AFDELING MATHEMATISCHE BESLISKUNDE (DEPARTMENT OF OPERATIONS RESEARCH) BW 39/77 NOVEMBER

B. DORHOUT

EXPERIMENTS WITH SOME ALGORITHMS FOR THE LINEAR ASSIGNMENT PROBLEM

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).

AMS(MOS) subject classification scheme (1970): 90B10, 90C05

Experiments with some algorithms for the linear assignment problem

by

B. Dorhout

ABSTRACT

This report is adapted from a Dutch report [5], written in 1973. Several algorithms for solving the linear assignment problem are compared with each other, in particular with respect to the computing times needed to solve problems of a test set.

In addition to some well known algorithms a recently-developed algorithm of the primal-dual type was tested. Because it seems to have very favourable properties it is described extensively in the report.

KEY WORDS & PHRASES: *linear assignment problem, comparison of algorithms*

1. INTRODUCTION

The *linear assignment problem* may be formulated as follows: Given real numbers a_{ij}, $i = 1,\dots,n$; $j = 1,\dots,n$, find such values of the variables x_{ij}, $i = 1,\dots,n$; $j = 1,\dots,n$, that

$$(1.1) \qquad z = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_{ij}$$

is minimized subject to the restrictions

$$(1.2) \qquad \sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,\dots,n,$$

$$(1.3) \qquad \sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,\dots,n,$$

$$(1.4) \qquad x_{ij} = 0 \text{ or } 1 \qquad i = 1,\dots,n;\ j = 1,\dots,n.$$

If (1.4) is replaced by

$$(1.5) \qquad x_{ij} \geq 0, \qquad i = 1,\dots,n;\ j = 1,\dots,n,$$

then linear programming theory tells us that among the possibly more than one optimal solutions there is always at least one, that satisfies (1.4). As a consequence also the altered problem is considered to represent a linear assignment problem. Because every solution to (1.2), (1.3), (1.4) describes one of the permutations of the integers 1,..,n, the linear assignment problem plays an important role in the solution to several kinds of sequencing problems. Direct applications are for example found in the field of transportation, where certain scheduling problems can be formulated as linear assignment problems [18]. Because these problems often cover several periods of time, they may have very large dimensions. Other sequencing problems can be solved by branch-and-bound methods, in which linear assignment problems occur as relaxations of the original problem [9]. In these cases large numbers of similar assignment problems have to be solved.

From the preceding it will be clear that it is very useful to have efficient algorithms for solving the linear assignment problem. For this reason the performances of several algorithms were compared, in particular with regard to computation time.

2. ALGORITHMS

In accordance with the second formulation of section 1 it is possible to solve the linear assignment problem by linear programming methods. In fact all successful algorithms make use of linear programming techniques.

The dual linear programming problem of the linear assignment problem is to maximize

$$(2.1) \qquad -\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$$

subject to the constraints

$$(2.2) \qquad -u_i + v_j \le a_{ij}, \qquad i = 1,\dots,n;\ j = 1,\dots,n.$$

Optimal feasible solutions to the original and the dual problem may be connected by the *complementary slackness* property described in the next theorem.

THEOREM 2.1. *A necessary and sufficient condition for a feasible solution to problem* (1.1), (1.2), (1.3), (1.5) *to be optimal is that there is a feasible solution to the dual problem* (2.1), (2.2) *with the complementary slackness property*

$$(2.3) \qquad (u_i+a_{ij}-v_j)\, x_{ij} = 0, \qquad i = 1,\dots,n;\ j = 1,\dots,n.$$

The known efficient algorithms for the linear assignment problem may be divided into *primal* algorithms and *primal-dual* algorithms.

In each step of a primal algorithm one has a feasible solution to (1.2), (1.3), (1.5) and values of the dual variables which satisfy (2.3).

If the values of the dual variables satisfy (2.2) then the optimal solution is found. Otherwise a new step is made. The number of steps is finite.

The primal-dual algorithms construct a sequence of feasible solutions to (2.2) with values of x_{ij} which satisfy (2.3) and (1.5). Also the x_{ij} satisfy

$$(2.4) \qquad \sum_{j=1}^{n} x_{ij} \leq 1, \qquad i = 1,\dots,n$$

and

$$(2.5) \qquad \sum_{i=1}^{n} x_{ij} \leq 1, \qquad j = 1,\dots,n.$$

As soon as a primal solution is obtained that satisfies (1.2) and (1.3) it is optimal. The successive values of

$$(2.6) \qquad w = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}$$

form a monotone non-decreasing sequence. Finite bounds for the number of steps may be calculated.

The best-known primal algorithms are the simplex algorithm for transportation problems, due to DANTZIG [3] and refined by SRINIVASAN and THOMPSON [15], an algorithm due to BALINSKY and GOMORY [1], and KLEIN's negative cycle algorithm [10]: As a comparative study of FLORIAN and KLEIN [7] revealed that the last mentioned algorithms give worse results than the primal-dual Hungarian algorithms, these methods were not considered.

The best-known primal-dual algorithm is the Hungarian algorithm, due to KUHN [11], in the version of MUNKRES [12], described [13] and programmed [14] by SILVER. A primal-dual algorithm for solving the general minimal cost network flow problem, due to BUSACKER and GOWEN [2], was worked out by TOMIZAWA [17] as well for the linear assignment problem as for the more general problem, and by TABOURIER [16], only for the linear assignment problem. Three versions of the Hungarian algorithm, the Tomizawa algorithm, the Tabourier algorithm and a revised form of the Tomizawa algorithm were programmed and tested.

Before mentioning the results we will describe the algorithm based on the ideas of Basacker and Gowen. For this purpose we consider the linear assignment problem as a network problem. We start from a directed network with a source P_0, a sink Q_0, and intermediate nodes $P_1,\dots,P_n$, $Q_1,\dots,Q_n$. There are arcs from P_0 to $P_1,\dots,P_n$, from $Q_1,\dots,Q_n$ to Q_0, each having capacity 1 and unit costs 0, and uncapacitated arcs from each P_i, $i = 1,\dots,n$, to each Q_j, $j = 1,\dots,n$, with unit costs a_{ij} (figure 2.1).

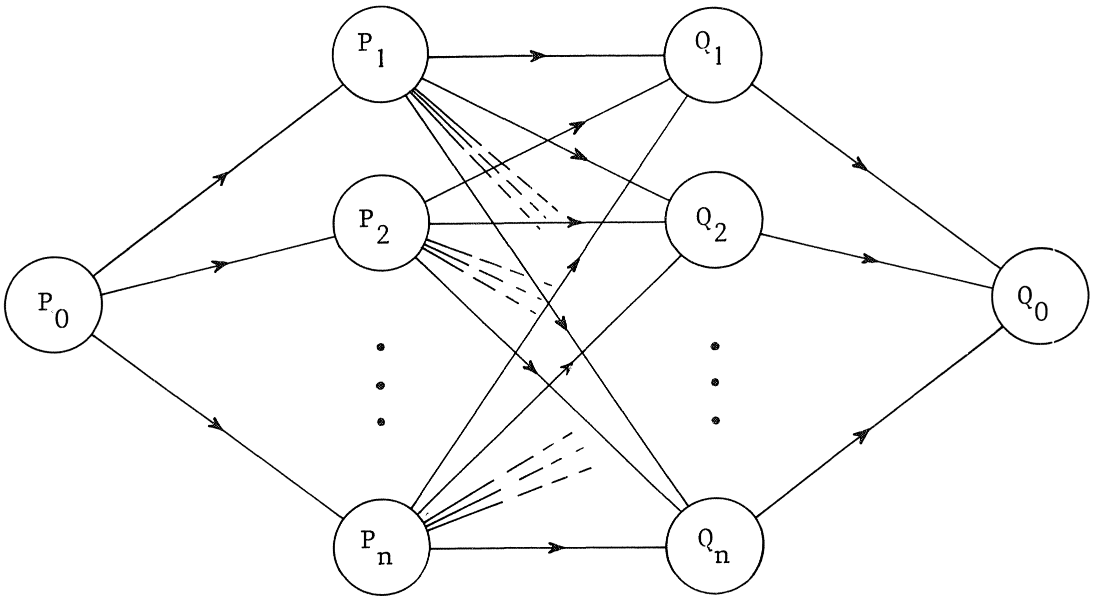

Figure 2.1. The linear assignment network.

Only nonnegative flow in the direction of the arrows is allowed. The sum of the flow into an intermediate node must be equal to the sum of the flows out of it. We will denote the quantities of flow in the arcs from P_0 to P_i, $i = 1,\dots,n$, from P_i to Q_j, $i = 1,\dots,n$; $j = 1,\dots,n$ and from Q_j, $j = 1,\dots,n$, to Q_0 by x_{0i}, x_{ij}, and x_{j0}, and their capacities by c_{0i}, c_{ij}, and c_{j0} respectively. Solving the linear assignment problem is then equivalent to determining a minimum cost flow from P_0 to Q_0 with integer flow values and total flow value $w = n$.

Given a certain feasible flow pattern, a *flow augmenting path* consists of a sequence of arcs (P_0,P_{i_1}), (P_{i_1},Q_{j_1}), $(P_{i_2},Q_{j_1}),\dots,(P_{i_r},Q_{j_r})$, (Q_{j_r},Q_0) with the property that $x_{0i_1} < 1$, $x_{i_{k+1}j_k} > 0$, $k = 1,\dots,r-1$, and $x_{j_r0} < 1$. If one moves along the nodes of this path via the arcs of it, the arcs (P_0,P_{i_1}), (P_{i_k},Q_{j_k}), $k = 1,\dots,r$, and (Q_{j_r},Q_0) are passed in the direction of the arrows. These are called *forward arcs*. The other arcs are called *backward arcs*. The total flow from source to sink may be augmented by augmenting the flow in the forward arcs by an amount δ and diminishing the flow in the backward arcs by the same amount. Then the costs of the total flow are augmented by the sum of the costs of the forward arcs, diminished by the sum of the costs of the backward arcs per unit of flow.

In the same way we define an augmenting cycle. This is a path with a given orientation, for which the starting point and the endpoint coincide. The arcs with the given orientation are forward arcs, the others are bakcward arcs. In the forward arcs one must have $x_{ij} < c_{ij}$, in the backward arcs $x_{ij} > 0$. The costs of augmenting the flow along the cycle are equal to the sum of the costs of the forward arcs, diminished by the sum of the costs of the backward arcs per unit of flow.

<u>THEOREM 2.2</u>. *A feasible flow having value* w *minimizes cost over all feasible flows of value* w *if and only if there are no augmenting cycles having negative cost.*

<u>PROOF</u>. It is obvious that a flow pattern that admits an augmenting cycle with negative cost cannot be of minimal cost. On the other hand, if one diminishes a minimal cost flow with a given non-minimal cost flow, this results in a sum of augmenting cycles with respect to the given flow. As the cost of this sum is negative it will contain at least one augmenting cycle with negative cost.

As a consequence of this theorem we have

<u>THEOREM 2.3</u>. (Busacker and Gowen). *If a minimum cost flow is augmented*

along a minimum cost flow augmenting path, the new flow is a minimum cost flow for the new flow value.

PROOF. We start with a flow that contains no augmenting cycles with negative costs. If the new flow is not a minimum cost flow, then it will contain a flow augmenting cycle with negative cost which contains at least one backward arc that was flowless before. So this is an arc which is a forward arc of the minimum cost flow augmenting path. But then this path would not have been a minimum cost path, for the part of the path that coincides with (part of) the negative cycle in the opposite direction, could have been replaced by the rest of the cycle, thus giving a path with less costs than those of the minimum cost path.

As a consequence of theorem 2.3 the linear assignment problem may be solved by the following *basic algorithm*:

1. Start with flow 0.
2. Determine a minimum cost flow augmenting path.
3. Augment the flow along the minimum cost path as much as possible. If the flow value is n, stop: the optimal solution is found. Otherwise go to step 2.

REMARK. After each iteration of the basic algorithm all x-values are equal to 0 or 1. In effect, at the start of the algorithm all x-values are 0, and in each subsequent iteration, flow is augmented by 1 unit in the forward arcs and diminished by 1 unit in the backward arcs of a flow-augmenting path. As a result the algorithm is finished after exactly n iterations.

THEOREM 2.4. *The optimal solution of a linear assignment problem does not change if a constant is added to all elements of a row (a column) of the cost matrix.*

PROOF. Trivial.

Application of theorem 2.4 leads to some modifications of the basic algorithm.

Firstly consider the situation, where a very large number M is defined, e.g. $M = \sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|$ and Mi is added to all elements a_{ij} of row i of the cost matrix, for $i = 1,\dots,n$. Then in the ith iteration of the basic algorithm the flow on (P_0,P_i) is augmented. The same sequence of flow augmenting paths would have been obtained by subsequently augmenting the flow on $(P_0,P_1),\dots,(P_0,P_n)$. We infer that in each iteration it is possible to choose the first arc of a minimum cost flow augmenting path arbitrarily from the flowless arcs (P_0,P_i). In the same way one is free in the choice of the last arc.

As a second consequence of theorem 2.4 one may reduce the cost matrix: first diminish all numbers in the rows of the cost matrix by their respective minima and then do the same with the columns. Both problems have the same optimal solution(s). After the reduction all coefficients are nonnegative and each row and each column contains at least one zero. Then one may choose an independent set of zeroes, i.e. a set of zeroes with the property that no row or column contains more than one of them. With each of these $a_{ij} = 0$ there corresponds a zero cost flow augmenting path (P_0,P_i), (P_i,Q_j), (Q_j,Q_0). Flow may be augmented along all of these paths because they do not have any P_i, $i \neq 0$, or Q_j, $j \neq 0$, in common.

Now *Tabourier's algorithm* [16] may be described as follows:

1. Start with flow 0.
2. Reduce the cost matrix and determine a maximal independent set of zeroes of it. Augment the flow along all corresponding zero cost flow augmenting paths by 1 unit.
3. Try to find a flowless arc (P_0,P_k). If there is no such arc, stop: the optimal solution is found. Otherwise determine a minimum cost flow augmenting path having (P_0,P_k) as its first arc.
4. Augment the flow along the minimum cost flow augmenting path with one unit. Go to step 3.

The minimum cost flow augmenting path in step 3 is determined by means of a shortest path algorithm, applied to a modified network. This is derived

from the original one by adding arcs (Q_j,P_i) for all pairs (P_i,Q_j) with $x_{ij} = 1$ and deleting all arcs (P_0,P_i) except (P_0,P_k) and deleting all saturated arcs $(Q_j?Q_0)$, i.e. all arcs with $x_{j0} = 1$. The lengths of the added arcs (Q_j,P_i) are $- a_{ij}$ and the lengths of the other arcs are equal to the corresponding unit costs in the original network.
As some of the arcs of the modified network may have negative lengths the shortest path problems are solved by Ford's algorithm [8], which requires $0(n^3)$ addition and comparison operations. So the number of operations required by Tabourier's algorithm is $0(n^4)$.

In *Tomizawa's algorithm* [17] at iteration k a modified network is obtained from the original one by adding arcs (Q_j,P_i) for all arcs (P_i,Q_j) with $x_{ij} = 1$ and by deleting all arcs (P_0,P_i) and (Q_j,Q_0) except (P_0,P_k) and (Q_k,Q_0). The arcs (P_i,Q_j), $i = i,\dots,k$; $j = 1,\dots,k$ have lengths $\bar{a}_{ij} = a_{ij} + u_i - v_j$ and all other arcs are of length 0. The values of the dual variables u_i and v_j are chosen in such a way that the $\bar{a}_{ij}$ are nonnegative. Therefore it is possible to apply Dijkstra's shortest path algorithm [4] that takes only $0(n^2)$ addition and comparison operations [6]. So the total number of these operations in Tomizawa's algorithm is $0(n^3)$.

In describing Tomizawa's algorithm we use expressions a := b which stand for "a is replaced by b". Furthermore in each iteration for all pairs (P_i,Q_j) with $x_{ij} = 1$ we define $\xi_i = j$ and $\eta_j = i$. We define $\xi_i = 0$ for all i with $x_{ij} = 0$, $j = 1,\dots,n$ and $\eta_j = 0$ for all j with $x_{ij} = 0$, $i = 1,\dots,n$.
The algorithm consists of the following steps:

1. Start with flow 0. k = 1.
   $v_1 = a_{11}$, $u_1 = 0$, $x_{11} = 1$.
2. k := k + 1.
   $v_k = \min_{i = 1,\dots,k-1} (a_{ik}+u_i)$, $u_k = -\min_{j = 1,\dots,k} (a_{kj}-v_j)$
   $\bar{a}_{ij} = a_{ij} + u_i - v_j$, $\qquad i = 1,\dots,k;\ j = 1,\dots,k.$
3. Define a set $T = \{1,\dots,k\}$ and labels $\lambda_j = k$, for all $j \in T$. In the modified network the shortest distance to P_k is $\Delta u_k = 0$. Nodes Q_j, $j = 1,\dots,k$ may be reached along one-arc paths of lengths $\Delta v_j = \bar{a}_{kj}$. Furthermore $\Delta u_i = \infty$, $i = 1,\dots,k-1$.

4. $\Delta v_m = \min\{\Delta v_j \mid j \in T\}$. If $m = k$ then go to 5.
   $T := T - \{m\}$, $i := \eta_m$, $\Delta u_i := \Delta v_m$. For all $j \in T$, for which $\bar{a}_{ij} + \Delta u_i < \Delta v_j$ make $\lambda_j := i$ and $\Delta v_j := \bar{a}_{ij} + \Delta u_i$.
   Repeat this step.

5. Construct the shortest path from P_k to Q_k: going backward from $Q_j = Q_k$ this path contains in turn arcs (P_i,Q_j) with $i = \lambda_j$ and (Q_j,P_i) with $j = \xi_i$ until P_k is reached. In the original network the flow is changed and new values of the dual variables are determined by
   $u_i := u_i + \min(\Delta u_i,\Delta v_k)$, $\quad i = 1,\dots,k,$
   $v_j := v_j + \min(\Delta v_j,\Delta v_k)$, $\quad j = 1,\dots,k.$
   If $k < n$ then go to 2.

6. The problem is solved. The u_i, $i = 1,\dots,n$ and v_j, $j = 1,\dots,n$ represent an optimal solution of the dual problem.

The validity of the algorithm follows from the observation that at the end of the kth iteration the u_i and v_j satisfy

$$(2.7) \qquad a_{ij} + u_i - v_j \geq 0, \qquad i = 1,\dots,r;\ j = 1,\dots,r,$$

and

$$(2.8) \qquad a_{ij} + u_i - v_j = 0 \quad \text{if } x_{ij} = 1, \qquad i = 1,\dots,r;\ j = 1,\dots,r,$$

with $r = k$.

In fact, at the end of step 1 these conditions are clearly satisfied. Now assume that they are satisfied at the end of iteration k-1, with $r = k-1$. We will show that they are also valid at the end of the kth iteration.
The determination of the values of u_k and v_k in step 2 assures that $\bar{a}_{ij} \geq 0$, $i = 1,\dots,k$; $j = 1,\dots,k$, during iteration k. In order to prove that (2.7) is valid for $r = k$ after the changes in step 5 we consider the shortest distances $\bar{\Delta} u_i$ and $\bar{\Delta} v_j$ to *all* P_i, $i = 1,\dots,k$, and Q_j, $j = 1,\dots,k$, respectively. They satisfy the shortest path optimality conditions

$$(2.9) \qquad \bar{a}_{ij} + \bar{\Delta u}_i - \bar{\Delta v}_j \geq 0, \quad i = 1,\dots,k;\ j = 1,\dots,k.$$

Now if we define $S = \{i \mid i = n_j,\ j \in T,\ i \neq 0\}$, $\bar{S} = \{1,\dots,k\}-S$, $\bar{T} = \{1,\dots,k\}-T$, then Dijkstra's algorithm, performed in the steps 4, has the property that in each execution of this step the sets $\bar{S}$ and $\bar{T}$ are extended with the index of one point to which the shortest distance from P_0 is definitively determined. The distances to the points with the subsequently added indices form two monotone nondecreasing sequences. At the moment that $m = k$ in step 4, and Dijkstra's algorithm stops, the following relations hold: $\Delta v_k = \bar{\Delta v}_k$, $\Delta u_i = \bar{\Delta u}_i \leq \Delta v_k$, for all $i \in \bar{S}$, $\Delta v_j = \bar{\Delta v}_j \leq \Delta v_k$, for all $j \in \bar{T}$, $\Delta u_i \geq \bar{\Delta u}_i \geq \Delta v_k$, for all $i \in S$, and $\Delta v_j \geq \bar{\Delta v}_j \geq \Delta v_k$, for all $j \in T$. If for the moment we denote the values of u_i and v_j at the end of step 5 by $u_i^!$ and $v_j^!$ respectively, then

$$a_{ij} + u_i^! - v_j^! = \bar{a}_{ij} + \min(\Delta u_i,\Delta v_k) - \min(\Delta v_j,\Delta v_k),$$
$$i = 1,\dots,k;\ j = 1,\dots,k.$$

Hence, by (2.9) we have for $i \in \bar{S}$, $j \in \bar{T}$:

$$a_{ij} + u_i^! - v_j^! = \bar{a}_{ij} + \bar{\Delta u}_i - \bar{\Delta v}_j \geq 0,$$

and for $i \in \bar{S}$, $j \in T$:

$$a_{ij} + u_i^! - v_j^! = \bar{a}_{ij} + \bar{\Delta u}_i - \Delta v_k \geq \bar{a}_{ij} + \bar{\Delta u}_i - \bar{\Delta v}_j \geq 0.$$

As for $i \in S$, $j = 1,\dots,k$

$$a_{ij} + u_i^! - v_j^! = \bar{a}_{ij} + \Delta v_k - \min(\Delta v_j,\Delta v_k) \geq \bar{a}_{ij} \geq 0$$

we conclude that (2.7) holds for $r = k$.

At the end of step 5 of an iteration $\Delta u_i = \Delta v_j$ for all (P_i,Q_j) for which $x_{ij} = 1$ at the start of step 2. So there is $a_{ij} + u_i^! - v_j^! = \bar{a}_{ij} = 0$. Furthermore for all (P_i,Q_j) on the shortest path $\Delta v_j = \bar{a}_{ij} + \Delta u_i$, with $\Delta u_i \leq \Delta v_k$ and $\Delta v_j \leq \Delta v_k$. Hence $a_{ij} + u_i^! - v_j^! = \bar{a}_{ij} + \Delta u_i - \Delta v_j = 0$ on these arcs. As flows are only changed on the arcs of the shortest path the new solution also satisfies (2.8) for r = k.

Observe that it is *essentially necessary* to apply *Dijkstra's algorithm*, if one wishes to stop the shortest path computations as soon as the shortest path to Q_0 is found (m = k in step 4).

The properties of Dijkstra's algorithm are used more fully if, like in Tabourier's algorithm, the modified network contains all arcs (Q_j,Q_0) for which $x_{j0} = 0$. Although the networks considered are greater the number of steps in Dijkstra's algorithm is greatly reduced in most cases, because in general there are less arcs in the shortest paths to Q_0. A second improvement is obtained if, like in Tabourier's method, one starts with a reduction of the cost matrix. The revised algorithm that results is described below. In order to prove the validity of this algorithm only minor changes have to be made in the proof of the original algorithm. It will therefore be omitted here.

The steps of the revised algorithm are:

1. Start with flow 0.

   Reduce the cost matrix:

   $u_i = -\min\limits_{j=1,\dots,n} a_{ij}, \qquad i = 1,\dots,n,$

   $v_j = \min\limits_{i=1,\dots,n} (a_{ij}+u_i), \quad j = 1,\dots,n,$

   Define a set $J = \{1,\dots,n\}$

   $i := 0.$

2. $i := i + 1$. If there is an index $j \in J$ with $a_{ij} + u_i - v_j = 0$ then $x_{ij} := 1$. $J := J - \{j\}$. Repeat this step up to and including $i = n$.
3. Try to find a flowless arc (P_0,P_k). If there is no such arc go to 7.

   $u_k = -\min\limits_{j=1,\dots,n} (a_{kj}-v_j)$

   $\bar{a}_{ij} = a_{ij} + u_i - v_j, \qquad i = 1,\dots,n; \quad j = 1,\dots,n.$

4. Define a set $T = \{1,\dots,n\}$ and labels $\lambda_j = k$, for all $j \in T$.
   In the modified network the shortest distance to P_k is $\Delta u_k = 0$.
   Nodes Q_j, $j = 1,\dots,n$ may be reached along one-arc paths of lengths $\Delta v_j = \bar{a}_{kj}$.
   Furthermore $\Delta u_i = \infty$, $\quad i = 1,\dots,n$.

5. If $\Delta v_m = \min\{\Delta v_j \mid j \in T\}$ then go to 6 in the case, that $\eta_m = 0$
   $T := T - \{m\}$, $\quad i := \eta_m$, $\quad \Delta u_i := \Delta v_m$.
   For all $j \in T$, for which $\bar{a}_{ij} + \Delta u_i < \Delta v_j$ make $\lambda_j := i$
   and $\Delta v_j := \bar{a}_{ij} + \Delta u_i$.
   Repeat this step.

6. Construct the shortest path from P_k to Q_m. In the original network the flow is changed and new values of the dual variables are determined by
   $u_i := u_i + \min(\Delta u_i, \Delta v_m)$, $\qquad i \in \{i \mid x_{0i} = 1\}$
   $v_j := v_j + \min(\Delta v_j, \Delta v_m)$, $\qquad j = 1,\dots,n$.
   Go to 3.

7. The problem is solved. The u_i, $i = 1,\dots,n$, and v_j, $j = 1,\dots,n$, represent an optimal solution of the dual problem.

Example for the revised Tomizawa algorithm.

Given is the cost matrix

$$\begin{bmatrix} 7 & 12 & 9 & 11 & 5 \\ 5 & 10 & 7 & 8 & 12 \\ 14 & 15 & 13 & 12 & 8 \\ 8 & 13 & 11 & 14 & 7 \\ 10 & 9 & 7 & 6 & 13 \end{bmatrix}$$

The first $\bar{a}_{ij}$ are shown in table 2.1. The assignments made in steps 2 are characterized by asterisks.

λ_j	$\cancel{3}$ 1	3	$\cancel{3}$ 1	3	3
Δu_i \ Δv_j	$\cancel{6}$ $\textcircled{2}$	4	$\cancel{4}$ 3	4	$\textcircled{0}$
0	2	4	3	6	0^*
2	0^*	2	1	3	7
0	6	4	4	4	0
	1	3	3	7	0
3	4	0^*	0	0	7

Table 2.1. The first tableau

The first flowless arc, found in step 3, is (P_0,P_3). According to step 4 we define $T = \{1,\ldots,5\}$ and $\lambda_j = 3$, $j = 1,\ldots,5$, with $\Delta v_j = \bar{a}_{3j}$, $j = 1,\ldots,5$, $\Delta u_3 = 0$. The infinite values of the other Δu_i are deleted. In step 5 we find $\Delta v_5 = \min\{\Delta v_j \mid j \in T\}$. $T := T - \{5\}$ as represented by the circle. $i := 1$, $\Delta u_1 := \Delta v_5 (= 0)$, $\Delta v_1 := 2+0$, $\lambda_1 := 1$, $\Delta v_3 := 3$, $\lambda_3 := 1$. In the second application of step 5 we find $\Delta v_1 = \min\{\Delta v_j \mid j \in T\}$, $T := T - \{1\}$, $i := 2$, $\Delta u_2 := \Delta v_1 (= 2)$. No labels are changed. In the third application of step 5 we have $\Delta v_3 = \min\{\Delta v_j \mid j \in T\}$. As $\eta_3 = 0$ (there is no asterisk in column 3), the minimum cost flow augmenting path is constructed. It consists, in reverse order, of the arcs (Q_3,Q_0), $(P_{\lambda_3},Q_3)=(P_1,Q_3)$, $(P_1,Q_{\xi_1})=(P_1,Q_5)$, $(P_{\lambda_5},Q_5)=(P_3,Q_5)$, and (P_0,P_3). The transition to the $\bar{a}_{ij}$ - tableau at the end of the next step 3 is represented by

$$\begin{bmatrix} 0 & 1 & 0 & 3 & 0^* \\ 0^* & 1 & 0 & 2 & 9 \\ 4 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 5 & 1 \\ 5 & 0^* & 0 & 0 & 10 \end{bmatrix}$$

Table 2.2 shows the final tableau of the second iteration

λ_j	4	4	2	3	1
Δu_j \ Δv_j	⓪	1	⓪	1	⓪
0	0	1	0*	3	0
0	0*	1	0	2	9
0	4	1	1	1	0*
0	0	1	1	5	1
1	5	0*	0	0	10

Table 2.2. The second tableau

After step 6 we have

$$\begin{bmatrix} 0 & 0 & 0^* & 2 & 0 \\ 0^* & 0 & 0 & 1 & 9 \\ 4 & 0 & 1 & 0 & 0^* \\ 0 & 0 & 1 & 4 & 1 \\ 6 & 0^* & 1 & 0 & 11 \end{bmatrix}$$

So the optimal solution is $x_{41} = x_{52} = x_{23} = x_{34} = x_{15} = 1$.

3. COMPUTATIONAL RESULTS

Several algorithms were programmed in ALGOL 60 and tested on an EL - X8 computer. Some results are given in table 3.1, which shows CPU time intervals between entry and exit of the ALGOL 60 procedures. The test problems are characterized by four digit numbers abcd, where ab represents the value of n, and digit c has the following meaning:

c = 0: a_{ij} random integers, $0 \le a_{ij} < 10$
c = 1: a_{ij} random integers, $0 \le a_{ij} < 50$
c = 2: a_{ij} random integers, $0 \le a_{ij} < 250$

algorithm / problem	H1	H2	H3	Ta	To	R	S
2501		1.85	1.92	4.41	2.70	2.14	11.74
2502		1.57	1.66	4.33	2.48	2.25	10.03
2503		2.03	2.10	4.79	2.59	2.21	9.22
2511		2.96	2.53	7.53	3.47	2.20	10.85
2512		2.32	1.95		2.88	1.74	8.62
2513		2.71	2.12		2.78	1.97	12.27
2521		4.19			2.40	1.62	11.64
2522		4.71			2.51	2.11	9.72
2523		2.56			2.24	1.39	8.95
2531		4.08			3.24	1.81	11.62
2532		4.33			2.96	1.82	10.02
2533		2.72			3.43	1.66	10.08
5001		4.50	5.33		12.94	10.96	19.73
5002		5.84			12.65	10.49	25.94
5003		6.19			12.98	11.44	21.50
5011		12.03			16.80	7.88	31.74
5012		13.17			17.51	9.79	37.56
5013		10.12			17.17	8.67	34.07
5021		22.72	18.29		20.60	9.34	33.11
5022	35.06	23.48	16.88	50.34	21.36	9.10	44.46
5023	39.97	27.43	19.77	49.01	23.99	12.72	38.90
5031	93.63	56.56	36.31	54.22	20.45	10.19	33.15
5032	55.22	35.79	24.63	41.87	20.63	9.09	31.76
5041	37.14	27.17	21.77		25.86	10.75	42.11
5051		97.50	76.56		30.37	13.17	36.91
7511		25.30	27.84		64.16	32.80	75.73
7512		25.52	24.54		63.97	24.97	66.22
7513		25.69	24.34		50.16	21.96	72.63
7531		163.79	105.30		69.96	19.08	67.32
7532		145.55	104.39		77.09	22.96	95.31

Table 3.1

Computation times (in seconds)

c = 3: a_{ij} random real numbers, $0 \le a_{ij} < 1$
c = 4: a_{ij} integers taken from a scheduling problem, $0 \le a_{ij} < 40$
c = 5: a_{ij} equal to the a_{ij}, applied with c = 4, augmented with random perturbations ε_{ij}, $0 \le \varepsilon_{ij} < .01$.

The headings of the columns represent the algorithms:

H1:	Hungarian method as described in [14]
H2:	Improved version of H1, the cost matrix is transformed in each iteration
H3:	More simple version of H2, no tranformations but updating of the dual variables
Ta:	Tabourier's algorithm
To:	Tomizawa's algorithm with cost matrix reduction start
R:	Revised Tomizawa algorithm
S:	Simplex transportation algorithm.

The main conclusion that can be drawn from our results is that the revised Tomizawa algorithm is superior to all other tested methods. For this reason a slightly changed version of the ALGOL 60 procedure that was tested is included in an appendix. As this procedure is programmed in a rather straightforward way one may expect still better results if more sophisticated programming techniques are applied.

It appears that for the Hungarian method computation times decrease with increasing dual degeneracy. Only problems with a very high degree of dual degeneracy were solved within competitive times. The other algorithms are rather insensitive in this respect.

As with growing n computation times do not grow as fast for the simplex algorithm as for the Hungarian algorithm, it is perhaps worthwhile to develop primal simplex methods which take advantage ofthe special structure of the linear assignment problem. Although several ideas of [15] are realised in the simplex procedure that was tested it can certainly be programmed better.

REFERENCES

[1] BALINSKI, M.L. & R.E. GOMORY, *" A primal method for the assignment and transportation problems",* Man. Sci. 10 (1964), 578-593.

[2] BUSACKER, R.G. & P.J. GOWEN, *"A Procedure for Determining a Family of Minimal-Cost Network Flow Patterns",* ORO Technical Report 15, Johns Hopkins University, 1961.

[3] DANTZIG, G.B., *Linear programming and extensions,* Princeton University Press, Princeton (1963).

[4] DIJKSTRA, E.W., *"A note on two problems in connection with graphs",* Num. Math. 1 (1959), 269-271.

[5] DORHOUT, B., *"Het lineaire toewijzingsprobleem: vergelijking van algoritmen",* Report BN 21, Mathematisch Centrum, Amsterdam.

[6] DREYFUS, S.E., *"An appraisal of some shortest-path algorithms",* Operations Res. 17 (1969), 395-412.

[7] FLORIAN, M. & M. KLEIN, *"An experimental evaluation of some methods of solving the assignment problem",* CORS 8 (1970), 101-108.

[8] FORD, L.R. & D.R. FULKERSON, *Flows in Netwerks,* Princeton University Press, Princeton (1962).

[9] GEOFFRION, A.U. & R.E. MARSTEN, *"Integer Programming Algorithms: A Framework and State-of-the-Art-Survey",* Man. Sci. 18 (1972), 465-491.

[10] KLEIN, M., *"A primal method for minimal cost flows with applications to the assignment and transportation problems",* Man. Sci. 14 (1967), 205-220.

[11] KUHN, H.W., *"The Hungarian method for the assignment problem",* Nav. Res. Log. Quart. 2 (1955), 83-97.

[12] MUNKRES, J., *"Algorithms for the assignment and transportation problems",* J. Soc. Industr. Appl. Math. 5 (1957), 32-38.

[13] SILVER, R., *"An Algorithm for the Assignment Problem",* Comm. ACM, 3 (1960), 605-606.

[14] SILVER, R., *Algorithm* 27, Assignment, Comm. ACM, 3 (1960), 603-604.

[15] SRINIVASAN, V. & G.L. THOMPSON, *"Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm"*, Journal of the ACM, 20 (1973), 194-213.

[16] TABOURIER, Y., *"Un algorithme pour le problème d'affectation"*, RAIRO 6 (1972), 3-15.

[17] TOMIZAWA, N., *"On Some Techniques Useful for Solution of Transportation Network Problems"*, Networks 1 (1971), 173-194.

[18] UEBE, G., *Optimale Fahrpläne*, Springer, Berlin (1970).

APPENDIX: An ALGOL 60 - procedure for the revised Tomizawa algorithm

```
real procedure assignment(n,i,j,aij,x,u,v); value n;
integer n,i,j; real aij; integer array x; array u,v;
comment
formal parameters:
n:      <arithmetical expression>: the size of the problem,
i:      <variable>:
        integer identifier denoting a row of the cost matrix,
j:      <variable>:
        integer identifier denoting a column of the cost matrix,
aij:    <arithmetical expression>:
        the cost of using the element in row i and column j,
x:      <array identifier>:
        a one-dimensional integer array x[1:n],
        exit: a rowwise stored optimal solution of the problem:
              x[i]=j means that the element in row i and column j belongs
              to the solution,
u,v:    <array identifier>:
        a one-dimensional array u[1:n], resp. v[1:n],
        exit: the arrays u and v contain the optimal solution
              of the dual problem, that corresponds with the final
              primal solution,
assign:= the cost of the optimal solution;

begin integer io,jo,k,ko,up;
  real a,dj,h,s,min,ui,vj,giant;
  integer array y, lab,td[1:n]; real array d[1:n];
  for j:= 1 step 1 until n do x[j]:= y[j]:= lab[j]:= 0;
  min := 0;
  for i:= 1 step 1 until n do
  begin j:= jo:= 1; a:= aij; giant:= a; ui:= -a;
    if giant<a then giant:= a;
    for j:= 2 step 1 until n do
    begin a:= aij; if giant<a then giant:= a;
      if a+ui<0 then
```

```
        begin ui:= -a;jo:= j end
      end;
      u[i]:= ui; if i=1 or min+ui>0 then min:= -ui;
      if y[jo]=0 then begin x[i]:=jo; y[jo]:= i end
    end;
    giant:= (giant-min)*n;
    up:= 0;
    for j:= 1 step 1 until n do if y[j]≠0 then v[j]:= 0 else
    begin v[j]:= giant; up:= up+1; td[up]:= j end;
    for i:= 1 step 1 until n do
    begin ui:= u[i];
      for k:= 1 step 1 until up do
      begin j:= td[k]; vj:= v[j]; a:= aij+ui;
        if a<vj then begin v[j]:= a; lab[j]:= i end
      end
    end;
    for k:= 1 step 1 until up do
    begin j:= td[k]; i:= lab[j];
      if i≠0 then
      begin if x[i] =0 then
        begin x[i]:= j; y[j]:= i end
      end
    end;
    for io:= 1 step 1 until n do if x[io]=0 then
    begin min:= giant; i:= io;
      for j:= 1 step 1 until n do
      begin d[j]:= dj:= aij-v[j]; lab[j]:= io; td[j]:= j;
        if dj<min then begin min:=dj; jo:= j end
      end;
      u[io]:= -min; td[jo]:= n; td[n]:= jo; up:= n-1;
      for i:= y[jo] while i>0 do
      begin h:= min+u[i]; min:= giant;
        for k:= 1 step 1 until up do
        begin j:= td[k]; s:= h+aij-v[j]; dj:= d[j];
      if s<dj then begin d[j]:= dj:= s; lab[j]:= i end;
      if dj<min then begin min:= dj; ko:= k end
    end;
    jo:= td[ko]; td[ko]:= td[up]; td[up]:= jo; up:= up-1
  end;
  u[io]:= -min;
  for k:= up+2 step 1 until n do
  begin j:= td[k]; h:= d[j]-min;
    v[j]:= v[j]+h; i:= y[j]; u[i]:= u[i]+h
  end;
  for j:= jo while j>0 do
  begin y[j]:= i:= lab[j]; jo:= x[i]; x[i]:= j end
 end;
 s:= 0; for j:= 1 step 1 until n do s:= s+v[j]-u[j];
 assignment:= s
end;
```