FINDING THE MAXIMAL INCIDENCE MATRIX OF A LARGE GRAPH

by

M. Overton
A. Proskurowski

STAN-CS-75-509
SEPTEMBER 1975

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

Finding the Maximal Incidence Matrix of a Large Graph

Michael Overton and Andrzej Proskurowski
Computer Science Department
Stanford University

Abstract

The paper deals with the computation of two canonical representations of a graph. A computer program is presented which searches for "the maximal incidence matrix" of a large connected graph without multiple edges or self-loops. The use of appropriate algorithms and data structures is discussed.

This research was supported in part by National Science Foundation grant DCR72-03752 A02 and by the Office of Naval Research contract NR 044-402.

1. Introduction.

The notion of the maximal incidence matrix as a canonical representation of a graph was introduced in [1]. An algorithm to search for this matrix (a graph being given by--any of its incidence matrices) was presented there together with a computer program which performed the search.

In this paper we briefly review basic ideas of [1] and discuss another "maximal incidence matrix" of a graph. Our main concern is the application of the search algorithm to large graphs and an efficient use of computer memory when representing graphs and carrying on the search. A variety of arrays and linked lists will be employed in order to limit the amount of parameters passed along with the recursive subroutine calls. We have developed a computer program written in ALGOL W that maintains the data structures and performs the search. The program is presented and its functions are discussed.

2. Basic Notions.

In order to use concrete phrases when discussing the problem and the proposed solution, let us define our basic vocabulary.

A graph will mean two sets N (of nodes) and E (of edges), together with a function F (the incidence function) which ascribes an edge $a \in E$ to some unordered pair of nodes n_1 and n_2,

$$F(n_1,n_2) = F(n_2,n_1) = a .$$

We constrain the function F to be partially defined (in particular, not defined for $n_1 = n_2$ thus excluding graphs with self-loops) and require that F is single-valued, i.e., graphs do not have multiple edges. Nodes n_1 and n_2 are said to be adjacent and the edge a is said to be incident to nodes n_1 and n_2 . The valence of a node n_1 is the number of edges incident to it, and will be denoted $\text{valence}(n_1)$. A graph is connected if for every pair of nodes $u,v \in N$ there exists a sequence of adjacent nodes n_i $(i = 0,\ldots,k)$ such that $n_0 = u$,

$n_k = v$ and $F(n_{i-1}, n_i)$ is defined for all $i = 1, \ldots, k$. In the following we shall consider only connected graphs, for simplicity.

We shall label elements of the sets of nodes N and edges E by consecutive integers beginning with 1 . We shall represent a graph by listing entries of its incidence function which is a shorthand for its incidence matrix: a sparse binary matrix of $n = |N|$ columns corresponding to the nodes and $e = |E|$ rows, each corresponding to an edge. The element $M(p,i)$ of the incidence matrix M equals 1 if the edge label-led p and node labelled i are incident, and 0 otherwise. We will denote edge labels p, q, r and node labels i, j , k . The p-th row of the matrix, corresponding to the edge labelled p , will be referred to as $M(p,*)$ and the i-th column, corresponding to the node labelled i , will be referred to as $M(*,i)$.

An important notion for our discussion is that of isomorphic graphs. Two graphs, $G_1 = (N_1,E_1,F_1)$ and $G_2 = (N_2,E_2,F_2)$, are said to be isomorphic if they may be represented by identical sets $N_1 = N_2$ and $E_1 = E_2$, and identical function $F_1 = F_2$. With our assumption about labelling sets N and E , this means that the labels in one of the graphs may be permuted in a way transforming the incidence function into a form identical with the other. In terms of the incidence matrices this means exchanging columns and rows of one matrix so as to get a matrix identical with the other one.

Let us consider incidence matrices of a graph which have rows arranged lexicographically in descending order. Then, for a given graph, we can define an ordering relation on the class of row-ordered incidence matrices. For two unequal matrices M_1 and M_2 we say that M_1 is row-greater than M_2 if the first row of M_1 that differs from the corresponding row of M_2 is lexicographically greater. A matrix not less than any other matrix in this class will be called the row-maximal incidence matrix of the graph, or the "romim" for short.

The notion of romim was introduced in [1] under the name of "maximal incidence matrix" and its existence proved.

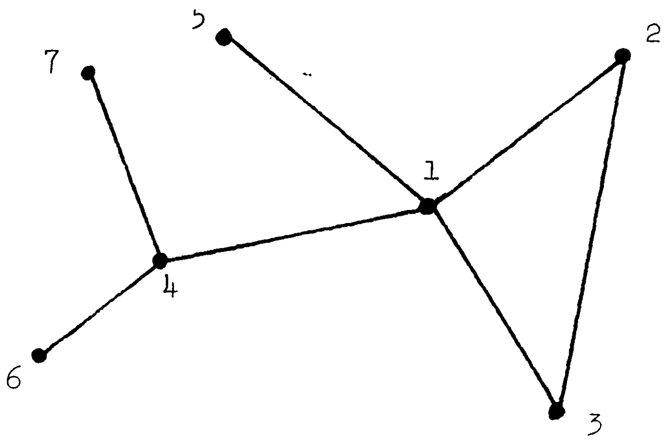
Considering columns of an incdence matrix as bit strings read top-to-bottom we may order them in descending lexicographic order. For a given graph let us define a relation column-greater than on the class of column ordered incidence matrices. A matrix not less (in the sense of column-ordering) than any other matrix in the class will be called the column-maximal incidence matrix of the graph, or simply the "comim".

Fact 2.1. For a given graph there always exists a column-maximal incidence matrix defined as above.

Proof 2.1. Given a graph we can always fix the labelling of the edges and then order the columns of the incidence matrix lexicographically. Thus, for all possible labellings (permutations) of edges we obtain a set of corresponding column-ordered incidence matrices. Since the set is finite, we have an element that is not less than any other element of the set. This is the comim. ❑

It must be pointed out that the two definitions describe two different quantities. We give an example of a graph and its romim and comim (Figure 2.1). By inspection, the matrices are not equal.

(a)



(b) Nodes:

1	2	3	4	5	6	7
1	1					
1		1				
1			1			
1				1		
	1	1				
			1		1	
			1			1

(c)

1	4	2	3	5	6	7
1	1					
1		1				
1			1			
1				1		
	1				1	
	1					1
		1	1			

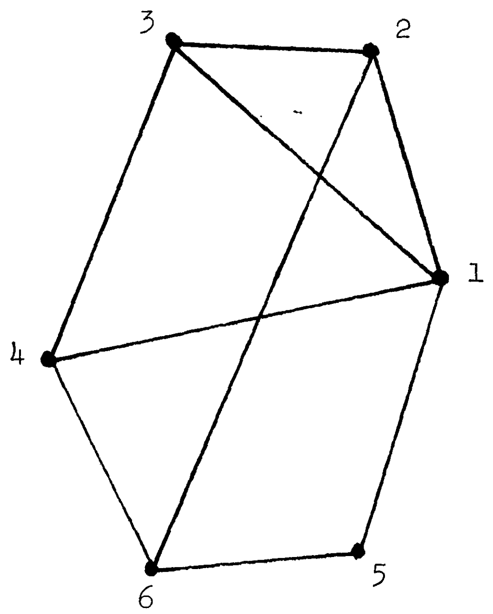
Figure 2.1. Example of a graph (a) with unequal romim (b) and comim (c).

3. The Search.

It is easy to describe a brute force method to find the maximal incidence matrix. By listing all possible labellings of nodes of a graph, lexicographically ordering the rows of the corresponding incidence matrices, and saving the "maximal matrix so far", the romim is obtained. Similarly, by listing all possible labellings of the edges and ordering the columns of the incidence matrices the comim is obtained. However, there often exist clear indications of which permutations should be considered as leading to the proper labelling. A depth-first search procedure to find the (row-) maximal incidence matrix was proposed in [2]. It labels nodes of the given graph and selects the best choices to be labelled tentatively leaving the other possibilities still to be examined. The search may be represented by a search tree where nodes of the tree correspond to the labels to be assigned. When the search arrives at a leaf of the tree (i.e., when all nodes of the graph are labelled), the incidence matrix "maximal so far" is compared with the result of the tentative labelling and -- if it is inferior -- replaced by the newly found one.

The main role in the process of labelling nodes of a graph is played by the priority vector. It is a one dimensional array which for every unlabelled node gives an indication of its suitability to be labelled next. This indication is calculated from the incidence matrix based upon how a node is connected with the labelled nodes. To formalize this we introduce a notion of the priority vector for assignment of the label m. The element $PRIVEC_m(i)$, where $2 \le m \le i \le n$, is a bit string which at every position $1 \le j < m$ has 1 if the node i is adjacent to node j and 0 otherwise. Figure 3.1 gives an example of a graph (a) and the priority vectors (b) for consecutive instances of labelling the nodes.

(a)



(b)

	$PRIVEC_m$(1)	(2)	(3)	(4)	(5)	(6)
m = 2		1	1	1	1	0
3			11	10	10	01
4				101	100	010
5					1000	0101
6						01011

Figure 3.1. A graph and the priority vector corresponding to the labelling (1,2,3,4,5,6) .

Let us define a labelling of the nodes of a graph to be _privet-proper_ if any incidence matrix of the graph with nodes arranged by this labelling has nonincreasing priority vectors, i.e., for every i, j and m such that $2 \le m \le i \le j \le n$ we have $\mathrm{PRIVEC}_m(i) \ge \mathrm{PRIVEC}_m(j)$.

The importance of privet-proper labellings of nodes is stressed by Theorem 3.1 (stated and proved for the _romim_ in [1]).

Theorem 3.1. For a given graph the labelling of the nodes that results in the maximal incidence matrix (_romim_ or _comim_) is privet-proper. However a matrix with a privet-proper node labelling is not necessarily a maximal matrix.

It is worth noting that the property of the priority vector stated in Theorem 3.1 holds true for both _romim_ and _comim_. Let us state two lemmas that will simplify proof of the theorem. Lemma 3.2 expresses an intuitively obvious fact that we want "as many ones as possible" in the lefthand upper corner of the incidence matrix.

Lemma 3.2. For a given incidence matrix and a given column i define S_i to be the set of all rows with their first 1 in column i . Then, for the maximal incidence matrix, _romim_ or _comim_, any row between the first row in S_i and the last row in S_i is also in S_i . We call the set S_i simply a block i of rows in the maximal incidence matrix (note that block i may be empty).

Proof 3.2. Assume the contrary: that for a maximal matrix M_1 there exist a column i and rows p, q, r with $p < r < q$, such that the first 1's of rows p and q are in column i and the first 1 of row r is in column $k \ne i$.

(i) Suppose M_1 is the _romim_. If $k < i$ then a matrix with rows p and r swapped is row-greater than M_1 , and if $k > i$ then a matrix with rows q and r swapped is row-greater than M_1 , so M_1 is not the _romim_.

(ii) Suppose M_1 is the comim. If $k < i$ then swapping rows p and ,
(relabelling corresponding edges) and column ordering the matrix results in a matrix column greater than M_1 . Similarly if $k > i$ then swapping rows q and r and column ordering leads to the contradiction. Cl

Actually it is obvious that this block structure of the incidence matrix holds for every row-ordered incidence matrix (see Figure 3.2).

```
______________
1 1
1   1                       block 1
1     1
1       1
~ m m - - - -
  1     1                   block 2
  1       1
- - - - - a -
    l l                     block 3
- - - - a - -                            block 4 is empty
        1   1               block 5
_ _ _ _ - - -
          1 1               block 6
```

Figure 3.2. A row-ordered incidence matrix of a graph displays the block structure.

The second lemma states the conservative property of the priority vector with respect to the assigned label.

<u>Lemma 3.3</u>. For a given incidence matrix and two nodes i and j $(i < j)$ we have, for all $2 \leq \ell \leq m \leq i$,

$$\mathrm{PRIVEC}_{\ell}(i) > \mathrm{PRIVEC}_{\ell}(j) \Rightarrow \mathrm{PRIVEC}_m(i) > \mathrm{PRIVEC}_m(j)$$

The proof is trivial and is left as an exercise for the reader. □

Now we can prove Theorem 3.1 for both romim and comim.

Proof 3.1. Assume the contrary: the given maximal incidence matrix M_1 does not have a privet-proper node-labelling. Thus there exist $m \leq i < j$ such that $\mathrm{PRIVEC}_m(i) < \mathrm{PRIVEC}_m(j)$. According to Lemma 3.3 this implies

$$\mathrm{PRIVEC}_i(i) < \mathrm{PRIVEC}_i(j)$$

which means that there is a position $k < i$ such that the k-th bit in $\mathrm{PRIVEC}_i(i)$ equals 0 and the k-th bit in $\mathrm{PRIVEC}_i(j)$ equals 1 , with the first k-l bits in the same in both $\mathrm{PRIVEC}_i(i)$ and $\mathrm{PRIVEC}_i(j)$. Thus in block k of M_1 (Lemma 3.2) all rows have 0's in column i and there is a row p in the block with a 1 in column j . We will now prove that M_1 may be rearranged in different ways leading to matrices M_2 and M_3 , each greater than M_1 , in the sense of row- and column-ordering, respectively. This will contradict our assumption that M_1 is a maximal incidence matrix.

(i) Suppose M_1 is the romim. Then swapping columns i and j (relabelling corresponding nodes), and ordering rows within blocks 1,...,k-l we obtain a matrix with the blocks 1,...,k-l identical with those of M_1 . In the block k , however, row p is greater than it was before, and no other row in this block has been changed. Thus, ordering block k we get a matrix M_2 that is row-greater than M_1 .

(ii) Suppose M_1 is the comim. Consider blocks 1,...,k-l ; because of the definition of k there cannot be a row with a 1 in column i without another row in the same block with a 1 in column j , and vice versa. In each block if there is a row p with a 1 in column i and a row q with a 1 in column j , such that p < q , then interchange rows p and q (relabel the corresponding edges). There must be at least one such block or else the columns would not be in order. Then the new column j is greater than column i of M_1 , the new column i is less than column i of M_1 and all other columns are unchanged. Thus, ordering the columns lexicographically, we obtain a matrix M_3 greater than M_1 . This completes the proof. Cl

We can now recall from [2] how the algorithm for finding the romim works.

At any stage m , the priority vector gives the indications for the assignment of label m . These indications may appear in two forms;

(1) There is exactly one node pretending to the label m since it uniquely has the highest value of the corresponding element of the priority vector;

(2) There are several nodes for which the corresponding elements of the priority vector have the highest value. These nodes are called equal pretenders.

The situation of (1)is clear and implies assigning label m to the pretender, thus increasing the number of labelled nodes. Calculating the priority vector for the rest of the unlabelled nodes again and again gives the situation (1) or (2) and eventually results in the incidence matrix, maximal for the original labelling 1,2,...,m-1 .

In the situation (2) there are more pretenders that have to be tried as node m . Successively one by one all of the equal pretenders are assigned the label m and, after proceeding as in situation (1), a matrix maximal for every labelling is calculated. The greatest of these matrices is stored as the incidence matrix maximal for labelling 1,2,...,m-1 . The maximal matrix of the graph is identical with the solution of the problem of finding for the matrix maximal for m = 1 (no nodes labelled).

The algorithm is based on two recursive procedures, CHOOSE and PRETEND. Procedure CHOOSE computes the priority vector and makes the right choice for the next label if there is only one pretender; if there are several it calls PRETEND. Procedure PRETEND mades various tentative choices for the next label, calling CHOOSE for each. The process is initiated by examining the valences of the nodes and calling CHOOSE with each node of highest valence as the initial choice. It is clear that for both romim and comim the node labelled first must be a node of highest valence.

We must correct here the algorithm of [2] which applies a valence check in situation (2) to narrow down the number of pretenders. In the example of the graph in Figure 2.1 this would result in M_2 rather than M_1, in spite of the fact that M_1 is row-greater than M_2. Our present algorithm omits this check.

However, the valence check employed--in the algorithm is useful for determining the comim, making the search for the comim more efficient than the search for the romim. This is elaborated in the next section.

4. Pruning the Search Tree for the Comim.

It is attractive to search for the comim rather than the romim because of the following theorem:

Theorem 4.1. Let M_1 be the comim for some graph with nodes numbered $1,\ldots,n$. Then for all $i < j$:

$$\mathrm{PRIVEC}_i(i) = \mathrm{PRIVEC}_i(j) \Rightarrow \mathrm{valence}(i) \geq \mathrm{valence}(j) .$$

Thus if on the i-th decision level two nodes are equal pretenders but have different valences, the node with the higher valence should be chosen.

Proof. Assume the contrary, that is, there exist $i < j$ such that $\mathrm{PRIVEC}_i(i) = \mathrm{PRIVEC}_i(j)$ and $\mathrm{valence}(i) < \mathrm{valence}(j)$. Consider blocks $1,\ldots,i-1$ of M_1 (cf. Lemma 3.2); because the priority vectors are equal there cannot be a row with a 1 in column i without a row in the same block with a 1 in column j, and vice versa. Relabel the edges in the following way. Interchange the pairs of rows, in the blocks $1,\ldots,i-1$, which have 1's in columns i and j, and also move the remaining rows with a 1 in column j up following block $i-1$. The new column j is greater than the column i of M_1, because $\mathrm{valence}(j) > \mathrm{valence}(i)$. Columns $1,\ldots,i-1$ remain unchanged, so after ordering the columns we obtain M_2 column-greater than M_1, which is a contradiction. ❑

Theorem 3.1 showed that the same search tree leading to privet-proper labellings of nodes can be used for both the romim and the comim. Theorem 4.1 shows that the comim search tree can be significantly pruned by considering the valences when encountering equal pretenders.

When arriving at a leaf of the search tree we have a privet-proper node labelling and have built up an incidence matrix of the graph with this node label-line;. It remains to label the edges. In the case of the romim search it is clear that ordering the rows of the matrix results in the row-maximal incidence matrix for this node labelling. It turns out that for the comim search as well, ordering the rows of the matrix results in the column-maximal matrix for the labellings. This result is stated in Theorem 4.2.

Theorem 4.2. Let a graph with a privet-proper labelling of nodes be given by an incidence matrix. Then ordering the rows of the matrix results in the column-maximal matrix for the labelling.

To prove this theorem, consider the row-ordered matrix. Lemma 4.3 shows that such a matrix has a column block structure analogous to the row block structure described in Section 3. Furthermore, Lemma 4.4 shows that such a matrix is column-ordered. The final step will be to prove that no other permutation of the rows gives a matrix which is column-greater than the row-ordered matrix.

Lemma 4.3. A row-ordered incidence matrix of a graph with a privet-proper labelling of the nodes has the following two properties:

(A) For every row in the matrix, a 0 in between two 1's has a 1 above it in the same column.

(B) The highest 1 in any column is not lower than the highest 1 in any succeeding column.

Proof 4.3.

(A) Assume that row p has a 0 in column k and 1's in columns i and j, where $i < k < j$, and that there is no 1 in column k prior to row p. As the given matrix is row ordered, rows with a

1 in column k must have the other 1 in column $\ell > i$ (see Figure 4.1). But this implies that $\mathrm{PRIVEC}_k(j) > \mathrm{PRIVEC}_k(k)$, which is not possible since the labelling is privet-proper.

$$
\begin{array}{c|cccc}
 & i & \ell & k & j \\
\hline
 & & & 0 & \\
 & & & \vdots & \\
p & & & 0 & \\
 & 1 & & 0 & 1 \\
 & & 1 & 1 & \\
\end{array}
$$

Figure 4.1

(B) Suppose the highest 1 in column i is in row p and the highest 1 in column j is in a higher row q , with $i < j$ and $p > q$. Let the other 1 of row q be in column k . If $k < i$ then we have a situation which contradicts (A) (see Figure 4.2), and if $k > i$ then rows p and q are out of order, which is not possible. ❑

$$
\begin{array}{c|ccc}
 & k & i & j \\
\hline
q & 1 & 0 & 1 \\
p & & 1 & \\
\end{array}
$$

Figure 4.2

<u>Lemma 4.4</u>. A row-ordered incidence matrix of a graph with a privet-proper labelling of the nodes is column-ordered.

<u>Proof 4.4</u>. Recall from Section 2 that we are concerned only with connected graphs without multiple edges.

We will show that every two columns of the matrix are in order. Consider the highest 1's in columns i and j with $i < j$. By

Lemma 4.3B the highest 1 in column i is not lower than the highest 1 in column j . We claim that it is in fact higher, except for the case $i = 1$, $j = 2$. Suppose the contrary -- then columns i and j have their highest 1's in the same row, say row p . Suppose further $i<j-1$. Then there is a column k $(i < k < j)$ with a 0 in row p , so by Lemma 4.3A there must be a 1 in column k higher than row p -- however this violates Lemma 4.3B since the highest 1 in column i is in row p . Otherwise $i = j-1$, but then nodes $1,\ldots,i-1$ are not connected to nodes $i,\ldots,n$. This can be seen by considering Figure 4.3, where submatrix M_1 must be all 0's since the rows are ordered, and submatrix M_2 must be all 0's because of Lemma 4.3B and the fact that the 1% in row p are the highest in columns i and j. Thus the assumption that the graph is connected is violated.

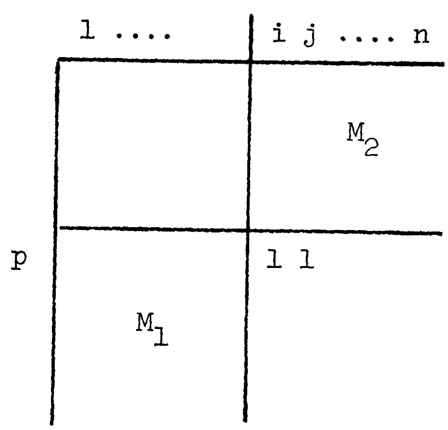


Figure 4.3

. Inthecase $i = 1$, $j = 2$ the first column must have 1's in the first two rows and the second column must have a 0 in the second row since the graph has no parallel edges and the first node chosen must be a node of greatest valence. Thus column 1 is greater than column 2 (except in the trivial case of a graph consisting of only one edge). □

Now let us prove Theorem 4.2.

<u>Proof 4.2</u>. By Lemma 4.3 any row-ordered incidence matrix M_1 of a graph with a privet-proper labelling of the nodes is column-ordered. Thus it is sufficient to show that no other permutation of the rows gives a matrix M_2 which is column greater.

Suppose the contrary. Let column k be the first column differing in M_1 and M_2, and let the first element of column k differing in M_1 and M_2 be in row p. Since M_2 is column greater than M_1, clearly $M_1(p,k) = 0$ and $M_2(p,k) = 1$. Because column k differs in M_1 and M_2 only by a permutation of elements p,...,e, there exists $q > p$ such that $M_1(q,k) = 1$. Therefore row p of M_1 has a 1 to the left of column k, say in column $i < k$, since the matrix is row-ordered. As column i is the same in both matrices we have $M_2(p,i) = M_1(p,i) = 1$. The other 1 in row p of M_1 must lie to the right of column k, say in column $j > k$; otherwise, if $j < k$, then $M_2(p,j) = M_1(p,j) = 1$ and there would be three 1's in row p of M_2. Thus we have $M_2(p,*) > M_1(p,*)$. Hence there exists $r < p$ such that $M_2(p,*) = M_1(r,*)$ (with the 1's in columns i and k), since M_1 and M_2 differ only by a permutation of rows and M_1 is row-ordered (see Figure 4.4).

$$
\begin{array}{c|ccc|cc}
 & & M_1 & & M_2 & \\
\hline
 & i & k & j & i & k \\
r & 1 & 1 & & & \\
p & 1 & 0 & 1 & 1 & 1 \\
q & & 1 & & & \\
\end{array}
$$

Figure 4.4

But then $M_2(r,*) = M_1(r,*)$ and we have two identical rows in M_2, which contradicts the assumption that the graph has no multiple edges. This completes the proof of Theorem 4.2. $\square$

We have now shown that the comim is a row-ordered matrix with a privet-proper node labelling. The example of Figure 2.1 shows that if M_1 and M_2 are two row-ordered matrices with privet-proper labellings it is possible for M_1 to be row greater than M_2 and M_2 column greater than M_1. However because of Theorem 4.1 we can (confine our attention to row-ordered matrices with privet-proper labellings and with

$$[\,\mathrm{PRIVEC}_i(i) = \mathrm{PRIVEC}_i(j) \text{ and } i < j\,] \Rightarrow \mathrm{valence}(i) \geq \mathrm{valence}(j)\ .$$

At first sight it might seem that if M_1 and M_2 are two such matrices then M_1 is row greater than M_2 if and only if M_1 is column greater than M_2. However this is not the case and Figure 4.5 gives a counterexample.

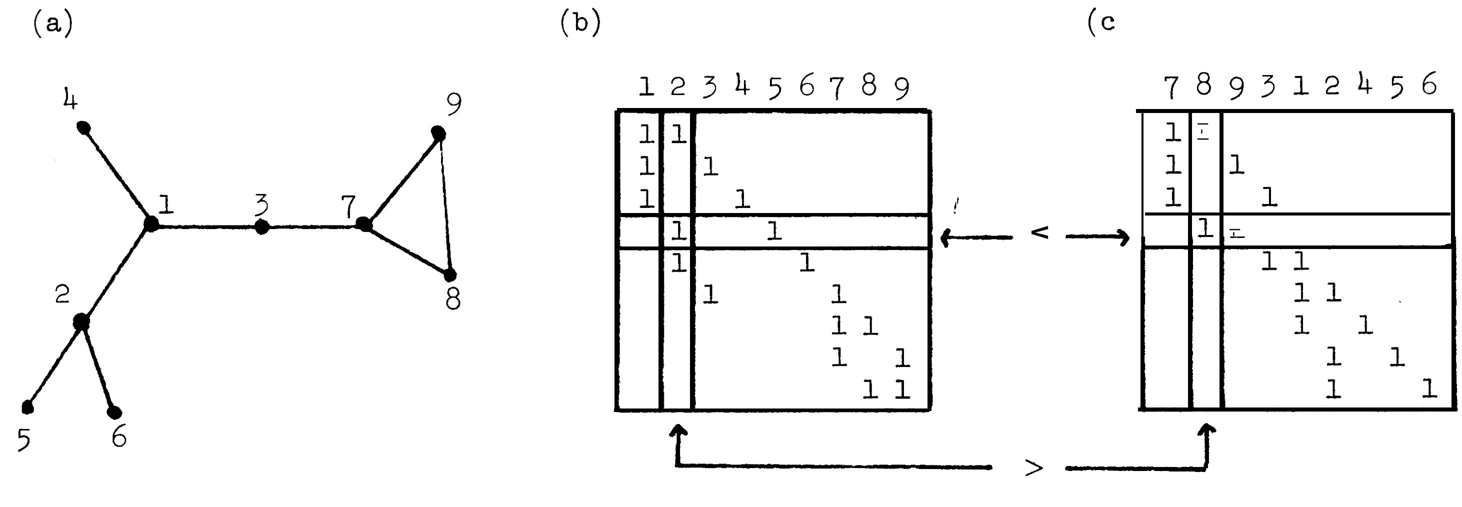


Figure 4.5. Two incidence matrices of a graph (a). Matrix (b) is column greater than (c), but (c) is row greater than (b).

5. Data Structures.

A data structure for the search of the maximal incidence matrix of small graphs by means of this algorithm was proposed and used in [2]. The incidence matrix of a graph with n nodes and e edges was represented by e words. If edge i was incident to nodes j and k , then the i-th word was a bit string with 1's only in positions j and k . Thus one row of the incidence matrix was stored in one word of computer memory. Such a representation facilitated manipulating the matrix by logical operations on the bit strings. However, the number of nodes in the graph was limited by the number of bits in the computer word. In the data structure proposed here the number of nodes is limited only by the computer integer range and the size of computer memory.

In the present implementation the incidence matrix INCMAT is stored in an e x 2 integer array (twice the storage of the old representation). If edge i is incident to nodes j and k then the i-th row is an unordered pair of integers j and k .

During the search it is necessary to order the rows of INCMAT. The ordering is achieved by introducing an integer vector NEXTEDGE which transforms INCMAT into a linked list. This vector is dimensioned from 0 to e with NEXTEDGE(i) = i+1 initially, except NEXTEDGE(e) = -1 . As nodes are labelled, the edges incident to them are "pulled up to the top of the list". The pointer LASTLABELLED points to the last such edge pulled up; initially LASTLABELLED is set to zero. More precisely, when procedure CHOOSE is entered, with say node p chosen to be the next labelled node, procedure PULLUP is called, which scans down the linked list (INCMAT, NEXTEDGE) starting from LASTLABELLED, and upon encountering an edge incident to p , deletes the edge from the list, inserts it following the edge at LASTLABELLED, and updates LASTWELLED. The two nodes incident to the edge are interchanged if necessary so that p is in the first column, the other node is examined, and the priority vector PRIVEC is modified accordingly.

When a leaf of the search tree is reached the new candidate for the maximal matrix must be calculated from the linked list representing the incidence matrix. This means that the rows of the incidence matrix must be lexicographically sorted. The entries in the first column of INCMAT are in order determined by the label permutation found. The second column, however, requires sorting of entries within blocks (cf. Lemma 3.2) to obtain an ordered matrix. Now the new matrix may be compared with MAXMAT, the maximal matrix found so far. This testing -- in the sense of row ordering -- is an easy task for the chosen data structure. It suffices simply to compare the two-element rows of the new matrix with those of MAXMAT one at a time. The test in the sense of column-ordering is not as obvious, and will be described in Section 6.

The priority vector PRIVEC does not have to be stored in a way described in Section 3, with the number of bits in each element equal to the number of nodes labelled so far. Instead it is stored here in an integer array called PRIVEC, whose entries are node numbers and which is broken into a number of logical blocks. (Now we are talking about blocks in PRIVEC, not the ones defined in Lemma 3.2.) At any stage of the labelling process all nodes within a block have equal priority, and nodes within one block have higher priority than nodes within another block further down the vector. There may be a block of nodes that have not been assigned any priority yet -- the last part of the PRIVEC may contain only zeros. This block is referred to as the empty block. In the priority vector described in Section 3, when a node p is labelled one more bit is added to every element of the vector: 1 to those elements corresponding to nodes adjacent to p and 0 to all other elements. In the data structure described here, when a node p is labelled, any node adjacent to p is either added to the empty block if it is not already in PRIVEC or marked in PRIVEC if it is already there. Such nodes are found by procedure PULLUP, described earlier. After all nodes adjacent to p have been found, procedure SHUFFLE is called. This procedure scans PRIVEC and shuffles the entries within each block so that the elements marked by the action of PULLUP are moved to the top of the block and the unmarked elements are moved to the bottom. If these two sets of elements are both nonempty the block is then split into two blocks, since the marked elements have higher priority than the unmarked elements. After all blocks have been shuffled, the empty block is checked for the presence of any new elements. If some were added by the pull up operation, a new block is created to accommodate them and the remaining zero elements become the new empty block.

In order to avoid searching the entire vector PRIVEC every time an edge incident to p is pulled up, a new vector CROSSREF is introduced. This is the cross reference to PRIVEC: at any time, if PRIVEC(i) = j > 0 then CROSSREF(j) = i .

The description of the PRIVEC blocks is stored in a list of records, pointed to by BLOCKLIST. Each record contains an integer field BLOCKPTR and a link NEXTBLOCK. The BLOCKPTR fields are integers pointing to the first element of each of the blocks in PRIVEC. The integer BLOCKPTR (BLOCKLIST) points to the first element in the highest priority block of PRIVEC; this element is not necessarily the first element of PRIVEC as will be explained shortly. The pointer EMPTYBLOCK points to the last record in the list. The integer BLOCKPTR (EMPTYBLOCK) points to the first zero element of PRIVEC, unless every node has been entered in PRIVEC in which case the pointer will have value n+1 .

Initially BLOCKLIST is set to point to a list of two blocks, the first containing a node of maximum valence and the second the empty block. At any stage in the search the pointer BLOCKLIST points to a list of at least two records. The initial data structures for a certain incidence matrix are shown in Figure 5.1.

After initialization, procedure CHOOSE is called. The node with the highest priority is considered to be labelled and procedures PULLUP and SHUFFLE are called to perform the actions described earlier. The resulting data structures are illustrated in Figure 5.2.

At this point the block containing the labelled node is deleted from the block list. (In fact the deletion is done in between PULLUP and SHUFFLE since it is a bit simpler to do so, but this makes no difference.) Now another node must be labelled so the first block of the modified block list is examined. If it contains only one element, CHOOSE is called. If'it contains more than one element there are several pretenders to the label, so PRETEND is called. Then PRETEND will call CHOOSE several times, each time with the first block split into two blocks, one containing a, single chosen pretender and the other containing the remaining pretenders. In the comim search the valence check may reduce the number of calls to CHOOSE (see Section 6).

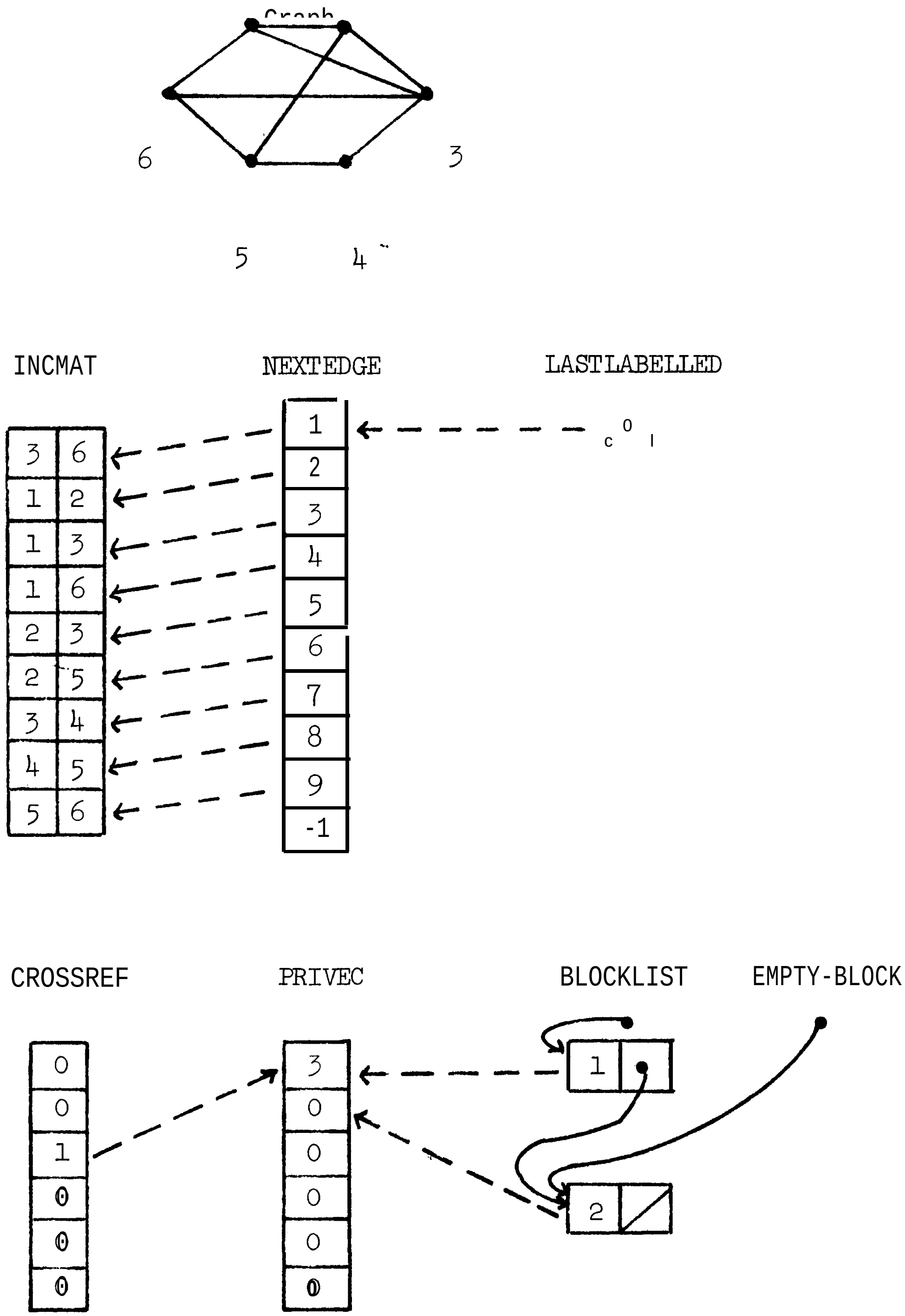


Figure 5.1. Example of initial structure.

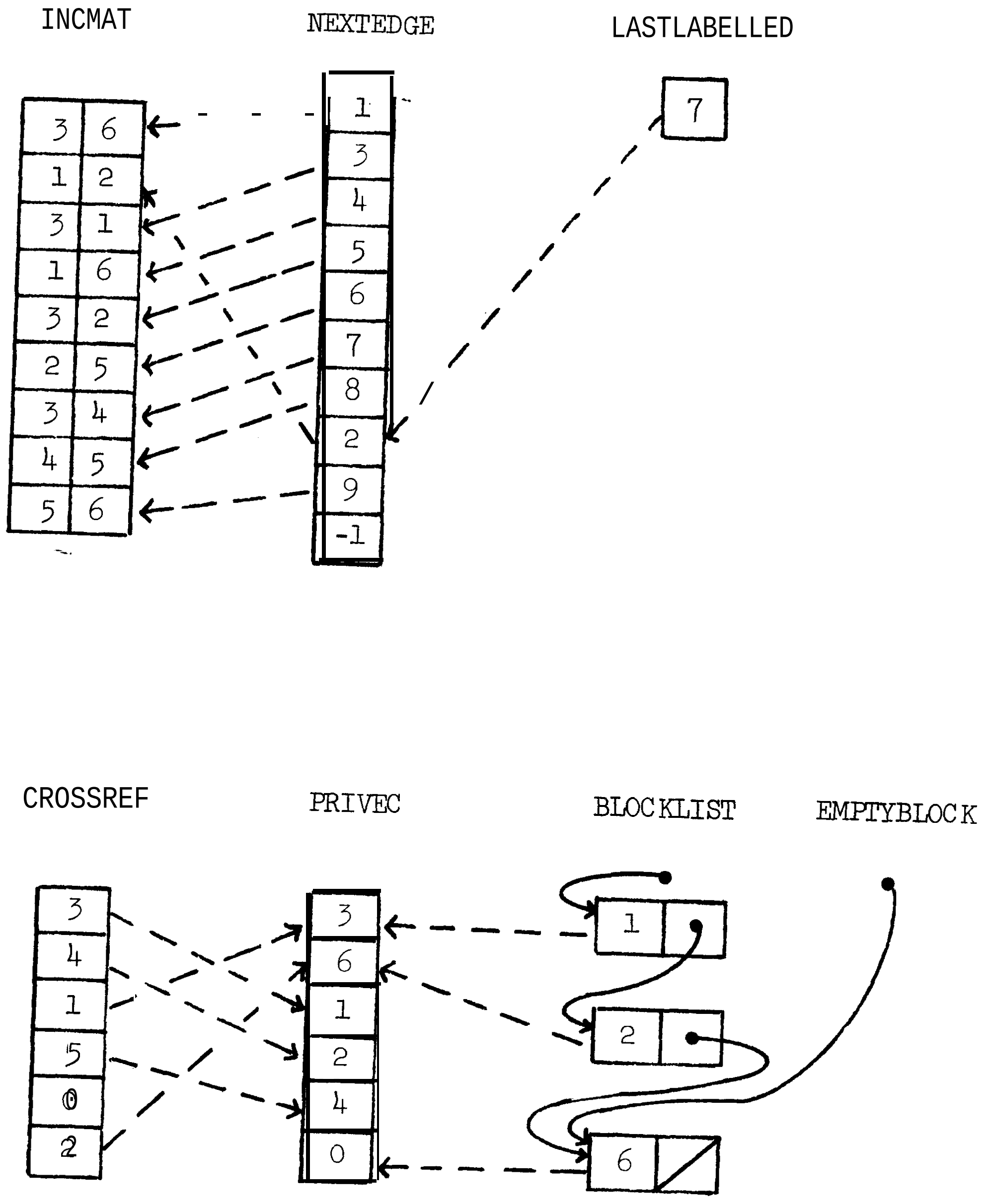


Figure 5.2. In first call of CHOOSE, after PULLUP and SHUFFLE.

Because of the recursive nature of the search, the crucial question that one must ask here is: <u>how much must be kept on the stack</u>? The answer is that when CHOOSE is calling itself or calling PRETEND only three words must be passed (as value parameters): FIRSTBLOCK, EMPTYBLOCK, and LASTLABELLED; when PRETEND is calling CHOOSE (i.e., at a branch in the search tree), only a copy of the block list structure must be passed in addition. At no time is it necessary to have more than one instance of INCMAT, NEXTEDGE, PRIVEC or CROSSREF. This is very important, since these arrays may be large and the search tree deep. It is not necessary to keep a copy of INCMAT or NEXTEDGE because any changes made to the linkedlistonly reorder the edges or reverse the pair of nodes incident to an edge, producing an incidence matrix as valid as the original one. It is not necessary to keep a copy of PRIVEC or CROSSREF because the only changes made to PRIVEC take the form either of shuffling elements within a block, or of adding elements to the empty block. Note that splitting a block does not affect PRIVEC but only inserts a new record in the block list. Since elements within one block have equal priority the shuffling does not destroy the priority information. Any elements added to the empty block of PRIVEC at a lower level may be deleted on return by saving a pointer to the empty block before the call; corresponding new CROSSREF entries may be deleted at the same time. An actual new copy of the block list structure need be made only when PRETEND calls CHOOSE, since this is the only point where the search tree branches.

The example of Figures 5.1 and 5.2 is continued in Figures 5.3 and 5.4, illustrating the situation after PRETEND has been called by CHOOSE, 'and after CHOOSE has been called again.

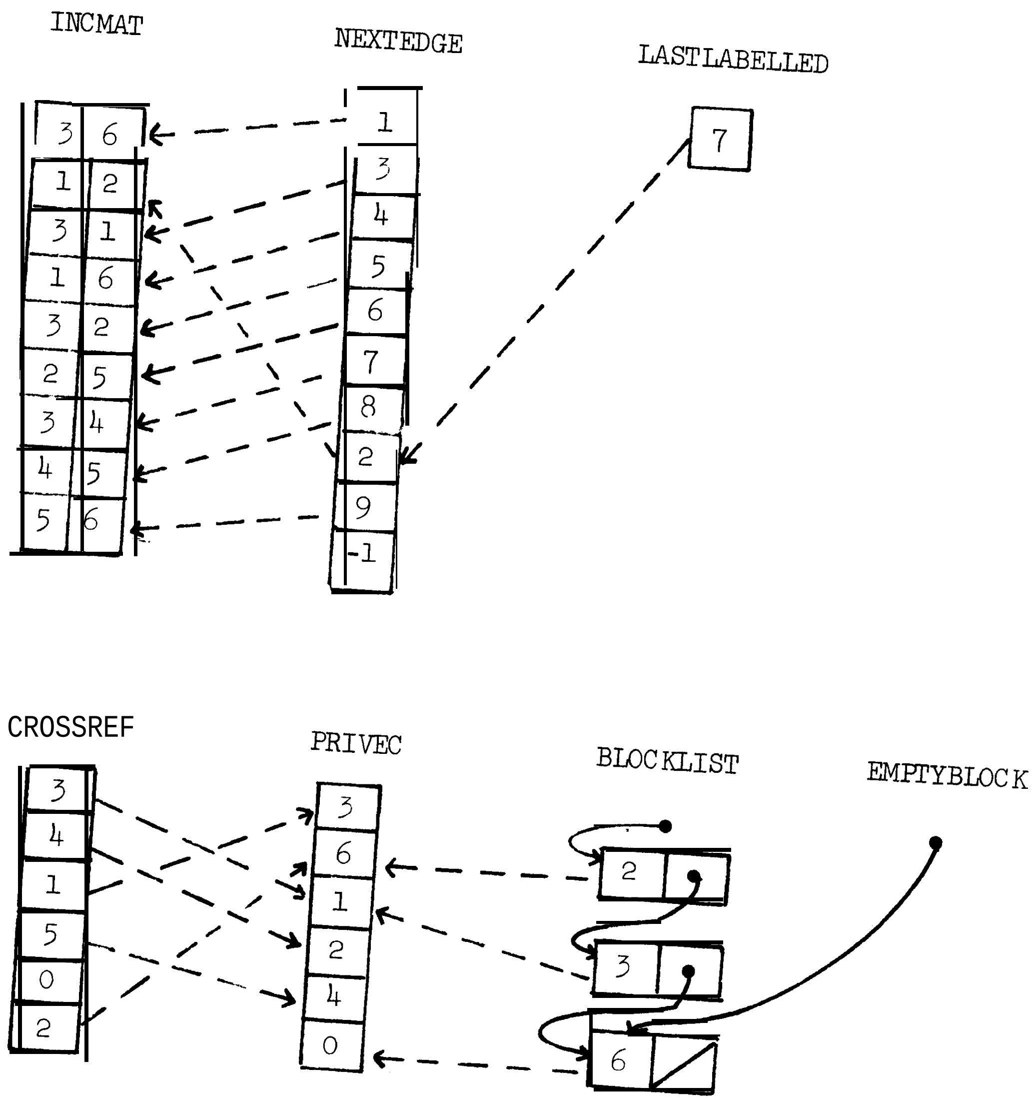


Figure 5.3. In first call of PRETEND, just before next call of' CHOOSE.

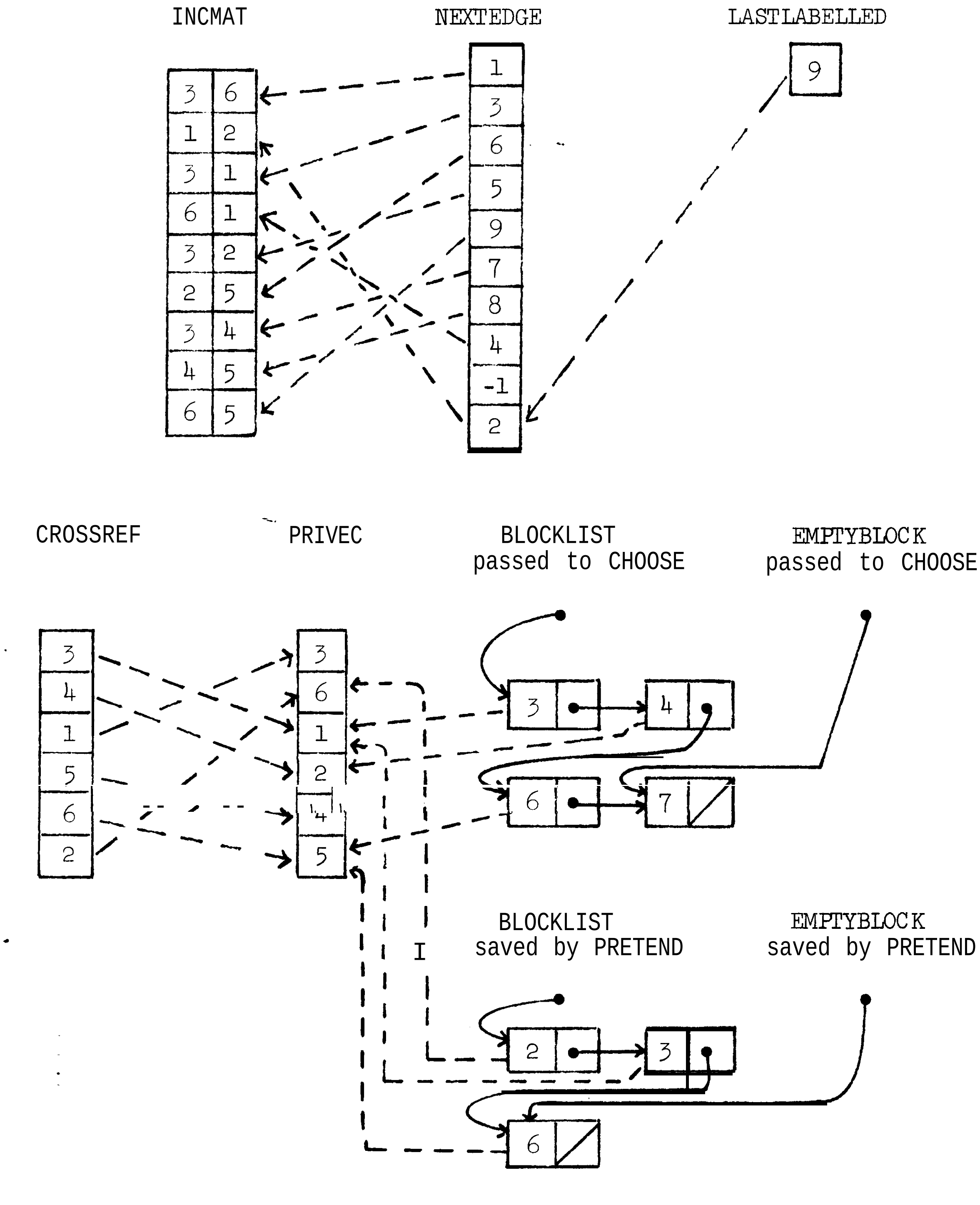


Figure 5.4. In second call of CHOOSE, just before next call of CHOOSE.

6. Differences in Implementation of the Search for the Romim and the Comim.

The previous sections showed that the same basic search tree can be used for both the romim and the comim, and furthermore that the search tree for the comim can be significantly pruned by making use of the node valences. The pruning is done by considering the valences of the pretenders at the beginning of procedure PRETEND. The maximum valence of the pretenders is found, and CHOOSE is called only for the pretenders with this valence. The valences of all the nodes were computed at the beginning of the program and stored in the array VALVEC.

The data structures described in Section 5 are particularly well suited for the romim search. With a slight modification they may also be used in the search for the comim. At a leaf of the search tree the new matrix found must be compared with the maximal matrix so far. In the romim case the maximal matrix so far is stored in MAXMAT, an array with the same format as INCMAT, and as explained in Section 5 it is then very easy to do the necessary row comparison. However the column comparison for the comim search would be very inefficient using this structure. A solution is to translate the row-ordered incidence matrix found into an array of n linked lists, each corresponding to a column and listing the rows with a 1 in this column. Then with the maximal matrix so far stored in a similar array of linked lists MAXMATCOL, the column comparison of the two matrices simply requires a series of scans down the lists. The matrix comparison, together with a replacement of the maximal matrix so far if necessary, is done by one of two versions of procedure UPDATE -- one for the romim and one for the comim.

Advantages of the comim search are demonstrated by the running times of an ALGOL W program which implements the search and data structures described. One of the parameters to the program is a logical variable whose value specifies whether to calculate the romim or the canim. The "records and references" dynamic storage feature of ALGOL W is used for the lists BLOCKLIST and MAXMATCOL. Integer arrays are used for all the other list structures since they do not change size dynamically. The program, listed in Appendix A, was run for several graphs on an IBM 370/168. The results are summarized in Table 6.1. The computer printouts and an explanation of the choice of graphs are given in Appendix B.

GRAPH	NODES	EDGES	ROMIM TIME	ROMIM LEAVES	COMIM TIME	COMIM LEAVES
1	6	9	.01	8	.01	4
2	7	7	.02	24	.01	4
3	17	22	.12	24	.02	2
4	18	24	.12	24	.03	2
5	19	29	.34	144	.10	16
6	22	30	2.13	576	.03	1
7	23	31	3.10	1152	.04	2
8	24	32	7.03	3456	.06	6
9	50	78	> 600		.81	48

Table 6.1. Summary of the results of sample runs.
Time is shown in seconds.

We see from the results that the comim search is substantially faster than the romim search. However the data structures in the program were designed primarily for the romim search. We could expect significant improvements in the performance of the comim search if more suitable data structures were used. A particularly attractive idea is to compare the maximal matrix so far with the new incidence matrix found as it is built up, and thus have the possibility of abandoning unuseful labellings early. This could be done if more of the priority information was kept. The idea of abandoning labellings early might also be applicable to the romim search, for example if the edges were labelled first instead of the nodes.

Highly symmetric graphs (graphs with many automorphisms) will require search trees with a large number of redundant leaves corresponding to automorphic permutations of nodes. A way to eliminate some of these leaves by keeping track of automorphic permutations as the search progresses is discussed in [3].

Acknowledgment.

We are grateful to Prof. D. E. Knuth for his constructive criticism and encouragement.

References.

[1] Proskurowski, Andrzej, "The Maximal Incidence Matrix of a Graph," Technical Report No. 70, December 1973, Royal Institute of Technology, Stockholm.

[2] Proskurowski, Andrzej, "Search for the Unique Incidence Matrix of a Graph," BIT 2 (14), 1974.

[3] Proskurowski, Andrzej, "Graph Symmetries in the Search for the Maximal Incidence Matrix," Technical Report No. 75, April 1974, Royal Institute of Technology, Stockholm.

Appendix A

The Program.

```
COMMENT

FINDING THE MAXIMAL INCIDENCE MATRIX OF A LARGE GRAPH
        MICHAEL OVERTON AND ANDRZEJ PROSKUROWSKI
             COMPUTER SCIENCE DEPARTMENT
                 STANFORD UNIVERSITY
                       JULY 1975

BEGIN

PROCEDURE KLAPPERSLANGE (INTEGER ARRAY INCMAT,MAXMAT(*,*);
 INTEGER ARRAY PERMUTATION(*); INTEGER VALUE NODES,EDGES;
 LOGICAL VALUE ROMIM; INTEGER RESULT LEAVES);
 COMMENT TAKES THE INCIDENCE MATRIX INCMAT OF A GRAPH AND RETURNS THE
  (ROMIM OR COMIM) MAXIMAL MATRIX MAXMAT AND THE LABEL PERMUTATION;
 COMMENT LEAVES IS SET TO THE NUMBER OF LEAVES IN THE SEARCH TREE;
       BEGIN
       INTEGER ARRAY NEXTEDGE(0::EDGES);
        COMMENT THESE POINTERS TRANSFORM INCMAT INTO A LINKED LIST;
       INTEGER LASTLABELLED;
        COMMENT POINTS TO LAST EDGE "PULLED UP"  BY NODE LABELLING;

       INTEGER ARRAY PRIVEC,CROSSREF,VALVEC(1::NODES);
        COMMENT PRIVEC IS THE PRIORITY VECTOR, CROSSREF THE CROSS
        REFERENCE TO PRIVEC, AND VALVEC THE VECTOR OF VALENCES;

       RECORD BLOCK(INTEGER BLOCKPTR; REFERENCE(BLOCK) NEXTBLOCK);
       REFERENCE(BLOCK) BLOCKLIST,EMPTYBLOCK;
        COMMENT BLOCKLIST POINTS TO THE LIST OF BLOCKS OF PRIVEC.
        EMPTYBLOCK POINTS TO THE LAST ELEMENT OF THE LIST;

       RECORD INCEDGE(INTEGER EDGENO; REFERENCE(INCEDGE) NEXTONE);
       REFERENCE(INCEDGE) ARRAY MAXMATCOL(1::NODES);
        COMMENT HEADS OF THE LIST REPRESENTATION OF MAXMAT -
        USED ONLY IN THE COMIM SEARCH;

       PROCEDURE CHOOSE (REFERENCE(BLOCK) VALUE BLOCKLIST,EMPTYBLOCK;
        INTEGER VALUE LASTLABELLED);
         COMMENT LABEL THE ONLY ELEMENT OF THE FIRST BLOCK IN BLOCKLIST.
         REARRANGE THE INCIDENCE MATRIX AND MODIFY PRIVEC ACCORDINGLY
        BY CALLING PULLUP AND SHUFFLE;
             BEGIN

             PROCEDURE PULLUP (INTEGER VALUE CHOSEN);
              COMMENT SCAN DOWN INCIDENCE MATRIX STARTING FROM
             LASTLABELLED.  UPON ENCOUNTERING AN EDGE INCIDENT TO NODE
             CHOSEN, PROCEED TO "PULL UP" THE EDGE TO LASTLABELLED.
             AFTER LOOKING AT THE OTHER NODE OF THE EDGE, MODIFY PRIVEC
             ACCORDINGLY;
                    BEGIN
                    INTEGER P,PREV,PRIVECLAST;
                    PREV:=LASTLABELLED; P:=NEXTEDGE(LASTLABELLED);
                    PRIVECLAST:=BLOCKPTR(EMPTYBLOCK); COMMENT POINTS TO
                     THE FIRST ZERO ELEMENT OF PRIVEC;
                    WHILE P¬=-1 DO
                          BEGIN INTEGER FOUND,X;
                          FOUND:= IF INCMAT(P,1)=CHOSEN THEN 1 ELSE IF
                           INCMAT(P,2) = CHOSEN THEN 2 ELSE 0;
                          IF FOUND¬=0 THEN
```

```
                BEGIN
                INTEGER TEMP;
                COMMENT PUT CHOSEN NODE IN FIRST COLUMN;
                IF FOUND=2 THEN
                      BEGIN INCMAT(P,2):=INCMAT( P,1);
                      INCMAT(P,1 ):=CHOSEN; END;
                COMMENT MODIFY PRI VEC;
                X:=INCMAT(P,2);
                IF CROSSREF(X)=0 THEN COMMENT NOTI N
                 PRIVEC SO ADD IT;
                      BEGIN
                      PRIVEC(PRIVECLAST):=X;
                      CROSSREF(X):=PRIVECLAST;
                      PRIVECLAST:=PRIVECLAST+1;
                      END
                ELSE COMMENTALREADYI NPRIVECS O  M A R KIT;
                PRIVEC(CROSSREF(X)):=-PRIVEC(CROSSREF(X));
                COMMENT PULL EDGE  HIGHER UP IN LIST;
                IF PREV¬=LASTLABELLED THEN
                      BEGIN
                      TEMP:=NEXTEDGE(LASTLABELLED);
                      NEXTEDGE(LASTLABELLED):=P;
                      NEXTEDGE(PREV):=NEXTEDGE(P);
                      NEXTEDGE(P):=TEMP;
                      LASTLABELLED:=P;
                      P:=NEXTEDGE(PREV);
                      END
                ELSE COMMENT P U L L  U P NOTNECESSARY;
                      BEGIN
                      LASTLABELLED:=PREV:=P;
                      P:=NEXTEDGE(P);
                      END;
                END
          ELSE COMMENT CHOSEN N  O  T FOUND IN EDGE;
                BEGIN
                PREV: =P; P:=NEXTEDGE(P);
                END
          END;
      COMMENT IF ANY NEW ELEMENTS H  A  V  E BEEN ADDED TO PRIVEC
       THEN CREATE A NEW BLOCK FOR T H E M ;
      IF PRIVECLAST>BLOCKPTR(EMPTYBLOCK) T H E N
       EMPTYBLOCK:=NEXTBLOCK(EMPTYBLOCK):=
        BLOCK(PRIVECLAST,NULL);
      END PULLUP;

PROCEDURE SHUFFA  L(REFERENCE(BLOCK)VALUE P);
COMMENT SCAN DOWN LIST OF BLOCK POINTERS.  FOR ANY BLOCK
CONTAINING NEGAT IVE ELEMENTS, S H U F F L E THE BLOCK SPLITTING
I T INTO TWO BLOCKS, THE FIRST CONTAINING THE NEGATI V E
ELEMENTS A  N  D THE SECOND THE POSITIVE - ALSO RESET THE
NEGATIVE ELEMENTS TO POSITIVE;
WHILE NEXTBLOCK(P)¬= N U L L DO
      BEGININTEGER A ,BORDER; COMMENTAFTERSHUFFL I N G T H E
      BLOCK EL EMENT S, BORDER WILL B E  J H E INDE X OF THE FIRST
      NONNEGATIVE ELEMENT ;
      BORDER:=BLOCKPTR(P);
      FOR I:= BLOCKPTR(P) UNTIL BLOCKPTR(NEXTBLOCK(P)) - 1
       D CIFPRIVEC(I)<0THEN C O M M E N T MOVEMARKED
      NODE UP TO NEGATIV  EHALF O  F BLOCK;
            BEGIN IF I¬= BORDER THEN
```

```
                    BEGIN INTEGER TEMP;
                    TEMP:=PRIVEC(I);
                    PRIVEC(I):=PRIVEC(BORDER);
                    PRIVEC(BORDER):= - TEMP;
                    CROSSREF(PRIVEC(I)):=I;
                    CROSSREF(PRIVEC(BORDER)):=BORDER;
                    END
                ELSE PRIVEC(I):=-PRIVEC(I);
                BORDER:=BORDER+1;
                END;
            COMMENT IF BOTH THE POSITIVE AND NEGATIVE HALVES
             OF THE BLOCK ARE NONEMPTY THEN SPLIT THE BLOCK;
            IF (BORDER¬=BLOCKPTR(P)) AND (BORDER¬=BLOCKPTR(
             NEXTBLOCK(P))) THEN NEXTBLOCK(P):=BLOCK(BORDER,
             NEXTBLOCK(P));
            P:=NEXTBLOCK(P);

            END SHUFFLE;

        INTEGER NEWNODES;
        NEWNODES:=BLOCKPTR(EMPTYBLOCK) ; COMMENT POINTER TO
         FIRST ZERO ENTRY IN PRIVEC TO BE USED FOR RESTORING;
        IF TRACE THEN WRITE ("ENTER CHOOSE WITH CHOSEN NODE =",
         PRIVEC(BLOCKPTR(BLOCKLIST)));
        PULLUP(PRIVEC(BLOCKPTR(BLOCKLIST)));
        COMMENT NOW THE FIRST BLOCK HAS BEEN DEALT WITH SO DELETE
         IT; BLOCKLIST:=NEXTBLOCK(BLOCKLIST);
        IF NEXTBLOCK(BLOCKLIST)¬=NULL THEN COMMENT THERE ARE STILL
         UNLABELLED NODES;
            BEGIN INTEGER PRETENDERS;
            SHUFFLE(BLOCKLIST);
            COMMENT   PRIVEC HAS NOW BEEN UPDATED AS REQUIRED BY
             THE LABELLING OF THE NODE.  IF THE FIRST BLOCK OF
             THE MODIFIED PRIVEC CONTAINS ONLY ONE ELEMENT THEN
             LABEL IT BY CALLING CHOOSE - OTHERWISE THERE ARE
             SEVERAL PRETENDERS;
            PRETENDERS:=BLOCKPTR(NEXTBLOCK(BLOCKLIST))-BLOCKPTR(
             BLOCKLIST);
            IF PRETENDERS = 1 THEN CHOOSE(BLOCKLIST,EMPTYBLOCK,
             LASTLABELLED) ELSE PRETEND(BLOCKLIST,EMPTYBLOCK,
             LASTLABELLED,PRETENDERS);
            COMMENT IT IS NOT NECESSARY TO PASS A NEW COPY OF THE
             BLOCK LIST;
            END
    ELSE COMMENT ALL NODES HAVE BEEN LABELLED SO CALCULATE
       THE INCIDENCE MATRIX FOUND AND UPDATE MAXMAT IF NECESSARY;
        IF RCMIM THEN UPDATE_ROMIM ELSE UPDATE_COMIM;
       COMMENT RESTORE PRIVEC AND CROSSREF WHICH HAVE BEEN
        MODIFIED BY SEARCH ON DEEPER LEVELS.  DELETE THE NEW
        NODES FROM PRIVEC AND DELETE THE CORRESPONDING
        ENTRIES IN CROSSREF;
       WHILE (NEWNODES<=NODES) AND (PRIVEC(NEWNODES)¬=0) DO
            BEGIN
            CROSSREF(PRIVEC(NEWNODES)):=0;
            PRIVEC(NEWNODES):=0;
            NEWNODES:=NEWNODES+1;
            END;
       IF TRACE THEN WRITE("EXIT CHOOSE") ;
       END CHOOSE;
```

```
PROCEDURE PRETEND (REFERENCE(BLOCK) VALUE BLOCKLIST,EMPTYBLOCK;
 INTEGER VALUE LASTLABELLED,PRETENDERS);
 COMMENT ASSIGN NEXT LABEL TO EACH OF THE PRETENDERS IN TURN BY
 CREATING A NEW BLOCK CONTAINING THE CHOSEN ELEMENT ONLY AND
 CALLING CHOOSE;
      BEGIN

      REFERENCE(BLOCK) PROCEDURE COPY (REFERENCE(BLOCK) VALUE P;
       REFERENCE(BLOCK) RESULT Q);
       COMMENT COPY THE LIST POINTED TO BY P , RETURN A POINTER
       TO IT AS THE PROCEDURE VALUE, AND SET Q TO POINT TO THE
       LAST ELEMENT OF THE LIST;
       IF P = NULL THEN
            BEGIN Q:=NULL; NULL END
       ELSE IF NEXTBLOCK(P) = NULL THEN
            BEGIN Q:=BLOCK(BLOCKPTR(P),NULL); Q END
       ELSE BLOCK(BLOCKPTR(P),COPY(NEXTBLOCK(P),Q));

      REFERENCE(BLOCK) HEAD,TAIL;COMMENT POINTERS TO NEW INSTANCES
       OF BLOCKLIST AND EMPTYBLOCK;
      INTEGER ARRAY BLOCKPRETS(1::PRETENDERS);
      INTEGER MAX,KEPT;
      IF TRACE THEN WRITE("ENTER PRETEND WITH",PRETENDERS,
       "  PRETENDERS");
      COMMENT COPY THE FIRST BLOCK OF BLOCKLIST (CONTAINING THE
       PRETENDERS) TO BLOCKPRETS;
      FOR I:=1 UNTIL PRETENDERS DO BLOCKPRETS(I):=PRIVEC(BLOCKPTR(
       BLOCKLIST)+I-1);
      COMMENT INTRODUCE A NEW BLOCK FOR THE CHOSEN NODE;
      NEXTBLOCK(BLOCKLIST):=BLOCK(BLOCKPTR(BLOCKLIST)+1,NEXTBLOCK(
       BLOCKLIST));

      COMMENT FOR THE COMIM ONLY FIND THE STRICT SET OF PRETENDERS
       TO THE NEXT LABEL BY CONSIDERING THE VALENCES;
      IF ¬RCMIM THEN
            BEGIN INTEGER V;
            COMMENT FIND THE MAX VALENCE OF THE PRETENDERS - KEPT
             IS THE NUMBER OF PRETENDERS WITH THE MAX VALENCE:;
            MAX:=KEPT:=0;
            FOR I := 1 UNTIL PRETENDERS DO
                  BEGIN
                  V:=VALVEC(PRIVEC(BLOCKPTR(BLOCKLIST)+I-1));
                  IF V>MAX THEN BEGIN MAX:=V; KEPT:=1; END
                  ELSE IF V=MAX THEN KEPT:=KEPT +1;
                  END;
            IF TRACE THEN WRITE("VALENCE CHECK:",KEPT,
             " PRETENDER(S) TO BE CONSIDERED");
            END;
      IF RCMIM OR (VALVEC(BLOCKPRETS(1))=MAX) THEN
       COMMENT CALL CHOOSE PASSING THE FIRST PRETENDER - IT
        IS NECESSARY TO PASS A NEW COPY OF THE BLOCKLIST
        BECAUSE OF THE TENTATIVE ASSIGNMENT UNLESS (FOR
        THE COMIM) ONLY ONE PRETENDER HAS THE MAX VALENCE;
       IF ¬RCMIM AND (KEPT=1) THEN
        CHOOSE(BLOCKLIST,EMPTYBLOCK,LASTLABELLED)
       ELSE
             BEGIN
             HEAD:=COPY(BLOCKLIST,TAIL);
             CHOOSE(HEAD,TAIL,LASTLABELLED);
             END;
```

```
FOR CHOSEN:=2 UNTIL PRETENDERS DO
  IF RCMIM OR (VALVEC(BLOCKPRETS(CHOSEN))=MAX) THEN
       BEGIN INTEGER I; LOGICAL FOUND;
       I:=1; FOUND:=FALSE;
       COMMENT PREVIOUS CALLS TO CHOOSE MAY HAVE CHANGED THE
        ORDER OF THE ELEMENTS IN THE FIRST BLOCK OF PRIVEC
        SO IT IS NECESSARY TO SEARCH FOR THE CHOSEN NODE;
       WHILE ¬FOUND DO IF PRIVEC(BLOCKPTR(BLOCKLIST)+I)=
        BLOCKPRETS(CHOSEN) THEN
             BEGIN COMMENT INTERCHANGE CHOSEN NODE WITH THE
             PREVIOUSLY CHOSEN NODE IN THE FIRST POSITION OF
             THE BLOCK;
             PRIVEC(BLOCKPTR(BLOCKLIST)+I):=PRIVEC(BLOCKPTR(
              BLOCKLIST));
             PRIVEC(BLOCKPTR(BLOCKLIST)):=BLOCKPRETS(CHOSEN);
             CROSSREF(PRIVEC(BLOCKPTR(BLOCKLIST)+I)):=
               BLOCKPTR(BLOCKLIST)+I;
             CROSSREF(BLOCKPRETS(CHOSEN)):=
               BLOCKPTR(BLOCKLIST);
             FOUND:=TRUE;
             END
        ELSE I:=I+1;
       COMMENT CALL CHOOSE PASSING THE PRETENDER - IT
        IS NECESSARY TO PASS A NEW COPY OF THE BLOCK LIST
        BECAUSE OF THE TENTATIVE ASSIGNMENT UNLESS (FOR
        THE CUMIM) ONLY ONE PRETENDER HAS THE MAX VALENCE;
       IF ¬RCMIM AND (KEPT=1) THEN
        CHOOSE(BLOCKLIST,EMPTYBLOCK,LASTLABELLED)
       ELSE
             BEGIN
             HEAD:=COPY(BLOCKLIST,TAIL);
             CHOOSE(HEAD,TAIL,LASTLABELLED);
             END;
       END;
IF TRACE THEN WRITE("EXIT PRETEND");
END PRETEND;

PROCEDURE UPDATE_RCMIM;
 COMMENT COMPARE THE INCIDENCE MATRIX OBTAINED BY NEW LABELLING TO
 THE MAXIMAL MATRIX FOUND SO FAR (MAXMAT) AND REPLACE THE LATTER
 IF NECESSARY;
 COMMENT THIS IS FOR THE ROMIM ONLY;
      BEGIN
      COMMENT BECAUSE OF THE ACTION OF PULLUP, THE FIRST COLUMN
       OF INCMAT IS ARRANGED IN THE DESIRED (LINKED) ORDER WITH
       THE LABEL PERMUTATION GIVEN BY CROSSREF;
      COMMENT IN THIS PROCEDURE THE TERM "BLOCK" IS USED TO MEAN
       A SECTION OF INCMAT WITH ALL ELEMENTS OF THE FIRST COLUMN
       EQUAL, AND THE RELATION "MAXMAT>NEW MATRIX" IS USED TO MEAN
       MAXMAT IS BETTER THAN THE NEW MATRIX;
      INTEGER ARRAY RELABELLED(1::MAXVAL);
      COMMENT RELABELLED IS USED FOR SORTING THE SECOND COLUMN OF
       THE CURRENT BLOCK OF INCMAT BEING EXAMINED;
      INTEGER I,J,JO,K; COMMENT I POINTS TO INCMAT, J AND JO TO
       MAXMAT AND K TO RELABELLED;
      INTEGER ELT1,COMP;
      COMMENT ELT1 IS THE ELEMENT IN THE FIRST COLUMN OF THE
       CURRENT BLOCK OF INCMAT BEING EXAMINED;
      COMMENT COMP IS SET POSITIVE IF MAXMAT>NEW MATRIX
       AND NEGATIVE IF MAXMAT<NEW MATRIX;
```

```
PROCEDURE SORT;
 COMMENT SORT RELABELLED FROM 1 TO K IN ASCENDING ORDER BY
 STRAIGHT INSERTION SORT;
  FOR L:=2 UNTIL K DO
       BEGIN INTEGER KEY,I;
       I:=L-1;
       KEY:=RELABELLED(L);
       WHILE (I > 0) AND (KEY < RELABELLED(I)) DO
              BEGIN
              RELABELLED(I+1):=RELABELLED(I);
              I:=I-1;
              END;
       RELABELLED(I+1):=KEY;
       END;

PROCEDURE COMPARE;
 COMMENT COMPARE THE PART OF THE INCIDENCE MATRIX IN
  RELABELLED TOGETHER WITH THE FIRST COLUMN ELEMENT
  CROSSREF(ELT1) WITH THE CORRESPONDING BLOCK OF MAXMAT.
  COMP IS SET POSITIVE IF MAXMAT IS BIGGER AND NEGATIVE
   IF MAXMAT IS SMALLER;
       BEGIN
       WHILE (COMP=0) AND (J<J0+K) DO
              BEGIN
              COMP:=-((MAXMAT(J,1)-CROSSREF(ELT1)) * NODES +
               MAXMAT(J,2)-RELABELLED(J-J0+1));
              COMMENT WATCH OUT FOR OVERFLOW FOR LARGE GRAPHS;
              J:=J+1;
              END;
       IF COMP¬=0 THEN COMMENT SET J TO FIRST ENTRY IN MAXMAT
        DIFFERENT FROM THAT IN THE NEW MATRIX;
        J:=J-1;
       END COMPARE;

LOGICAL FIRSTSAME;
COMP:=0;
I:=NEXTEDGE(0);
J0:=1; K:=0;
WHILE (I¬=-1) AND (COMP <= 0) DO COMMENT CONTINUE UNLESS
 IT IS ESTABLISHED THAT MAXMAT > NEW MATRIX;
       BEGIN
       FIRSTSAME:=TRUE;
       J:=J0:=J0+K; K:=0;
       ELT1:=INCMAT(I,1);
       WHILE FIRSTSAME DO COMMENT COPY SECOND
        COLUMN OF A BLOCK OF INCMAT TO RELABELLED;
              BEGIN
              K:=K+1;
              RELABELLED(K):=CROSSREF(INCMAT(I,2));
              COMMENT CONTINUE TILL FIRST ELEMENT OF COPIED
               EDGE CHANGES;
              FIRSTSAME:=(NEXTEDGE(I)¬=-1) AND
                         (INCMAT(NEXTEDGE(I),1) = ELT1);
              I:=NEXTEDGE(I);
              END;
       SORT;
       IF COMP=0 THEN COMPARE;
       IF COMP<0 THEN COMMENT NEW MATRIX > MAXMAT SO REPLACE
        THE LATTER;
```

```
          FOR M:=J UNTIL J0+K-1 DO
                BEGIN
                MAXMAT(M,1):=CROSSREF(ELT1);
                MAXMAT(M,2):=RELABELLED(M-J0+1);
                END;
          END;
     IF LEAFTRACE THEN
          BEGIN
          WRITE("LEAF OF SEARCH TREE - LABEL PERMUTATION IS:");
          FOR I:=1 UNTIL NODES DO
                BEGIN IF I REM 12 = 1 THEN IOCONTROL(2);
                WRITECN(PRIVEC(I));
                END;
          END;
     IF COMP<0 THEN
          BEGIN COMMENT UPDATE PERMUTATION;
          FOR I:=1 UNTIL NODES DO PERMUTATION(I):=PRIVEC(I);
          IF TRACE THEN WRITE("MAXMAT UPDATED");
          END
     ELSE IF TRACE THEN WRITE("MAXMAT NOT UPDATED");
     IF TRACE THEN WRITE(" ");
     LEAVES:=LEAVES+1;
     END UPDATE_RCMIM;

PROCEDURE UPDATE_CCMIM;
 COMMENT COMPARE THE INCIDENCE MATRIX OBTAINED BY NEW LABELLING TO
 THE (CCMIM) MAXIMAL MATRIX FOUND SO FAR (STORED IN MAXMATCOL)
 AND REPLACE THE LATTER IF NECESSARY;
     BEGIN
     COMMENT BECAUSE OF THE ACTION OF PULLUP, THE FIRST COLUMN
      OF INCMAT IS ARRANGED IN THE DESIRED (LINKED) ORDER WITH
      THE LABEL PERMUTATION GIVEN BY CROSSREF;
     COMMENT IN THIS PROCEDURE THE TERM "BLOCK" IS USED TO MEAN
      A SECTION OF INCMAT WITH ALL ELEMENTS OF THE FIRST COLUMN
      EQUAL, AND THE RELATION "MAXMAT>NEW MATRIX" IS USED TO MEAN
      MAXMAT IS BETTER THAN THE NEW MATRIX;
     INTEGER ARRAY RELABELLED(1::MAXVAL);
     COMMENT RELABELLED IS USED FOR SORTING THE SECOND COLUMN OF
      THE CURRENT BLOCK OF INCMAT BEING EXAMINED;
     INTEGER I,J0,K; COMMENT I,J0 POINT TO INCMAT,
      AND K TO RELABELLED;
     INTEGER ELT1,REL1,LASTCOMPARED,COMP;
     COMMENT ELT1 IS THE ELEMENT IN THE FIRST COLUMN OF THE
      CURRENT BLOCK OF INCMAT BEING EXAMINED, AND REL1 IS ITS
      RELABELLED VALUE.  LASTCOMPARED IS THE LAST COLUMN OF THE
      MATRIX COMPARED SO FAR;
     COMMENT COMP IS SET POSITIVE IF MAXMAT>NEW MATRIX
      AND NEGATIVE IF MAXMAT<NEW MATRIX;
     REFERENCE(INCEDGE) ARRAY INCMATCOL,COLTAIL(1::NODES);
      COMMENT INCMATCOL IS FOR THE HEADS OF THE (TEMPORARY)
      LIST REPRESENTATION OF INCMAT AND COLTAIL IS FOR THE TAILS;

     PROCEDURE SORT;
      COMMENT SORT RELABELLED FROM 1 TO K IN ASCENDING ORDER BY
      STRAIGHT INSERTION SORT;
       FOR L:=2 UNTIL K DO
           BEGIN INTEGER KEY,I;
           I:=L-1;
           KEY:=RELABELLED(L);
           WHILE (I > 0) AND (KEY < RELABELLED(I)) DO
```

```
            BEGIN
            RELABELLED(I+1):=RELABELLED(I);
            I:=I-1;
            END;
        RELABELLED(I+1):=KEY;
        END;

PROCEDURE TRANSREL1;
 COMMENT EXPAND THE LIST OF INCMATCOL CORRESPONDING TO
 COLUMN REL1;
  FOR L:=1 UNTIL K DO
   COLTAIL(REL1):=NEXTONE(COLTAIL(REL1)):=
   INCEDGE(JC+L-1,NULL);

PROCEDURE TRANSRELABELLED;
 COMMENT EXPAND THE LISTS CORRESPONDING TO THE COLUMNS
 SPECIFIED IN RELABELLED;
  FOR L:=1 UNTIL K DO
   COLTAIL(RELABELLED(L)):=
   NEXTONE(COLTAIL(RELABELLED(L))):=
   INCEDGE(JC+L-1,NULL);

PROCEDURE COMPARE;
 COMMENT COMPARE COLUMNS LASTCOMPARED+1 THROUGH REL1 OF
 THE TWO LIST STRUCTURES INCMATCOL AND MAXMATCOL.
 COMP IS SET POSITIVE IF MAXMATCOL IS BIGGER THAN
 INCMATCOL, I.E. MAXMAT>NEW MATRIX, AND NEGATIVE IF
 IT IS SMALLER;
        BEGIN
        REFERENCE(INCEDGE) P1,P2;
        INTEGER Q;
        LOGICAL ENDCOLUMN;
        Q:=LASTCOMPARED;
        WHILE (COMP=0) AND (Q<REL1) DO
            BEGIN
            Q:=Q+1;
            P1:=MAXMATCOL(Q);
            P2:=NEXTONE(INCMATCOL(Q));
            ENDCOLUMN:=FALSE;
            WHILE (COMP=0) AND ¬ENDCOLUMN DO
                BEGIN
                IF P1=NULL THEN
                    BEGIN
                    IF P2=NULL THEN ENDCOLUMN:=TRUE
                    ELSE COMP:=-1
                    END
                ELSE IF P2=NULL THEN COMP:=1
                ELSE
                    BEGIN
                    COMP:=EDGENO(P2)-EDGENO(P1);
                    P1:=NEXTONE(P1); P2:=NEXTONE(P2);
                    END;
                END;
            END;
        LASTCOMPARED:=REL1;
        END COMPARE;

LOGICAL FIRSTSAME;
COMP:=0;
LASTCOMPARED:=0;
```

```
I:=NEXTELGE(I );
JO:=1; K:=0;
FOR L: = A UNTIL NODES DO INCMATCOL(L):=COLTAIL(L):=
 INCEDGE(0,NULL):   COMMENT DUMMY RECOR 3;
WHILE (I¬=-1) AND (COMP <= 0) DO COMMENT CONT INUE UNLESS
  II 1  5 ESTABLISHED THAT MAXMAT > NEW M A T R I X ;
     BEGIN
     FIRSTSAME:=TRUE;
     JO:=JO+K; K:=0;
     ELT1:=INCMAT(I,1);
     REL1:=CROSSREF(ELT1);
     WHILE FIRSTSAME DO COMMENT COPY SECOND
      COLUMN OF  A  BLOCK OF INCMAT TO RELABELLED:
          BEGIN
          K:=K+1;
          RELABELLED(K):=CROSSREF(INCMAT(I,2));
          COMMENT CONTINUE TILL F I R S T ELEMENT OF COPIED
           EDGE C HANGES;
          FIRSTSAME:=(NEXTEDGE(I)¬=-1)A N D
                          (INCMAT(NEXTEDGE(I),1) = ELT1);
          I:=NEXTEDGE(I);
          END;
     TRANSREL1;
     IF COMP=0 THEN COMPARE;
    ~IF   COMP<=0   THEN
          BEGIN
          SORT;
          TRANSRELABELL EI):
          END;
     END;
I fLEAFTRACE THEN
     BEGIN
     WRITE("LEAF OF SEARCH TREE -LABEL PERMUTATION IS:"):
     FOR I:=1 UNTIL NODES D  O
          BEGIN I  F I REM 1  2 =  1 THEN IOCONTROL(2);
          WRITECN(PRIVEC(I));
          END;
     END;
I F COMP<0 THEN
     BEGIN COMMENT UPDATE PERMUTATION;
     FOR I:=1 UNTIL NODES DO PERMUTATION(I):=PRIVEC(I);
     COMMENT UPDATE MAXMATCOL;
     FOR I:=1 UNTIL NODES DO
      MAXMATCOL(I):=NEXTONE(INCMATCOL(I));
      COMMENT DROP DUMMY RECORD;
     IF TRACE THEN WRITE("MAXMAT UPDATED");
     END
ELSE IF TRACE THEN WRI TE("MAXMAT NOT UPDATED");
IF TRACE THEN WRITE(" ");
LEAVES:=LEAVES+1;
END UPDATE_COMIM;

PROCEDURE TRANSLATE_MAXMAT;
  COMMENT TRANSLAT E THE (COMIM) L IST REPRESENTAT ION
  MAXMATCOL I NTO THE ARRAY MAXMAT;
   FOR I:=1 UNTIL NODES DO
        BEGIN
        REFERENCE( INCEDGE) P;
        P:=MAXMATCOL(I);
        WHILE P¬= NULL DO
```

```
            BEGIN
            IF MAXMAT(EDGENO(P),1)=NODES+1 THEN
             MAXMAT(EDGENO(P),1):=I
             ELSE MAXMAT(EDGENO(P),2):=I;
            P:=NEXTONE(P);
            END;
        END;

PROCEDURE PRINT (REFERENCE(BLOCK) VALUE B,E; INTEGER VALUE L);
 COMMENT PRINT OUT THE DATA STRUCTURE CONTENTS IF DEBUG IS SET;
 IF DEBUG THEN
        BEGIN
        WRITE("BLOCK LIST IS:");
        WHILE B¬=NULL DO
            BEGIN
            WRITEON(BLOCKPTR(B));
            B:=NEXTBLOCK(B);
            END;
        WRITE("EMPTYBLOCK IS:");
        IF EMPTYBLOCK=NULL THEN WRITEON("NULL ") ELSE
         WRITEON(BLOCKPTR(E));
        WRITE("PRIVEC IS:");
        FOR I:=1 UNTIL NODES DO WRITEON(PRIVEC(I));
        WRITE("CROSSREF IS:");
        FOR I:=1 UNTIL NODES DO WRITEON(CROSSREF(I));
        WRITE("NEXTEDGES:");
        FOR I:=0 UNTIL EDGES DO WRITEON(NEXTEDGE(I));
        WRITE("LASTLABELLED=",L);
        WRITE(" ");
        END;

INTEGER MAXVAL; COMMENT MAXIMUM VALENCE OF A NODE;
COMMENT INITIALIZE DATA STRUCTURES;
FOR I:=1 UNTIL EDGES DO FOR J:=1 UNTIL 2 DO MAXMAT(I,J):=NODES+1;
IF ¬ROMIM THEN FOR I:=1 UNTIL NODES DO MAXMATCOL(I):=NULL;
LEAVES:=0;
COMMENT USE NEXTEDGE TO MAKE INCMAT A LINKED LIST;
FOR I:=0 UNTIL EDGES-1 DO NEXTEDGE(I):=I+1;
NEXTEDGE(EDGES):=-1;
LASTLABELLED:=0;
FOR I:=1 UNTIL NODES DO PRIVEC(I):=CROSSREF(I):=VALVEC(I):=0;
COMMENT CALCULATE VALENCES OF NODES;
MAXVAL:=0;
FOR I:=1 UNTIL EDGES DO FOR J:=1 UNTIL 2 DO
        BEGIN
        VALVEC(INCMAT(I,J)):=VALVEC(INCMAT(I,J))+1;
        IF VALVEC(INCMAT(I,J)) > MAXVAL THEN MAXVAL:=VALVEC(
         INCMAT(I,J));
        END;
COMMENT INITIALIZE BLOCK LIST TO A LIST OF TWO BLOCKS, THE FIRST
 CONTAINING A NODE WITH THE HIGHEST VALENCE AND THE SECOND THE
 EMPTY BLOCK;
EMPTYBLOCK:=BLOCK(2,NULL); BLOCKLIST:=BLOCK(1,EMPTYBLOCK);
COMMENT CHOOSE FOR THE FIRST LABELLED NODE EACH OF THE NODES WITH
 THE HIGHEST VALENCE IN TURN;
FOR I:=1 UNTIL NODES DO IF VALVEC(I) = MAXVAL THEN
        BEGIN
        PRIVEC(1):=I;
        CROSSREF(I):=1;
        CHOOSE(BLOCKLIST,EMPTYBLOCK,LASTLABELLED);
```

```
              COMMENT RESTORE CROSSREF;  CROSSREF(I):=0;
              END;
         IF ¬ROMIM THEN TRANSLATE_MAXMAT;
         END KLAPPERSLANGE;

 INTEGER NODES,EDGES,LEAVES;
 INTEGER ARRAY INCMAT,MAXMAT(1::100,1::2);
 INTEGER ARRAY PERMUTATION(1::100);
 LOGICAL DEBUG,TRACE,LEAFTRACE,ROMIM;
 INTFIELDSIZE:=3;
 LEAFTRACE:=FALSE;
 TRACE:=FALSE;
 DEBUG:=FALSE;
 READ(NODES,EDGES,ROMIM);
 WHILE NODES>0 DO
        BEGIN
        IOCONTROL(3);
        IF ROMIM THEN WRITE("* * ROMIM * *") ELSE WRITE("* * COMIM * *");
        WRITE("NUMBER OF NODES =",NODES,"     NUMBER OF EDGES =",
         EDGES);
        FOR I:=1 UNTIL EDGES DO FOR J:=1 UNTIL 2 DO READON(INCMAT(I,J));
        WRITE("INCIDENCE MATRIX IS:");
        FOR I:=1 UNTIL EDGES DO WRITE(INCMAT(I,1),INCMAT(I,2));
        KLAPPERSLANGE(INCMAT,MAXMAT,PERMUTATION,NODES,EDGES,ROMIM,LEAVES);
        WRITE("MAXIMAL MATRIX IS:");
        FOR I:=1 UNTIL EDGES DO WRITE(MAXMAT(I,1),MAXMAT(I,2));
        WRITE("LABEL PERMUTATION IS :");
        FOR I:=1 UNTIL NODES DO
              BEGIN IF I REM 12 = 1 THEN IOCONTROL(2);
              WRITEON(PERMUTATION(I));
              END;
        WRITE("NUMBER OF LEAVES IN SEARCH TREE =",LEAVES);
        READ(NODES,EDGES,ROMIM);
        END;
 END.
```

Appendix B

Sample Runs.

The examples given below are not intended to serve the purpose of a systematic analysis of the program's performance. However they illustrate the difference in efficiency of the <u>romim</u> and <u>comim</u> searches. An attempt at a more systematic analysis of the basic search algorithm, by means of "random graphs", is given in [1]

Maximal incidence matrices were computed for several graphs. Graph 1 is the example used throughout Section 5 (see Figure 5.1 (a)); a trace of the program flow is shown for this example only. Graph 2 was mentioned in Section 2 as having unequal <u>romim</u> and <u>comim</u> (see Figure 2.1). Graph 3 represents the structure of an electrical filter, which indicates a possible application. Graphs 6, 7 and 8 are subgraphs of Graph 9, which is the largest graph we have tested and has arbitrarily chosen edges.

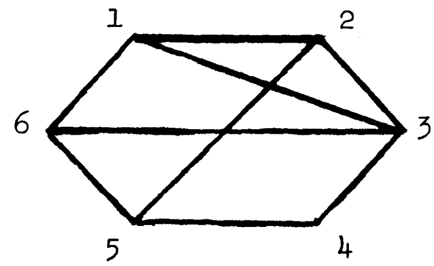
The results of the computer runs follow.

```
* * ROMIM * *
NUMBER OF NODES =   6          NUMBER OF EDGES =   9
INCIDENCE MATRIX IS:
   3     6
   1     2
   1     3
   1     6
   2     3
   2     5
   3     4
   4     5
   5     6
ENTER CHOOSE WITH CHOSEN NODE =   3
ENTER PRETEND WITH   4     PRETENDERS
ENTER CHOOSE WITH CHOSEN NODE =   6
ENTER CHOOSE WITH CHOSEN NODE =   1
ENTER CHOOSE WITH CHOSEN NODE =   2
ENTER CHOOSE WITH CHOSEN NODE =   4
ENTER CHOOSE WITH CHOSEN NODE =   5
LEAF OF SEARCH TREE - LABEL PERMUTATION IS:
   3     6     1     2     4     5
MAXMAT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
ENTER CHOOSE WITH CHOSEN NODE =   1
ENTER PRETEND WITH   2     PRETENDERS
ENTER CHOOSE WITH CHOSEN NODE =   6
ENTER CHOOSE WITH CHOSEN NODE =   2
ENTER CHOOSE WITH CHOSEN NODE =   4
ENTER CHOOSE WITH CHOSEN NODE =   5
LEAF OF SEARCH TREE - LABEL PERMUTATION IS:
   3     1     6     2     4     5
MAXMAT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
ENTER CHOOSE WITH CHOSEN NODE = 2
ENTER CHOOSE WITH CHOSEN NODE =   6
ENTER CHOOSE WITH CHOSEN NODE =   4
ENTER CHOOSE WITH CHOSEN NODE =   5
LEAF OF SEARCH TREE - LABEL PERMUTATION IS:
   3     1     2     6     4     3
MAXMAT NOT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT PRETEND
EXIT CHOOSE
ENTER CHOOSE WITH CHOSEN NODE =   2
ENTER CHOOSE WITH CHOSEN NODE =   1
ENTER CHOOSE WITH CHOSEN NODE =   6
ENTER CHOOSE WITH CHOSEN NODE =   4
```



```
ENTER CHOOSE WITH CHOSEN NODE =  5
LEAF OF SEARCH TREE - LABEL PERMUTATION IS:
   3    2    1    6    4    5
MAXMAT NOT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
ENTER CHOOSE WITH CHOSEN NODE =  4
ENTER PRETEND WITH  3    PRETENDERS
ENTER CHOOSE WITH CHOSEN NODE =  1
ENTER PRETEND WITH  2    PRETENDERS
ENTER CHOOSE WITH CHOSEN NODE =  6
ENTER CHOOSE WITH CHOSEN NODE =  2
ENTER CHOOSE WITH CHOSEN NODE =  5
LEAF OF SEARCH TREE - LABEL PERMUTATION IS:
   3    4    1    6    2    5
MAXMAT NOT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
ENTER CHOOSE WITH CHOSEN NODE =  2
ENTER CHOOSE WITH CHOSEN NODE =  6
ENTER CHOOSE WITH CHOSEN NODE =  5
LEAF OF SEARCH TREE - LABEL PERMUTATION IS:
   3    4    1    2    6    5
MAXMAT NOT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT PRETEND
EXIT CHOOSE
ENTER CHOOSE WITH CHOSEN NODE =  6
ENTER CHOOSE WITH CHOSEN NODE =  1
ENTER CHOOSE WITH CHOSEN NODE =  2
ENTER CHOOSE WITH CHOSEN NODE =  5
LEAF OF SEARCH TREE - LABEL PERMUTATION IS:
   3    4    6    1    2    5
MAXMAT NOT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
ENTER CHOOSE WITH CHOSEN NODE =  2
ENTER CHOOSE WITH CHOSEN NODE =  1
ENTER CHOOSE WITH CHOSEN NODE =  6
ENTER CHOOSE WITH CHOSEN NODE =  5
LEAF OF SEARCH TREE - LABEL PERMUTATION IS:
   3    4    2    1    6    5
MAXMAT NOT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
```

```
EXIT PRETEND
EXIT CHOOSE
EXIT PRETEND
EXIT CHOOSE
MAXIMAL MATRIX IS:
   1     2
   1     3
   1     4
   1     5
   2     3
   2     4
   3     6
   4     6
   5     6
LABEL PERMUTATION IS :
   3     1     6     2     4     5
NUMBER OF LEAVES IN SEARCH TREE =   8

000.03 SECONDS IN EXECUTION
```

```
* * COMIM * *
NUMBER OF NODES =   6        NUMBER OF EDGES =   9
INCIDENCE MATRIX IS:
  3    6
  1    2
  1    3
  1    6
  2    3
  2    5
  3    4
  4    5
  5    6
ENTER CHOOSE WITH CHOSEN NODE =   3
ENTER PRETEND WITH   4    PRETENDERS
VALENCE CHECK:   3    PRETENDER(S) TO BE CONSIDERED
ENTER CHOOSE WITH CHOSEN NODE =   6
ENTER CHOOSE WITH CHOSEN NODE =   1
ENTER CHOOSE WITH CHOSEN NODE =   2
ENTER CHOOSE WITH CHOSEN NODE =   4
ENTER CHOOSE WITH CHOSEN NODE =   5
LEAF OF SEARCH TREE -- LABEL PERMUTATION IS:
  3    6    1    2    4    5
MAXMAT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
ENTER CHOOSE WITH CHOSEN NODE =   1
ENTER PRETEND WITH   2    PRETENDERS
VALENCE CHECK:   2    PRETENDER(S) TO BE CONSIDERED
ENTER CHOOSE WITH CHOSEN NODE =   6
ENTER CHOOSE WITH CHOSEN NODE =   2
ENTER CHOOSE WITH CHOSEN NODE =   4
ENTER CHOOSE WITH CHOSEN NODE =   5
LEAF OF SEARCH TREE -- LABEL PERMUTATION IS:
  3    1    6    2    4    5
MAXMAT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
ENTER CHOOSE WITH CHOSEN NODE =   2
ENTER CHOOSE WITH CHOSEN NODE =   6
ENTER CHOOSE WITH CHOSEN NODE =   4
ENTER CHOOSE WITH CHOSEN NODE =   5
LEAF OF SEARCH TREE - LABEL PERMUTATION IS:
  3    1    2    6    4    5
MAXMAT NOT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT PRETEND
EXIT CHOOSE
ENTER CHOOSE WITH CHOSEN NODE =   2
ENTER CHOOSE WITH CHOSEN NODE =   1
    E
```

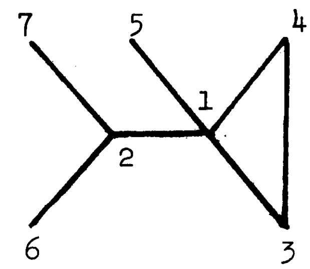
```
ENTER CHOOSE WITH CHOSEN NODE =   6
ENTER CHOOSE WITH CHOSEN NODE =   4
ENTER CHOOSE WITH CHOSEN NODE =   5
LEAF OF SEARCH TREE - LABEL PERMUTATION IS:
  3    2    1    6    4    5
MAXMAT NOT UPDATED

EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT CHOOSE
EXIT PRETEND
EXIT CHOOSE
MAXIMAL MATRIX IS:
  1    2
  1    3
  1    4
  1    5
  2    3
  2    4
  3    6
  4    4
  5    6
LABEL PERMUTATION IS  :
  3    1    6    2    4    5
NUMBER OF LEAVES IN SEARCH TREE =    4

000.02 SECONDS IN EXECUTION
```

```
* * ROMIM * *
NUMBER OF NODES =  7        NUMBER OF EDGES =  7
INCIDENCE MATRIX IS:
   1    2
   1    3
   1    4
   1    5
   2    6
   2    7
   3    4
MAXIMAL MATRIX IS :
   1    2
   1    3
   1    4
   1    5
   2    3
   4    6
   4    7
LABEL PERMUTATION IS :
   1    3    4    2    5    6    7
NUMBER OF LEAVES IN SEARCH TREE = 24

000.02 SECONDS IN EXECUTION
```

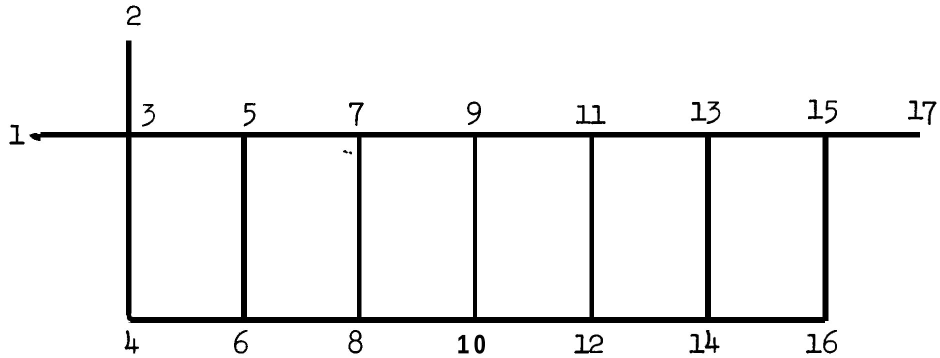


```
* * CUMIM * *
NUMBER OF NODES =  7          NUMBER OF EDGES =  7
INCIDENCE MATRIX  S:
  1     2
  1     3
  1     4
  1     5
  2     6
  2     7
  3     4
MAXIMAL MATRIX IS:
  1     2
  1     3
  1     4
  1     5
  2     6
  2     7
  3     4
LABEL PERMUIATION IS :
  1     2     3     4     5     6     7
NUMBER OF LEAVES IN SEARCH TREE =  4

000.01 SECONDS IN EXECUTION
```

```
* * ROMIM * *
NUMBER OF NODES  =  17          NUMBER OF EDGES  =  22
INCIDENCE MATRIX IS:
   1     3
   2     3
   3     4
   3     5
   4     6
   5     6
   5     7
   6     8
   7     8
   7     9
   8    10
   9    10
   9    11
  10    12
  11    13
  12    14
  13    14
  13    15
  14    16
  15    16
  15    17
  11    12
MAXIMAL MATRIX IS :
   1     2
   1     3
   1     4
   1     5
   2     6
   2     7
   3     6
   6     8
   7     8
   7     9
   8    10
   9    10
   9    11
  10    12
  11    12
  11    13
  12    14
  13    14
  13    15
  14    16
  15    16
  15    17
LABEL  PERMUTATION  IS :
   3     5     4     2     1     6     7     8     9    10    11    12
  13    14    15    16    17
NUMBER OF LEAVES IN SEARCH TREE = 2    4

000.12 SECONDS IN EXECUTION
```



```
* * COMIM * *
NUMBER OF NODES = 17          NUMBER OF EDGES = 22
INCIDENCE MATRIX IS:
   1     3
   2     3
   3     4
   3     5
   4     6
   5     6
   5     7
   6     8
   7     8
   7     9
   8    10
   9    10
   9    11
  10    12
  11    13
  12    14
  13    14
  13    15
  14    16
  15    16
  15    17
  11    12
MAXIMAL MATRIX IS:
   1     2
   1     3
   1     4
   1     5
   2     6
   2     7
   3     6
   6     8
   7     8
   7     9
   8    10
   9    10
   9    11
  10    12
  11    12
  11    13
  12    14
  13    14
  13    15
  14    16
  15    16
  15    17
LABEL PERMUTATION IS :
   3     5     4     2     1     6     7     8     9    10    11    12
  13    14    15    16    17
NUMBER OF LEAVES IN SEARCH TREE =   2

000.02 SECONDS IN EXECUTICN
```

```
* * ROMIN * *
NUMBER OF NODES = 15        NUMBER OF EDGES = 24
INCIDENCE MATRIX IS:
   1    2
   1   10
   1   12
   2    3
   3    4
   3   15
   4    5
   4   15
   5    6
   6    7
   7    8
   7   16
   8    9
   9   10
  10   11
  10   12
  11   12
  12   13
  12   18
  13   14
  14   15
  18   17
  17   16
  16    8
MAXIMAL MATRIX IS:
   1    2
   1    3
   1    4
   1    5
   1    6
   2    3
   2    4
   2    7
   3    8
   5    9
   6   10
   7   11
   8   12
   9   13
  10   14
  11   13
  11   15
  12   14
  12   16
  13   15
  14   16
  15   17
  16   18
  17   18
LABEL PERMUTATION IS :
  12   10    1   11   18   13    9    2   17   14    8    3
  16   15    7    4    6    5
NUMBER OF LEAVES IN SEARCH TREE = 24

000.12 SECONDS IN EXECUTION
```

* * COMIM * *

NUMBER OF NODES = 18   NUMBER OF EDGES = 24

INCIDENCE MATRIX IS:

1	2
1	10
1	12
2	3
3	4
3	15
4	5
4	15
5	6
6	7
7	8
7	16
8	9
9	10
10	11
10	12
11	12
12	13
12	16
13	14
14	15
16	17
17	18
18	8

MAXIMAL MATRIX IS:

1	2
1	3
1	4
1	5
1	6
2	3
2	4
2	7
3	8
5	9
6	10
7	11
8	12
9	13
10	14
11	13
11	15
12	14
12	16
13	15
14	16
15	17
16	18
17	18

LABEL PERMUTATION IS :

12 10 1 11 18 13 9 2 17 14 8 3
16 15 7 4 6 5

NUMBER OF LEAVES IN SEARCH TREE = 2

000.03 SECONDS IN EXECUTION

* * [illegible] * *
NUMBER OF NODES = 19   NUMBER OF EDGES = 29
INCIDENCE MATRIX IS:

1	3
1	6
2	3
3	4
4	5
5	6
6	7
7	2
3	10
3	11
4	11
4	12
5	15
5	16
6	14
9	18
9	19
8	9
9	10
10	11
11	12
12	13
13	14
14	15
15	16
16	17
17	18
18	19
19	8

MAXIMAL MATRIX IS:

1	2
1	3
1	4
1	5
1	6
2	3
2	4
2	7
3	7
3	8
4	9
5	10
6	11
7	12
8	10
8	13
8	14
9	15
9	16
9	17
10	11
10	18
12	18
13	14
13	18
14	19
15	16

```
 15    17
 16    19
LABEL PERMUTATION IS :
  3    11    4    10    1    2    12    5    9    6    7    13
 15    16    19   18    8    14   17
NUMBER OF LEAVES IN SEARCH TREE =144

000.34 SECONDS IN EXECUTION
```

* * COMP 4 * *

NUMBER OF NODES = 19 NUMBER OF EDGES = 29

INCIDENCE MATRIX IS:

1	5
1	6
2	3
3	4
4	5
5	6
6	7
7	8
8	10
3	11
4	11
4	12
5	13
5	16
8	14
9	13
9	14
8	9
9	10
10	11
11	12
12	13
13	14
14	15
15	16
16	17
17	18
18	19
19	8

MAXIMAL MATRIX IS:

1	2
1	3
1	4
1	5
1	6
2	3
2	4
2	7
3	7
3	8
4	9
5	10
6	11
7	12
8	13
8	13
8	14
9	15
9	16
9	17
10	11
10	18
12	13
13	14
13	16
14	19
15	16

```
 15    19
 16    17
LABEL PERMUTATION IS :
   3    11     4    10     1     2    12     5     9     6     7    13
  15    16    18    19     8    14    17
NUMBER OF LEAVES IN SEARCH TREE = 16

 00.10 SECONDS IN EXECUTION
```

```
* * DOMIN * *
NUMBER OF NODES = 22        NUMBER OF EDGES = 30
INCIDENCE MATRIX IS:
   1     2
   2     3
   3     4
   4     5
   3     5
   3     6
   6     7
   4     7
   1     4
   1     8
   8     9
   9    10
  10    11
  11    12
  12    13
  13    14
   7    14
   2     5
   3    14
  14    15
  15    16
  16    17
  10    17
   3    11
   3    15
   4    18
   4    19
  19    20
  20    21
   7    22
 MAXIMAL MATRIX IS:
   1     2
   1     3
   1     4
   1     5
   1     6
   1     7
   1     8
   2     3
   2     4
   3     9
   3    10
   3    11
   3    12
   4     9
   5     6
   5    10
   5    13
   6    14
   7    10
   8    15
   8    16
   9    17
  10    18
  11    19
  13    15
  14    20
```

```
 16    20
 16    21
 17    21
 19    22
LABEL PERMUTATION IS :
  3     5     4     2    14    15     6    11     1     7    19    18
 13    16    12    10     8    22    20    17     9    21
NUMBER OF LEAVES IN SEARCH-TREE =576

002.13 SECONDS IN EXECUTION
```

```
* * COMIM * *
NUMBER OF NODES = 22        NUMBER OF EDGES = 30
INCIDENCE MATRIX IS:
   1     2
   2     3
   3     4
   4     5
   3     5
   3     6
   6     7
   4     7
   1     4
   1     8
   8     9
   9    10
  10    11
  11    12
  12    13
  13    14
   7    14
   2     5
   3    14
  14    15
  15    16
  16    17
  10    17
   3    11
   3    15
   4    18
   4    19
  19    20
  20    21
   7    22
MAXIMAL MATRIX IS:
   1     2
   1     3
   1     4
   1     5
   1     6
   1     7
   1     8
   2     3
   2     9
   2    10
   2    11
   2    12
   3     4
   4     9
   5     6
   5    10
   5    13
   6    14
   7    15
   7    16
   8    10
   9    17
  10    18
  11    19
  13    15
  14    20
```

```
16    20
16    21
17    21
19    22
LABEL PERMUTATION IS :
  3     4     5     2    14    15    11     6     A     7    19    18
 13    16    12    10     8    22    20    17     9    21
NUMBER OF LEAVES IN SEARCH TREE = 1

000.03 SECONDS IN EXECUTION
```

* * NORTH * *

NUMBER OF NODES = 23 NUMBER OF EDGES = 31

INCIDENCE MATRIX IS:

1	2
2	3
3	4
4	5
3	5
3	6
6	7
4	7
1	4
1	8
6	9
9	10
10	11
11	12
12	13
13	14
7	14
2	5
3	14
14	15
15	16
15	17
16	17
3	11
3	15
4	18
4	19
19	20
20	21
7	22
7	23

MAXIMAL MATRIX IS:

1	2
1	3
1	4
1	5
1	6
1	7
1	8
2	3
2	4
3	9
3	1
3	11
3	12
4	9
5	6
5	10
5	13
6	14
7	10
8	15
8	16
9	17
10	18
10	19
11	20

```
 13    15
 14    21
 16    21
 16    22
 17    22
 20    23
LABEL PERMUTATION IS  :
  3     5     4     2    14    15     6    11     1     7    19    18
 13    16    12    10     8    23    22    20    17     9    21
NUMBER OF LEAVES IN SEARCH TREE =1152

303.10 SECONDS IN EXECUTION
```

* * CCMIN * *

NUMBER OF NODES = 23   NUMBER OF EDGES = 31

INCIDENCE MATRIX IS:

1	2
2	3
3	4
4	5
3	5
3	6
6	7
4	7
1	4
1	8
a	9
9	10
10	11
11	12
12	13
13	14
7	14
2	5
3	14
14	15
15	16
16	17
10	17
3	11
7	15
4	18
4	19
19	20
30	21
7	22
7	33

MAXIMAL MATRIX IS:

1	2
1	3
1	4
1	5
1	6
1	7
1	8
7	3
2	9
2	10
2	11
2	13
3	4
4	9
5	6
5	10
5	13
6	14
7	15
7	16
8	10
9	17
10	18
10	19
11	20

```
 13   15
 14   21
 16   21
 16   22
 17   22
 20   23
LABEL PERMUTATION IS :
  3    4    5    2   14   15   11    6    1    7   19   18
 13   16   12   10    8   22   23   20   17    9   21
NUMBER OF LEAVES IN SEARCH TREE =    2

000.04 SECONDS IN EXECUTION
```

```
* * RCMIM * *
NUMBER OF NODES = 24          NUMBER OF EDGES = 32
INCIDENCE MATRIX IS:
  1    2
  2    3
  3    4
  4    5
  3    5
  3    6
  6    7
  4    7
  1    4
  1    8
  8    9
  9   10
 10   11
 11   12
 12   13
 13   14
  7   14
  2    5
  3   14
 14   15
 15   16
 16   17
 10   17
  3   11
  3   15
  4   18
  4   19
 19   20
 20   21
  7   22
  7   23
  7   24
MAXIMAL MATRIX IS:
  1    2
  1    3
  1    4
  1    5
  1    6
  1    7
  1    8
  2    3
  2    4
  3    9
  3   10
  3   11
  3   12
  4    9
  5    6
  5   10
  5   13
  6   14
  7   10
  8   15
  8   16
  9   17
 10   18
 10   19
```

```
10    20
11    21
13    15
14    22
16    22
16    23
17    23
21    24
LABEL PERMUTATION IS :
  3     5     4     2    14    15     6    11     1     7    19    18
 13    16    12    10     8    24    23    22    20    17     9    21
NUMBER OF LEAVES IN SEARCH TREE =3456

007.03 SECONDS IN EXECUTION
```

* * COMIN * *

NUMBER OF NODES = 24   NUMBER OF EDGES = 32

INCIDENCE MATRIX IS:

1	2
2	3
3	4
4	5
3	5
3	6
6	7
4	7
1	4
1	8
8	3
9	10
10	11
11	12
12	13
13	14
7	14
2	5
3	14
14	15
15	16
16	17
10	17
3	11
3	15
4	18
4	19
19	20
20	31
7	22
7	33
7	34

MAXIMAL MATRIX IS:

1	2
1	3
1	4
1	5
1	6
1	7
1	8
2	3
2	9
2	10
2	11
2	12
3	4
4	9
5	6
5	10
5	13
6	14
7	15
7	16
8	10
9	17
10	18
10	19

```
10    20
11    21
13    15
14    22
16    22
16    23
17    23
21    24
LABEL PERMUTATION IS :                        6
13    16    12     2    14    15    11    24     1     7    19    18
NUMBER OF LEAVES10     8    22    23                20    17     9    21
                  IN SEARCH TREE =     6

000.06 SECONDS IN EXECUTION
```

* * COMIM * *
NUMBER OF NODES = 50   NUMBER OF EDGES = 78
INCIDENCE MATRIX IS:

1	2
2	3
3	4
4	5
3	5
3	6
6	7
4	7
1	4
1	8
8	9
9	10
10	11
11	12
12	13
13	14
7	14
2	5
3	14
14	15
15	16
16	17
10	17
3	11
3	15
4	18
4	19
19	20
20	21
7	22
7	23
7	24
7	25
7	26
7	27
8	28
9	29
10	30
30	31
29	32
28	33
27	34
26	35
26	34
26	33
25	32
24	31
22	30
35	36
36	37
36	38
38	39
39	40
11	41
12	42
11	43
12	44

11	45
12	46
13	47
14	48
15	49
9	48
8	47
7	46
6	45
48	49
1	50
37	48
36	47
25	46
21	42
11	19
12	20
23	25
23	26
23	27
25	27

MAXIMAL MATRIX IS:

1	2
1	3
1	4
1	5
1	6
1	7
1	8
1	9
1	10
1	11
2	12
2	13
2	14
2	15
2	16
3	4
3	5
3	6
3	17
4	5
4	7
5	18
6	19
7	18
7	20
7	21
8	12
8	22
8	23
8	24
9	12
9	25
10	26
11	27
12	13
12	22
12	28
12	29

```
 13    29
 14    28
 14    30
 14    31
 15    29
 15    32
 17    33
 19    23
 19    29
 19    32
 19    34
 19    35
 20    36
 21    37
 22    38
 22    39
 23    40
 24    38
 24    41
 24    42
 25    29
 26    27
 26    43
 29    43
 29    44
 29    45
 30    36
 30    40
 30    41
 32    46
 33    41
 34    46
 37    40
 37    42
 37    47
 39    48
 41    43
 43    49
 47    49
 49    50
LABEL PERMUTATION IS :
  7     4    25    23    27    46    26    14     6    22    24     3
  5     1    19    18    32    34    12    33    35    15    13    48
 45    30    31     2    11     8    50    20    29    42    44    28
 36    49    16    47     9    37    10    41    43    21    38    17
 39    40
NUMBER OF LEAVES IN SEARCH TREE = 48

000.81 SECONDS IN EXECUTION
```