stichting
mathematisch
centrum

AFDELING MATHEMATISCHE BESLISKUNDE BW 25/73 JULY

B.J. LAGEWEG

AN ALGORITHM FOR A MAXIMUM WEIGHTED COMMON PARTIAL TRANSVERSAL

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS(MOS) subject classification scheme (1970): 05B40, 90C35

Table of contents

0. Abstract

An efficient primal-dual algorithm is presented for determining a common partial transversal of two families of sets, maximizing the sum of the weights of its elements. The problem is a generalization of the linear assignment problem and a specialization of the problem of determining a maximum weighted common independent set of two matroids. An algorithm for determining a common partial transversal of maximum cardinality is inferred from the former one.

1. Introduction

$E = \{e_1,\dots,e_k,\dots,e_p\}$ is a finite set of elements. Let element e_k have a real valued weight c_k. The weight of a subset $F \subseteq E$ is the sum of the weights of the elements of F. Let $A = (A_1,\dots,A_i,\dots,A_m)$ and $B = (B_1,\dots,B_j,\dots,B_n)$ be two families of non-empty subsets of E.
A subset F of E is called a tranversal of family A if F consists of m distinct elements of E, one from each set A_i. F is a partial transversal of A if F is a tranversal of a subfamily of A. A common partial transversal of A and B is a set which is a partial transversal of both A and B.
The subject of this report is: determine a common partial transversal of A and B with a maximum weight.

An example of the problem is the following. E is a set of jobs, each to be processed during one time unit. The weights c_k indicate the priority of the jobs. A_i is the set of jobs that work party i is qualified to process. B_j is the set of jobs that can be processed on machine j. A maximum common partial transversal provides a set of jobs with maximum total priority, each job processed by a qualified work party on a suitable different machine.

It is wellknown that the partial transversals of family A are the independent sets of a matroid on E, a so-called transversal matroid. An efficient algorithm for determining a partial transversal of A with maximum weight therefore is the greedy algorithm [3].
The subject of this report is how to find a maximum common independent set of two transversal matroids. The more general problem of finding a maximum common independent set of two matroids, without restrictions on the kind of matroids involved, has been solved efficiently by Edmonds [2] and Lawler [6]. Some special classes of the latter problem have been solved efficiently also, e.g. the problem of constructing a maximum weighted directed tree in a directed graph, equivalent to finding a maximum common independent set of a graph matroid and a special transversal matroid (Edmonds [1]).
We here present an efficient algorithm for the maximum common partial transversal problem. If the subsets of family A are mutually disjoint, and the same holds for family B, the problem reduces to the linear assignment

problem and the algorithm simplifies to the Hungarian method [4][5].

In section 2 a linear programming formulation of the problem and its dual are used to state optimality conditions. The algorithm starts with constructing an initial primal and dual solution. The algorithm then searches for improved primal solutions given the dual solution, until no better one is found. In that case an improved dual solution is constructed given the operative primal one and the algorithm restarts searching for primal improvements. In section 3 this process is described as a search for an augmenting path in a directed tree, growed by means of a labelling process. Section 4 provides the proof of the method, in particular concerning its efficiency. In section 5 the case $c_k = 1$ for all elements of E is treated, i.e. an algorithm for determining a common partial transversal with maximum cardinality is derived from the preceding sections. Section 6 finally deals with details of the implementation of the algorithm and lists some computational results.

2. Description of the algorithm

We associate with the common transversal problem a graph G as follows: the vertex-set of G consists of the subsets $A = \{a_i \mid A_i \in \mathcal{A}\}$, $B = \{b_j \mid B_j \in \mathcal{B}\}$, $EA = \{ea_k \mid e_k \in \mathcal{E}\}$ and $EB = eb_k \mid e_k \in \mathcal{E}\}$; the edge-set D(G) of G contains an edge (ea_k, eb_k) for $k = 1,\ldots,p$, an edge (a_i, ea_k) for each $e_k \in A_i$ and an edge (b_j, eb_k) for each $e_k \in B_j$.

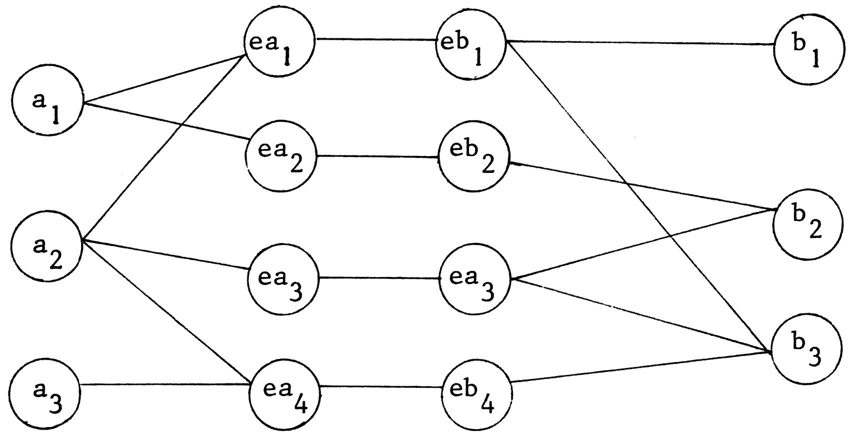


Fig.1: An example of graph G.

Definition. A set of edges M of the graph G is called a matching, if

1) at most one edge of M is incident to vertex x, for each $x \in A \cup B$;
2) if any edge of M is incident to $ea_k \in EA$ ($eb_k \in EB$), then exactly two edges of M are incident to ea_k (eb_k), one of which is (ea_k, eb_k).

We shall denote the set of vertices of G, incident to a matching M, by M_v.

If we assign weight c_k to edges (ea_k, eb_k) and weight zero to the other edges of G, the maximum common partial transversal problem clearly is equivalent to finding a matching (in the sense of the above definition) with a maximum weight in G.

We also can formulate the transversal problem as a zero-one linear programming problem. Let a variable x_{ijk} be one if element $e_k \in \mathcal{E}$ links subset A_i of family $\mathcal{A}$ with subset B_j of family $\mathcal{B}$, and zero else. The i.p.-

formulation of the transversal problem reads:

(1) Maximize $\sum_{k=1}^{p} \sum_{i|k\in A_i} \sum_{j|k\in B_j} c_k x_{ijk}$

subject to:

$$\sum_{k\in A_i} \sum_{j|k\in B_j} x_{ijk} \le 1, \ i = 1,\ldots,m.$$

$$\sum_{k\in B_j} \sum_{i|k\in A_i} x_{ijk} \le 1, \ j = 1,\ldots,n.$$

$$\sum_{i|k\in A_i} \sum_{j|k\in B_j} x_{ijk} \le 1, \ k = 1,\ldots,p.$$

(2) $x_{ijk} \ge 0, \ i = 1,\ldots,m; \ j = 1,\ldots,n; \ k = 1,\ldots,p.$

(3) x_{ijk} integer, $i = 1,\ldots,m; \ j = 1,\ldots,n; \ k = 1,\ldots,p.$

A solution (x_{ijk}) of (2) - (3) corresponds with a matching M in the graph G and vice versa:

$$x_{ijk} = 1 \iff (a_i, ea_k) \in M, \ (ea_k, eb_k) \in M, \ (eb_k, b_j) \in M.$$

The dual problem of linear programming problem (1) - (2) reads:

(4) Minimize $\sum_{i=1}^{m} u_i + \sum_{j=1}^{n} v_j + \sum_{k=1}^{p} w_k$

subject to:

$$(5)\quad \begin{cases} u_i + v_j + w_k \geq c_k, & k = 1,\dots,p; \\ & i \mid k \in A_i; \\ & j \mid k \in B_j. \\ u_i \geq 0, & i = 1,\dots,m. \\ v_j \geq 0, & j = 1,\dots,n. \\ w_k \geq 0, & k = 1,\dots,p. \end{cases}$$

The duality theorem of linear programming says, that a solution (x_{ijk}) satisfying (2) is optimal if and only if a solution (u_i,v_j,w_k) satisfying (5) exists such that the criterion values (1) and (4) are equal. Assuming (x_{ijk}) satisfies (3) also, there is a matching M with vertex set M_v corresponding with (x_{ijk}). The equality of the criterium values holds iff:

$$(6)\qquad x_{ijk} > 0 \Longrightarrow u_i + v_j + w_k = c_k$$

$$(7)\qquad u_i > 0 \Longrightarrow a_i \in M_v$$

$$(8)\qquad v_j > 0 \Longrightarrow b_j \in M_v$$

$$(9)\qquad w_k > 0 \Longrightarrow (ea_k, eb_k) \in M.$$

Thus we can solve the maximum common partial transversal problem by finding a pair of dual solutions (x_{ijk}) and (u_i,v_j,w_k) satisfying the optimality conditions (2), (3), (5), (6), (7), (8) and (9).

The algorithm starts with solutions violating only some of the conditions (7):

$$\begin{cases} M = \emptyset \quad (\Longleftrightarrow (x_{ijk}) = (0)). & \\ u_i = \max\{0, \max_{k \in A_i} c_k\} & , i = 1,\dots,m. \\ v_j = 0 & , j = 1,\dots,n. \\ w_k = 0 & , k = 1,\dots,p. \end{cases}$$

During the execution of the algorithm the conditions (7) are fulfilled one by one, and that by constructing an augmenting path in G, while any once satisfied condition remains satisfied.

Let S and T be sets of edges of G. We define the complement $\bar{S}(T)$ of S with respect to T as

$$\bar{S}(T) \equiv (S \cup T) \setminus (S \cap T),$$

i.e. edges of T, not yet belonging to S, are added to S and the other edges of T are removed out of S.
A path $P = (x_0,\ldots,x_r)$ in G is a sequence of vertices of G such that (x_i,x_{i+1}) is an edge of G and each vertex of G occurs at most once in P.
A path P of the form $(x_0,\ldots,x_{r-1},a_i)$ is called an augmenting path (AP) for a matching M, if $\bar{M}(\{(x_0,x_1),\ldots,(x_{r-1},a_i)\})$ satisfies all optimality conditions so far satisfied and in addition condition i of (7).

By constructing AP's successively ending in $a_1,a_2,\ldots$ and a_m, the algorithm will reach a pair of optimal solutions (x_{ijk}) and (u_i,v_j,w_k), and thus an optimal solution of the transversal problem.

3. The construction of an augmenting path

Let we have at our disposal a dual solution (u_i,v_j,w_k) and a matching M, corresponding with an integer solution (x_{ijk}). The pair of solutions satisfies all optimality conditions except some of (7). a_0 is a vertex of G which violates (7), i.e. $u_0 > 0$ and $a_0 \notin M_v$. If there is no such a vertex, the present matching is optimal and the algorithm ends. Else a_0 becomes the root of an arborescence, i.e. a directed tree, a part of which finally is meant to result in an AP.

Before describing the construction of an AP we need one more definition.

Definition. An edge (a_i,ea_k), resp. (ea_k,eb_k), resp. (eb_k,b_j) of G is called admissible with respect to a dual solution (u_i,v_j,w_k), if there is a path (a_i,ea_k,eb_k,b_j) in G, such that $u_i + v_j + w_k = c_k$.

Each edge of an AP belongs to a matching - either M or $\bar{M}(\{AP\})$ -, which we want to satisfy the conditions (6). Henceforth, only admissible edges are candidates for inclusion into an arborescense.

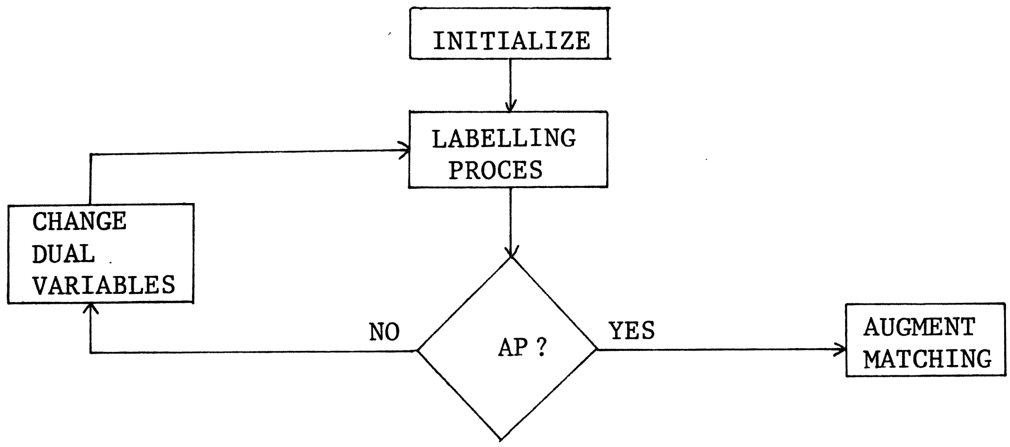


Fig.2. The construction of an AP

We now can describe the construction of an AP and the subsequent augmentation of the matching (fig. 2). Firstly a labelling process grows an arborescence rooted at a_0. When an AP is met during the labelling process, the labelling halts and a new matching is found by taking the complement of M with

respect to AP. If the labelling process ends by exhaustion, the growed arborescence has a maximum number of edges given the present dual solution. Now the values of (some of) the dual variables are changed, in that way that the labelling process can be continued, restarting from the arborescence growed before. After a finite number of changes of the dual variables the arborescence will contain an AP.
The labelling process below assigns labels to the vertices of G. The state of a vertex is either labelled or unlabelled. Two vertices are in the same state if they are both labelled or both unlabelled. Else they are in a different state.

We now state the algorithm for constructing an augmenting path, ending in a_0.

LABELLING PROCESS:

L1: Initialize all vertices of G as unlabelled and unscanned. Assign to a_0 label [0].

L2: Select a labelled, unscanned vertex x.
If there is no such a vertex the arborescence is maximal with regard to the present dual solution: go to DUAL VARIABLE CHANGE.
Vertex x becomes scanned; if $x \in A$ go to L3;
if $x \in EA$ go to L4; if $x \in EB$ go to L5;
if $x \in B$ go to L6.

L3: Let $x = a_i$. If $u_i = 0$, an AP has been found, starting in a_i: go to BREAKTHROUGH.
Assign label [i] to all unlabelled vertices $ea \in EA$, incident to an admissible edge (a_i, ea).
Go to L2.

L4: Let $x = ea_k$. If ea_k is matched, assign then label [k] to the match of ea_k in A, i.e. to vertex $a_i \in A$ such that $(a_i, ea_k) \in M$.
Else assign label [0] to vertex eb_k.
Go to L2.

L5: Let $x = eb_k$. Assign label [k] to all unlabelled vertices $b \in B$, incident to an admissible edge (eb_k, b).
If ea_k unlabelled and $w_k = 0$, assign then label [0] to ea_k.
Go to L2.

L6: Let $x = b_j$. If b_j is unmatched, an AP has been found, starting in b_j: go to BREAKTHROUGH.
Assign label [j] to the match of b_j in EB, i.e. to vertex $eb_k \in EB$ such that $(eb_k, b_j) \in M$.
Go to L2.

DUAL VARIABLE CHANGE:

D1: Compute $d_1 = \min_i \{u_i \mid a_i \text{ labelled}\}$.

D2: Compute $d_2 = \min_i \{ \min_{(k,j)} \{u_i + v_j + w_k - c_k \mid (a_i, ea_k) \in D(G), (eb_k, b_j) \in D(G), ea_k \text{ unlabelled}, eb_k \text{ and } b_j \text{ in the same state}\} \mid a_i \text{ labelled}\}$.

D3: Compute $d_3 = \min_k \{w_k \mid ea_k \text{ unlabelled}, eb_k \text{ labelled}\}$.

D4: Compute $d_4 = \min_k \{\min_j \{v_j \mid (eb_k, b_j) \in D(G), b_j \text{ unlabelled}\} - \min_j \{v_j \mid (eb_k, b_j) \in D(G), b_j \text{ labelled}\} \mid eb_k \text{ labelled}\}$.

D5: $d = \min \{d_1, d_2, d_3, d_4\}$.
Change the dual variables: if $(u_i', v_j'. w_k')$ denote the dual variables after the change, then:

$$u_i' = \begin{cases} u_i - d & , a_i \text{ labelled} \\ u_i & , \text{else.} \end{cases}$$

$$v_j' = \begin{cases} v_j + d & , b_j \text{ labelled} \\ v_j & , \text{else.} \end{cases}$$

$$w'_k = \begin{cases} w_k + d & , ea_k \text{ labelled, } eb_k \text{ unlabelled} \\ w_k - d & , ea_k \text{ unlabelled, } eb_k \text{ labelled} \\ w_k & , ea_k \text{ and } eb_k \text{ in the same state.} \end{cases}$$

D6: Determine the vertices of G which are starting-points for a further labelling by transferring them into the collection of unscanned vertices:

1. If $d = d_r$, each labelled vertex a_{i_0} such that for $i = i_0$ the minimum d_r is assumed in Dr (r = 1,2);
2. If $d = d_r$, each vertex eb_{k_0}, such that for $k = k_0$ the minimum d_r is assumed in Dr (r = 3,4).

D7: Go to L2.

BREAKTHROUGH:

Construct the augmenting path by running back in the growed arborescence from the breakthrough-point a_i or b_j into the reverse direction of the edges of the arborescence until the root a_0 is reached, i.e.:

go from a vertex $b \in B$ to ea_k if label $b_j = [k]$;

go from a vertex $eb_k \in EB$ to $\begin{cases} b_j \text{ if label } eb_k = [j],\ j > 0; \\ ea_k \text{ if label } eb_k = [0]; \end{cases}$

go from a vertex $ea_k \in EA$ to $\begin{cases} a_i \text{ if label } ea_k = [i],\ i > 0, \\ eb_k \text{ if label } ea_k = [0]; \end{cases}$

go from a vertex $a \in A$ to ea_k, if label $a = [k]$, $k > 0$, else stop.

Determine the new matching by removing the edges of the AP, already belonging to M, out of M, and adding the other edges of the AP to M.

4. Proof of correctness and efficiency

Firstly we prove some properties of the change of the dual variables (DVC), among others that a DVC does not violate any already satisfied optimality condition. Afterwards we show that an AP is known after the execution of finitely many DVC's. The correctness and finiteness of the algorithm immediately follow.

We start with a trivial observation. Let $u_i + v_j + w_k \geq c_k$ for each path (a_i,ea_k,eb_k,b_j) of G. If a_{i_0} is incident to the admissible path (a_{i_0},ea_k,eb_k,b_j), more precisely, if a_{i_0} is incident to the admissible edge (a_{i_0},ea_k) and $u_{i_0} + v_j + w_k = c_k$, then

$$u_{i_0} = \min_i \{u_i \mid (a_i,ea_k) \in D(G)\}.$$

The dual variables u_i of all vertices $a_i \in A$, adjacent to a vertex ea_k by an admissible edge, are equal. Likewise, if b_{j_0} is incident to the admissible path (a_i,ea_k,eb_k,b_{j_0}) then

$$v_{j_0} = \min_j \{v_j \mid (eb_k,b_j) \in D(G)\}.$$

<u>Lemma 1</u>: If an edge (a_i,ea_k), resp. (eb_k,b_j), of G belongs to M, then the vertices a_i and ea_k, resp. eb_k and b_j, are in the same state.

<u>Proof</u>: If ea_k has received a label, then also a_i (see L4). No matched vertex in A can receive a label unless his match in EA previously received a label.
A similar reasoning proves the lemma with respect to (eb_k,b_j).

<u>Lemma 2</u>: A DVC does not violate any optimality condition already satisfied.

<u>Proof</u>: The optimality conditions (2) and (3) are independent of the dual solution. Conditions (7), (8) and (9) are easily checked: a DVC makes no u_i positive (7); v_j becomes positive only if b_j is labelled, but then $b_j \in M_v$ (8); w_k becomes positive iff ea_k is labelled and eb_k unlabelled, but then $(ea_k,eb_k) \in M$ (9).

The change of the dual variable w_k can be split up in the change due to the state of ea_k and the change due to eb_k. Lemma 1 implies that if $x_{ijk} > 0$ then a_i and ea_k are in the same state, and eb_k and b_j also. If the two vertices, incident to an edge of G, are in the same state, the changes in the corresponding dual variables, due to the state of those vertices, are zero or equal in absolute value with opposite sign, so their sum is zero. Hence if $x_{ijk} > 0$, the resulting change of $u_i + v_j + w_k$ is zero and (6) is not violated.

u_i and w_k do not become negative by a DVC because of D1, D3 and D5. The remaining check concerns the nonnegativity of $u_i + v_j + w_k - c_k$ on each path (a_i, ea_k, eb_k, b_j) of G. We split up the change in $u_i + v_j + w_k$ in the change due to the states on (a_i, ea_k) and the change due to the states on (eb_k, b_j). Problems only arise if a decrease of the dual variables due to one of both edges is not compensated by the other one. We discern three cases:

1. a_i labelled, ea_k unlabelled, eb_k and b_j in the same state: condition (5) holds because of D2;
2. eb_k labelled, b_j unlabelled, a_i and ea_k in the same state: in this case eb_k is incident to an admissible path, say $(a_{i_1}, ea_k, eb_k, b_{j_1})$. Furthermore, b_{j_1} is labelled, either according to L5, or due to L6. Lemma 1 and D4 imply $v'_j \geq v'_{j_1}$ after the DVC. On the path $(a_i, ea_k, eb_k, b_{j_1})$ the vertices are pairwise in the same state, hence the total change of the dual variables is zero and

$$u'_i + v'_j + w'_k \geq u'_i + v'_{j_1} + w'_k = u_i + v_{j_1} + w_k \geq c_k;$$

3. a_i and eb_k labelled, ea_k and b_j unlabelled: in this case eb_k must have a label $\neq 0$, say $[j_1]$.
   b_{j_1} is labelled and on an admissible path, so according to D4 $u'_1 + v'_j + w'_k \geq u'_i + v'_{j_1} + w'_k$. The path $(a_i, ea_k, eb_k, b_{j_1})$ falls

under the first case above, hence $u_i' + v_j' + w_k' \geq c_k$.
This suffices to check (5) and concludes the proof of the lemma.

Lemma 3: The dual solution after a DVC admits the same arborescence as the solution before the DVC.

Proof: The value of the dual variables is only relevant to the labelling by L3 and L5.
Concerning L3, let ea_k have received a label from a_i, and (a_i, ea_k, e_{bk}, b_j) be an admissible path before the DVC. Now a_i and ea_k have the same state. If $ea_k \notin M_v$ then eb_k and b_j both are labelled according to L4 and L5, and the total change of the dual variables is zero, so (a_i, ea_k) still is admissible after the DVC. Else $ea_k \in M_v$ and $eb_k \in M_v$ and there is a vertex b_{j_1} with $(eb_k, b_{j_1}) \in M$, hence eb_k and b_{j_1} are in the same state and we see $(a_i, ea_k, eb_k, b_{j_1})$ is admissible before and after the DVC, having in mind the observation at the beginning of this section.
Concerning L5, let b_j have received a label from eb_k. If ea_k is labelled there is a path (a_i, ea_k, eb_k, b_j) fully labelled. Else there is a labelled vertex b_{j_1}, the match of eb_k in B. Before and after the DVC (eb_k, b_{j_1}) is admissible, $v_j = v_{j_1}$ and $v_j' = v_{j_1}'$, so (eb_k, b_{j_1}) is admissible. The last thing to check is that w_k remains zero if ea_k has received a label from eb_k. Well, ea_k and eb_k then are in the same state and D5 completes the proof.

Lemma 3 says that the old arborescence remains valid. The next thing to show is that the arborescence actually can be expanded after a DVC, or hids an AP so the labelling can be ended. Before proving this we show that no vertex twice receives a label. Vertices in EA or B only receive a label if they are unlabelled, so at most once. Vertices $a_i \in A$, except the root a_0, only receive a label from their match in EA, namely when that match, labelled and unscanned, is selected in L2. Once scanned, a vertex $ea \in EA$

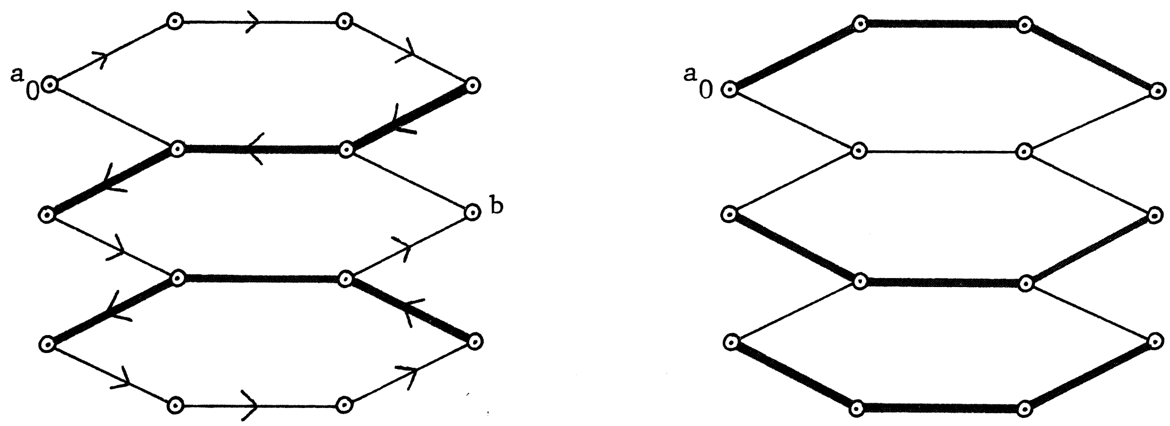
never becomes unscanned again. A vertex $eb_k \in EB$ can receive a label from ea_k if $eb_k \notin M_v$, or else from its match in B. For both vertices holds that they are only once selected in L2.
At the same time we observe that the labelled vertices, except the root a_0, can be divided into pairs, incident to the same edge of G. With any labelled vertex b_j corresponds its match in EB; with any labelled vertex ea_k corresponds either its match in A or eb_k. The changes of the dual variables of the paired vertices compensate each other in the dual criterion function (4). The total decrease of (4) by a DVC therefore is equal to d. Clearly $d > 0$, because otherwise the continuation of the labelling, as indicated by lemma 4 below, could have taken place before the DVC.
If the coefficients c_k are integral valued, $d \geq 1$ holds, and the maximum number of DVC's is equal to the criterion value of the initial dual solution.

Lemma 4: After a DVC either an AP is found before the labelling ends by exhaustion, or at least two more vertices receive a label.

Proof: The in D5 computed value d is equal to d_1, d_2, d_3 or d_4.
If $d = d_1$, a labelled vertex a_{i_0} with $u_{i_0} = 0$ is declared unscanned in D6 and we meet the situation in L3 that ends the labelling. If $d = d_2$, a labelled vertex a_{i_0} is adjacent to an unlabelled vertex ea_k by an edge which becomes admissible by executing the DVC. So ea_k and either eb_k or the match of ea_k in A receive a label.
If $d = d_3$, a w_{k_0} becomes zero and ea_{k_0} and its match in A receive labels.
If $d = d_4$, an edge (eb_{k_0}, b_j) with b_j unlabelled, becomes admissible after the DVC. Then either the match of b_j in EB receives a label, viz. if $b_j \in M_v$, or the labelling ends in situation L5. The proof that an AP really has been found when the labelling ends in the situations L3 or L5, is given in lemma 5.

<u>Lemma 5</u>: The arborescence will contain an AP after the execution of finitely many DVC's.

<u>Proof</u>: If the graph G has v vertices, lemma 4 and the fact that each vertex only at most once receives a label, imply that after (v-1)/2 DVC's a vertex a_i with $u_i = 0$, or a unmatched vertex b_j, or all vertices have received a label. In the latter case another DVC will yield a minimum $d = d_1$ for $i = i_0$, and u_{i_0} becomes zero. Thus after finitely many DVC's the algorithm will turn up at the breakthrough-routine. Yet to show is that the path P determined by this routine, conforms to the definition of an AP.



Matching M and AP with start in b, ending in root a_0

Matching M, constructed by taking the complement of M with respect to the AP

——— edge, not in matching

▬▬▬ edge, in matching

> direction of labelling of an edge in AP

<u>Fig. 3</u>. Example of an AP

First of all we check that taking the complement of the matching M with respect to P, produces a new matching $\bar{M}$, i.e. $\bar{M}$ satisfies (2) and (3) (figure 3).

Now each vertex of G occurs at most once in P, because each vertex has received a label at most one. We restrict our attention to vertices of G, incident to P, because to the other vertices of G the same edges of M and $\bar{M}$ are incident.

The root a_0 and a starting point b_j of P do not belong to M_v, so exactly one edge of $\bar{M}$ is incident with any of then. A starting point a_i of P belongs to M_v and not to $\bar{M}_v$.

To a vertex $x \in A \cup B$ on P, except the extremal vertices of P, two edges of P are incident. One of then is the edge of M incident to x and hence not contained in $\bar{M}$. Then the other one cannot belong to M and therefore belongs to $\bar{M}$. So exactly one edge of $\bar{M}$ is incident to x and $x \in \bar{M}_v$.

If ea_k is incident to P and $(ea_k,eb_k) \notin M$, there exists a path $(b_j,eb_k,ea_k,a_i) \subseteq P$, of which the edges are not contained in M. Those edges are the only ones incident to ea_k or eb_k in $\bar{M}$.

If ea_k is incident to P and $(ea_k,eb_k) \in M$, either there exists a path $(a_i,ea_k,eb_k,b_j) \subseteq P$, and the edges of this path belong to M and not to $\bar{M}$, or $(ea_k,eb_k) \notin P$.

In the latter case $(ea_k,eb_k) \in \bar{M}$. The vertex ea_k now is incident to P in a way as above described for $x \in A \cup B$: one of the two edges of P incident to ea_k belongs to M and the other one not, and after the breakthrough the same holds for $\bar{M}$, after exchanging the edges.

As the same considerations hold in the case of eb_k, $\bar{M}$ really is a matching.

The breakthrough does not concern the dual solution, hence with respect to condition (5) nothing changes. The other conditions, namely (6), (7), (8) and (9), remain satisfied by the construction of the arborescence. For it consists of admissible edges, which suffices to maintain (6). A vertex b_j, once incident to a matching, is incident to all succeeding matchings, and that proves

(8). If $w_k > 0$, the arborescence and hence the AP does not contain (ea_k, eb_k), so $(ea_k, eb_k) \in \bar{M}$ (9). If condition (i) of (7) was satisfied before the breakthrough, either $u_i > 0$, so a_i is no starting point of the AP and $a_i \in \bar{M}$, or $u_i = 0$. The same reasoning holds for the root a_0 of the arborescence: $a_0 \in \bar{M}$ or $u_0 = 0$. This completes the check of the optimality conditions and proves an AP has been found.

Theorem: The algorithm computes a maximum matching in a efficient way.

Proof: The above lemma's imply that after constructing at most m AP's a maximum matching has been found.
The efficiency has been proven, if the order of the amount of work to construct an AP is shown to be polynomial in n, m and p.
During the growth of an arborescence at most $m + n + 2p$ vertices receive a label. Labelling from vertices in $EA \cup B$ requires a time proportional to $p + n$, and labelling from a vertex $a \in A$ a time proportional to $p * n$, included the time to establish the admissibility of an edge. The time due to assigning a label to a vertex of A, also is proportional to $p * n$. Hence the total time for assigning labels to vertices of A and from vertices of A, is proportional to $m * p * n$. In a similar way the time involved in labelling vertices of EB is proportional to $p * n$. So the labelling requires a time of the order $m * n * p$. The time for determining a new matching is proportional to the length of the AP and therefore at most of the order of the labelling.
The order of time of a DVC depends on the computation of the minima d_2 and d_4, hence is $m * n * p$. As after at most $(m+n+2p)/2$ DVC's an AP has been met, the time needed for DVC's and also the time for constructing an AP, is proportional to
$(m+n+p) * m * n * p$. Q.E.D.

The algorithm itself therefore is of the order $m^2np\ (m+n+p)$.
Assuming $O(m) = O(n)$ and $p >> m$, the total computing time is proportional to m^3p^2.
In section 6 a tighter bound will be given.

5. A common partial transversal with maximum cardinality

We can determine a common partial transversal of A and B with maximum cardinality, i.e. with a maximum number of elements of E, by applying the algorithm of section 2 and 3 with $c_k = 1$ for all elements of E. The case however allows some simplifications.

To see this, assume we proceed according sections 2 and 3. We then construct an initial dual solution with $u_i = 1$ for all i, because each set A_i is non-empty. Next we start growing an arborescence, say with root a_0. Remark all edges up to now are admissible. If an unmatched vertex b_j receives a label, an AP has been found and we continue by determining the new matching and growing again an arborescence. The alternative is that the labelling ends by exhaustion, as all u_i are 1 and no a_i with $u_i = 0$ can receive a label. In that case the algorithm executes a DVC, resulting in $d = 1$ and $u_0 = 0$ after the DVC. So an AP containing zero edges has been found, starting and ending in a_0. In the sequel of the algorithm a_0 is unmatched and never will receive a label again, hence we know a_0 will be unmatched in the optimal solution.

Instead of executing a DVC and a breakthrough with AP = $\emptyset$, we therefore remove a_0 out of G, together with all edges of G, incident to a_0. In the reduced graph G' we maintain the initial dual solution (except $u_0 = 1$) and the matching of G. We continue the algorithm by growing a new arborescence in G' with root at any unmatched a_i.

We observe that this method always maintains the same dual solution in the operative graph G': u_i for all a_i in G', $v_j = 0$ for all j and $w_k = 0$ for all k. All edges present in G' are admissible and we do not need explicitly the dual solution.

Summarized the modifications with respect to the algorithm for growing an arborescence, presented in section 3, are the following:

The root a_0 is an unmatched vertex in the present, possibly reduced graph G;

L3: No breakthrough can occur, all u_i being one;
assigns labels to all unlabelled vertices adjacent to a_i;

L5: Assign labels to all unlabelled vertices adjacent to eb_k, included ea_k;

DUAL VARIABLE CHANGE:

Stop growing the arborescence. Construct a reduced graph G' by removing out of G vertex a_0 and all edges incident to a_0.

BREAKTRHOUGH:

The starting vertex of an AP always is a vertex $b \in B$.

The order of the above algorithm depends on the time for growing a maximum arborescence in G. Each edge adjacent to $x \in A \cup EB$ once is scanned, viz. during the labelling from x; the time needed for labelling from $x \in EA \cup B$ is proportional to $|EA| + |B|$ [1)] $= p + m$. Because $|B_j| \geq 1$, the total labelling time is proportional to the number of edges of G,

$$p + \sum_{i=1}^{m} |A_i| + \sum_{j=1}^{n} |B_j|.$$

The computing time of the maximum cardinality algorithm is proportional to the number of sets in family $\mathcal{A}$ times the number of edges of G.

Another kindred problem is how to select among the common partial transversals of $\mathcal{A}$ and $\mathcal{B}$ with maximum cardinality the one with maximum weight That set could be determined by applying the algorithm of section 2 and 3, replacing the original weights c_k by $c_k' = c_k + M$. M must be chosen that large that the partial transversal with more elements weights more than the one with less elements, e.g.

$$M = \sum_{k=1}^{p} |c_k|.$$

A more direct approach however is possible. Suppose the weights c_k' are

1) Y denotes the cardinality of a set Y.

used in the algorithm of sections 2 and 3. The initial values of $u_1,\dots,u_m$ are in the order of M. During a DVC the minimum d will be equal to d_2, d_3 or d_4, and not of order M, unless all sets involved with the computation of d_1, d_2 and d_3 are empty. In the latter case $d = d_1$ holds, say for $i = i_0$. We know that a_{i_0} will not be matched in the final solution.

Instead of adjusting the dual solution and afterwards computing a new matching, we can proceed as follows. We remark the method mentioned below can be applied also in the standard algorithm of section 2 and 3, and actually is applied in the ALGOL-procedure of the appendix.

We immediately determine the new matching $\bar{M}$ by means of the AP starting in a_{i_0}. Next we remove a_{i_0} and its incident edges from the graph G, and thus get a reduced graph G'. $\bar{M}$ satisfies condition (0) of (7) and all optimality conditions so far satisfied except condition (i_0) of (7). That condition however does not occur in G' because of the absence of a_{i_0} in G'. We thus evade a change of the dual variables of order M, and the variables u_i in G' still are of order M.

We do not have to specify M. We set $u_i = \max_k \{c_k \mid (a_i, ea_k) \in D(G)\}$ in the initial dual solution.

During a DVC we first of all compute $d' = \min \{\infty, d_2, d_3, d_4\}$. If $d' < \infty$ we adjust the dual solution and continue the labelling as usual. Else we compute $u_{i_0} = \min_i \{u_i \mid a_i \text{ labelled}\}$ and proceed as described above. When we employ the values of u_i derived from the original c_k, the labelling of a vertex a_i with $u_i = 0$ does not imply an AP has been found, as u_i really stands for $u_i + M = M > 0$.

6. Computational aspects

The algorithms of sections 2 and 3 and section 5 have been implemented in ALGOL and tested on the EL-X8 computer of the Mathematical Centre.
Some results for a series of small problems are given in table 1. In the randomly generated problems the expectation of the number of elements in a set A_i or B_j varies between 2 and 5.

Table 1.

p	m = n	Max. CPT algorithm		Max. card. algorithm
		DVC's	time [1)]	time [1)]
100	25	7	1.3	.4
		26	5.5	.9
100	50	32	5.5	.9
		70	18.7	2.1
100	75	24	4.5	1.6
		93	26.9	3.1

1) in seconds

We further restrict us in this section to the procedure for the maximum common partial transversal problem, included in the appendix.
It uses a recursive labelling, which enables us to code the labelling rather shortly. A different order of labelling, e.g. alternately from unscanned vertices in A and EB, could be preferable.

A second feature of the procedure is how it decides an edge to be admissible. The procedure initially computes, and if necessary updates, a set of pointers rb(k), k = 1,...,p, satisfying

$$v_{rb(k)} = \min_j \{v_j \mid (eb_k, b_j) \in D(G)\}.$$

An edge (a_i, ea_k) is admissible if $u_i + v_{rb(k)} + w_k = c_k$. An edge (eb_k, b_j) is admissible if $v_j = v_{rb(k)}$ and $(eb_k, b_{rb(k)})$ is admissible; we only have to decide if (eb_k, b_j) is admissible, when we already know there exists an admissible path $(a_i, ea_k, eb_k, b_{j_1})$ through eb_k, i.e. we know $(eb_k, b_{rb(k)})$ is admissible.
If eb_k is matched to b_j, the procedure sets rb(k) = j. For the rest rb(k) has to be updated only after executing a DVC. If for an index k $(ea_k, eb_k) \in M$, or ea_k and/or eb_k are labelled, or rb(k) is unlabelled, rb(k) does not change by a DVC. Else rb(k) has to be computed according to its definition. The time for a complete update of rb(k), is proportional to $\sum_i |B_i|$.
The use of rb(k) decreases the order of time for labelling during the growth of an arborescence: the labelling time now is proportional to the number of edges of G.

The most time consuming part of the algorithm is the DVC and in particular the computation of d_2 and d_4. However we can reduce somewhat the order of the time for the DVC.
Concerning d_4, we observe

$$\min_j \{v_j \mid (eb_k, b_j) \in D(G),\ b_j \text{ labelled}\} = v_{rb(k)},$$

and the time for computation of d_4 is proportional to $\sum_{j=1}^{n} |B_j|$.
To compute d_2 we use a second set of pointers minu (k), k = 1,...,p, satisfying

$$u_{\text{minu }(k)} = \min_i \{u_i \mid a_i \text{ labelled},\ (a_i, ea_k) \in D(G)\}.$$

If there is no such i, we set minu (k) = 0; we define $u_0 = \infty$.
This computation requires a time proportional to $\sum (A_i)$, summed up over all i with a_i labelled. During the next DVC's, as long as no breakthrough occurs, we only have to evaluate the influence upon minu (k) of those vertices a_i that have received a label since the preceding DVC. So the time to initially compute and to update afterwards minu (k) during the growth of

an arborescence is proportional to $\sum_{i=1}^{m} |A_i|$, whereas the time for labelling is proportional to $\sum_{i=1}^{m} |A_i| + \sum_{j=1}^{n} |B_j| + p$. d_2 now can be computed as

$$d_2 = \min_k \{\min_j \{u_{minu}(k) + v_j + w_k - c_k \mid (eb_k, b_j) \in D(G) \;\; eb_k \text{ and } b_j \text{ in the same state}\} \mid ea_k \text{ unlabelled}\},$$

in a time proportional to $\sum_{j=1}^{n} |B_j|$.

The construction of an AP therefore requires a time proportional to $\sum_i |A_i| + \sum_j |B_j| + p + (n+m+p) * \{\sum_j |B_j| + p\}$.

Assuming each element of E occurs at least once in family B, we can simplify this formula as $\sum_i |A_i| + (n+m+p) * \sum_j |B_j|$.

The computing time for the whole procedure is of the order

$$m * \{(n+m+p) * \sum_j |B_j| + \sum_i |A_i|\}.$$

If $\sum_i |A_i|$ is of the same order as $\sum_j |B_j|$, and $O(m) = O(n)$, the computing time is proportional to:

the number of sets of a family times
the number of sets of a family plus the number of elements of E times
the sum of the cardinalities of the sets of a family.

References

1. Edmonds, J., "Optimum Branchings", *J. Res. Nat. Bureau of Standards*, 71 B (1967), 223-240.

2. Edmonds, J., "Submodular functions, Matroids and Certain Polyhedra", pp. 69-87 in R. Guy (ed.), *Combinatorial Structures and their Applications*, Gordon and Breach, New York (1970).

3. Edmonds, J., "Matroids and the greedy algorithm", *Mathematical Programming*, 1 (1971), 127-136.

4. Ford Jr., L.R. and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton (1962).

5. Kuhn, H.W., "The Hungarian Method for the Assignment Problem", *Naval Res. Logist. Quart.*, 2 (1955), 83-97.

6. Lawler, E.L., "Optimal Matroid Intersections", pp. 233-234 in R. Guy (ed.), *Combinatorial Structures and their Applications* Gordon and Breach, New York (1970).

7. Mirsky, L., *Transversal Theory*, Academic Press, New York (1971).

Appendix

```
INTEGER PROCEDURE MAXIMUM COMMON PARTIAL TRANSVERSAL
(M,N,P,CARDA,C,PA,PB,LISTA,LISTB,RA,RB);
VALUE M,N,P,CARDA; INTEGER M,N,P,CARDA;
INTEGER ARRAY C,PA,PB,LISTA,LISTB,RA,RB;

COMMENT
INPUT:   M, RESP. N IS THE NUMBER OF SETS OF FAMILY A, RESP. B,
         P IS THE NUMBER OF ELEMENTS OF SET E,
         CARDA IS THE SUM OF THE CARDINALITIES OF THE SETS OF FAMILY A,
         ARRAY C CONTAINS THE WEIGHTS OF THE ELEMENTS OF SET E,
         PX[Q] CONTAINS THE ADDRESS OF THE FIRST ELEMENT
         OF SET Q OF FAMILY X IN LISTX (X=A,B),
         THE NEXT ADDRESSES OF LISTX, UNTIL A ZERO IS MET,
         CONTAIN THE OTHER ELEMENTS OF SET Q OF FAMILY X (X=A,B),
OUTPUT:  IF RA[K]=RB[K]=0 ELEMENT K DOES NOT OCCUR IN THE FINAL SOLUTION
         ELSE ELEMENT K MATCHES SET RA[K] OF FAMILY A AND SET RB[K] OF FAMILY B,
         RA[0] IS EQUAL TO THE CARDINALITY OF THE SOLUTION AND
         MAXIMUM COMMON PARTIAL TRANSVERSAL TO ITS WEIGHT;

BEGIN    INTEGER I,J,K,K1,G,I1,O,S,SI,SJ,D,ID,UI,VJ,CK,NA,ND,VMIN,NAI;
         BOOLEAN LEK; INTEGER ARRAY FAMA[1:CARDA+M],NEWLABEL[1:10],
         F,MATCHA[1:M],U,NEXTA[0:M],MATCHB,LB[1:N],V[0:N],LEA,LEB,CW[1:P];

         PROCEDURE LABEL A(I); VALUE I; INTEGER I;
         BEGIN    INT S,UMIN; NEXTA[NA]:= NA:= I; UMIN:= U[I];
                  IF UMIN=0 THEN BREAK A(I); S:= F[I];
                  FOR K:= FAMA[S] WHILE K > 0 DO
                  BEGIN    S:= S+1; IF LEA[K]<0 THEN
                           BEGIN  I1:= RA[K]; IF I1≠0 THEN
                                  BEGIN   IF U[I1]=UMIN THEN
                                          BEGIN LEA[K]:= I; LABEL A(I1) END
                                  END     ELSE IF CW[K]-V[RB[K]]=UMIN THEN
                                  BEGIN LEA[K]:= I; LABEL EB(K,0) END
                           END
                  END
         END;

         PROCEDURE LABEL EB(K,L); VALUE K,L; INTEGER K,L;
         BEGIN    INT S,VMIN; LEB[K]:= L; VMIN:= V[RB[K]]; S:= PB[K];
                  FOR J:= LISTB[S] WHILE J > 0 DO
                  BEGIN    S:= S+1; IF LB[J]<0 ^ V[J]=VMIN THEN
                           BEGIN    LB[J]:= K; L:= MATCHB[J];
                                    IF L=0 THEN BREAK B(J);
                                    LABEL EB(L,J); IF LEA[L]<0 ^ CW[L]=C[L] THEN
                                    BEGIN LEA[L]:= 0; LABEL A(RA[L]) END
                           END
                  END
         END;
```

```
PROCEDURE BREAK A(I); VALUE I; INTEGER I;
BEGIN   FOR I:= I,LEA[K1] WHILE I ≠ 0 DO
        BEGIN   K:= MATCHA[I]; MATCHA[I]:= K1; RA[K1]:= I;
                K1:= K; IF K1=0 THEN GOTO BREAK
        END;
        RA[K1]:= 0; BREAK B(LEB[K1])
END;

PROCEDURE BREAK B(J); VALUE J; INTEGER J;
BEGIN   FOR J:= J,LEB[K1] WHILE J ≠ 0 DO
        BEGIN MATCHB[J]:= K1:= LB[J]; RB[K1]:= J END;
        BREAK A(LEA[K1])
END;

PROCEDURE DUAL CHANGE;
BEGIN   INTEGER ARRAY MINU[1:P];
        PROC MINV;
        BEGIN   VMIN:= CK+1; SJ:= PB[K];
                FOR J:= LISTB[SJ] WHILE J>0 DO
                BEGIN   SJ:= SJ+1; IF LB[J]<0 THEN
                        BEGIN VJ:= V[J]; IF VJ<VMIN THEN VMIN:= VJ END
                END;
                CK:= CK-VMIN; IF CK≥0 THEN NEWLABELLING(LEK)
        END;
        PROC NEWLABELLING(NEG); BOOLEAN NEG;
        BEGIN   IF CK>0 THEN BEGIN Q:= 1; D:= D-CK END
                ELSE IF Q<10 THEN Q:= Q+1;
                NEWLABEL[Q]:= IF NEG THEN -K ELSE K
        END;

        FOR K:= 1 STEP 1 UNTIL P DO MINU[K]:= 0; ID:= ND:= 0;
DC0:    D:= U[ID]; Q:= 1; I1:= ND; NEXTA[NA]:= 0;
        FOR I1:= NEXTA[I1] WHILE I1>0 DO
        BEGIN   UI:= U[I1]; IF UI<D THEN BEGIN D:= UI; ID:= I1 END;
                S:= F[I1]; FOR K:= FAMA[S] WHILE K>0 DO
                BEGIN S:= S+1; IF UI<U[MINU[K]] THEN MINU[K]:= I1 END
        END;
        FOR K:= 1 STEP 1 UNTIL P DO
        BEGIN   LEK:= LEB[K]≥0; IF LEA[K]≥0≡LEK THEN
                BEGIN   G:= MINU[K]; IF G≠0 THEN
                        BEGIN CK:= D-U[G]+CW[K]; IF CK≥0 THEN MINV END
                END     ELSE IF LEK THEN
                BEGIN   CK:= C[K]-CW[K]; G:= MINU[K]; IF G≠0 THEN
                        BEGIN UI:= U[G]-U[RA[K]]; IF UI<CK THEN CK:= UI END;
                        CK:= D-CK; IF CK≥0 THEN NEWLABELLING(FALSE);
                        CK:= D+V[LEB[K]]; MINV
                END
        END;
        IF U[ID]=D THEN BREAK A(ID); ND:= NA;
        FOR I1:= NEXTA[0],NEXTA[I1] WHILE I1≠0 DO U[I1]:= U[I1]-D;
        FOR J:= 1 STEP 1 UNTIL N DO IF LB[J]>0 THEN V[J]:= V[J]+D;
        FOR K:= 1 STEP 1 UNTIL P DO
        BEGIN   LEK:= LEA[K]<0; IF LEK≡LEB[K]≥0 THEN
                CW[K]:= IF LEK THEN CW[K]+D ELSE CW[K]-D ELSE
                IF LEK ∧ RA[K]=0 THEN
                BEGIN   G:= RB[K]; IF LB[G]>0 THEN
                        BEGIN   VMIN:= V[G]; SJ:= PB[K];
                                FOR J:= LISTB[SJ] WHILE J>0 DO
                                BEGIN   SJ:= SJ+1; VJ:= V[J]; IF VJ<VMIN THEN
                                        BEGIN RB[K]:= J; VMIN:= VJ END
```

```
                                                END
                                    END
                        END
                END;
                FOR Q:= Q STEP -1 UNTIL 1 DO
                BEGIN   K:= NEWLABEL[Q]; IF K<0 THEN LABEL EB(-K,LEB[-K]) ELSE
                        IF LEA[K]<0 THEN
                        BEGIN   LEA[K]:= IF LEB[K]>0 ∧ CW[K]=C[K] THEN 0 ELSE MINU[K];
                                I1:= RA[K]; IF I1=0 THEN LABEL EB(K,0) ELSE LABEL A(I1)
                        END
                END;
                GOTO DC0
        END DUAL CHANGE;


INIT:   K1:= NA:= Q:= 0; U[0]:= V[0]:= ₁₀6;
        FOR I:= 1 STEP 1 UNTIL M DO U[I]:= F[I]:= MATCHA[I]:= 0;
        FOR J:= 1 STEP 1 UNTIL N DO V[J]:= MATCHB[J]:= 0;
        FOR K:= 1 STEP 1 UNTIL P DO
        BEGIN   CW[K]:= CK:= C[K]; SI:= PA[K]; SJ:= LISTB[PB[K]];
                IF CK≤0 ∨ LISTA[SI]≤0 ∨ SJ≤0 THEN RA[K]:= -1 ELSE
                BEGIN   RA[K]:= 0; RB[K]:= SJ; FOR I:= LISTA[SI] WHILE I > 0 DO
                        BEGIN   SI:= SI+1; F[I]:= F[I]+1;
                                IF CK>U[I] THEN BEGIN MATCHA[I]:= K; U[I]:= CK END
                        END
                END
        END;
        FOR I:= 1 STEP 1 UNTIL M DO
        BEGIN   K:= MATCHA[I]; IF K=0 THEN GOTO IN1;
                F[I]:= Q:= Q+F[I]+1; FAMA[Q]:= 0;
                IF RA[K]≠0 THEN GOTO IN0; SJ:= PB[K];
                FOR J:= LISTB[SJ] WHILE J>0 DO
                IF MATCHB[J]≠0 THEN SJ:= SJ+1 ELSE
                BEGIN MATCHB[J]:= K; RB[K]:= J; RA[K]:= I; GOTO IN1 END;
        IN0:    NEXTA[NA]:= NA:= I;
        IN1:
        END;
        FOR K:= 1 STEP 1 UNTIL P DO IF RA[K]≥0 THEN
        BEGIN   SI:= PA[K]; FOR I:= LISTA[SI] WHILE I > 0 DO
                BEGIN SI:= SI+1; Q:= F[I]:= F[I]-1; FAMA[Q]:= K END
        END;
        NEXTA[NA]:= 0; NAI:= NEXTA[0];

AUGMENTING PATH:
        FOR I:= NAI WHILE I ≠ 0 DO
        BEGIN   FOR J := 1 STEP 1 UNTIL N DO LB [J]:= -1;
                FOR K := 1 STEP 1 UNTIL P DO LEA[K]:= LEB[K]:= -1;
                NA:= MATCHA[I]:= 0; NAI:= NEXTA[I];
                LABEL A(I); DUAL CHANGE;
        BREAK:
        END;

        CK:= 0; FOR K:= 1 STEP 1 UNTIL P DO
        IF RA[K]≤0 THEN RA[K]:= RB[K]:= 0 ELSE
        BEGIN K1:= K1+1; CK:= CK+C[K] END;
        RA[0]:= K1; MAXIMUM COMMON PARTIAL TRANSVERSAL := CK

END MAXIMUM COMMON PARTIAL TRANSVERSAL;
```