WEST0156	229	no	no	198	197	32

TABLE 3
Number of Pivots and Base Exchanges

Problem	# Pivots				# Base Exchanges			
	Original	Revised	Relax.	New	Original	Revised	Relax.	New
na18	44	44	44	44	3	3	3	3
EV–6	326	326	274	295	18	18	15	14
p199	307	307	267	295	14	14	13	14
IMPCOL A	799	799	669	787	6	6	6	6
IMPCOL B	150	150	90	146	0	0	0	0
IMPCOL C	8162	8162	1290	6426	1	1	0	1
IMPCOL D	84841	84841	9349	76096	3	3	3	3
IMPCOL E	2070	2070	1846	2015	16	16	16	16
WEST0067	21	21	21	21	0	0	0	0
WEST0132	334	334	262	322	4	4	4	4
WEST0156	233	233	227	227	6	6	6	6

in Q. The results are confirmed in Table 5 which shows the effect for the variants of Step B

TABLE 4
Computation Time

Problem	Original	Revised	Relax.	New
	s	s	s	s
na18	3.6	3.1	3.5	3.07
EV–6	133.72	90.78	89.53	84.25
p199	156.17	117.37	116.15	109.62
IMPCOL A	177.7	139.58	135.13	118.33
IMPCOL B	30.45	21.05	20.53	18.27
IMPCOL C	119.42	92.18	75.15	76.72
IMPCOL D	1488.87	1060.83	829.82	878.02
IMPCOL E	542.57	400.02	368.1	357.33
WEST0067	22.6	19.02	19.73	18.23
WEST0132	114.93	93.12	91.67	86.02
WEST0156	144.27	107.28	99.35	91.37

in terms of computation time.

From our experiments, we can conclude that there are two winners, namely the relaxed algorithm and the new algorithm, where in general, the new algorithm seems to behave better. Both versions provide a powerful algorithm for computing the CCF of a layered mixed matrix and improve the original version significantly. We further expect the running time to decrease considerably when implementing our code in a programming language like C and using sophisticated pointer structures rather than exclusively working on data structures based on *Mathematica* lists.

In order to further exploit what really happens when executing the algorithm on the above problems, we compiled some additional computational details. Table 6 describes,

TABLE 5
Computation Time for Step B

Problem	Revised	Relax.	New
	s	s	s
na18	0.35	0.3	0.2
EV–6	32.57	24.76	22.9
p199	30.68	24.25	22.4
IMPCOL A	58.22	49.6	39.28
IMPCOL B	9.23	7.36	6.82
IMPCOL C	65.1	47.6	52.4
IMPCOL D	790.38	542.95	632.67
IMPCOL E	120.57	84.98	81.63
WEST0067	2.31	1.6	1.27
WEST0132	16.73	11.9	10.43
WEST0156	41.02	28.07	26.2

how Step A and Step B (combined with the new algorithm) behave on the given problems. The table lists the number of arcs of the assignment initially computed on columns of Q (**$\#B_C$–Assignment**) and rows of T (**$\#B_T$–Assignment**) and the final distribution of the assignment on Q and T. Column **Augm. calls** gives the number of augmentings along shortest paths which are needed in order to yield the final independent assignment. We also calculated the change in the number of nonvanishing coefficients during the matrix transformation (**Entries**). The last three columns list the computational time consumed by Step A and Step B and the time needed for the augmenting phase and the decomposition.

TABLE 6
Applying the new algorithm

Problem	$\#B_C$ Ass.		$\#B_T$–Ass.		Augm.	En-	Time			
	init	final	init	final	calls	tries	Step A	Step B	Augm.	Dcomp.
							s	s	s	s
na18	4	9	9	9	6	−2	0.07	0.2	1.33	0.96
EV–6	103	119	86	86	17	+ 0	1.77	22.9	24.6	25.03
p199	102	116	85	85	15	+1	1.73	22.4	22.63	54.72
IMPCOL A	137	171	57	57	35	−60	1.23	39.28	39.23	29.45
IMPCOL B	57	58	31	31	2	−8	0.31	6.82	1.23	6.62
IMPCOL C	135	136	18	18	2	−144	0.2	52.4	4.7	16.22
IMPCOL D	424	425	58	58	2	+24	1.78	632.67	104.25	134.38
IMPCOL E	178	223	141	141	46	+157	6.91	81.63	161.7	74.03
WEST0067	15	31	55	55	17	+9	0.65	1.27	7.33	5.57
WEST0132	55	93	118	118	39	−14	2.8	10.43	37.52	26.98
WEST0156	112	135	92	94	26	−21	1.98	26.2	24.97	30.7

From our computational experiments we can draw the conclusion that the procedures Step A and Step B produce good starting assignments such that the number of calls of Step 2 and Step 3 (which are rather time consuming procedures) could remain small. The findings of the test further indicate that the computation of an initial assignment is dominated by the elimination operations, so that the algorithm still spends much of the computation time for Step B. On the other hand, a comparable amount of computation time is consumed in the few augmenting steps from the initial assignment to the optimal one. Also, the decomposition phase is still relatively slow, since we did not put much effort in its implementation and the data structures.

A. Appendix: Implementation Manual. The program code of our implementation for computing the CCF is available via anonymous ftp from `www.kurims.kyoto-u.ac.jp`. Please change to the directory `pub/paper/member/murota` and read the `README` file. Alternatively you can check the homepage under `http://www.kurims.kyoto-u.ac.jp/~murota`.

The given directories include the new algorithm which — as a mathematica package — is named `ccf.m`. Also, we provide a file `ccfstat.m` which creates, in addition to computing the

CCF, a file `stat.log` of computation statistics. For demonstrating purposes, all matrices used in the computational experiments are given in a subdirectory `data`.

When running a mathematica session, include the package `ccf.m` (or `ccfstat.m`) by simply typing "<< ccf.m" in the Mathematica prompt. The variables used in the code are protected such that we do not worry about conflicting variable setting caused by previous computations. Typing "CCF[*filename*]" will execute the algorithm on the input matrix given in *filename*. The resulting decomposition will be displayed on the screen and in a file `ccf.out` in a sparse matrix format.

The sparse matrix format used for the input matrices as well as for the file `ccf.out` is as follows: The first line gives the number of columns. The second and the third line contain the number of rows in Q and in T, respectively. Separated by a line with a zero, the matrix coefficients are described. Each line contains, from left to the right, the number of the row, the number of nonvanishing entries in that row and a pair (j, m_j) consisting of the column number for each such entry as well as the value of that coefficient. For the submatrix T for simplicity, the symbol entries all get the value zero.

REFERENCES

[BR91] R. A. Brualdi and H. J. Ryser, *Combinatorial Matrix Theory*, Cambridge University Press, London, 1991.

[DER86] I. S. Duff, A. M. Erisman and J. K. Reid, *Discrete Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[DGL89] I. S. Duff, R. G. Grimes and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[DGL92] I. S. Duff, R. G. Grimes and J. G. Lewis, *Users' Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)*, TR/PA/92/86, CERFACS, Toulouse Cedex, France, 1992.

[DR78] I. S. Duff and J. K. Reid, *An implementation of Tarjan's algorithm for the block triangularization of a matrix*, ACM Trans. Math. Software, 4 (1978), pp. 137–147.

[DM59] A. L. Dulmage and N. S. Mendelsohn, *A structure theory of bipartite graphs of finite exterior dimension*, Trans. Roy. Soc. Canada, Section III, 53 (1959), pp. 1–13.

[E94] N. R. E. Emms, *An Implementation of the Combinatorial Canonical Form Decomposition Algorithm for Layered Mixed Matrices*, Dissertation for Master Thesis, Kyoto Univ., 1994.

[EGLPS87] A. M. Erisman, R. G. Grimes, J. G. Lewis, W. G. Poole, Jr. and H. D. Simon, *Evaluation of orderings for unsymmetric sparse matrices*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 600–624.

[Gu76] F. Gustavson, *Finding the Block Lower Triangular Form of a Sparse Matrix*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, 1976, pp. 275–289.

[Ho76] T. D. Howell, *Partitioning using PAQ*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, 1976, pp. 23–37.

[Mu87] K. Murota, *Systems Analysis by Graphs and Matroids – Structural Solvability and Controllability*, Springer-Verlag, 1987.

[Mu93] K. Murota, *Mixed Matrices – Irreducibility and Decomposition*, in Combinatorial and Graph Theoretic Problems in Linear Algebra, R. A. Brualdi, S. Friedland and V. Klee, eds., The IMA Volumes in Mathematics and Its Applications, Vol. 50, Springer-Verlag 1993, pp. 39–71.

[Mu96] K. Murota, *Structural approach in systems analysis by mixed matrices – An exposition for index of DAE*, in ICIAM 95 (Proc. Third Intern. Congr. Indust. Appl. Math., Hamburg, Germany, July 3-7, 1995), K. Kirchgässner, O. Mahrenholtz and R. Mennicken, eds., Mathematical Research, Vol. 87, Akademie Verlag, 1996, pp. 257-279.

[MI85] K. Murota and M. Iri, *Structural solvability of systems of equations — A mathematical formulation for distinguishing accurate and inaccurate numbers in structural analysis of systems*, Japan J. Appl. Math., 2 (1985), pp. 247–271.

[MIN87] K. Murota, M. Iri and M. Nakamura, *Combinatorial canonical form of layered mixed matrices and its application to block-triangularization of systems of equations*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 123–149.

[PF90] A. Pothen and C. J. Fan, *Computing the block triangular form of a sparse matrix*, ACM Trans. Math. Software, 16 (1990), pp. 303–324.

[YTK81] K. Yajima, J. Tsunekawa and S. Kobayashi, *On equation-based dynamic simulation*, Proc. World Congr. Chem. Eng., Montreal, V (1981).