Chapter 7

A Computer Implementation of the Blossom Algorithm

In this chapter we discuss a computer implementation of the blossom algorithm we described in Chapter 3. The program was written in PL/1; the reader is assumed to have some knowledge of this programming language. (See [I2] for the language specifications). The design of the program was influenced somewhat by BLOSSOM I (Edmonds, Johnson, Lockhart [E7]), a FORTRAN implementation of a generalization of the blossom algorithm. A special acknowledgement is due to Professor Ellis L. Johnson, who has contributed to both the design and the details of this computer code.

In the next three sections we describe the data structure used and discuss the way the program handles such problems as manipulating trees and blossoms and shrinking subgraphs. Following this we discuss the code itself and in the last section of the chapter we discuss storage requirements and experimental results obtained concerning the algorithm. The program itself is listed in the Appendix.

Throughout the chapter, we refer to statements in the program by means of the PL/1 statement numbers. T and F are bit strings of length one having the values '1'B and '0'B respectively and are used as logical constants having the values "true" or "false".

7.1. Storage of the Graph.

NEDGE and NNODE are binary full words that hold the number of edges and nodes respectively of the graph G. They

do not change throughout the execution of the program, in particular they do not reflect the shrinking of subsets of nodes or the creation of pseudonodes. The edge set of the graph is the set of integers 1, 2,...,NEDGE; the node set of the graph is a set of NNODE pointer variables which point to the structures holding the information about the nodes.

The graph is represented by an array of edges. EDGES (Statement 4) is an array of NEDGE structures which contain the following information for each edge J.

C(J) is a single precision floating point variable which holds the current "reduced cost". That is, it holds the value $c_J - y(\psi(J)) - y(Q^0(J))$ where c_J is the cost assigned to edge J and y is the current dual solution. Determining the equality subgraph and computing the bound for a dual variable change are facilitated by having this value stored. Initially C(J) should simply be the cost of the edge J; the program (Statements 333 to 346) subtracts the value of the initial dual solution while initializing.

X(J) is a binary halfword that holds the current value of the matching for the edge J.

STATUS(J) is a set of 16 one bit switches available for recording the status of edge J. Only four are used by the algorithm, they are:

EQ(J) = T or F according as J does or does not belong to the current equality subgraph;

SHRNK (J) = T or F according as J has or has not been shrunk in forming a pseudonode;

FRST(J) = T or F according as J does or does not belong to the alternating forest or to some component of $G^+(X)$;

ZER(J) allows the edge J to be omitted from consideration during execution of the program. Any edge J for which ZER(J) = T will be completely ignored, any edge J for which ZER(J) = F will be processed normally. This feature is intended to facilitate processing of subgraphs of the graph G.

ENDS(J, *) and ORIGENDS(J, *) are arrays consisting of two pointers. ORIGENDS holds pointers representing the nodes of G with which J is incident and does not change throughout the execution of the algorithm. ENDS reflects any pseudonodes that have been formed. Thus, where R is the nested family of sets described in Chapter 3, if $J \in E(G \times R)$ then ENDS holds pointers to the nodes of $G \times R$ with which J is incident. If $J \notin E(G \times R)$ then the pointers in ENDS point to the pseudonode corresponding to the minimal member of R which contains $\psi(J)$.

The variables for the real nodes of the graph are stored in an array NODELST (Statement 3). However they are referred to by means of the based structure NODE (Statement 6). Handling the nodes in this way simplifies the treatment of pseudonodes while at the same time allows the algorithm to be as economical with storage as possible.

For each node P, real or pseudo, we have the following values.

P -> DEF is a binary halfword holding the deficiency of the current matching at the node P, that is, it holds the value $b_P - x(\delta(P))$ where b_P is the degree constraint of P and x is the current matching. If P is contained in a pseudonode, this value may be too large or too small by 1 however this situation is corrected when we expand the pseudonode or correct the matching within it.

P -> STATUS is a set of 16 one bit switches which reflect the status of node P. Nine of these are actually used.

P -> REAL = T or F according as P is a real node or a pseudonode.

P -> CONSTEQ = T if the degree constraint for node P is an equation, P -> CONSTEQ = F if the degree constraint for node P is an inequality. Thus P -> CONSTEQ = T or F according as $P \in V^{=}$ or $V^{\leq}$.

P -> DEFIC = T or F according as P does or does not belong to the alternating forest.

P -> ODD = T if P is an odd node of the alternating forest, otherwise P -> ODD = F.

P -> YRTO = T if P belongs to the alternating forest and the tree containing P is rooted at a real node $i \in V^{\leq}$ for which $y_i = 0$ or at a pseudonode containing a node $i \in V^{\leq}$ for which $y_i = 0$.

P -> BLOS = T if the components of $G^{+}(x)$ containing P contains an odd polygon, otherwise it is false.

P -> DCHNG, P -> INPATH and P -> EXPANDED are all used by the algorithm and will be discussed later.

P -> Y is a single precision floating point variable used to hold the current dual variable of the node P. If P is a pseudonode, then it holds the dual variable of the subset of the nodes of G which form the pseudonode.

P -> TREE, P -> EDGEDN and P -> STACKUP are used for representation of the trees and blossoms of the algorithm and their use is described in the next two sections.

7.2. Tree Handling.

The manipulation of trees and forests is an important part of the blossom algorithm. There are three properties which we wish our data structure which represents trees to satisfy. First it should provide an easy means of finding the path in the tree from any node of the tree to the root, second it should provide a reasonable means of examining all the nodes and edges of a tree and third it should make convenient such operations as rerooting trees, growing trees and removing portions of trees. The structure used is the "triply linked tree" developed by Johnson [J2]. A description of this structure also appears in Knuth [K3], p. 352.

We are actually storing a planar representation of the tree. We think of a tree being rooted "at the bottom" and consisting of various "levels" of nodes according to their distance from the root (see Figure 7.1).

For any node P of the tree other than the root, P -> DN is the node adjacent with P in the level immediately below P, P -> EDGEDN is the edge of the tree joining P and P -> EDGEDN. If P is the root of a tree then P -> DN = NULL and P -> EDGEDN = 0.

P -> UP is the leftmost node adjacent with P in the tree belonging to the level of the tree immediately above the level containing P, if such a node exists. Otherwise P -> UP = NULL.

P -> RT is the first node Q to the right of P in the level of the tree containing P which satisfies P -> DN = Q -> DN. If no such node Q exists, then P -> RT = NULL. Observe that if P is the root of a tree then P -> RT = NULL.

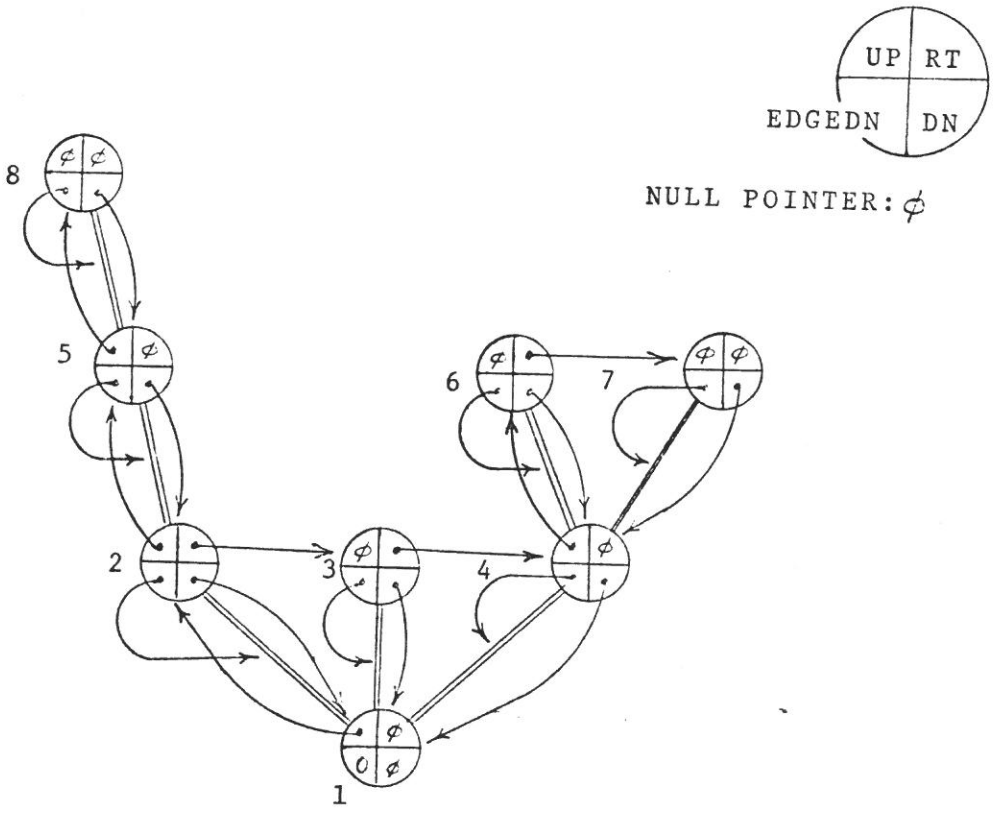


Figure 7.1 Triply Linked Tree

We now describe some of the procedures used by the program in manipulating trees.

ADDON(Q1, Q2, J) (Statements 195-202) uses the edge J which joins nodes Q1 and Q2 to attach a tree rooted at Q2 to the tree containing Q1. It also sets FRST(J) = T to indicate that J is now an edge of the forest.

REMOVE(P1) (Statements 158-175) does the following. P1 is a node belonging to some tree, REMOVE removes P1 and the portion of the subtree above P1 from the tree, thereby creating a new tree rooted at P1. If P1 is already a root, it simply returns having done nothing. Otherwise it finds the other pointers equal to P1 and modifies them appropriately. It sets P1 -> DN = NULL and sets FRST(P1 -> EDGEDN) = F, indicating this edge is no longer part of the forest.

REROOT(P1) (Statements 176-194) reroots the tree containing P1 at P1. This it does by travelling down the path in the tree from P1 to the root, successively removing the portion of the tree above each node in the path and adding that portion to the portion previously removed.

UPSCAN(P1, UPCALL, SUBRUB, DNCALL, SUBRDN) (Statements 203-234) is a routine which scans through all the nodes of the tree containing P1 which are above P1. These nodes are scanned according to the following rule: UPSCAN always tries to move up the tree; if it cannot do this, it tries to go to the right and then continue moving up; if it cannot

do this it goes down and then tries to go to the right. For example, it would encounter the nodes of the tree of Figure 7.1 in the following order: 1, 2, 5, 8, 5, 2, 3, 4, 6, 7, 4, 1. UPCALL and DNCALL are one bit strings, if UPCALL = T then the first time each node is encountered, UPCALL calls the procedure SUBRUP passing it a pointer to the node. If DNCALL = T then the last time each node is encountered the procedure SUBRDN is called and passed as a parameter a pointer pointing to the node.

Thus depending on the procedures SUBRUP and SUBRDN, UPSCAN can perform a great many functions. The procedures described in Statements 235-324 are all used by means of UPSCAN. We describe the purpose of these procedures in Section 7.5 when describing the main procedure.

The final procedure we discuss in this section is more than just a tree manipulating subroutine. It performs augmentations and at the same time (optionlly) helps construct the new alternating forest.

AUGMENT(P1,R1,DELTAX,DESTROY,ODDB) (Statements 40-59) alternately subtracts and add DELTAX to the value of X(Q1 -> EDGEDN) for each node Q1 ≠ R1 in the path from P1 to R1 in the tree containing these nodes. The pointers Q1 -> DN are used to trace down the path. If ODDB (BIT(1)) equals F then the procedure starts with a subtraction, if T then it starts with an addition.

If DESTROY (BIT(1)) = T then everytime an edge becomes

zero, the portion of the tree above that edge is removed and broken into nonzero components. This is done by using UPSCAN, passing it the procedure NONDEFIX which is called the last time each node is encountered. If DESTROY = F then none of this is done.

NONDEFIX(P1) (Statements 250-258) updates the indicators for the node P1. If there is an edge J down from P1 in the tree such that X(J) = 0 then P1 is removed from the tree.

7.3. Blossoms, Shrinking and Pseudonodes.

One of the central problems encountered in implementing the blossom algorithm is the problem of shrinking. It has even been suggested (Balinski B[1] p. 232) that the amount of storage required to handle this process would make computer implementation of the blossom algorithm impractical. As was shown by BLOSSOM I and as is shown again by the program of this chapter, such is not the case. An upper bound on the amount of storage required to hold all the information necessary for whatever amount of shrinking is done by the algorithm is only slightly greater than half the amount of storage used to store the information required for the real nodes; in practise we generally require considerably less.

A blossom consists of a special type of alternating tree together with an edge J which forms an odd polygon; this is how it is stored. There is one node R in a

blossom at which the current matching restricted to the edges of the blossom is deficient, the tree is rooted at this node. Since R is the root of a tree, we normally have R -> EDGEDN = 0. When representing a blossom we let R -> EDGEDN = J. Thus storing a blossom is no more difficult than storing a tree.

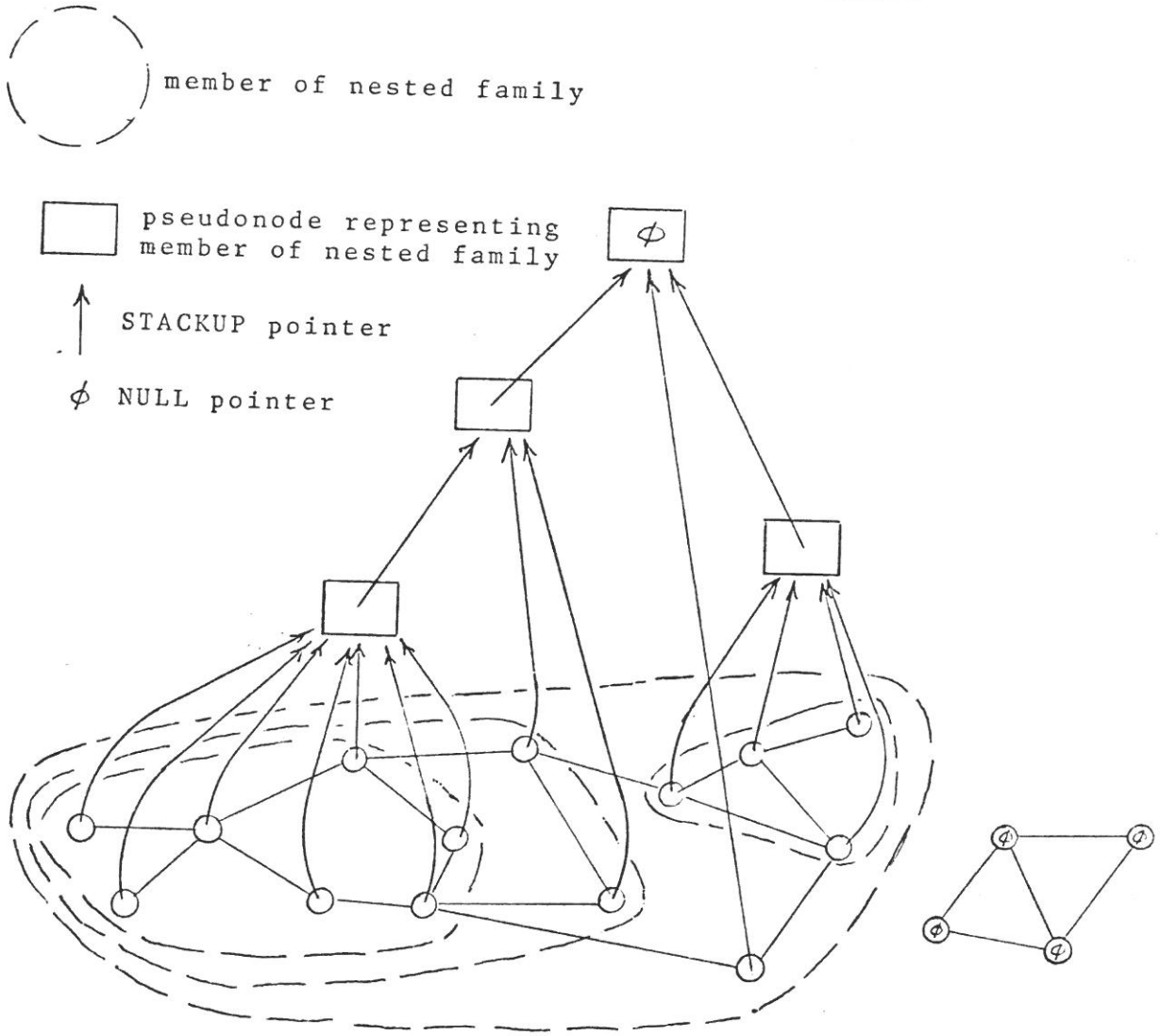
(Components of $G^+(X)$ containing an odd polygon are also stored in this fashion, the only difference being that the root of these components is not deficient. P -> BLOS=T for every node P in such a component.)

Now we describe the way in which the nested family of shrinkable sets is represented. For each member P of the nested family we have allocated (in Statements 508-516) a structure P -> PSEUDO. The first seven words of P -> PSEUDO are used in the same way as the seven words of NODE are used. (The maximal members of the noted family are the current pseudo nodes.) However there is in addition an eighth word P -> ROOT which is the root of the blossom associated with P, that is, the node at which the matching restricted to the edges of the blossom is deficient.

For any real node P1, P1 -> STACKUP is a pointer to the structure associated with the minimal member of the nested family containing P1 if such a set exists, otherwise P1 -> STACKUP = NULL. For any member P of the nested family, P -> STACKUP is a pointer to the structure associated with the minimal member of the family properly containing P,

if such a set exists, otherwise P -> STACKUP = NULL. (See Figure 7.2).

Figure 7.2 Nested Family Representation



Frequently we wish to know the maximal member of the nested family containing a node P, that is, the current pseudonode containing P.

SURF(P) (Statements 13-25) returns the value of the pointer corresponding to the maximal member of the nested family containing P, if such a set exists. If no such set

exists it simply returns P. In Statements 17-22 it searches up the chain of STACKUP pointers starting with P -> STACKUP until a null pointer is found. It uses PNEST to count the number of members of the nested family which contain P. This value is not used by the algorithm itself, but the maximum "depth of nesting" is stored in RUNSTAT(3) to provide one indication of the amount of work done by the algorithm.

We now describe the operations performed by the program in shrinking a blossom (Statements 506-558). Figure 3.7 may help to clarify this process. J is an edge joining nodes P1 and P2 which are both even nodes (or in some cases odd nodes) of the same alternating tree. R3 is the first common node of the paths in the tree from P1 and P2 to the root of the tree. R2 is the last node belonging to the blossom in the path in the tree from R3 to the root of the tree. (These nodes have been determined earlier in the program.) We call the path from R3 to R2 the <u>stem</u> of the blossom. The blossom consists of the polygon plus the stem plus any components of $G^{+}(X)$ containing a node of the polygon or stem.

In Statements 508-516 we allocate the pseudonode P for the blossom and initialize most of its variables. In Statements 519-534 we "mark" the polygon and the stem of the blossom by letting P3 - > INPATH = T. This is to make it possible to identify all the nodes of the blossom. Then

in Statement 537 we call UPSCAN, passing it the routines UPBLOSS and DNBLOSS. These routines (Statements 296-329) do two things. UPBLOSS set P1 -> STACKUP = P for every node of the blossom. DNBLOSS removes any portions of the tree above the blossom and adjoins them to P. These routines rely on the order in which UPSCAN scans the nodes. SHRNKNG is a one bit switch which is T or F according as the next node to be scanned can or cannot be expected to be part of the blossom. Thus whenever UPBLOSS detects a node P1 not in the polygon or stem for which X(P1->EDGEDN)=0, PX is set equal to P1 and SHRNKNG is set equal to F. From then on nothing is done to successive nodes until DNBLOSS is passed the node PX. Then the subtree rooted at PX is removed from the blossom and adjoined to P, SHRNKNG is set T and the process continues.

If SHRNKNG=T then when UPBLOSS is passed a node of the polygon or stem or a node P1 for which X(P1 -> EDGEDN) $\neq$ 0 it sets P1 -> STACKUP = P, indicating that this node is part of the blossom.

When UPSCAN has completed its scan the program removes the blossom from the tree and replaces it with the pseudonode P, together with any portions of the tree that DNBLOSS may have adjoined to P (Statements 538-543). Finally, R2 -> EDGEDN is set equal to J and the blossom is represented completely.

Now all that remains to be done is to update ENDS to reflect the new pseudonode. This is done in Statements 545-556. At the same time SHRNK(J1) is set equal to T

for any edge such that ENDS(J1,1) and ENDS(J1,2) are nodes of the blossom.

7.4. Parameters Passed and Returned.

In this section we describe the parameters passed and returned when using this code. It should be pointed out that this program is designed to be used by other programs as a subroutine and consequently there is no provision for card input or printer output (except for the trace feature). Thus unlike BLOSSOM 1 this program requires a suitable driver program to prepare its data and output its results if we simply want to solve matching problems.

Parameters Passed.

*NEDGE (BINARY FIXED (16)) holds the number of edges of the graph.

*NNODE (BINARY FIXED (16)) holds the number of nodes of the graph.

*NODELST is an array of NNODE structures having the format described in NODE (Statement 6). Each structure is seven words long and holds the following information. Let $I \in \{1,2, \ldots, NNODE\}$ and let P = ADDR(NODELST(I)).

- P -> DEF(BINARY FIXED (15)) holds the degree constraint of the node P.

- P -> CONSTEQ(BIT(1)) should be T or F according as the constraint at node P is an equation or an inequality.

- P -> REAL, P -> DEFIC (BIT(1)) should be set equal to T initially.

- P -> DCHNG, YRTO, INPATH, EXPANDED, ODD and BLOS (BIT(1)) should initially all be set equal to F.

- P -> Y (DECIMAL FLOAT (SHORT)) holds the initial dual variable of the node. This dual solution must be feasible. If P -> Y is set equal to half the absolute value of the largest edge cost for every node P then this starting dual solution is feasible.

- P -> TREE.UP,RT,DN and STACKUP(POINTER) should all be set equal to NULL.

- P -> EDGEDN (BIN FIXED(16) should be initially zero.

*EDGES is an array of NEDGE structures, each six full words long holding the following information. Let J ε {1,2,..., NEDGE}.

- C(J) (DECIMAL FLOAT (SHORT)) is the cost of edge J (not the "reduced cost; this is computed by the algorithm).

- X(J) (BINARY FIXED(15)) should be set to zero.

- ZER(J), EQ(J), SHRNK(J) and FRST(J) should all be initially set to F.

- ENDS(J,*), ORIGENDS(J,*) (POINTER)should be the nodes of the graph with which J is incident.

* RUNSTAT (Statement 5) is an array of 10 binary halfwords. The only entry used for input is RUNSTAT(10). A trace of the execution of the program is or is not printed out according as RUNSTAT(10) = 1 or 0. This trace, if obtained, prints a message concerning each edge used by the algorithm together with the values of the matching and the dual solution any time they are changed.

The main use of RUNSTAT is to return statistics concerning the problem solved to the program which invoked BLOSSOM.

These input specifications were based upon the assumption that we were using the zero matching as a starting solution.

Parameters Returned.

The parameters are returned in the following state. Let P be any node, real or pseudo.

- P -> DEF is the deficiency of the current matching X at the node P.

- P -> STATUS is set to reflect the status of P at termination.

- P -> TREE, EDGEDN holds the tree and blossom structure of the final solution.

- P -> STACKUP points to the pseudonode representing the minimal member of the nested family containing P, if such a set exists, otherwise it is null.

- P -> Y is the final dual variable of the node P if P is real, or the final dual variable of the corresponding member of the nested family if P is a pseudo node.

Notice that the invoking program is returned both the optimal dual solution and the final nested family. (This was desired in Chapter 6.)

Let J be any edge of the graph.

- C(J) is the final reduced cost of the edge J.

- X(J) is the maximum matching, that is, the answer to the problem.

- STATUS(J) reflects the status of edge J at termination.

- ENDS(J,*),ORIGENDS(J,*) are both as they were originally, the nodes of the graph met by J.

*RUNSTAT (BINARY FIXED (15)) (Statement 5) is an array of ten indicators showing the amount of work done in solving the problem. The values returned are as follows:

- RUNSTAT(1) is the number of dual variable changes;

- RUNSTAT(2) is the number of times a blossom was shrunk;

- RUNSTAT(3) is the deepest nest of pseudonodes formed (or equivalently, the longest chain of STACKUP pointers);

- RUNSTAT(4) is the number of times pseudonodes were expanded (in Step 9e of the blossom algorithm);

- RUNSTAT(5) is the number of times the forest was grown without making an augmentation (Steps 3a and 7 of the blossom algorithm);

- RUNSTAT(6) is the number of two tree augmentations (Step 4 of the blossom algorithm);

- RUNSTAT(7) is the number of one tree augmentations (Step 5b of the blossom algorithm);

- RUNSTAT(8) is the number of times a component of $G^{+}(X)$ containing an odd polygon was added to the forest (Step 3c of the blossom algorithm);

- RUNSTAT(9) is the number of so called "pseudo augmentations", augmentations which move a deficiency to a node i of $V^{\leq}$ for which $y_i = 0$ (Step 7a of the blossom algorithm);

- RUNSTAT(10) is returned with the value zero or one according as the matching returned is or is not feasible, if RUNSTAT(10) = 1 when returned then the algorithm terminated with a Hungarian forest.

7.5. The Main Procedure.

Now we describe the main procedure itself. The code follows fairly closely the description of the blossom algorithm given in Section 3.8.

Statements 325-349 are for initialization, reduced costs are computed and EQ(J) is set for each edge J reflecting whether or not the edge belongs to the equality subgraph. A procedure FN(J) (Statements 26-39) is used in computing reduced costs. It calculates the sum of the dual variables of the ends of J and of all members of the nested family which contain both ends of J.

Statements 350-634 constitute the "edge processing" loop of the program. JCNT is used to cycle through the edges. Anytime we finish considering an edge, whether or not we have been able to make use of it, we go to ENDA (Statement 633) where JCNT is set equal to 1 + MOD(JCNT,NEDGE).

Whenever we are able to use the edge JCNT (to augment, grow the forest or shrink), LASTJ is set equal to JCNT. If JCNT ever "cataches up" with LASTJ then we have made a complete cycle through the edges without having been able to do anything so we go to Statement 636 and attempt to change the dual variables.

Statements 350-372 test each edge JCNT to see if it belongs to the equality subgraph, has not been shrunk, is not in the forest and meets an even node P1 of the alternating forest F^1. If JCNT violates any of these criteria we go to ENDA. If the other end P2 of JCNT is an odd node of F^1 then it is of no use to us and we go to ENDA, if P2 is an odd node of F^0 then we go to ODDGROW (Statement 581). Otherwise we set J=JCNT and go to POLYSTEP(Statement 566), GROWSTEP(Statement 559) or DXCALC(Statement 382) depending on the status of P2.

DXCALC:(Statement 382) J joins even nodes P1 and P2 of the forest F, in Statements 382-400 we determine whether they belong to the same or to different trees. At the same time we compute D1 and D2, bounds on the amount of augmentation that can be made. INPATH is used to mark the nodes in the paths from P1 and P2 to the roots of their respective trees. If P1 and P2 belong to different trees then R1 and R2 are the roots of the two trees. If P1 and P2 belong to the same tree then R1 is the root of the tree and R2 is the first common node of the two paths.

If P1 and P2 belong to different trees, then we go to TWOTREE (Statement 401) where we perform the augmentation described in Step 4b of the blossom algorithm compute the new forest and go to ENDA.

If P1 and P2 belong to the same tree then we go to ONETREE (Statement 427). There we determine whether or not an augmentation is possible. If not we go to DEFBLOSS

(Statement 506) where we shrink. If we can make an augmentation we do so, update the tree and then update the forest. At this point we may have to shrink, if so we go to DEFBLOSS where we do so. We may have created a component of $G^+(X)$ containing an odd polygon and no deficient node. If so (Statements 497-505) we find the root R2, store the polygon forming edge J as R2 -> EDGEDN and call UPSCAN passing it the procedure BLOSSIND. BLOSSIND (Statements 259-264) simply sets the node STATUS indicators correctly.

We have already discussed the shrinking procedure in Section 7.3.

GROWSTEP:(Statements 559-565) J joins an even node P1 of the forest to a node P2 not in the forest. We simply grow the forest. ADDFIX (Statements 235-244) sets the STATUS indicators for the nodes added to the forest. (This corresponds to Step 3b of the blossom algorithm.)

POLYSTEP:(Statements 566-580) J joins an even node P1 of the forest to a node P2 of a component of $G^+(X)$ which contains an odd polygon. First we find the root of this component and hence the polygon forming edge J1. Then we add this component (minus J1) to the forest just as in GROWSTEP. Then we replace J with J1, P1 and P2 with the ends of J1 and go to DXCALC(Statement 382). (This corresponds to Step 3c of the blossom algorithm.)

ODDGROW:(Statements 581-632). Edge J meets an even node P1 of F^1 and an odd node P2 of F^0. Statements 585-597 add a suitable portion of the tree containing P2 to the tree containing P1. SETYRTO is a procedure used by UPSCAN

to set P -> YRTO = YROOTO for all nodes scanned. Thus we first set YROOTO correctly. (This corresponds to Step 7 of the blossom algorithm.)

We may now have a tree in the forest containing two deficient nodes P1 and P2. If this is the case, we make a so called "pseudo augmentation" to remedy this (Statements 599-632). These steps also update the forest. (This portion of the code corresponds to Step 7a of the blossom algorithm.)

This completes the description of the main edge processing loop. If we make a complete cycle through the edges without being able to make use of any edge then we go to DUALCHNGE(Statement 636) where we attempt to change the dual variables. FAIL is a one bit switch which is used to indicate whether or not we have an optimal feasible matching. Initially FAIL=F, if we discover a node in a tree of F^1 then FAIL is set equal to T.

In Statements 637-665 we compute EPS2, a bound imposed by the nodes on the amount of dual change that can be made. (EPS2 equals the minimum of ε_3, ε_4 of Step 9a of the blossom algorithm.)

If FAIL=F, thus the current matching is feasible, we go to CORRECTION(Statement 925) where we correct the matching in the pseudo nodes. If EPS2=0 then we need make no dual variable change; we go to NODEBND(Statement 786) where we either reroot a tree or expand an odd pseudonode of the forest.

Otherwise (Statements 670-705) we compute EPS1, the

bound on the amount of dual change determined by the dual constraints corresponding to the edges. (ESP1 equals the minimum of ϵ_1, ϵ_2 of Step 9a of the blossom algorithm.) Then we let EPS = MIN(EPS1, EPS2). If EPS = 10^{10} (infinity for our purposes) then the forest is Hungarian, no feasible matching exists, we go to CORRECTION and terminate. Otherwise (Statements 706-780) we make a change of dual variables and update the reduced costs accordingly. (For any pseudo node P., P -> DCHNG is used to ensure that we only change its dual variable once.) If the bound on the dual change was imposed by a constraint corresponding to an edge JX then we can now immediately make use of the edge; we set JCNT equal JX and return to the start of the edge processing loop. (Statement 350).

If the bound on the dual change was imposed by a constraint corresponding to a real node PX of the forest, then we now have PX -> Y = 0. If PX is the root of the tree, we simply reset YRTO for the nodes of the tree. Otherwise we go to AUG(Statement 599) and make a pseudo augmentation. This process corresponds roughly to Step 9d of the blossom algorithm, although in the computer code we do not insist that all trees of F^1 rooted at nodes $i \in V^{\leq}$ for which $y_i = 0$ be moved to F^0, we simply handle one each time.

If the bound on the dual change was imposed by a constraint corresponding to a pseudonode PX, then PX -> Y = 0 and PX is an odd pseudonode of the forest that has to be

expanded. This we do in Statements 797-924.

The first thing done is to call EXPAND, a procedure (Statements 60-145) that first updates ENDS so as to no longer reflect the existence of pseudonode P and then "corrects" the matching within the blossom corresponding to P so that it is compatible with the matching of the graph containing P. This procedure also forms the nucleus of the final matching correction step (corresponding to Step 12 of the blossom algorithm). Notice that for any calls to AUGMENT in EXPAND we have DESTROY = F, thus the blossom does not have its structure destroyed.

EXPAND set JIN equal to the edge J of the graph incident with P for which X(J) = 1 and sets BROOT equal to the node of the blossom met by JIN. If P -> EDGEDN = JIN then we have the easier case, JIN is the unique edge of the forest meeting P. This case is handled in Statements 803-842. Otherwise two edges of the forest meet P, this case is handled in Statements 843-924.

ADDBLOS,DEFFIX(Statements 265-295) are routines called by UPSCAN to "unshrink" a blossom and update the status indicators. Their operation is similar to that of UPBLOSS, DNBLOSS. Initially SHRNKNG(BIT(1)) = T. For each node P1 that ADDBLOS is passed, it sets P1 -> STACKUP = NULL, thereby removing the reference to the pseudonode. Then it proceeds, setting the status of each node to indicate that it belongs to the forest, until it finds a node P1 which would have become an even node of the forest for which

X(P1 EDGEDN) = 0. When this happens PX is set equal to P1 and SHRNKNG is set equal to F. From then on the status of each node encountered is set to indicate that the node does not belong to the forest. DEFFIX breaks the blossom up at edges J for which X(J) = 0. When DEFFIX is passed the node PX it sets SHRNKNG = T and the process continues.

The final part of the program is the step (Statements 925-945) where we correct the matching in the pseudo nodes prior to terminating. For each pseudonode P, P -> EXPANDED is used to ensure that we do not try to correct the matching for the pseudonode more than once.

This completes the description of the program.

7.6. Experimental Results

This program was compiled under version 5.2B of the OS/360 PL/1 F level compiler, OPT=1 and was tested on a large number of contrived graphs. Then a series of tests on "random graphs" was run to obtain the experimental results described here.

The random graph generator accepted as input the number of nodes and edges desired in the graph together with a range for the degree constraints and a range for the edge costs (integers were used for edge costs in these tests). It generated the graph by successively joining each node of the graph to some other node until sufficient edges had been created. Thus the test graphs had multiple edges but no loops. The random graph generator also accepted

a parameter specifying the desired probability of a node belonging to $\bar{V}$.

An option of the random graph generator was to create a file containing the information about each graph in a form suitable as input to BLOSSOM I, the earlier Fortran implementation of the matching algorithm. This enabled comparative tests to be run between the two programs.

The driver program then invoked BLOSSOM to solve the matching problem. Following this a test was made of the matching and dual solution returned by BLOSSOM to ensure that they were feasible solutions satisfying the complementary slackness conditions for optimality.

The results of these tests are listed in Table 7.1 They were run on an IBM/360 model 75 at the University of Waterloo. Thirty two graphs were run on both BLOSSOM I and the code described here, two random graphs were generated with each set of specifications. We required the degree constraint be satisfied with equality at each node. In addition, six "large" graphs were run on the code of this chapter.

One of the most striking observations is that even though the value of the edge costs do not enter into our theoretical bound, the number of different edge costs drastically affects the run time of the code. The reason for this seems to be that the more different edge costs we have, the more dual variable changes that have to be done to obtain an optimal solution, and dual variable changes are practically (although not theoretically) time consuming.

A second observation is that the number of pseudos formed during the course of execution of the code tends to be relatively small. The entries in the table give the total number formed during the execution of the code, the number present at termination is often considerably smaller.

The BLOSSOM program of this chapter does run faster than BLOSSOM I (The ratio of its execution time to that of BLOSSOM I seems to decrease as the number of edges of the graph increases). This is not surprising, however, for BLOSSOM I treats directly a more general form of the matching problem than is treated by the code of this chapter. These more general problems can be reduced to problems solvable with the code of this chapter; however this involves significantly, though algebraically, increasing the number of edges and nodes.

The BLOSSOM procedure itself requires 33K bytes of storage. Storage of the graph requires $28 \times v + 24 \times e + 32 \times p$ bytes of storage, where v is the number of nodes, e is the number of edges and p is the maximum number of pseudonodes present at any one time in the execution. The various PL/1 library routines required to run BLOSSOM add to these storage estimates however. When run with the random graph generator and driver, the [100 node-1000 edge] graphs required 148K bytes of storage, the [1000 node-4000 edge] graphs required 238K bytes of storage.

Since the computer code uses fixed word arithmetic, it may not be able to solve a problem if the number of significant digits of some of the values used becomes too large.

The degree constraints and values of the matching are integers stored as binary half words and so can be no larger than 32767. The value X(J) for any edge J can never be larger than the smaller degree constraint of its ends, so as long as the degree constraints range from 1 to 32767 we will have no difficulty handling these values.

The edge costs and dual variables are stored as hexadecimal (base 16) floating point numbers having six significant hexadecimal digits. Any edge costs stored by the computer are rational numbers. If we multiply all edge costs by a positive constant we do not affect the solution set of the problem. Hence we can assume that the edge costs have been multiplied by a large enough number so that they are all integer. As was shown in the proof of (3.10.7) if our starting dual variables are integer valued then all dual variables computed during the execution of the algorithm will be integer or half integer valued. If the degree constraint of every node is an inequality (that is, $V^{=} = \phi$ and $V^{\leq} = V$) then all dual variables are nonnegative and so no dual variable needs to be larger than the largest edge cost. Thus if the edge costs are integers from the range - 1,048,576 to 1,048,576 then the dual variables will be integers and half integers from the same range. These numbers are represented exactly by six hexadecimal digits so we can be sure that the computer code will solve such problems.

In the case that $V^{=} \neq \phi$, and consequently some dual variables are allowed to become negative, we may in fact require dual variables considerably larger than the largest edge cost. Consequently the establishment of a bound on the magnitude of

the dual variables is more complicated. For an analysis of a situation of this sort, see Edmonds, Johnson, Lockhart [E7].

If higher precision were required for some problem it would be a straightforward matter to replace all binary half words with full words and all floating point numbers with double precision floating point numbers. Then degree constraints could range from 1 to 2,147,483,648 and if the edge costs were integers from the range -4×10^{15} to 4×10^{15} we could guarantee a correct solution.

No. of Nodes	No. of Edges	Range of b_i	Range of c_j	Elapsed Time	Blossom I Elapsed Time	No. of Shrinkings	No. of Dual Variable Change
30	150	1	1-1000	4.9, 6.9 sec.	6.5, 9.2 sec.	1,5	28,39
30	150	1	1-5	0.5, 0.7	1.1, 1.3	0,0	2,3
30	500	1-7	1-500	22.6,19.0	29.4,25.6	2,3	39,33
30	500	1-77	1-500	25.5,24.0	32.0,32.1	6,5	44,41
50	200	1-10	1-10	3.7, 4.6	5.5, 6.2	4,3	10,15
50	500	1-10	1-10	5.8, 5.8	12.0,11.2	5,5	6,7
50	200	1-100	1-10	3.2, 5.7	4.8, 7.4	0,5	10,18
50	200	1-10	1-9999	23.8,16.7	29.4,19.5	11,6	96,66
100	300	1-2	1-10	10.6, 8.1	24.4,23.7	27,19	16,16
100	1000	1-2	1-10	20.3, 8.8	54.7,15.9	26, 6	9,5
100	300	50-150	1-10	9.7,11.5	13.4,16.7	8,13	16,19
100	300	1-2	1-9999	50.5,45.5	62.3,53.8	11, 7	138,124
300	1500	1	1-10	38.1,21.6	65.9,39.8	11, 6	13,7
300	1500	7-77	1-10	44.0,51.3	102.6,112.4	15,13	18,12
300	1500	2	1-100	135.1,125.6	182.4,172.9	0,0	68,70
300	1500	100	1-10	20.5,18.4	36.8,31.9	0,0	8,7
** 500	5000	7	1-10	137.0,123.4		31,28	5,6
** 500	5000	1	1-10	84.2,177.4		36,61	3,5
*1000	4000	1-2	1-10	143.5,184.8		11,33	14,14

** Run with all nodes $i \in V^{\leq}$.
* Run with half nodes in $V^{\leq}$.

APPENDIX

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST
                    /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */
 1                  BLOSSOM: PROC(NEDGE,NNODE,NODELST,EDGES,RUNSTAT);

                    /*************************************************************
                     *************   VARIABLE DECLARATIONS   ******************
                     *************************************************************/
 2      1           DCL  (NEDGE,NNODE) BIN FIXED(16);
 3      1           DCL 1 NODELST(* /*NNODE*/),   /* ACTUAL STORAGE FOR NODE VARS */
                          2 FILL(7) BIN FIXED(16);
 4      1           DCL  1 EDGES(* /*NEDGE*/),
                           2 C FLOAT,
                           2 X BIN FIXED(15),
                           2 STATUS,
                                3 FILL BIT(12),
                               (3 ZER,
                                3 EQ,
                                3 SHRNK,
                                3 FRST) BIT(1),
                          (2 ENDS(2),
                           2 ORIGENDS(2)) PTR;
 5      1           DCL RUNSTAT(10) BIN FIXED (15);
                         /*   RUNSTAT(1)=NO.OF DUAL CHANGES,
                              RUNSTAT(2)=NO. OF SHRINKINGS,
                              RUNSTAT(3)=DEEPEST NEST OF PSEUDONODES FORMED,
                              RUNSTAT(4)=NO. OF EXPANSIONS,
                              RUNSTAT(5)=NO. OF TIMES FOREST GROWN,
                              RUNSTAT(6)=NO. OF TWO TREE AUGMENTATIONS,
                              RUNSTAT(7)=NO. OF ONE TREE AUGMENTATIONS,
                              RUNSTAT(8)=NO. OF TIMES POLYGON ADDED TO THE FOREST,
                              RUNSTAT(9)=NO. OF PSEUDO AUGMENTATIONS,
                              RUNSTAT(10)= 0 IF MATCHING IS FEASIBLE,
                                           1 IF MATCHING IS NOT FEASIBLE.
                              RUNSTAT(10) IS PASSED WITH VALUE 0 IF NO TRACE IS DESIRED,
                                          WITH VALUE 1 IF A TRACE IS REQUIRED.  */
 6      1           DCL  1 NODE BASED(P),
                           2 BASICS,
                                3 DEF BIN FIXED(15),
                                3 STATUS,
                                     4 FILL BIT( 7),
                                    (4 REAL,
                                     4 DCHNG,
                                     4 YRTO,
                                     4 INPATH,
                                     4 EXPANDED,
                                     4 CONSTEQ,
                                     4 ODD,
                                     4 DEFIC,
                                     4 BLOS)BIT(1),
                                3 Y FLOAT,
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

                                   3 TREE,
                                     (4 UP,
                                      4 RT,
                                      4 DN) PTR,
                                   3 EDGEDN BIN FIXED(16),
                                   3 STACKUP PTR;
  7     1          DCL  1 PSEUDO BASED(P),
                          2 BASICS(7) BIN FIXED (16),
                          2 ROOT PTR;
  8     1          DCL((EPS,EPS1,EPS2,Z) FLOAT, JX BIN FIXED(16)) STATIC;
  9     1          DCL SURF ENTRY RETURNS(PTR);
 10     1          DCL ((P,P1,P2,P3,R1,R2,R3,PX,BROOT,Q1,Q2,Q3)PTR,
                     (I,J,J1,J2,K,JCNT,LASTJ,JIN,JON) BIN FIXED(16),
                     DELTAX BIN FIXED(15),
                     (D1,D2,D3) BIN FIXED(15))STATIC;
 11     1          DCL  (T INIT ('1'B), F INIT ('0'B)) STATIC BIT (1),
                     (ODDB , POLYBIT,SHRNKNG,YROOT0,TRACE,FROMEX,NOCHECK,FAIL)
                       STATIC BIT(1);
 12     1          FMT:  FORMAT(SKIP,A,F(6),A); /* USED FOR TRACING */

                   /*****************************************************************
                    ***********  GENERAL PURPOSE SUBROUTINES  ***********************
                    *****************************************************************/
 13     1          SURF: PROC (P) RETURNS(PTR);
                        /* PROCEDURE TO FIND HIGHEST LEVEL PSEUDO NODE CONTAINING P.  */
                        /* PNEST IS USED TO COUNT STACK DEPTH. */
 14     2               DCL (P,P1 STATIC) PTR;
 15     2               DCL PNEST BIN FIXED(15) STATIC;

 16     2               PNEST=0;
 17     2               P1=P;
 18     2               DO WHILE (P1->STACKUP ¬=NULL);
 19     2  1                 P1=P1->STACKUP;
 20     2  1                 PNEST=PNEST+1;
 21     2  1                 END;
 22     2               IF PNEST > RUNSTAT(3) THEN RUNSTAT(3)=PNEST;
 24     2               RETURN(P1);
 25     2          END SURF;
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

 26     1          FN:PROC(EDGE);
                        /*  THIS PROCEDURE EVALUATES THE SUM OF THE DUAL VARIABLES
                            ON EACH END OF AN EDGE AND ON ODD SETS CONTAINING
                            THE EDGE.*/
 27     2              DCL (P1,P2) STATIC PTR, EDGE BIN FIXED(16), SUM STATIC FLOAT;

 28     2              P1=ORIGENDS(EDGE,1);
 29     2              P2=ORIGENDS(EDGE,2);
 30     2              SUM=P1->Y + P2->Y;
 31     2              IF ¬SHRINK(EDGE) THEN GO TO END;
 33     2
 34     2              P1=ENDS(EDGE,1);
 35     2    1         DO WHILE (P1¬=NULL);
 36     2    1              SUM=SUM+P1->Y;
 37     2    1              P1=P1->STACKUP;
 38     2                   END;
 39     2          END:RETURN(SUM);
                    END FN;

 40     1          AUGMENT:PROC(P1,R1,DELTAX,DESTROY,ODDB);
                        /*  THIS PROCEDURE AUGMENTS ALONG THE PATH FROM P1 TO THE ROOT R1
                            BY AMOUNT DELTAX.  IF DESTROY = T THEN THE TREE GETS BROKEN
                            UP AT EDGES J FOR WHICH THE NEW X(J) = 0,  IF DESTROY = F
                            THEN THIS DOES NOT HAPPEN.  WE START AUGMENTING WITH AN
                            ADDITION OR A SUBTRACTION DEPENDING ON WHETHER ODD = T OR F.*/
 41     2              DCL (P1,R1) PTR,DELTAX BIN FIXED(15),(DESTROY,ODDB) BIT(1),
                           (Q1,Q2) STATIC PTR;

 42     2              Q1=P1;
 43     2              DO WHILE (Q1¬=R1);
 44     2    1              Q1->INPATH=F;
 45     2    1              J1=Q1-> EDGEDN;
 46     2    1              Q2=Q1->DN;
 47     2    1              IF ODDB THEN X(J1)=X(J1)+DELTAX;
 49     2    1              ELSE DO;
 50     2    2                   X(J1)=X(J1)-DELTAX;
 51     2    2                   IF DESTROY THEN
 52     2    2                    IF X(J1)=0 THEN CALL UPSCAN(Q1,F,NONDEFIX,T,NONDEFIX);
                            /*  THIS REMOVES EVERYTHING ABOVE AND SPLITS TREE
                                      INTO ITS POSITIVE COMPONENTS.  */
 54     2    2                   END;
 55     2    1              Q1=Q2;
 56     2    1              ODDB=¬ODDB;
 57     2    1         END;
 58     2              R1->INPATH=F;
 59     2          END AUGMENT;
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

 60     1          EXPAND:PROC (P);
                     /* THIS PROCEDURE EXPANDS A PSEUDONODE.
                     SPECIFICALLY IT
                          1)  CORRECTS EDGE ENDS SO THAT THEY NO LONGER REFLECT
                              EXISTENCE OF PSEUDONODE
                          2)  AUGMENT SO THAT MATCHING CORRECT BUT STACKUP STILL
                              ACKNOWLEDGES PSEUDONODE */
 61     2          DCL P PTR, DESTROY BIT(1),((P1,P2) PTR, (I,J) BIN FIXED(16), IN BIT(1))
                     STATIC;

 62     2                   DESTROY=F;
                            /* CORRECT EDGE ENDS */
 63     2                   JIN=0;/*  JUST IN CASE THERE IS NO EDGE IN WITH X(J) =1. */
 64     2                   DO J=1 TO NEDGE;
 65     2     1                  IN=F; /*  INDICATES PARITY OF NO. OF EDGE ENDS IN P. */
 66     2     1                  IF ENDS(J,1)¬=P THEN GO TO P2TEST;
 68     2     1                  IN=T;
 69     2     1                  SHRNK(J)=F;
 70     2     1                  P1=ORIGENDS(J,1);
 71     2     1            LPP1: IF P1 -> STACKUP=P THEN DO;
 73     2     2                        ENDS (J,1)=P1; GO TO P2TEST;
 75     2     2                        END;
 76     2     1                  P1=P1 -> STACKUP; GO TO LPP1;
 78     2     1           P2TEST: IF ENDS (J,2) ¬=P THEN GO TO LPEND;
 80     2     1                  IN=¬IN;
 81     2     1                  P2=ORIGENDS (J,2);
 82     2     1            LPP2: IF P2 -> STACKUP=P THEN DO;
 84     2     2                        ENDS (J,2)=P2;
 85     2     2                        GO TO LPEND; END;
 87     2     1                  P2=P2->STACKUP;  GO TO LPP2;
 89     2     1           LPEND: IF IN THEN
 90     2     1                     IF X(J)=1 THEN /* THIS IS THE EDGE INTO THE BLOSSOM*/
 91     2     1                      JIN=J;
 92     2     1                  END;
 93     2                   IF JIN=0 THEN DO; /*CHECK FOR <= NODE WITH Y=0 */
 95     2     1                  DO I = 1 TO NNODE;
 96     2     2                        P1=ADDR(NODELST(I));
 97     2     2                        IF P1->CONSTEQ THEN GO TO ENDSCH;
 99     2     2                        IF P1->Y ¬= 0 THEN GO TO ENDSCH;
                                    /*  OTHERWISE WE SEE IF P1 IS CONTAINED IN P. */
101     2     2                      BROOT=P1;
102     2     2                      DO WHILE(BROOT->STACKUP ¬= NULL);
103     2     3                           IF P=BROOT->STACKUP THEN GO TO LAB7;
                                    /* BROOT IS A SUITABLE NODE FOR RECEIVING A DEFICIENCY*/
105     2     3                           BROOT=BROOT->STACKUP;
106     2     3                           END;
107     2     2            ENDSCH:END;
                            /* P CONTAINS NO <= NODE WITH Y=0, SO CURRENT MATCHING O.K.*/
108     2     1                  RETURN;
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

109     2    1          LAB7:R1=P->ROOT;
110     2    1               IF BROOT=R1 THEN RETURN; /* CURRENT MATCHING IS CORRECT */
112     2    1               R1->DEF=0;
113     2    1               BROOT->DEF=1;
114     2    1               P1=BROOT;
115     2    1               GO TO AGMNT; /* MATCHING CORRECTION SET UP */
116     2    1               END;
                        /* ONE END OF JIN HAS STACKUP=P, THE OTHER DOES NOT.  LET
                        P1 BE THAT NODE */
117     2               P1=ENDS (JIN,1); IF P1-> STACKUP=P THEN
119     2               P2=ENDS (JIN,2);
120     2               ELSE DO; P2=P1; P1=ENDS (JIN,2); END;
                        /* NOW P1 IS THE SURPLUS NODE */
124     2               BROOT=P1;            /* VARIABLES RETURNED TO BLOSSOM */
125     2               R1=P->ROOT;
126     2               R1->DEF=0; /* WE WILL CLEAR UP THIS DEFICIENCY */
127     2               IF P1 = R1 THEN RETURN;
129     2          AGMNT:
                        DELTAX=1;
130     2               ODDB=F; /* START WITH SUBTRACTION */
131     2               CALL AUGMENT (P1,R1,DELTAX,DESTROY,ODDB);
132     2                    IF¬ODDB THEN /* WE WENT CORRECT DIR'N AROUND POLYGON */
133     2                         RETURN;
134     2                    J=R1->EDGEDN;
135     2                    P1=ENDS (J,1);
136     2                    P2=ENDS (J,2);
137     2               ODDB=F; /* NORMAL CASE */
138     2               IF P1->ODD THEN ODDB=T; /* ABNORMAL CASE */
140     2               CALL AUGMENT (P1,R1,DELTAX,DESTROY,(ODDB));
141     2               CALL AUGMENT (P2,R1,DELTAX,DESTROY,(ODDB));
142     2               X(J)=X(J)+DELTAX;
143     2               IF ODDB THEN X(J)=X(J)-2*DELTAX; /* CORRECT A BAD GUESS */
145     2               END;

146     1          XOUT:PROC;  /* PRINTS CURRENT SOLUTION */
147     2               PUT EDIT('*BLOSS - CURRENT MATCHING :')(SKIP,A);
148     2               PUT EDIT(X)(SKIP,20 F(5));
149     2               END XOUT;

150     1          YOUT:PROC;   /* PRINTS CURRENT DUAL NODE VARS */
151     2               PUT EDIT('*BLOSS - CURRENT NODE DUAL VARIABLES :')(SKIP,A);
152     2               PUT SKIP;
153     2               DO IY = 1 TO NNODE;
154     2    1               P=ADDR(NODELST(IY));
155     2    1               PUT EDIT(P->Y)(F(10,2));
156     2    1               END;
157     2               END YOUT;
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

                    /*********************************************************************
                     **********  GENERAL TREE HANDLING ROUTINES **********************
                     *********************************************************************/

158     1           REMOVE: PROC (P1);
                         /*   SUBROUTINE TO REMOVE P1 FROM THE TREE CONTAINING IT.  */
159     2                DCL (P1,(P2,P3) STATIC) PTR;

160     2                P2=P1->DN;
161     2                IF P2=NULL THEN GO TO RET /* FOR P1 IS THE ROOT OF ITS TREE */;
163     2                P3=P2->UP;
164     2                IF P3=P1 THEN GO TO EASY;
166     2                DO WHILE (P3->RT ¬=P1);
167     2    1               P3=P3->RT;
168     2    1               END;
                         /*  NOW WE HAVE FOUND P1 */
169     2                P3->RT=P1->RT;
170     2                GO TO RET;
171     2           EASY:P2->UP=P1->RT;
172     2           RET: P1->RT=NULL;
173     2                P1->DN=NULL;
174     2                FRST(P1->EDGEDN)=F;
175     2           END REMOVE;

176     1           REROOT: PROC (P1);
                         /* SUBROUTINE WHICH REROOTS THE TREE CONTAINING P1 AT P1.  */
177     2                DCL   ((P2,P3,PX) PTR,(J,J3) BIN FIXED(16)) STATIC,  P1 PTR;

178     2                P2=P1->DN;
179     2                IF P2=NULL THEN RETURN /* FOR P1 IS ALREADY A ROOT. */;
181     2                J=P1->EDGEDN;
182     2                PX=P1;
183     2                CALL REMOVE (P1);
184     2           LP:  P3=P2->DN;
185     2                J3=P2->EDGEDN;
186     2                CALL REMOVE (P2);
187     2                CALL ADDON (PX,P2,J);
188     2                IF P3=NULL THEN RETURN /* FOR P2 WAS THE ROOT.  */;
190     2                PX=P2;
191     2                P2=P3;
192     2                J=J3;
193     2                GO TO LP;
194     2           END REROOT;
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

195     1           ADDON:PROC(Q1,Q2,J);
                         /*  ADDON ATTACHES THE TREE ROOTED AT Q2 TO NODE Q1 BY MEANS
                             OF EDGE J.  IT REQUIRES THAT Q2 BE THE ROOT OF A TREE. */
196     2                DCL (Q1,Q2) PTR,  J BIN FIXED(16);

197     2                Q2->RT=Q1->UP;
198     2                Q2->DN=Q1;
199     2                Q2->EDGEDN=J;
200     2                Q1->UP=Q2;
201     2                FRST(J)=T;
202     2           END ADDON;

                    /*********************************************************************
                     ***************   THE UPSCAN ROUTINES  ***********************
                     *********************************************************************/

203     1           UPSCAN:PROC(P1,UPCALL,SUBRUP,DNCALL,SUBRDN);

                         /*  UPSCAN GOES THROUGH ALL THE NODES ABOVE P1 IN THE TREE
                             CONTAINING P1 AND IF UPCALL=T THEN CALLS SUBRUP FOR
                             EACH NODE IN THE TREE AS IT REACHES IT COMING UP.
                             IF DNCALL = T THEN SUBRDN IS CALLED FOR EACH NODE AS
                             IT IS ENCOUNTERED COMING DOWN.  */
204     2                DCL (P1,(Q1,Q2) STATIC ) PTR,(UPCALL,DNCALL) BIT(1),
                             (SUBRUP,SUBRDN) ENTRY;

205     2                IF UPCALL THEN CALL SUBRUP(P1);
207     2                Q1=P1;
208     2           MVUP:Q2=Q1->UP;
209     2                IF Q2¬=NULL THEN DO;
211     2     1     CALLUP:    IF UPCALL THEN CALL SUBRUP(Q2);
213     2     1                Q1=Q2;
214     2     1                GO TO MVUP;
215     2     1                END;
216     2           ENDTST: IF Q1=P1 THEN DO;
218     2     1                IF DNCALL THEN CALL SUBRDN(Q1);
220     2     1                RETURN;
221     2     1                END;
222     2                Q2=Q1->RT;
223     2                IF Q2¬=NULL THEN DO;
225     2     1                IF DNCALL THEN CALL SUBRDN(Q1);
227     2     1                GO TO CALLUP;
228     2     1                END;
229     2                Q2=Q1;
230     2                Q1=Q1->DN;
231     2                IF DNCALL THEN CALL SUBRDN(Q2);
233     2                GO TO ENDTST;
234     2           END UPSCAN;
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

235    1        ADDFIX: PROC (Q1);
                    /*   THIS PROCEDURE SETS ODD,DEFIC AS APPROPRIATE FOR THE NODE
                         Q1.  IT DEPENDS ON Q1->TREE.DN.  */
236    2            DCL (Q1,Q2 STATIC) PTR;

237    2            Q1->BLOS,Q1->INPATH=F;
238    2            Q2=Q1->DN;
239    2            Q1->DEFIC=Q2->DEFIC;
240    2            Q1->YRTO = Q2->YRTO;
241    2            IF ¬Q1->DEFIC THEN Q1->ODD=F;
243    2            ELSE Q1->ODD=¬Q2->ODD;
244    2        END ADDFIX;

245    1        POLYFIX:  PROC(P1);
                    /*  PROC CALLED BY UPSCAN TO
                        1)  SET P1->STACKUP = NULL;
                        2)  SET P1->INPATH = F.        */
246    2            DCL P1 PTR;

247    2            P1->STACKUP=NULL;
248    2            P1->INPATH=F;
249    2        END POLYFIX;

250    1        NONDEFIX: PROC (P1);
                /* THIS IS A PROCEDURE DESIGNED TO BE CALLED BY UPSCAN WHICH CORRECTS
                   THE STATUS INDICATORS AND SPLITS A TREE WITH NON-DEFICIENT ROOT
                   INTO POSITIVE COMPONENTS */
251    2              DCL P1 PTR;

252    2              P1 -> DEFIC, P1 -> ODD = F;
253    2              P1 -> BLOS = F;
254    2              P1 -> YRTO = F;
               /* INDICATORS NOW CORRECT */
255    2              IF P1 -> DN¬=NULL /* I.E. IT EXISTS */
256    2                 THEN  IF X(P1->EDGEDN)=0 /* I.E. WE HAVE PLACE FOR
                           DETACHING */
257    2                    THEN CALL REMOVE(P1);
258    2              END NONDEFIX;
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

 259     1          BLOSSIND:PROC (P1);
                    /* PROCEDURE TO INDICATE THAT P1 IS A NODE IN A NON-DEFICIENT
 260     2             BLOSSOM, AND THE EDGE DOWN IS IN A SIMILAR STATE */
                       DCL P1 PTR;
 261     2
 262     2                FRST (P1 -> EDGEDN)=T;
 263     2                P1 -> ODD, P1 -> DEFIC, P1->INPATH = F;
 264     2                P1 -> BLOS=T;
                    END;

 265     1          ADDBLOS:PROC(P1);
                         /* PROCEDURE CALLED WHEN EXPANDED BLOSSOM HAS BEEN ADDED
                         TO A DEFIC TREE.  IT SETS ODD UNTIL A ZERO EVEN EDGE
                         IS FOUND,  WHEN IT SETS THINGS UP FOR DEFFIX TO SPLIT
                         THINGS INTO NONZERO COMPONENTS. NOCHECK IS USED TO AVOID
                         TRYING TO SET DEFIC AND ODD FOR THE ROOT WHEN UPSCAN IS
 266     2               STARTED AT THE ROOT OF A TREE.  */
                         DCL(P1,P2 STATIC)PTR;
                         /*  PX AND SHRNKNG ARE USED AS EXT. VARS. */
 267     2
 268     2               P1->STACKUP=NULL;
 269     2               P1->BLOS,P1->INPATH=F;
 271     2     1         IF NOCHECK THEN DO;
 274     2                   NOCHECK=F;  RETURN;  END;
 276     2               IF ¬SHRNKNG THEN GO TO LAB1;
 277     2               P2=P1->DN;
 278     2               IF P2->ODD THEN
 280     2     1              IF X(P1->EDGEDN)=0 THEN DO; /* DETACH */
 282     2     1                   SHRNKNG=F;PX=P1;
 283     2     1              LAB1:P1->DEFIC,P1->ODD = F;
 284     2     1                   RETURN;
 285     2                         END;
 286     2               P1->DEFIC=P2->DEFIC;
 287     2               P1->ODD=¬P2->ODD;
                         RETURN;

 288     2          DEFFIX:ENTRY(P1);
 289     2               IF ¬SHRNKNG THEN /*POSSIBLE DETACHMENT */
 290     2                  IF X(P1->EDGEDN)=0 THEN CALL REMOVE(P1);
 292     2               IF P1=PX THEN SHRNKNG=T;
 294     2               RETURN;
 295     2          END ADDBLOS;
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

296     1           UPBLOSS:PROC (P1);
                    /* UPBLOSS AND DNBLOSS DO MOST OF THE WORK REQUIRED TO
                       SHRINK A BLOSSOM.  WE USE PX (PTR) AND SHRNKNG (BIT(1)) AS
                       DEFINED IN BLOSSOM.  WE ASSUME THAT R2 IS THE ROOT OF THE
                       BLOSSOM AND P IS THE PSEUDONODE BEING CREATED.  */

297     2                DCL P1 PTR;

298     2                IF ¬ SHRNKNG THEN RETURN;
300     2                IF P1=R2 THEN GO TO BFIX;
302     2                IF ¬ P1 -> INPATH THEN
303     2                     IF X(P1 -> EDGEDN)=0 THEN DO;
305     2   1                      SHRNKNG=F; /* STOP SHRINKING */
306     2   1                      PX=P1 /* SAVE NODE FOR DNBLOSS*/;
307     2   1                      RETURN;
308     2   1                      END;
309     2              BFIX: P1 -> STACKUP=P;
310     2                 RETURN;

311     2           DNBLOSS:ENTRY (P1);
312     2                P1->INPATH=F;  /* TURN OFF PATH INDICATOR */
313     2                IF P1¬=PX THEN /* NO SNIPPING TO BE DONE, SO */
314     2                RETURN;
315     2                K=P1 -> EDGEDN;
316     2                CALL REMOVE (P1);
317     2                CALL ADDON (P,P1,K);
318     2                SHRNKNG=T;  /*  RESUME SHRINKING */
319     2                RETURN;
320     2                END;

321     1           SETYRT0: PROC(P1);
                         /*  PROCEDURE CALLED BY UPSCAN TO SET P1->YRT0 EQUAL TO
                             THE GLOBAL VARIABLE YROOT0.   */
322     2                DCL P1 PTR;

323     2                P1 -> YRT0=YROOT0;
324     2           END SETYRT0;
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

                  /*************************************************************
                   ***********  INITIALIZATION  ********************************
                   *************************************************************/
 325     1              IF RUNSTAT(10)=1 THEN TRACE=T;  ELSE TRACE=F;
 328     1              IF TRACE THEN DO;
 330     1    1              CALL XOUT;  CALL YOUT;  END;
                        /*   GENERATE THE INITIAL EQUALITY SUBGRAPH.  */
 333     1              IF TRACE THEN DO;
 335     1    1              PUT EDIT('*BLOSS - EDGES IN EQUALITY SUBGRAPH:')(SKIP,A);
 336     1    1              PUT SKIP;  END;
 338     1              DO J=1 TO NEDGE;
 339     1    1              C(J) = C(J) - FN(J);  /* CALCULATE REDUCED COST */
 340     1    1              IF C(J) = 0 THEN EQ(J)=T;  ELSE EQ(J) = F;
 343     1    1              IF TRACE THEN IF EQ(J) THEN PUT EDIT(J)(F(5));
 346     1    1              END;
                        /*  END OF EQUALITY SUBGRAPH GENERATION */
 347     1              RUNSTAT=0;
 348     1              NOCHECK,FROMEX = F;
 349     1              JCNT,LASTJ=1;
 350     1         A:
                  /*************************************************************
                   *************  FIRST LEVEL EDGE ANALYSIS   ******************
                   *************************************************************/
 351     1              IF ¬EQ(JCNT) THEN GO TO ENDA;
 352     1              IF ZER(JCNT) THEN GO TO ENDA;
 354     1              IF SHRNK(JCNT) THEN GO TO ENDA;
 356     1              IF FRST(JCNT) THEN GO TO ENDA;
                        /*
                             OTHERWISE WE HAVE AN EDGE WHICH IS IN THE EQUALITY SUB-
                             GRAPH WHICH CAN TAKE ON A NONZERO VALUE AND SO FAR HAS
                             NOT BEEN SHRUNK AND IS NOT IN THE FOREST.*/
                        /*
                             WE NOW ANALYZE THE EDGE.  IN ORDER FOR IT TO BE USEFUL
                             ONE END MUST BE AN EVEN NODE OF A DEFICIENT TREE IN THE
                             FOREST FOR WHICH THE ROOT IS NOT A <= NODE WITH Y=0. */
 358     1              P1=ENDS(JCNT,1);  P2=ENDS(JCNT,2);
 360     1              IF P1->DEFIC THEN
 361     1                   IF ¬P1->YRT0 THEN
 362     1                        IF ¬P1->ODD THEN GO TO DEFOUT;
 364     1              IF ¬P2->DEFIC THEN GO TO ENDA;
 366     1              IF P2->ODD THEN GO TO ENDA;
 368     1              IF P2->YRT0 THEN GO TO ENDA;
 370     1              P3=P2; P2=P1; P1=P3; /* INTERCHANGE POINTERS FOR P2 DEF. OUT ND.*/
 373     1         DEFOUT: /* IF THE OTHER END OF THE EDGE IS AN ODD NODE OF
                             THE FOREST THEN THE EDGE IS OF NO USE TO US,
                             UNLESS IT IS IN A TREE WHOSE ROOT IS <= WITH Y = 0. */
                        IF P2->ODD THEN
 374     1                   IF ¬P2->YRT0 THEN GO TO ENDA;
 376     1                   ELSE GO TO ODDGROW;
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE,  16-03-73   */

STMT LEVEL NEST

377     1               J,LASTJ=JCNT;       /* FOR WE ARE ABOUT TO ACCOMPLISH SOMETHING*/
378     1               IF P2->BLOS THEN GO TO POLYSTEP;
380     1               IF ¬P2->DEFIC THEN GO TO GROWSTEP;
                     /*  OTHERWISE EDGE(J) JOINS TWO EVEN NODES OF THE FOREST */
                     /*  FIRST WE SEE IF THEY ARE IN TREES WITH DISTINCT ROOTS,
                         IF SO WE CAN SIMPLY AUGMENT,  OTHERWISE WE MAY HAVE TO
                         SHRINK.  AT THE SAME TIME WE COMPUTE HOW MUCH THE VALUES
                         ON THE PATH CAN BE CHANGED.*/

                    /*****************************************************************
                    ************   SECOND LEVEL EDGE ANALYSIS  ***********************
                    *****************************************************************/
382.    1               DXCALC: D1,D2,D3=32767;
                          /* NOW FIND PATH FROM P1 TO THE ROOT*/
383     1               R1=P1;
384     1               R1->INPATH=T;
385     1               DO WHILE (R1->DN ¬= NULL);
386     1   1                 IF ¬R1->ODD THEN D1 = MIN(D1,X(R1->EDGEDN));
388     1   1                 R1=R1->DN;
389     1   1                 R1->INPATH=T;
390     1   1                 END;
                         /*    SIMILARLY,  FIND PATH FROM P2 TO ITS ROOT R2;  IF A
                               POLYGON IS FORMED,  R2 WILL BE THE ROOT OF THE POLYGON. */
391     1               R2=P2;
392     1               DO WHILE(¬R2->INPATH);
393     1   1                 R2->INPATH=T;
394     1   1                 IF P2->DN=NULL THEN /* WE ARE AT THE ROOT */ GO TO TWOTREE;
396     1   1                 IF ¬R2->ODD THEN D2=MIN(D2,X(R2->EDGEDN));
398     1   1                 R2=R2->DN;
399     1   1                 END;
                         /* WE MUST HAVE A COMMON ROOT TO THE TWO TREES SO WE */
400     1               GO TO ONETREE;
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

                  /*****************************************************************
                  ***********   TWO TREE AUGMENTATION   ***********************
                  *****************************************************************/
401     1         TWOTREE:
                    /*  IF WE MADE IT TO HERE,  R1 AND R2 ARE DIFFERENT SO WE
                        AUGMENT BY AMOUNT */
                    RUNSTAT(6)=RUNSTAT(6)+1;
402     1           DELTAX=MIN(D1,D2,P1->DEF,P2->DEF);
403     1           CALL AUGMENT(P1,R1,DELTAX,T,(F));
404     1           CALL AUGMENT(P2,R2,DELTAX,T,(F));
405     1           X(J)=X(J)+DELTAX;
406     1           R1->DEF=R1->DEF - DELTAX;
407     1           R2->DEF=R2->DEF - DELTAX;
408     1           IF TRACE THEN DO;
410     1   1           PUT EDIT('*BLOSS - EDGE ',J,' USED FOR 2 TREE AUGMENTATION')
                            (R(FMT));
411     1   1           CALL XOUT;   END;
                  /*  NOW CORRECT STATUS INDICATORS IN THE TREE*/
413     1           IF R1->DEF = 0 THEN CALL UPSCAN(R1,F,NONDEFIX,T,NONDEFIX);
415     1           IF R2->DEF = 0 THEN CALL UPSCAN(R2,F,NONDEFIX,T,NONDEFIX);
                  /*  FINALLY INCORPORATE J INTO THE FOREST */
417     1         JADD:   IF P1->DEFIC THEN /* P2 CANNOT BE IN A DEFICIENT TREE, ADD
418     1                 ON TO P1*/DO;P3=P2;P2=P1;P1=P3;END;
423     1           CALL REROOT(P1);CALL ADDON(P2,P1,J);
425     1           CALL UPSCAN (P1,T,ADDFIX,F);
426     1           GO TO ENDA;

                  /*****************************************************************
                  ************  SINGLE TREE AUGMENTATION   ********************
                  *****************************************************************/
427     1         ONETREE:  /*  FIND BOTTLENECK IN STEM OF BLOSSOM */
                      R3=R2;
428     1             DO WHILE (R2->DN¬=NULL);
429     1   1             IF ¬R2->ODD THEN DO;
431     1   2                   IF X(R2->EDGEDN) = 1 THEN /* WE HAVE FOUND THE
432     1   2                   START OF A BLOSSOM,  SO*/ GO TO DEFBLOSS;
433     1   2                   D3=MIN(D3,X(R2->EDGEDN));
434     1   2                   END;
435     1   1               R2=R2->DN;
436     1   1           END;
                  /* AT THIS POINT,  AN AUGMENTATION IS POSSIBLE,  SINCE NO EVEN
                      EDGE IN THE STEM HAS X=1,  UNLESS R1->DEF = 1.*/
437     1           IF R1->DEF = 1 THEN GO TO DEFBLOSS;
                    /*  OTHERWISE ITS AUGMENTATION TIME.*/
439     1           RUNSTAT(7)=RUNSTAT(7)+1;
440     1           DELTAX=MIN(D1,D2,FLOOR(R1->DEF/2),FLOOR(D3/2));
441     1           ODDB=F; /* NORMAL CASE */
442     1           IF P1->ODD THEN  DO;       /* ABNORMAL CASE */
444     1   1                   ODDB=T;  DELTAX=MIN(DELTAX,X(J));  END;
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

                    CALL AUGMENT(P1,R3,DELTAX,T,(ODDB));
                    CALL AUGMENT(P2,R3,DELTAX,T,(ODDB));
                    X(J)=X(J)+DELTAX;
                    IF ODDB THEN X(J) = X(J) -2*DELTAX; /* CORRECT A BAD GUESS */
                    POLYBIT=F;
                    IF P1->DEFIC THEN
                      IF P2->DEFIC THEN
                        IF X(J)>0 THEN POLYBIT=T;/* WE HAVE A NONZERO POLYGON */
                    DELTAX=DELTAX+DELTAX; /* STEM GETS DOUBLE AUGMENTATION */
                    R1->DEF=R1->DEF - DELTAX;
                    ODDB=F;
                    IF R3->ODD THEN ODDB=T;
                    CALL AUGMENT(P3,R1,DELTAX,T,(ODDB));
                    IF TRACE THEN DO;
                         PUT EDIT('*BLOSS - EDGE ',J,' USED FOR 1 TREE AUGMENTATION')
                              (R(FMT));
                         CALL XOUT;
                         END;
                    /*  DISASSEMBLE THE TREE IF ROOT NO LONGER DEFICIENT. */
                    IF R1->DEF = 0 THEN CALL UPSCAN(R1,F,NONDEFIX,T,NONDEFIX);
                    ELSE IF FROMEX THEN DO; /* WE HAVE EXPANDED PSEUDO ODD NODE
                              AND MUST ENSURE THAT TREE IS CORRECT. */
                              NOCHECK=T; /* IGNORE ROOT,  SET F BY ADDBLOS. */
                              PX=NULL; SHRNKNG=T; /* GLOBALS FOR ADDBLOS-DEFFIX */
                              CALL UPSCAN(R1,T,ADDBLOS,T,DEFFIX);
                              END;
                    FROMEX=F;
                    IF ¬POLYBIT THEN /* INCORPORATE J INTO THE FOREST */
                      IF X(J)>0 THEN GO TO JADD;
                      ELSE DO;  FRST(J)=F;  GO TO ENDA;   END;
                    /* OTHERWISE WE HAVE A POLYGON WITH NONZERO EDGES */
                    IF R3 -> DEFIC THEN DO;
                         R2=R3;
                         DO WHILE (R2->DN¬=NULL);
                              IF ¬R2 -> ODD THEN
                                   IF X(R2->EDGEDN)=1
                                        THEN GO TO DEFBLOSS;
                              R2=R2->DN;
                              END;
                         GO TO DEFBLOSS;  /* ROOT OF TREE ROOT OF BLOSSOM */
                         END;
                    ELSE DO;  R2=R3;
                    LB1:  IF R2->DN = NULL THEN GO TO LB2;
                         R2=R2->DN;
                         GO TO LB1;
                    LB2:  R2-> EDGEDN=J;
                         CALL UPSCAN (R2,T,BLOSSIND,F);/* SHOW A NONZERO COMPONENT */
                         GO TO ENDA;                   /* CONTAINING AN ODD POLYGON */
                         END;
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

                    /*****************************************************************
                     ********   PSEUDO NODE CREATION ***************************
                     *****************************************************************/
506      1          DEFBLOSS:  /* HERE P1 AND P2 ARE THE TWO ENDS OF THE
                       EDGE J WHICH FORMS A BLOSSOM, R2 IS THE ROOT OF THE STEM
                       AND R3 IS THE ROOT OF THE POLYGON.  ALL WE NEED DO IS
                       SHRINK IT */
                       RUNSTAT(2)=RUNSTAT(2)+1;
507      1             FROMEX=F; /* IN CASE IT WAS SET T BY PSEUDO EXPANSION */
508      1             ALLOCATE PSEUDO; /* CREATE A PSEUDO NODE */
509      1                  DEF=1;
510      1                  REAL,INPATH,ODD,BLOS,EXPANDED=F;
511      1                  CONSTEQ,DEFIC=T;
512      1                  Y=0E0;
513      1                  STACKUP=NULL;
514      1                  ROOT=R2;
515      1                  UP,RT,DN=NULL;
516      1                  YRT0=R2->YRT0;
517      1             IF TRACE THEN PUT EDIT('*BLOSS - EDGE ',J,' FORMS PSEUDONODE',
                             UNSPEC(P))(SKIP,A,F(6),A,F(10));
                       /* INDICATE NODES IN PATHS FROM P2, P1 TO R2 */
519      1             P3=P1;
520      1             DO WHILE (P3¬=R2);
521      1   1              P3 -> INPATH = T; P3=P3->DN; END;
524      1             R2->INPATH=T;
525      1             P3=P2;
526      1             DO WHILE (¬P3->INPATH);
527      1   1              P3 -> INPATH = T; P3=P3->DN; END;
                       /* TURN OFF INPATH IN UNUSED PART OF STEM */
530      1             P3=R2->DN;
531      1             DO WHILE(P3¬=NULL);
532      1   1                P3->INPATH=F;  P3=P3->DN;  END;
535      1             SHRINKNG=T; /* WE ARE SHRINKING */
536      1             PX=NULL; /* PREP. FOR CALL OF UPSCAN */
537      1             CALL UPSCAN (R2,T,UPBLOSS,T,DNBLOSS);
538      1             K=R2->EDGEDN; P3=R2->DN;
540      1             CALL REMOVE (R2);
541      1             IF P3 ¬= NULL THEN CALL ADDON (P3,P,K);
543      1               ELSE P->EDGEDN = 0;
544      1             R2->EDGEDN=J; /* THIS IS THE EDGE THAT FORMED THE BLOSSOM;
                          NOW FIX ALL EDGES SO THAT ENDS IS CORRECT */
545      1             DO J1=1 TO NEDGE;
546      1   1              IF SHRNK(J1) THEN GO TO ENDC;
548      1   1              P1=ENDS(J1,1); P2=ENDS (J1,2);
550      1   1              IF P1->STACKUP=NULL THEN /* NO CHANGE */
551      1   1                   GO TO ENDXB;
552      1   1              SHRNK(J1)=T;
553      1   1              ENDS(J1,1)=P;
554      1   1         ENDXB:  IF P2->STACKUP¬=NULL THEN
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

555      1    1              ENDS(J1,2)=P;
556      1    1              ELSE SHRNK(J1)=F;
557      1    1         ENDC:   END;
                        /* NOW SHRINKING IS COMPLETE */
558      1              GO TO ENDA;

                    /*************************************************************
                     ***********NORMAL FOREST GROWTH ***************************
                     *************************************************************/
559      1           GROWSTEP:  /* WE GROW TREE BY USING J TO ADD A NONDEFICIENT
                                  TREE */
                        RUNSTAT(5)=RUNSTAT(5)+1;
560      1              IF TRACE THEN PUT EDIT('*BLOSS - EDGE ',J,' USED TO GROW FOREST')
                                (R(FMT));
562      1              CALL REROOT(P2);
563      1              CALL ADDON(P1,P2,J);
564      1              CALL UPSCAN (P2,T,ADDFIX,F);
565      1              GO TO ENDA;
                    /*************************************************************
                     ************ ADJUNCTION OF POLYGON TO THE FOREST *********
                     *************************************************************/
566      1           POLYSTEP:  /* FIND ROOT OF COMPONENT */
                        RUNSTAT(8)=RUNSTAT(8)+1;
567      1              IF TRACE THEN PUT EDIT('*BLOSS - EDGE ',J,' USED TO ADD NONZERO PO
                    LYGON TO THE FOREST')(R(FMT));
569      1              P3=P2;
570      1              DO WHILE (P3->DN¬=NULL);
571      1    1              P3=P3->DN;
572      1    1              END;
573      1              J1=P3->EDGEDN; /*J1 IS THE EDGE WHICH FORMED THE POLYGON */
574      1              CALL REROOT (P2);
                            /* REROOT THE COMPONENT AND ADD IT TO TREE */
575      1              CALL ADDON (P1,P2,J);
576      1              CALL UPSCAN (P2,T,ADDFIX,F);
577      1              P1=ENDS(J1,1);
578      1              P2=ENDS(J1,2);
579      1              J=J1;
580      1              GO TO DXCALC;
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

                    /*************************************************************
                     ********** PSEUDO FOREST GROWTH ***************************
                     *************************************************************/
 581      1         ODDGROW:  /* AN EDGE J HAS BEEN FOUND JOINING P1 IN F1 TO P2 IN F0*/
                        RUNSTAT(5)=RUNSTAT(5)+1;
 582      1
 583      1         J,LASTJ = JCNT;
                    IF TRACE THEN PUT EDIT('*BLOSS - EDGE ',J,' USED FOR PSEUDO FOREST
                   GROWTH')(R(FMT));
                        /* FIND FIRST NODE IN PATH FROM P2 TO ITS ROOT HAVING A ZERO
                           DOWN EDGE,  OR IF NO SUCH EDGE EXISTS,  THEN WE FIND THE
                           ROOT OF THE TREE CONTAINING P2. */
 585      1                R1=P2;
 586      1                DO WHILE(R1->DN ¬= NULL);
 587      1    1                IF X(R1->EDGEDN)=0 THEN GO TO ROOTADD;
 589      1    1                R1=R1->DN;
 590      1    1                END;
                        /*  R1 IS THE ROOT,  ALL EDGES IN PATH HAVE X>0. */
 591      1         ROOTADD:
                        Q3=R1->DN;
 592      1                CALL REMOVE(R1);
 593      1                CALL REROOT(P2);
 594      1                CALL ADDON(P1,P2,J);  /* TREES NOW CONSOLIDATED */
 595      1                YROOT0 = F;
 596      1                CALL UPSCAN(P2,T,SETYRT0,F);
 597      1                IF Q3 ¬= NULL THEN /*  WE HAD A ZERO EDGE */GO TO ENDA;
                        /* NOW AUGMENT SO AS TO GET DEFICIENCY TO THE ROOT */
                        /*  Q1 WILL BE THE LAST NODE FOR WHICH THE DOWN EDGE BECOMES 0 */
 599      1         AUG: D1=32767;
 600      1                RUNSTAT(9)=RUNSTAT(9)+1;
 601      1                R2=R1;
 602      1                DO WHILE (R2->DN ¬= NULL);
 603      1    1                IF ¬R2->ODD THEN DO;
 605      1    2                     J1=R2->EDGEDN;
 606      1    2                     IF X(J1) <= D1 THEN DO;
 608      1    3                          D1=X(J1);
 609      1    3                          Q1=R2;
 610      1    3                          END;
 611      1    2                     END;
 612      1    1                R2=R2->DN;
 613      1    1                END;
 614      1              DELTAX=MIN(D1,R2->DEF);
                       /* NOW WE AUGMENT */
 615      1              CALL AUGMENT(R1,R2,DELTAX,F,(F));
 616      1              R2->DEF = R2->DEF - DELTAX;
 617      1              R1->DEF=R1->DEF + DELTAX; /* WE INCREASE DEF AT THIS NODE. */
 618      1              IF TRACE THEN DO;
 620      1    1                PUT EDIT('*BLOSS - PSEUDO AUGMENTATION')(SKIP,A);
 621      1    1                CALL XOUT;
 622      1    1                END;
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

623     1               IF D1 > DELTAX THEN /*  NO EDGE IN PATH BECAME ZERO */
624     1                    GO TO TADD;
                      /* ELSE EVERYTHING ABOVE Q1 GETS REMOVED AND REROOTED AT R1 */
625     1                 CALL REMOVE (Q1);
626     1                 YROOT0=F;
627     1                 CALL UPSCAN(R2,T,SETYRT0,F);
628     1                 IF R2->DEF=0 THEN
629     1                 CALL UPSCAN(R2,F,NONDEFIX,T,NONDEFIX); /* ALSO SETS YTR0=F */
630     1               TADD:  CALL REROOT(R1);
631     1                 YROOT0 = T;
632     1                 CALL UPSCAN(R1,T,SETYRT0,F);
                    /*************************************************************
                     *********  END OF MAIN PROCESSING LOOP  ******************
                     *************************************************************/
633     1               ENDA:JCNT=1 + MOD(JCNT,NEDGE);
634     1                    IF JCNT¬=LASTJ THEN /* CONTINUE PROCESSING */ GO TO A;
                         /* WHENEVER AN EDGE IS MADE USE OF IN THE MAIN LOOP,  LASTJ
                         IS SET EQUAL TO THE INDEX OF THE EDGE.  IF JCNT EVER 'CATCHES
                         UP' WITH LASTJ THEN WE HAVE MADE A COMPLETE CYCLE THROUGH THE
                         EDGES WITHOUT FINDING ANY EDGES WHICH WE CAN USE SO WE PROCEED
                         TO ATTEMPT A CHANGE OF DUAL VARIABLES.  */
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

                  /*************************************************************
                   ********** DUAL VARIABLE CHANGE ROUTINE *********************
                   *************************************************************/
636     1             DUALCHNGE:
                      /*  NOW EXAMINE NODES,  IF NO SURFACE NODE IS IN A DEFIC TREE
                         THEN WE ARE DONE.  FAIL IS SET TRUE IF WE DISCOVER THAT
                         THIS IS NOT THE CASE. */
                      FAIL=F;
637     1             EPS1,EPS2=1E10;/*RIDICULOUSLY LARGE VALUES*/
                  /*************************************************************
                   **********  DETERMINATION OF NODE BOUND  ********************
                   *************************************************************/
638     1             PX=NULL;
639     1         LF: DO I = 1 TO NNODE;
640     1   1             P1=ADDR(NODELST(I));
641     1   1             P2=SURF(P1);/*  HIGHEST LEVEL PSEUDONODE CONTAINING P1 */
642     1   1             IF¬P2->DEFIC THEN GO TO ENDF;
644     1   1             IF P2->YRTO THEN GO TO ENDF;
                          /* ELSE WE HAVE NOT YET GOT A FEASIBLE MATCHING. */
646     1   1             FAIL=T;
647     1   1                   IF ¬P2->ODD THEN
648     1   1                     IF ¬P1->CONSTED THEN
649     1   1                       IF P1->Y < EPS2 THEN DO;
651     1   2                         PX=P1;
652     1   2                         EPS2=P1->Y;
653     1   2                         END;
654     1   1             IF P2->REAL THEN /* NOT IN A PSEUDO NODE */ GO TO ENDF;
                          /* OTHERWISE CHECK THE PSEUDO NODE */
656     1   1             P2->DCHNG=F;/* NO DUAL CHANGE MADE YET ON THIS NODE */
657     1   1             IF¬P2->ODD THEN GO TO  ENDF;
659     1   1                  Z=P2->Y / 2E0;
660     1   1                  IF Z<EPS2 THEN DO;
662     1   2                       PX=P2;
663     1   2                       EPS2=Z;
664     1   2                       END;
665     1   1    ENDF:  END LF;
666     1             IF ¬FAIL THEN /* WE ARE FINISHED */ GO TO CORRECTION;
668     1             IF EPS2= 0 THEN GO TO NODEBND;/* NO NEED TO CHECK EDGES,
                             WE ALREADY HAVE OUR BOUND. */
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

                    /*************************************************************
                     **********  DETERMINATION OF EDGE BOUND  ****************
                     ************************************ ************************/
                         /*  NOW CHECK EDGES FOR A BOUND ON EPS */
670      1               JX=0;
671      1           LD: DO J=1 TO NEDGE;
672      1     1             IF EQ(J) THEN GO TO ENDD; /*  IGNORE EDGES IN EQ SUBGRAPH */
674      1     1             IF SHRNK(J) THEN GO TO ENDD;
676      1     1             IF ZER(J) THEN GO TO ENDD;
678      1     1             P1=ENDS(J,1);
679      1     1             IF ¬P1->DEFIC THEN GO TO TRY2;
681      1     1             IF P1->YRTO THEN GO TO TRY2;
683      1     1             P2=ENDS(J,2 );
684      1     1             IF ¬P1->ODD THEN GO TO TESTP2;
686      1     1         TRY2:P1=ENDS(J,2);
687      1     1             IF ¬P1->DEFIC THEN GO TO ENDD;
689      1     1             IF P1->YRTO THEN GO TO ENDD;
691      1     1             P2=ENDS(J,1);
692      1     1             IF P1->ODD THEN GO TO ENDD;
                         /*  AT THIS POINT P1 IS AN EVEN NODE OF A DEFIC TREE, AS LONG
                         AS P2 IS NOT AN ODD NODE WE HAVE FOUND AN EDGE OF INTEREST */
694      1     1         TESTP2:IF P2->ODD THEN
695      1     1                    IF ¬P2->YRTO THEN GO TO ENDD;
697      1     1                Z=- C(J);
698      1     1                IF P2->DEFIC THEN
699      1     1                  IF ¬P2->YRTO THEN Z=Z/2E0;
                              /* ELSE J HAS JUST ONE END IN THE FOREST*/
701      1     1                IF Z>=EPS1 THEN GO TO ENDD;
703      1     1                JX=J;
704      1     1                EPS1=Z;
705      1     1           ENDD:     END LD;
                         /*************************************************************
                          **********  MAKE ACTUAL CHANGE IN DUAL VARS. ************
                          *************************************************************/
706      1                    EPS=MIN(EPS1,EPS2);
707      1                    IF EPS=1E10 THEN /*  FOREST IS HUNGARIAN */ DO;
709      1     1                   RUNSTAT(10)=1;
710      1     1                   GO TO CORRECTION;   END;
                         /*  HERE WE GO ON A CHANGE OF DUAL VARIABLES*/
712      1                    IF TRACE THEN PUT EDIT('*BLOSS - DUAL VARIABLE CHANGE')(SKIP,A);
714      1                LG : DO I=1 TO NNODE;
715      1     1              P1=ADDR(NODELST(I));
716      1     1              P2=SURF(P1);
717      1     1              IF ¬P2->DEFIC THEN GO TO ENDLG;
719      1     1              IF P2->YRTO THEN GO TO ENDLG;
721      1     1              IF P2->ODD THEN DO;
723      1     2                   P1->Y=P1->Y + EPS;
724      1     2                   IF ¬P2->REAL THEN
725      1     2                     IF ¬P2->DCHNG/*P2->Y HAS NOT YET BEEN CHANGED */ THEN DO;
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

727     1    3                  P2->Y=P2->Y-2E0 * EPS;
728     1    3                  P2->DCHNG=T;
729     1    3                  IF TRACE THEN PUT EDIT('      PSEUDO ',UNSPEC(P2),
                                ' DUAL VAR.',P2->Y)(SKIP,A,F(10),A,F(10,1));
731     1    3                  END;
732     1    2             END;
733     1    1        ELSE /* P2 IS AN EVEN NODE */ DO;
734     1    2             P1->Y=P1->Y - EPS;
735     1    2             IF ¬P2->REAL THEN
736     1    2               IF ¬P2->DCHNG THEN DO;
738     1    3                  P2->Y=P2->Y +2E0 * EPS;
739     1    3                  P2->DCHNG= T;
740     1    3                  IF TRACE THEN PUT EDIT('      PSEUDO ',UNSPEC(P2),
                                ' DUAL VAR.',P2->Y)(SKIP,A,F(10),A,F(10,1));
742     1    3                  END;
743     1    2             END;
744     1    1   ENDLG: END LG;
745     1             RUNSTAT(1)=RUNSTAT(1)+1;
746     1             IF TRACE THEN CALL YOUT;
748     1             IF TRACE THEN PUT EDIT('*BLOSS - EDGES IN EQUALITY SUBGRAPH')
                         (SKIP,A);
                      /* NOW CALCULATE NEW REDUCED COSTS */
750     1             DO J=1 TO NEDGE;
751     1    1          IF SHRNK(J) THEN GO TO MSG;
753     1    1          IF FRST(J) THEN GO TO MSG;
755     1    1          IF ZER(J) THEN GO TO ENDX;
757     1    1          P1=ENDS(J,1);  P2=ENDS(J,2);
759     1    1          IF P1->DEFIC THEN
760     1    1            IF ¬P1->YRTO THEN DO;
762     1    2              IF P1->ODD THEN C(J)=C(J)-EPS;
764     1    2              ELSE C(J)=C(J)+EPS;
765     1    2              END;
766     1    1          IF P2->DEFIC THEN
767     1    1            IF ¬P2->YRTO THEN DO;
769     1    2              IF P2->ODD THEN C(J) = C(J) - EPS;
771     1    2              ELSE C(J)=C(J) + EPS;
772     1    2              END;
773     1    1        MSG: IF C(J)=0 THEN DO;
775     1    2              EQ(J)=T;
776     1    2              IF TRACE THEN PUT EDIT(J)(F(5));
778     1    2              END;
779     1    1          ELSE EQ(J)=F;
780     1    1        ENDX;END;
781     1             IF EPS1 = EPS THEN DO;JCNT,LASTJ=JX;
784     1    1          GO TO A; /* RETURN TO MAIN LOOP AND ACCOMPLISH SOME -
                                    THING.  */
785     1    1        END;
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

                    /*************************************************************
                     ********** REROOT A TREE SO ROOT HAS Y=0  *********************
                     *************************************************************/
 786     1          NODEBND:
 787     1               IF PX -> REAL THEN GO TO PSEUDOEX;
                         /* OTHERWISE PX IS A REAL <= EVEN NODE */
 788     1               IF TRACE THEN PUT EDIT('*BLOSS - REROOT A TREE')(SKIP,A);
 790     1               R1=SURF(PX);
 791     1               JCNT,LASTJ=1;
 792     1               IF R1->DN^=NULL THEN GO TO AUG;  /* FOR R1 IS NOT A ROOT,
                         OTHERWISE R1 IS A ROOT.  */
 794     1               YROOT0 = T;
 795     1               CALL UPSCAN (R1,T,SETYRT0,F);
 796     1               GO TO A;

                    /*************************************************************
                     **********  PSEUDO NODE EXPANSION ROUTINE *********************
                     *************************************************************/
 797     1          PSEUDOEX:
                         P=PX;
 798     1               RUNSTAT(4)=RUNSTAT(4)+1;
 799     1               IF TRACE THEN PUT EDIT('*BLOSS - EXPAND PSEUDONODE ',UNSPEC(P))
                                 (SKIP,A,F(10));
 801     1               CALL EXPAND(P);  /* EXPAND THE PSEUDONODE */
                         /*   R1 IS THE ROOT OF THE BLOSSOM,
                              JIN IS THE EDGE FOR WHICH X(JIN)^=0,
                              BROOT IS THE EDGE OF JIN IN THE BLOSSOM, */
 802     1               J=R1->EDGEDN; /* BLOSSOM FORMING EDGE */

                    /*************************************************************
                     ********  CASE 1:  1 EDGE INTO PSEUDO NODE  ***************
                     *************************************************************/
 803     1          EX1: IF P->EDGEDN = JIN THEN DO; /* EASY CASE,  ONE EDGE INTO P */
 805     1     1              CALL REROOT(BROOT);
 806     1     1              P1=P->DN;
 807     1     1              CALL REMOVE (P);
 808     1     1              CALL ADDON (P1,BROOT,JIN);
 809     1     1              PX=NULL;SHRNKNG=T; /* GLOBAL VARS FOR ADDBLOS, DEFFIX */
 811     1     1              CALL UPSCAN(BROOT,T,ADDBLOS,T,DEFFIX);
 812     1     1              FREE P->PSEUDO;
 813     1     1         JTST: /*  CAN WE TREAT J IN A NORMAL FASHION? */
                              JCNT,LASTJ=J;
 814     1     1              IF X(J)=0 THEN GO TO A;
 816     1     1              P1=ENDS(J,1);  P2=ENDS(J,2);
 818     1     1              IF P1->DEFIC THEN
 819     1     1                IF P2->DEFIC THEN  /* BOTH ENDS IN DEFIC TREE */
 820     1     1                  GO TO DXCALC;
 821     1     1              ELSE /* P2 NOT  DEFIC */  GO TO GROWSTEP;
                                /*  ELSE P1 IS NOT IN A DEFICIENT TREE */
```

```
/*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

822     1    1              IF P2->DEFIC THEN DO;  /* SWITCH POINTERS */
824     1    2                   P3=P2;P2=P1;P1=P3;
827     1    2                   GO TO GROWSTEP;
828     1    2                   END;
                            /* OTHERWISE NEITHER IS IN A DEFICIENT TREE,  ARE THEY
                               IN DIFFERENT NONZERO COMPONENTS? */
829     1    1                 P3=P1;
830     1    1                 DO WHILE(P3->DN¬=NULL);
831     1    2                   P3=P3->DN;  END;
833     1    1                 R3=P2;
834     1    1                 DO WHILE (R3->DN ¬= NULL);
835     1    2                   R3=R3->DN;   END;
837     1    1                 IF R3¬=P3 THEN /* DIFFERENT CMPNTS */ GO TO GROWSTEP;
                              /*  ELSE WE INDICATE A NONDEFIC BLOSSOM */
839     1    1                 R3->EDGEDN=J;
840     1    1                 CALL UPSCAN(R3,T,BLOSSIND,F);
841     1    1                 GO TO ENDA;
842     1    1          END; /* OF EASY CASE  */

                      /*************************************************************
                       ********  CASE 2:  2 EDGES INTO PSEUDO NODE  *************
                       *************************************************************/
                           /*  NOW HARDER CASE : WE HAVE A DOWN EDGE AND AN UP EDGE */
843     1                  JDN=P->EDGEDN;
844     1                  Q3=P->DN;
845     1                  Q1=ENDS(JDN,1); /* FIND EDGE OF JDN IN THE BLOSSOM */
846     1                  IF Q3 = Q1 THEN Q1=ENDS(JDN,2);
                           /*  Q1 IS TO BE THE NEW ROOT OF THE BLOSSOM */
848     1                  CALL REROOT(Q1);
                           /*  REMOVE TOP PART OF TREE */
849     1                  Q2=P->UP;
850     1                  CALL REMOVE(Q2);
                           /*  ADD BLOSSOM TO THE TREE */
851     1                  CALL REMOVE(P);
852     1                  FREE P->PSEUDO;
853     1                  CALL ADDON (Q3,Q1,JDN);
854     1                  CALL UPSCAN(Q1,T,ADDFIX,F);/* LABEL NODES ODD AND EVEN */
855     1                  IF BROOT->ODD THEN DO; /* THINGS WORK OUT EASILY */
857     1    1        SIMPFIN:  PX=NULL; SHRNKNG=T; /*GLOBAL VARS FOR ADDBLOS-DEFFIX. */
859     1    1                  CALL UPSCAN(Q1,T,ADDBLOS,T,DEFFIX);
860     1    1                  CALL ADDON(BROOT,Q2,JIN); /* ADD THE TOP OF THE TREE */
861     1    1                  GO TO JTST; /*  CONTINUE AS IN EASY CASE. */
862     1    1                  END;
                           /*  OTHERWISE WE MAY HAVE A POLYGON IN THE PATH,  OR WE MAY
                           JUST NEED J IN THE PATH.  FIRST LABEL NODES IN POLYGON */
863     1                  P1=ENDS(J,1);  P2=ENDS(J,2);
865     1                  DO WHILE (P1¬=Q3);
866     1    1                  P1->INPATH=T;  P1=P1->DN;  END;
869     1                  DO WHILE(¬P2->INPATH);
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

870     1    1              P2->INPATH= 1;P2=P2->DN; END;
873     1                R1=P2; /*  ROOT OF THE POLYGON */
                         /*  TURN INPATH OFF IN STEM */
874     1                DO WHILE (P2->DN¬=Q3);
875     1    1              P2=P2->DN;  P2->INPATH=F;  END;
                         /*  NOW P->INPATH = T IFF P IS IN THE POLYGON */
                         /*  IF BROOT IS LABELLED EVEN,  AND THE PATH FROM BROOT
                         TO Q1 CONTAINS AT MOST ONE POLYGON NODE THEN POLYGON
                         IS IN PATH,  OTHERWISE NOT. */
878     1                P1=BROOT;
879     1                DO WHILE(¬P1->INPATH);
880     1    1              P1=P1->DN;
881     1    1              IF P1=Q1 THEN /*  AT MOST ONE PGON NODE IN PATH */
882     1    1                GO TO POLYCASE;
883     1    1              END;
884     1                P2=P1->DN;
885     1                IF ¬P2->INPATH THEN GO TO POLYCASE;
                         /* OTHERWISE ALL WE HAVE TO DO IS REMOVE P1->EDGEDN
                         FROM POLYGON AND REPLACE IT WITH J AND WE CAN TREAT AS
                         SIMPLE CASE */
887     1                P2=ENDS(J,1);
888     1                P3=P2; /*  SEE IF P2 IS END WE WANT FOR ADDON */
889     1                DO WHILE(P3->INPATH);
890     1    1              IF P3=P1 THEN /* CORRECT, SO */  DO;
892     1    2                   R3=ENDS(J,2);
893     1    2                   GO TO FIN1;
894     1    2                   END;
895     1    1              P3=P3->DN;
896     1    1              END;
                         /* OTHERWISE WE HAD IT BACKWARDS */
897     1                R3=ENDS(J,1);
898     1                P2=ENDS(J,2);
899     1           FIN1:J1=P1->EDGEDN;
900     1                CALL REMOVE(P1);
901     1                CALL REROOT(P2);
902     1                CALL ADDON (R3,P2,J);
903     1                J=J1;
904     1                GO TO SIMPFIN;
905     1           POLYCASE: /*  HERE WE HAVE A POLYGON IN PATH, LABEL PATH FROM
                         BROOT TO POLYGON OR STEM CORRECTLY,
                            FIRST MARK NODES IN STEM.  */
                         DO WHILE(R1¬=Q3);
906     1    1              R1->INPATH=T;  R1=R1->DN;  END;
909     1                IF BROOT->INPATH THEN /* NO FIXING NECESSARY */ GO TO WINDUP;
911     1                P1=BROOT;
912     1                DO WHILE (¬P1->INPATH);
913     1    1              P2=P1;  P1=P1->DN;  END;
916     1                P1->ODD=¬P1->ODD;
917     1                CALL UPSCAN(P2,T,ADDFIX,F);
```

```
        /*THE BLOSSOM ALGORITHM:  MAIN PROCEDURE.  16-03-73   */

STMT LEVEL NEST

918     1             P1->ODD=¬P1->ODD;
919     1           WINDUP: CALL ADDON (BROOT,Q2,JIN);
920     1             P1=ENDS(J,1);  P2=ENDS(J,2);
                     /*  SET UP FOR RETURN TO MAIN LOOP. */
922     1             CALL UPSCAN(Q1,T,POLYFIX,F);
923     1             FROMEX=T;
924     1             GO TO DXCALC;

                    /*************************************************************
                     ********  FINAL CORRECTION OF MATCHING IN PSEUDOS  ********
                     *************************************************************/
925     1           CORRECTION: IF TRACE THEN PUT EDIT('*BLOSS - CORRECT MATCHING IN PSEUDO
                    NODES')(SKIP,A);
927     1             IF TRACE THEN CALL XOUT;
929     1             DO I=1 TO NNODE;
930     1     1         P1=ADDR(NODELST(I));
931     1     1       EXP1: IF P1->STACKUP=NULL THEN GO TO EXPEND;
933     1     1             P2=P1->STACKUP;
934     1     1             IF P2-> EXPANDED THEN GO TO EXPEND;
936     1     1             P3=P2->STACKUP;
937     1     1             DO WHILE((P3¬=NULL)&(¬ P3->EXPANDED));
938     1     2                  P2=P3;  P3=P3->STACKUP;
940     1     2                  END;
941     1     1             CALL EXPAND (P2);  /* EXPAND AND  KEEP   THE BLOSSOM */
942     1     1             P2->EXPANDED=T;
943     1     1             GO TO EXP1;
944     1     1          EXPEND:  END;
945     1           END BLOSSOM;
                    /*************************************************************************/
                    /*******************    END OF BLOSSOM ALGORITHM   ********************/
                    /*************************************************************************/
```

References

[B1] M.L. Balinski, K. Spielberg, "Methods for Integer Programming: Algebraic, Combinatorial, and Enumerative", in Progress in Operations Research Vol. III, J. Aronofsky (ed.), Wiley, New York, N.Y. 195-292 (1969).

[B2] C. Berge, "Sur le couplage maximum d'un graph", C.R. Acad. Sci. Paris 247, 258-259 (1958).

[B3] C. Berge, The Theory of Graphs and Its Applications, Methuen, London, England (1962).

[B4] G. Birkhoff, S. MacLean, A Survey of Modern Algebra, Third ed., Macmillan, New York, N.Y. (1965).

[B5] R.G. Busacker, T.L. Saaty, Finite Graphs and Networks, McGraw-Hill, New York, N.Y. (1965).

[C1] C. Carathéodory, "Über den Variabilitätsbereich der Koeffizienten von Potenzreihen, die gegebene Werte nicht annehment", Math. Ann. 64, 95-115 (1907).

[D1] G. B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, N.J. (1963).

[E1] J. Edmonds, "Paths, Trees and Flowers", Canadian J. Math. 17, 449-467 (1965).

[E2] J. Edmonds, "Maximum Matching and a Polyhedron with 0, 1-vertices", J. Res. Nat. Bur. of Standards 69B (Math. and Math. Phys) No. 1, 125-130 (1965).

[E3] J. Edmonds, "An Introduction to Matching" Notes on lectures given at Ann Arbor, Michigan (1967)

[E4] J. Edmonds, "Optimum Matchings", in manuscript.

[E5] J. Edmonds, E.L. Johnson, "Matching: A Well-Solved Class of Integer Linear Programs", preprint: summary appears in Combinatorial Structures and their Applications, Gordon and Breach, New York, N.Y. 89-92 (1970).

[E6] J. Edmonds, E.L. Johnson, "Matching, Euler Tours and the Chinese Postman", I.B.M. Research Report RC 3783 (1972), to appear in Math. Programming.

[E7] J. Edmonds, E.L. Johnson, S. Lockhart, "Blossom I: A Computer Code for the Matching Problem", to appear.

[G1] B. Grünbaum, Convex Polytopes, Interscience, London, England (1967).

[H1] G. Hadley, Linear Programming, Addison-Wesley, Reading, Mass. (1962).

[H2] F. Harary, Graph Theory, Addison-Wesley, Reading, Mass. (1969).

[I1] E. Isaacson, H. Keller, Analysis of Numerical Methods, John Wiley and Sons, New York, N.Y. (1966).

[I2] I.B.M. Systems 1360 Operating System, PL/1(F) Language Reference Manual, C28-8201 (1970)

[J1] E.L. Johnson, "Programming in Networks and Graphs", Univ. of Calif., Berkely Research Report ORC 65-1 (1965).

[J2] E.L. Johnson, "Networks and Basic Solutions", Operations Research 14, 619-623 (1966).

[K1] V. Klee, C. Witzgall, "Facets and Vertices of Transportation Polytopes", Boeing Scientific Rsch. Lab. Doc. D1-82-0662, (1967).

[K2] D. König, Theorie der endlichen und unendlichen Graphen, Acad. Verl. M.B.H., Leipzig (1936). Reprint, Chelsea Publishing Company, New York, N.Y. (1950).

[K3] D. E. Knuth, The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison-Wesley, Reading, Mass. (1968).

[R1] R.T. Rockafellar, Convex Analysis, Princteon University Press, Princeton, N.J. (1969).

[S1] J. Stoer, C. Witzgall, Convexity and Optimization in Finite Dimensions I, Springer-Verlag, Berlin, Heidelberg (1970).

[T1] W.T. Tutte, "The Factorization of Linear Graphs", J. London Math. Soc. 22, 107-111 (1947).

[T2] W.T. Tutte, "The Factors of Graphs", Canadian J. Math. 4, 314-328 (1952).

[T3] W.T. Tutte, "A Short Proof of the Factor Theorem for Finite Graphs", Canadian J. Math. 6, 347-352 (1954).