September, 1979 LIDS-R-945

A New Algorithm for the Assignment Problem *

by

Dimitri P. Bertsekas

Dept. of Electrical Engineering and Computer Science
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

July 1979

* Work supported by Grant NSF ENG-7906332

Abstract

We propose a new algorithm for the classical assignment problem. The algorithm resembles in some ways the Hungarian method but differs substantially in other respects. The worst case computational complexity of one implementation of the algorithm for full dense, all integer, NxN problems is $O(N^3)$ - the same as the Hungarian method. Its average complexity, however, seems to be considerably better. In a large number of randomly generated problems the algorithm has consistently outperformed an efficiently coded version of the Hungarian method by a broad margin. The factor of improvement increases with problem dimension and reaches an order of magnitude for N equal to several hundreds.

1. Introduction

The assignment problem was among the first linear programming problems to be studied extensively. It arises often in practice and it is a fundamental problem in network flow theory since a number of other problems, such as the shortest path, weighted matching, transportation, and minimum cost flow problems can be reduced to it ([1], p.186, 187). It is characteristic in this respect that the first specialized method for the assignment problem, namely Kuhn's Hungarian method [2], was subsequently extended for solution of much more general network flow problems. Furthermore, some of its main ideas were instrumental in the development of more general methods such as the out-of-kilter and nonbipartite matching methods. This suggests that the assignment problem is not only an important problem in itself, but also represents a suitable testing ground for new computational ideas in network flow theory. It is for this reason that we restrict attention to the assignment problem even though the ideas of this paper have extensions to more general problems.

In practice the assignment problem is currently solved by either specialized forms of the simplex method [3] - [5], or by means of versions of Kuhn's Hungarian method [6], [7]. There exist several competing codes for assignment and in fact there seems to be some controversy regarding the relative merits of simplex codes and primal-dual (i.e. Hungarian) codes [6], [8]. A recent well documented computational study [7] finds simplex and primal-dual codes roughly comparable. From our own analysis and computational experience (see Section 3) it appears that the reported performance of primal-dual methods can be significantly improved by efficient implementation. We do not know whether the same is true for simplex codes. We note that we did not

have access to any existing special code for the assignment problem so our reasoning is based exclusively on documentation by other researchers rather than personnal experience.

The computational complexity of many of the existing codes is unknown and in fact some of these codes ([3], [6]) are proprietary. It is known that the complexity of the Hungarian method for full dense, all integer, NxN assignment problems is $0(N^3)$ ([1], p. 205), but the complexity of the implementation of [7] which is reported to be comparable to the best simplex type methods, is apparently worse than $0(N^3)$ (see Section 3). To our knowledge there is no simplex type method with complexity as good as $0(N^3)$.

The purpose of this paper is to propose a new method for solving the assignment problem. We show in the next section that the worst case complexity of the pure form of the method for full dense, all integer, NXN problems with the elements of the assignment matrix taking values in the interval [0,R] is $0(N^3) + 0(RN^2)$. It appears, however, that for all values of R, large or small, the method performs at its best when it is combined with the Hungarian method. This combination is described in Section 4 and the worst case complexity of the resulting method is $0(N^3)$. Its average complexity, however, seems to be substantially better than both $0(N^3)$ as well as the average complexity of the Hungarian method. This is demonstrated by means of extensive computational experiments with randomly generated problems. These experiments show that the new method consistently outperforms an efficiently implemented version of the Hungarian method [$0(N^3)$ complexity] by a broad margin. Indeed, out of more than a thousand randomly generated problems solved with $N \geq 20$ we did not find a single problem where

our method did not work faster than the Hungarian method. Furthermore, the factor of improvement increases with N thereby suggesting a better average complexity. For large problems with N being several hundreds our method can converge ten or more times faster than the Hungarian method.

Since we have been unable to characterize analytically the average complexity of our method we cannot claim to fully understand the mechanism of its fast convergence. This seems to be a difficult problem and in fact we do not know of a corresponding average complexity result for the Hungarian method. On heuristic grounds, however, it appears that the new method owes its good performance principally to a phenomenon which we refer to as outpricing. This is explained more fully in the next section but basically it refers to a property of the method whereby during the course of the algorithm the prices of some sinks are increased by large increments - much larger than in the Hungarian method. As a consequence these sinks are temporarily or permanently outpriced by other sinks and are in effect driven out of the problem in the sense that they do not get labeled and scanned further - at least for relatively long time periods. As a result the algorithm requires fewer row operations since in effect it deals with a problem of lower dimension.

There are a number of open questions regarding the ideas of this paper. The performance of our algorithm has been extensively tested with full dense problems but not with large sparse problems. It will become evident to the reader that our method is at least as suitable for exploiting sparsity as the Hungarian method, but we have not yet developed a sparse problem code. Common sense suggests also that problem sparsity will enhance the outpricing phenomenon thereby inducing even better performance for our

method, but this remains to be verified. We also expect that algorithms similar to the one of the present paper can be developed for other more general problems such as transportation, minimum cost flow problems, etc. Such algorithms are currently under investigation. Some preliminary work is reported in [9].

2. The Pure Form of the Algorithm

Consider a bipartite graph consisting of two sets of nodes S and T each having N elements, and a set of directed links L with elements denoted (i,j) where $i \in S$ and $j \in T$. We refer to elements of S and T as sources and sinks respectively. Each link (i,j) has a weight a_{ij} associated with it. By an assignment we mean a subset X of links such that for each source i (sink j) there is at most one link in X with initial node i (terminal node j). We say that a source i is unassigned under X if $(i,j) \notin X$ for all $(i,j) \in L$. Otherwise we say that source i is assigned under X. We use similar terminology for sinks. We wish to find an assignment that maximizes $\sum_{(i,j)\in X} a_{ij}$ over all assignments X of cardinality N.

Throughout the paper we will assume the following:

a) There exists at least one assignment of cardinality N.

b) The weights a_{ij} are nonegative integers not all zero, and the maximal weight will be denoted by R, i.e.

$$R = \max_{(i,j)\in L} a_{ij} > 0$$

c) For each source i there are at least two links (i,j) in L.

The nonnegativity assumption on the weights involves no loss of generality since if we add a constant to all weights the problem remains essentially the same. Assumption c) also involves no loss of generality,

since if for some source i there is only one link $(i,j)\varepsilon L$, then (i,j) is certainly part of any optimal assignment and as a result nodes i and j can be removed from the problem thereby reducing dimensionality. We use assumption c) for convenience in stating our algorithm.

The assignment problem can be embedded into the linear program

$$(1) \qquad \text{maximize} \quad \sum_{(i,j)\varepsilon L} a_{ij}\, x_{ij}$$

$$\text{subject to} \quad \sum_{(i,j)\varepsilon L} x_{ij} = 1 \quad , \qquad \forall\, i = 1,\ldots,N$$

$$\sum_{(i,j)\varepsilon L} x_{ij} = 1 \quad , \qquad \forall\, j = 1,\ldots,N \quad .$$

The corresponding dual problem in the vectors $m = (m_1,\ldots,m_N)$, $p = (p_1,\ldots,p_N)$ is

$$(2) \qquad \text{minimize} \quad \sum_{i=1}^{N} m_i + \sum_{j=1}^{N} p_j$$

$$\text{subject to } m_i + p_j \geq a_{ij} \quad , \qquad \forall\, (i,j)\varepsilon L.$$

The scalars p_j and $(a_{ij} - p_j)$ will be referred to as <u>prices</u> and <u>profit margins</u> respectively. From complementary slackness we have that if (i,j) is part of an optimal assignment, then for any optimal solution (m,p) of the dual problem we have

$$(3) \qquad m_i = \alpha_{ij} - p_j = \max \{\alpha_{in} - p_n \,|\, \text{all } n \text{ with } (i,n)\varepsilon L\}$$

i.e., at an optimum source i is assigned to a sink j offering maximum profit margin relative to the price vector p.

The Hungarian method solves the dual problem by generating for $k = 0,1,\ldots$ vectors m^k, p^k together with a corresponding assignment X^k.

For each k the vectors m^k, p^k are dual feasible and for all $(i,j) \in X^k$ the complementary slackness condition (3) is satisfied. These features are also shared by the algorithm we propose. In the Hungarian method if a source i or sink j is assigned under some $X^{\bar{k}}$ it continues to be assigned under all X^k, $k \geq \bar{k}$. In our method this is also true for sinks j but it is not true for sources i. Sources may come in and out of the current assignment with attendant changes in the vectors m and p. Another difference of our method over the Hungarian method is the manner in which the dual variables are incremented. In the Hungarian method every change in the vectors m and p induces a reduction of the value of the dual cost function. In our method when changes are made on the values of m and p all that is guaranteed is that the objective function value will not increase.

Both the Hungarian method and the algorithm we propose involve flow augmentations along alternating paths, and changes in the values of dual variables. The roles of these two devices are, however, somewhat different in the two methods. The Hungarian method is geared towards assigning the unassigned sources by searching directly for an augmenting path along links (i,j) with $m_i + p_j = a_{ij}$. Dual variable changes are effected only when no more progress can be made towards generating such an augmenting path. In our method the roles of searching for an augmenting path and changing the dual variables are in effect reversed. Primary consideration is given to increasing the prices of assigned sinks as much as is possible without violating the complementary slackness constraint. The aim here is to make the prices of unassigned sinks "competitive" with those of assigned sinks. Augmentation is in some sense incidental and takes place either when it

can be effected at essentially no computational cost, or when it is no more possible to continue the process of increasing prices of assigned sinks without violating the complementary slackness constraint.

We now describe formally our method. A more specific implementation together with an initialization procedure will be given in Section 4.

The method is initialized with any integer vectors m^o, p^o that are dual feasible and with X^o being the empty assignment.

For $k = 0,1,\ldots$, given (m^k, p^k, X^k) satisfying for all $i \in S$

$$m_i^k + p_i^k = a_{ij} \qquad \text{if } (i,j) \in X^k$$

$$m_i^k + p_n^k \geq a_{in} \qquad \forall\ (i,n) \in L$$

we stop if X^k has cardinality N. Otherwise we use the following procedure to generate $(m^{k+1}, p^{k+1}, X^{k+1})$ satisfying for all $i \in S$

$$m_i^{k+1} + p_j^{k+1} = a_{ij} \qquad \text{if } (i,j) \in X^{k+1}$$

$$m_i^{k+1} + p_n^{k+1} \geq a_{in} \qquad \forall\ (i,n) \in L \ .$$

(k+1)st Iteration of the Algorithm

Choose a source $\bar{i}$ which is unassigned under X^k. Compute the maximum profit margin

$$\bar{m} = \max \{a_{\bar{i}j} - p_j^k \mid (\bar{i}, j) \in L\} \tag{4}$$

and find a sink $\bar{j}$ such that

$$\bar{m} = a_{\bar{i}\bar{j}} - p_{\bar{j}}^k \quad . \tag{5}$$

Compute also the "second maximum" profit margin

$$\tilde{m} = \max \{a_{\bar{i}j} - p_j^k \mid (\bar{i}, j) \in L, \quad j \neq \bar{j}\} \quad . \tag{6}$$

Proceed as described for the following two cases:

Case 1 ($\bar{m} > \tilde{m}$, or $\bar{m} = \tilde{m}$ and sink $\bar{j}$ is unassigned under X^k):

Set

$$m_i^{k+1} = \begin{cases} m_i^k & \text{for } i \neq \bar{i} \\ \tilde{m} & \text{for } i = \bar{i} \end{cases} \tag{7}$$

$$p_j^{k+1} = \begin{cases} p_j^k & \text{for } j \neq \bar{j} \\ p_j^k + \bar{m} - \tilde{m} & \text{for } j = \bar{j} \end{cases} \quad . \tag{8}$$

If $\bar{j}$ is unassigned under X^k, add $(\bar{i}, \bar{j})$ to X^k, i.e.

$$X^{k+1} = X^k \cup \{(\bar{i}, \bar{j})\} \quad .$$

If $(\hat{i}, \bar{j}) \in X^k$ for some $\hat{i} \in S$, obtain X^{k+1} from X^k by replacing $(\hat{i}, \bar{j})$ by $(\bar{i}, \bar{j})$, i.e.

$$X^{k+1} = [X^k \cup \{(\bar{i}, \bar{j})\}] - \{(\hat{i}, \bar{j})\} \quad .$$

This completes the (k+1)st iteration of the algorithm.

Case 2 ($\bar{m} = \tilde{m}$ and, for some $\hat{i} \in S$, $(\hat{i}, \bar{j}) \in X^k$)

Give the label "0" to $\bar{i}$. Set

$$m_{\bar{i}}^k \leftarrow \bar{m} \quad (9)$$ †

$$\pi_j = \infty \quad , \quad j = 1,\ldots,N,$$

and perform the following labeling procedure (compare with [1], p.205).

Step 1 (Labeling): Find a source i with an unscanned label and go to Step 1a, or find a sink $j \neq \bar{j}$ with an unscanned label and $\pi_j = 0$, and go to Step 1b. If no such source or sink can be found go to Step 3.

Step 1a: Scan the label of source i as follows. For each $(i,j) \in L$ for which $m_i^k + p_j^k - a_{ij} < \pi_j$ give node j the label "i" (replacing any existing label) and set $\pi_j = m_i^k + p_j^k - a_{ij}$. Return to Step 1.

Step 1b: Scan the label on the sink $j \neq \bar{j}$ with $\pi_j = 0$ as follows. If j is unassigned under X^k go to Step 2. Otherwise identify the unique source i with $(i,j) \in X^k$, and give i the label "j". Return to Step 1.

Step 2 (Augmentation): An augmenting path has been found that alternates between sources and sinks, originates at source $\bar{i}$ and terminates at the sink j identified in Step 1b. The path can be generated by "backtracing" from label to label starting from the terminating sink j to the originating source $\bar{i}$. Add to X^k all links on the augmenting path that are not in X^k and remove from X^k all links on the path that are in X^k to obtain X^{k+1}. Set m^{k+1}, p^{k+1} as in (7) and (8). This completes the (k+1)st iteration of the algorithm.

† The notation A←B means that the current value of the variable A is changed to the current value of B.

Step 3 (Change of dual variables): Find

$$\delta = \min\{\pi_j \mid j \in T, \ \pi_j > 0\} \quad .$$

Set

$$m_i^{k+1} = \begin{cases} m_i^k - \delta & \text{if } i \text{ has been labeled} \\ m_i^k & \text{if } i \text{ has not been labeled} \end{cases}$$

[Recall here that $m_{\bar{i}}^k$ was set equal to $\bar{m}$ in the initialization of the labeling procedure c.f. (9)]. Set

$$p_j^{k+1} = \begin{cases} p_j^k + \delta & \text{if } \pi_j = 0 \\ p_j^k & \text{if } \pi_j > 0 \end{cases}$$

Obtain X^{k+1} from X^k by replacing $(\hat{i}, \bar{j})$ by $(\bar{i}, \bar{j})$, i.e.

$$X^{k+1} = [X^k \cup \{(\bar{i}, \bar{j})\}] - \{(\hat{i}, \bar{j})\} \quad .$$

This completes the (k+1)st iteration of the algorithm.

Notice that, by contrast with the Hungarian method (see Section 3), the iteration need not terminate with a flow augmentation. It can also terminate with a change in the dual variables (Case, 1, $\bar{j}$ is assigned, or Case 2, Step 3) but in this case the source $\bar{i}$ under consideration becomes assigned under X^{k+1}, while some other source which was assigned under X^k becomes unassigned under X^{k+1}.

Unfortunately the description of the algorithm is quite complicated. For this reason some explanatory remarks are in order. Case 1 is easy to understand so we concentrate on Case 2. (Actually Case 1 is a degenerate special case of Case 2 as the reader can verify. We have decided to state separately the two cases in order to enhance clarity). In Case 2 we basically try to find an augmenting path not containing $\bar{j}$ from source $\bar{i}$

to an unassigned sink. There are two possibilities. Either an augmenting path will be found through Step 2 of the labeling procedure, or else a change in the dual variables will be effected through Step 3. In the first case the link $(\hat{i}, \bar{j})$ will be retained in X^{k+1} and the sink at which the augmenting path terminates will be assigned under X^{k+1} as shown in Figure 1. In the second case the link $(\hat{i}, \bar{j})$ will be replaced by $(\bar{i}, \bar{j})$ in X^{k+1} and no new sink will be assigned under X^{k+1} as shown in Figure 2. The dual variables will change, however, by the minimum amount necessary to obtain $m_i^{k+1} + p_j^{k+1} = a_{ij}$ for some labeled source i and labeled but unscanned sink j. A similar (but not identical) labeling procedure is used in the Hungarian method (see Section 3). In the Hungarian method after a change in dual variables occurs the labeling procedure continues until an augmenting path is found. By contrast in our method the labeling procedure terminates with a change in the dual variables and a new iteration is started with a new unassigned sink.

The order in which unassigned sources are chosen by the algorithm is not essential for the convergence result of Proposition 1. However, from the practical point of view it is important that a scheme that ensures fairness for all unassigned sources be utilized. In subsequent examples as well in our implementation of the method the scheme adopted is one whereby a list of unassigned sources is maintained by the algorithm. At each iteration the source at the top of the list is chosen and if a new source becomes unassigned (Case 1, $\bar{j}$ assigned under X^k, or Case 2, Step 3) it is placed at the bottom of the list.

We illustrate the algorithm by means of an example.

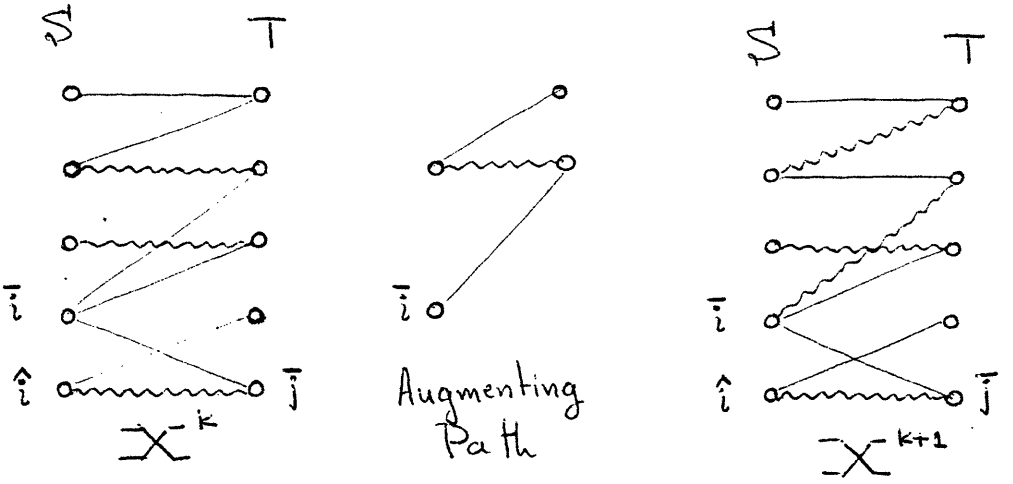

Figure 1: Case 2, Step 2. ∘⁓∘ = Assigned link,
∘——∘ = Unassigned link (i,j) with $m_i + p_j = a_{ij}$,
∘ ∘ = Unassigned link (i,j) with $m_i + p_j > a_{ij}$.

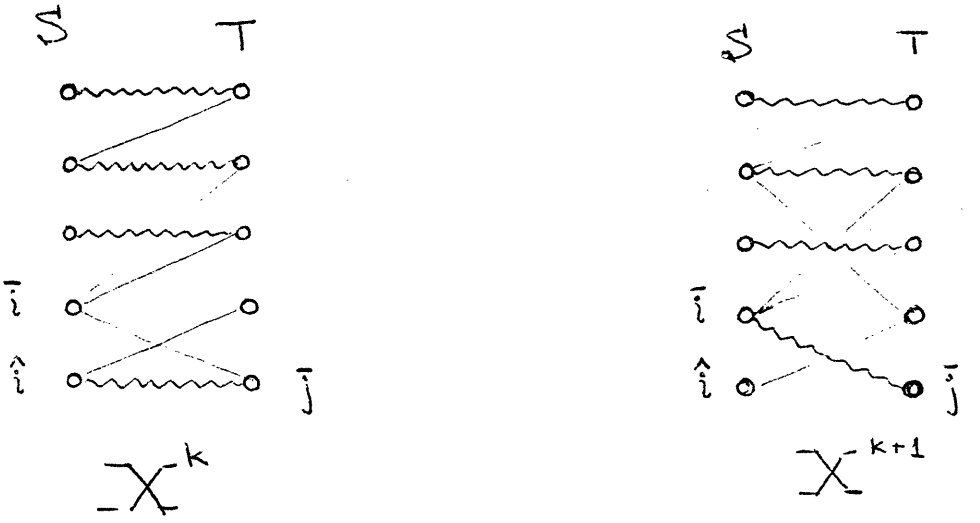

Figure 2: Case 2, Step 3. ∘⁓∘ = Assigned link,
∘——∘ = Unassigned link (i,j) with $m_i + p_j = a_{ij}$,
∘ ∘ = Unassigned link (i,j) with $m_i + p_j > a_{ij}$.

Example 1: Consider the 4x4 full dense problem represented by the following initial tableau.

j	m_i \ p_j	0	0	0	0
	10	1	3	6	1
	10	2	4	7	3
	10	2	5	7	2
	10	1	3	5	1

The matrix of weights is shown in the lower right portion of the tableau. The row above the matrix gives the initial prices arbitrarily chosen to be zero. The column to the left of the matrix shows the initial profit margins. We have chosen $m_i = 10$ for all i - one of the many choices satisfying feasibility. The extreme left column gives the sinks j, if any, to which sources are assigned. Here we are starting with the empty assignment. We describe successive iterations of the algorithm. The corresponding tableaus are given in Figure 3.

1st Iteration: We choose the unassigned source 1. We have $\bar{m} = 6$, $\tilde{m} = 3$, $\bar{j} = 3$. We are thus in Case 1.

2nd Iteration: We choose the unassigned source 2. We have $\bar{m} = \tilde{m} = 4$. Suppose $\bar{j} = 2$. Since $\bar{j}$ is unassigned we come again under Case 1. (If we had chosen $\bar{j} = 3$ then we would have come under Case 2. We would have

j	m_i \ p_j	0	0	3	0
3	3	1	3	6	1
	10	2	4	7	3
	10	2	5	7	2
	10	1	3	5	1

After 1st Iteration

j	m_i \ p_j	0	0	3	0
3	3	1	3	6	1
2	4	2	4	7	3
	10	2	5	7	2
	10	1	3	5	1

After 2nd Iteration

j	m_i \ p_j	0	1	3	0
3	3	1	3	6	1
	4	2	4	7	3
2	4	2	5	7	2
	10	1	3	5	1

After 3rd Iteration

j	m_i \ p_j	0	2	4	0
3	2	1	3	6	1
	4	2	4	7	3
	4	2	5	7	2
2	1	1	3	5	1

After 4th Iteration

j	m_i \ p_j	0	2	4	0
3	2	1	3	6	1
4	3	2	4	7	3
	4	2	5	7	2
2	1	1	3	5	1

After 5th Iteration

j	m_i \ p_j	0	2	4	0
3	2	1	3	6	1
4	3	2	4	7	3
2	3	2	5	7	2
1	1	1	3	5	1

After 6th Iteration

Figure 3

obtained the degenerate augmenting path (2,2) through Step 2 of the labeling procedure and the end result would have been the same).

3rd Iteration: We choose the unassigned source 3. Here $\bar{m} = 5$, $\tilde{m} = 4$, $\bar{j} = 2$. We are thus again in Case 1. But now source 2 will be driven out of the assignment and will be replaced by source 3.

4th Iteration: We choose the unassigned source 4. Here $\bar{m} = \tilde{m} = 2$. Suppose $\bar{j} = 2$. We are now in Case 2 with $\hat{i} = 3$. Applying the labeling procedure we label first source 4. A simple computation shows that sink 3 is labeled from source 4 and then source 1 is labeled from sink 3. No more labels can be scanned so we are in Step 3 of Case 2. Source 4 will enter the assignment and source 3 will be driven out. We have $\delta = 1$ and the corresponding tableau is shown in Figure 3.

5th Iteration: We choose the unassigned source 2. Here $\bar{m} = \tilde{m} = 3$. Suppose $\bar{j} = 4$. We are in Case 1 and (2,4) will be added to the assignment. (The result would be the same if $\bar{j} = 3$ in which case the degenerate augmenting path (2,4) would be obtained via Step 2 of Case 2).

6th Iteration: We choose the unassigned source 3. Here $\bar{m} = \tilde{m} = 3$. Suppose $\bar{j} = 3$. We are in Case 2 with $\hat{i} = 1$. Applying the labeling procedure we label first source 3. Sink 2 is labelled from source 3 and then source 4 is labeled from sink 2. Next sinks 1 and 4 are labeled from source 4. Sink 1 is unassigned and this yields the augmenting path (3,2), (4,2), (4,1) in Step 2 of Case 2. The algorithm terminates.

We make the following observations regarding the algorithm.

(a) The sequences $\{m_i^k\}$, $i = 1,\ldots,N$ are monotonically nonincreasing while the sequences $\{p_j^k\}$ are monotonically nondecreasing.

(b) If for some $\bar{k}$ a sink j is assigned under $X^{\bar{k}}$ then it remains assigned under X^k for all $k \geq \bar{k}$.

(c) At each iteration initiated with an unassigned source $\bar{i}$ one of two things can happen. Either the cardinality of the assignment increases by one (Case 1, $\bar{j}$ is unassigned, or Case 2, step 2), or else at least one variable m_i will decrease strictly by an integer amount and at least one price will increase strictly by an integer amount (Case 1, $\bar{j}$ is assigned, or Case 2, step 3).

(d) For every k and $i \in S$ we have

$$m_i^k + p_j^k = a_{ij} \qquad \text{if } (i,j) \in X^k \tag{10}$$

$$m_i^k + p_n^k \geq a_{in} \qquad \forall\ (i,n) \in L. \tag{11}$$

From (10) and (11) we see that dual feasibility and complementary slackness are maintained throughout the algorithm. Thus if the algorithm terminates (by necessity at an assignment of cardinality N), then the assignment and dual variables obtained are optimal. The following proposition shows that termination is guaranteed under the assumption made earlier that there exists at least one assignment of cardinality N.

Proposition 1: The algorithm of this section terminates at an optimal assignment in a finite number of iterations.

Proof: Assume that the algorithm does not terminate. Then after a finite number of iterations the cardinality of the current assignment will remain constant and at each subsequent iteration at least one variable m_i will decrease strictly and at least one price will increase strictly [observation c) above]. Hence the sets S_∞ , T_∞ defined by

$$S_\infty = \{i\varepsilon S \mid \lim_{k\to\infty} m_i^k = -\infty\}$$

$$T_\infty = \{j\varepsilon T \mid \lim_{k\to\infty} p_j^k = \infty\}$$

are nonempty. For all k sufficiently large the sinks in T_∞ are assigned under X^k [observation (b)], and they must be assigned to a source in S_∞ [observation (d)]. Furthermore, since the algorithm does not terminate, some source in S_∞ must be unassigned under X^k. It follows that the cardinality of S_∞ is strictly larger than that of T_∞ . From (11) it is clear that there cannot exist a link $(i,j)\varepsilon L$ such that $i\varepsilon S_\infty$ and $j\notin T_\infty$. Thus we have

$$\{j \mid (i,j)\varepsilon L, \quad i\varepsilon S_\infty\} \subset T_\infty$$

while S_∞ has larger cardinality than T_∞ . This contradicts the assumption that there exists an assignment of cardinality N. Q.E.D.

We now estimate the worst case computational complexity of the algorithm for full dense problems (i.e. for problems such that $(i,j)\varepsilon L$ for all $i\varepsilon S, j\varepsilon T$). By subtracting (11) from (10) we have for all k, $i\varepsilon S$ and $n\varepsilon T$

$$(12) \qquad p_j^k - p_n^k \geq a_{ij} - a_{in} \geq R \quad \text{if } (i,j)\varepsilon X^k.$$

Suppose that B_1 and B_2 are lower and upper bounds for all initial prices, i.e.

$$(13) \qquad B_1 \leq p_j^o \leq B_2 \ , \qquad \forall\ j\varepsilon T.$$

For each k there must be at least one unassigned sink, say $\bar{n}_k$ and we must have $p^k_{n_k} = p^o_{n_k} \leq B_2$. It follows from (12) and observation a) that

$$B_1 \leq p^o_j \leq p^k_j \leq R + B_2 \quad , \quad \forall \ k = 0,1,\ldots, \quad j \in T. \tag{14}$$

It is easy to see that there is an integer γ such that the kth iteration of the algorithm requires at most $\gamma n_k N$ computer operations where

$$n_k = \begin{cases} 1 & \text{in Case 1} \\ \text{number of labelled sources in Case 2} \end{cases}$$

There can be at most N iterations at which the cardinality of X^k increases so the number of operations needed for these is bounded by γN^3. At each iteration k for which the cardinality of X^k does not increase, n_k prices will be increased by at least unity. If follows from $n_k \leq N$ and (14) that the total number of these iterations is bounded by $R + (B_2 - B_1)$. Hence the total number of operations for these iterations is bounded by $\gamma(R + B_2 - B_1)N^2$. If we restrict the initial prices so that for some integer $\bar{\gamma}$ we have $B_2 - B_1 \leq \bar{\gamma}R$ (as indeed would be the case with any reasonable initialization procedure including the one used in our experiments) we obtain the upper bound $\gamma N^3 + \gamma(1 + \bar{\gamma})RN^2$ on the number of operations necessary for problem solution. Thus the worst case complexity of the pure form of the algorithm for full dense problems is

$$O(N^3) + O(RN^2) \quad . \tag{15}$$

While the worst case complexity of the algorithm is inferior to the one of the Hungarian method for large R, we would like to emphasize that, as experience with the simplex method has shown, worst case complexity and

average complexity of an algorithm can be quite different. Thus two algorithms with comparable worst case complexity can differ substantially in their performance in solving randomly generated problems or problems typically arising in practice. In our computational experiments with random problems we found that the algorithm often performed surprisingly better than the Hungarian method. The following example illustrates what we believe is the mechanism responsible for this superior performance.

Example 2: Consider the NxN full dense problem specified by the assignment matrix

$$\begin{bmatrix} N & & & & & \\ N & N-1 & & & & \\ N & N-1 & N-2 & & & \\ \cdot & \cdot & \cdot & & & \\ \cdot & \cdot & \cdot & & & \\ \cdot & \cdot & \cdot & 3 & & \\ \cdot & \cdot & \cdot & 2 & 2 & \\ N & N-1 & N-2 & 3 & 2 & 1 \end{bmatrix}$$

with all elements above the diagonal equal to zero. Let us trace the iterations of our algorithm for this problem starting with $p = 0$ and the empty assignment. In the first iteration source 1 is chosen, we have $\bar{m} = N$, $\tilde{m} = 0$, $\bar{j} = 1$ and, under Case 1, link (1,1) is added to the assignment while price p_1 is increased to N. In the second iteration source 2 is chosen, we have $\bar{m} = N-1$, $\tilde{m} = 0$, $\bar{j} = 2$ and, under Case 1, link (2,2) is added to the assignment while price p_2 is increased to N-1. Continuing in this manner we see that at the kth iteration link (k,k) will be added to the assignment and the price p_k is increased to k. Thus the algorithm terminates in N iterations and its computation time for this problem is $O(N^2)$.

If we apply the Hungarian method of the next section to the same problem with the same initial conditions we find that at every iteration except the first all sources will be scanned leading to a computation time $O(N^3)$. - essentially N times slower than with our method. This type of example does not depend on the initial prices as much as it may appear, since if the standard initialization procedure of the Hungarian method were adopted (see next section) then by adding a row of the form [N,N....N] and a column consisting of zeros in all positions except the last to the assignment matrix, the computation times of the two methods remain essentially unchanged for large N.

In analyzing the success of our method in this example we find that it is due to the fact that by contrast with the Hungarian method, it tends to increase prices by as large increments as is allowed by the complementary slackness constraint. Thus in the first iteration p_1 is increased by N. This has the effect of outpricing sink 1 relative to the other sinks in the sense that the price of sink 1 is increased so much that, together with source 1, it plays no further role in the problem. Thus in effect after the first iteration we are dealing with a problem of lower dimension. By contrast the Hungarian method in the first iteration will add link (1,1) to the assignment but will not change its price from zero. As a result source 1 and sink 1 are labelled and scanned at every subsequent iteration. Outpricing sink 1 has another important effect namely it allows a large price increase and attendant outpricing for sink 2 at the second iteration. This in turn allows outpricing sink 3 at the third iteration and so on. This illustrates that outpricing has the character of a chain phenomenon

whereby outpricing of some sinks enhances subsequent outpricing of other sinks.

The preceding example is obviously extreme and presents our method in the most favorable light. If, for example, the first row contained some non-zero elements other than N, the change in the price p_1 at the first iteration would be smaller than N. In this case the effect of outpricing, while still beneficial, would not be as pronounced and it would drive source 1 and sink 1 out of the problem only temporarily until the prices of other sources increase to comparable levels.

While we found that the algorithm of this section performs on the average substantially better than the Hungarian method for randomly generated problems, we often observed a pattern whereby the algorithm would very quickly assign most sinks but would take a disproportionally large number of iterations to assign the last few sinks. For example for N = 100 we observed some cases where 75% of the iterations were spent for assigning the lasttwo or three sinks. Predictably in view of the complexity estimate (15) this typically occured for large values of R. This points to the apparent fact that the beneficial effect of outpricing is most pronounced in the initial and middle phases of the algorithm but sometimes tends to be exhausted when there are only few unassigned sources. The remedy suggesting itself is to combine the algorithm with the Hungarian method so that if the algorithm does not make sufficiently fast progress a switch is made to the Hungarian method. A reasonable implementation of such a combined method will be presented in Section 4. Its worst case computational complexity is $O(N^3)$ - the same as for the Hungarian method. Its performance on randomly generated problems was found to be consistently superior to both the Hungarian method and the pure form of the algorithm of this section. Significantly, the factor of improvement over the Hungarian method increases with problem

dimension.

3. Implementations of the Hungarian Method

Since we intend to compare and combine our method with the Hungarian method we describe here the implementation that we used in our computational experiments.

The initial assignment X^o is taken to be the empty assignment and the initial dual variables are chosen according to the usual scheme

$$m_i^o = \max\{ a_{ij} | j \in T\} \quad , \quad i = 1,\ldots,N$$

$$p_j^o = \max \{a_{ij} - m_i^o | i \in S\} \quad j = 1,\ldots,N \quad .$$

Notice that the equations above imply

$$m_i^o + p_j^o \geq a_{ij} \quad , \quad \forall\ (i,j) \in L.$$

For $k = 0,1,\ldots,N-1$, given (m^k, p^k, X^k) satisfying for all $i \in S$

$$m_i^k + p_j^k = a_{ij} \quad \text{if } (i,j) \in X^k \ \dagger$$

$$m_i^k + p_n^k \geq a_{in} \quad \forall\ (i,n) \in L$$

we obtain $(m^{k+1}, p^{k+1}, X^{k+1})$ satisfying for all $i \in S$

$$m_i^{k+1} + p_j^{k+1} = a_{ij} \quad \text{if } (i,j) \in X^{k+1}$$

$$m_i^{k+1} + p_n^{k+1} \geq a_{in} \quad \forall\ (i,n) \in L$$

according to the following labelling procedure.

---

† Actually throughout the algorithm the stronger condition $m_i^k = \max\{a_{in} - p_n^k | (i,n) \in L\}$ holds for all k and $i \in S$. We state the algorithm in this form in order to emphasize that (m^k, p^k, X^k) need only satisfy the same conditions as in the algorithm of the previous section, thereby simplifying the transition from one algorithm to the other.

Step 0: Give the label "0" to all unassigned sources under X^k. Set $\pi_j = \infty$, $j = 1,\ldots,N$.

Step 1 (Labeling) : Find a source i with an unscanned label and go to Step 1a, or find a sink j with an unscanned label and $\pi_j = 0$ and go to Step 1b. If no such source or sink can be found, to to Step 3.

Step 1a: Scan the label of source i as follows. For each $(i,j)\in L$ for which $m_i^k + p_j^k - a_{ij} < \pi_j$ give node j the label "i" (replacing any existing label) and set $\pi_j = m_i^k + p_j^k - a_{ij}$. Return to Step 1.

Step 1b: Scan the label on the sink j with $\pi_j = 0$ as follows. If j is unassigned under X^k go to Step 2. Otherwise identify the unique source i with $(i,j)\in X^k$, and give i the label "j". Return to Step 1.

Step 2 (Augmentation): An augmenting path has been found that alternates between sources and sinks originating at a source unassigned under X^k and terminating at the sink j identified in Step 1b. The path is generated by "backtracing" from label to label starting from the terminating sink j. Add to X^k all links on the augmenting path that are not in X^k and remove from X^k all links on the augmenting path that are in X^k. This gives the next assignment X^{k+1}. Set $m^{k+1} = m^k$, $p^{k+1} = p^k$. (Note that m^k and p^k may have been changed through Step 3). This completes the iteration of the algorithm.

Step 3 (Change of Dual Variables): Find

$$\delta = \min\{\pi_j \mid j\in T,\ \pi_j > 0\} \quad . \tag{16}$$

Set

$$m_i^k \leftarrow m_i^k - \delta \qquad \text{for all } i\in S \text{ that are labeled}$$

$$p_j^k \leftarrow p_j^k + \delta \qquad \text{for all } j\in T \text{ with } \pi_j = 0$$

$$\pi_j \leftarrow \pi_j - \delta \qquad \text{for all } j\in T \text{ that are labeled and } \pi_j > 0.$$

and go to Step 1.

Notice that the labeling procedure will terminate only upon finding an augmenting path at Step 2 and therefore at each iteration the cardinality of the current assignment is increased by one. Thus X^N has cardinality N and is an optimal assignment. We now estimate the complexity of the algorithm. Scanning labels in Steps 1a and 1b requires at each iteration at most $0(N^2)$ operations for a full dense problem. The augmentation (Step 2) requires at most 0(N) operations. The number of operations needed at each iteration for Step 3 is 0(N) multiplied with the number of times Step 3 is performed at each iteration. We estimate this number as follows. The initial prices can be easily seen to satisfy

$$-R \leq p_n^o \leq 0 \qquad , \forall j \in T.$$

From (12) - (14) we see that we have

$$-R \leq p_j^o \leq p_j^k \leq R \qquad , \forall\ k = 0,1,\ldots,N-1, \quad j \in T$$

Since $m_i^o \leq R$ for all $i \in S$ and $\{m_i^k\}$ is monotonically decreasing it follows that

$$m_i^k \leq R \qquad \forall\ k = 0,1,\ldots,N-1, \quad i \in S.$$

During each iteration we have, using the two inequalities above, for each $j \in T$ and for some $i \in S$

$$\pi_j = m_i^k + p_j^k - a_{ij} \leq 2R.$$

Now the number of times that Step 3 is performed at each iteration is bounded by N as well as by the maximum value of π_j after the first label is scanned. Using the inequality above it follows that this number is bounded by $\min\{N,2R\}$. We thus obtain a bound $0(\min\{N,2R\}N)$ for the total number of

operations in Step 3 during a single iteration of the algorithm. Summing over N iterations yields the following complexity bound for full dense problems

$$(17) \qquad O(N^3) + O(\min\{N,2R\}N^2) \ .$$

This bound can also be written as $O(N^3)$.

There are other possible implementations of the Hungarian method. The one described in [7] performs Step 3 in a different way than we do. Instead of keeping track of the scalars π_j the entire matrix of "reduced cost coefficients" $c_{ij} = m_i^k + p_j^k - a_{ij}$ is maintained and updated each time Step 3 is performed. The increment of dual variable change δ of (16) is obtained from

$$\delta = \min\{c_{ij} | i \text{ is labelled, } j \text{ is unscanned}\}$$

The reduced cost coefficients are also updated to reflect the changes in m_i^k and p_j^k. Unfortunately for full dense problems this requires $O(N^2)$ operations thereby resulting in a complexity bound

$$(18) \qquad O(N^3) + O(\min\{N,2R\}N^3)$$

When R is small relative to N this is comparable to $O(N^3)$. But for large values of R the complexity bound reaches $O(N^4)$. This fact explains the strong dependence on R of the computation times given in [7], (see also Table 1). Furthermore it suggests that our implementation of the Hungarian method is superior to the one of [7] which was in turn found comparable to the best simplex and primal-dual codes currently available for the assignment problem.

The purpose of the preceding discussion has not been so much to demonstrate that our implementation of the Hungarian method is superior to other existing codes. In fact there isn't sufficient evidence to support

such a claim. Rather, our aim is to convince the reader that the Hungarian code that we have compared our algorithm with is quite efficient and at least comparable with state-of-the-art codes. To make this point more strongly we present in Table 1 computation times obtained using our Hungarian code, and computation times taken from [7] for NxN randomly generated problems with weights chosen according to a uniform distribution from [0,R]. The top entry in each cell gives the time in secs for our code. The middle and bottom entries give the time for the Hungarian code PDAC and the primal simplex code with alternating basis PACAB of [7] respectively. Each entry represents an average over ten full dense, random problems. The computation times did not vary greatly around the mean so we feel that for a sample size of ten it is valid to compare these times except for the fact that our times are on an IBM 370/168 computer while the times of [7] are on the CYBER 70/74 computer system at Georgia Tech. All programs were written in Fortran and compiled using the FTN compiler in the OPT=2 optimizing mode. Integer arighmetic has been used throughout and computation times exclude input and output operations. It is estimated in [7] that 1 solution second on a CDC for this type of problem is approximately equivalen to 1.43 seconds on the CYBER 70/74. Based on our experience with the IBM 370 and the CDC 6600, and data given in [4], we estimate that one solution second on the IBM 370 should be equivalent to somewhere between 2 and 3 seconds on the CYBER 70/74. This is also consistent with the computation times listed in Table 1 for R = 10 in which case the two Hungarian codes should be roughly equivalent since, for R = 10, Step 3 is hardly ever performed [compare also the estimates (17) and (18)]. It can be seen from Table 1 that even if we scale down the times of the Hungarian code of [7] by a factor of 3, our code comes out much superior for $R \geq 100$. Notice that

N \ R	10	100	1,000	10,000	100,000
50	0.038 0.092 0.258	0.058 0.290 0.464	0.070 0.537 0.478	0.073 0.715 0.502	0.073 0.704 0.484
100	0.103 0.253 1.246	0.343 1.079 2.505	0.430 2.795 2.987	0.475 5.365 2.849	0.484 6.576 3.153
150		1.036 3.639 6.917			1.498 24.382 7.868
200		2.383 7.355 12.375			3.695 53.395 16.316

TABLE 1: Computation times in secs of our Hungarian code on IBM 370 (top entry in each cell), the Hungarian code of [7] on CYBER 70/74 (middle entry), and the primal simplex code of [7] on CYBER 70/74 (bottom entry). Each entry is an average over 10 randomly generated full dense problems. One solution second on IBM 370 approximately equivalent to 2-3 seconds on CYBER 70/74.

the times given in Table 1 show remarkable agreement with qualitative predictions based on the complexity estimates (17) and (18).

4. Combinations with the Hungarian Method - Computational Results

As discussed at the end of Section 2 it appears advantageous to combine our new algorithm with the Hungarian method. A switch from the new algorithm to the Hungarian method is very simple in view of the similarities of the two methods. We have used the following scheme in our experiments:

We are making use of two lists of unassigned sources during execution of the algorithm. Each unassigned source is contained in one and only one of the two lists. We select at each iteration the unassigned source which is at the top of the first list. If in that iteration a new source becomes unassigned (Case 1, $\overline{j}$ assigned, or Case 2, Step 3) this source is placed at the bottom of the second list. Initially the first list contains all sources and the second list is empty. As the algorithm proceeds the size of the first list decreases while the size of the second list increases. When the first list is emptied the contents of the second list are transferred to the first and the second list becomes empty. We refer to the portion of the algorithm between the time that the first list is full to the time it is empty as a cycle. At the end of each cycle we compare the number of sources in the second list with the number of sources contained in the first list at the beginning of the cycle. If they are the same (implying that no augmentation occured during the cycle) a counter initially set at zero is incremented by one. The counter is also incremented by one if during the cycle Case 2, Step 3 was reached more than a fixed prespecified number of times (4 in our experiments) with the number of labelled sources being more than a fixed prespecified number (10 in our experiments).†

† Actually this last device does not seem to play an important role for practical purposes. It was introduced in order to make possible a proof of an $O(N^3)$ complexity bound for the combined algorithm.

At the point where the counter exceeds a prespecified threshold value a switch is made to the Hungarian method of the previous section. The threshold value was set at 0.1N in all of our experiments, but the average performance of the algorithm seems fairly insensitive to this value within broad limits. It is a straightforward but tedious exercise to show that the complexity of this combined algorithm is bounded by $O(N^3)$. The proof essentially consists of showing that at most $O(N^3)$ operations are necessary before a switch to the Hungarian method takes place. In almost all the problems we solved, the great majority (95-100%) of sinks were assigned by the new algorithm and the remainder by the Hungarian method after a switch was made. This was particularly true for small values of R when for most problems a switch to the Hungarian method was not necessary.

Finally regarding initialization we have in all cases chosen X^o = empty, and $p_j^o = 0$, $m_i^o = R$ for all i and j. However at the end of the first cycle (i.e. at the end of the Nth iteration) the prices of all unassigned sinks j are changed from $p_j^N = 0$ to

$$p_j^N = \max\{a_{ij} - m_i^N \mid i: \text{ assigned under } X^N\} \quad .$$

The remaining prices and all values m_i^N are left unchanged. This is in effect an initialization procedure quite similar to the one for the Hungarian method of the previous section. Its purpose is to reduce the prices of the unassigned sinks as much as possible without violating the complementary slackness constraint. It has worked quite well in our experiments.

Tables 2 and 3 show the results of our computational experiments with randomly generated full dense, NxN problems. Each entry represents an average over five problems, which were the same for all three methods and for

each N. The weights were chosen from a uniform distribution over [0,1] and subsequent multiplication by R(Table 2), or from a normal distribution N(0,1) and subsequent multiplication by Σ (Table 3). They were then truncated to the nearest integer. The programs were written in Fortran and compiled with the optimizing compiler in the OPT = 2 mode. The times given in the top entry of each cell refer to the IBM 370 at M.I.T. We give in the bottom entry of each cell the average number of sources scanned for each method (Case 1 in the new algorithm corresponds to one source scanned). The average computation time per source scanned does not differ much from one method to another, so the number of sources scanned represents a valid measure of comparison which is independent of the computer, compiler, programmer, and time of the day the run was made. The results clearly indicate that the combined method is overall superior to the others. The pure form of the new algorithm also appears superior to the Hungarian method, but not by as much as the combined method. Also the variance of computation time exhibited by the pure form of the algorithm is larger than those of the Hungarian and the combined methods. The combined method had the smallest variance in computation time over the three methods tested.

As a final comparison with existing methodology it is worth observing that the computation time of Table 2 for the combined method and 200x200 problems with weights in the range [0,100] is 0.526 seconds. There are five 200x200 NETGEN benchmark assignment problems with weights in the range [0,100] that have been solved by a number of presently available codes. The best solution times achieved [3], [7] range from 0.96 to 1.68 secs on a CDC 6600. Making an adjustment for the advantage in speed of the IBM 370 over the CDC 6600 we

conclude that our time is at least comparable and probably superior ([4] gives an advantage in speed of 5 to 6 for the IBM 370 over the CDC 6600 for network problems). Yet the NETGEN problems are only 3-12% dense while the problems we solved are 100% dense.

	Hungarian Method				Combined New Algorithm and Hungarian Method				Pure Form of the New Algorithm			
N \ R	30	100	1,000	100,000	30	100	1,000	100,000	30	100	1,000	100,000
50	.052 291	.052 289	.062 325	.064 330	.023 133	.023 136	.027 149	.027 147	.022 130	.030 174	.062 388	.056 373
100	.262 814	.319 1017	.406 1293	.459 1396	.091 285	.103 319	.119 382	.130 388	.091 285	.103 317	.311 1023	.300 1107
150	.578 1175	1.09 2442	1.32 2953	1.56 3356	.184 402	.288 593	.338 735	.348 712	.184 402	.288 593	.665 1469	.942 2217
200	.822 1208	2.30 4021	2.81 4986	3.59 5663	.276 467	.526 861	.637 1072	.644 994	.276 467	.526 861	1.33 1917	2.22 3833
300		6.16 7320		10.7 12765		1.39 1544		1.83 2076		1.39 1544		7.53 8582
400		12.5 11337		24.5 22625		2.40 2122		3.64 3183		2.40 2122		12.9 11084

TABLE 2: Top entry in each cell = Time in secs on IBM 370. Bottom entry = Number of sources scanned. Average over five NxN full dense problems with weights chosen by uniform distribution over [0,1] and subsequent multiplication by R and truncation to the nearest integer.

	Hungarian Method			Combined New Algorithm and Hungarian Method			Pure Form of the New Algorithm		
N \ Σ	30	100	10,000	30	100	10,000	30	100	10,000
50	.079 440	.085 449	.089 458	.024 135	.025 136	.026 140	.033 192	.037 233	.038 238
100	.419 1317	.455 1383	.487 1447	.091 285	.091 288	.094 283	.103 326	.118 388	.113 395
150	1.25 2868	1.40 3128	1.53 3265	.260 570	.267 599	.292 601	.342 728	.425 929	.420 1024
200	2.78 4975	3.07 5395	3.43 5700	.492 808	.486 819	.533 852	.603 975	1.00 1607	.800 1536
300	8.25 10101		10.0 11864	1.21 1318		1.15 1268	1.45 1576		2.21 2270
400	20.4 19103		24.8 22755	2.63 2140		2.45 2083	2.85 2350		9.55 8108

TABLE 3: Top entry in each cell = Time in secs on IBM 370. Bottom Entry = Number of sources scanned. Average over five NxN full dense problems with weights chosen by normal distribution N(0,1) and subsequent multiplication by Σ and truncation to the nearest integer.

References

[1] Lawler, E., Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, 1976.

[2] Kuhn, H.W., "The Hungarian Method for the Assignment Problem," Naval Research Logistics Quarterly, Vol. 2, 1955, pp. 83-97.

[3] Barr, R.S., Glover, F., and D. Klingman, "The Alternating Basis Algorithm for Assignment Problems", Math. Programming, Vol. 13, 1977, p.1.

[4] Bradley, G .H., Brown, G .G., and G.W. Graves, "Design and Implementation of Large Scale Primal Transhipment Algorithms", Management Science, Vol. 24, 1977, p.1.

[5] Helgason, R.V., and J.L. Kennington, "NETFLO Program Documentation", Technical Report IEOR 76011, Department of Industrial Engineering and Operations Research, Southern Methodist University, 1976.

[6] Hatch, R.S., "Bench Marks Comparing Transportation Codes Based on Primal Simplex and Primal-Dual Algorithms", Operations Research, Vol. 23, 1975, p.1167.

[7] McGinnis, L.F., "Implementation and Testing of a Primal-Dual Algorithm for the Assignment Problem", Industrial and Systems Engineering Report Series No. J-78-31, Georgia Institute of Technology, November 1978.

[8] Glover F., and D. Klingman, "Comment on a Note by Hatch on Network Algorithms", Operations Research, Vol. 26, 1978, p. 370.

[9] Bertsekas, D.P., "An Algorithm for the Hitchcock Transportation Problem", Proceedings of 18th Allerton Conference on Communication, Control, and Computing, Allerton Park, Ill., Oct. 1979.

PROCEEDINGS OF 17th ALLERTON CONFERENCE ON COMMUNICATION, CONTROL & COMPUTING
Allerton Park, Ill,
October 1979

AN ALGORITHM FOR THE HITCHCOCK TRANSPORTATION PROBLEM

DIMITRI P. BERTSEKAS
Dept. of Electrical Engineering and Computer Science
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

Abstract: An algorithm proposed by the author for the classical assignment problem [1] is generalized for solution of the linear uncapacitated transportation problem.

1. INTRODUCTION

In an earlier paper [1] we proposed a new algorithm for solving the classical assignment problem. The algorithm was shown via computational experimentation to offer substantial computational savings over the Hungarian method. It is thus natural to consider extensions of this algorithm to more general network flow problems. The present paper provides such an extension for the classical linear uncapacitated transportation problem, commonly referred to as the Hitchcock problem. As is well known [2], the general minimum cost flow problem with capacity constraints on the links can be reduced to the Hitchcock problem. Thus the algorithm of this paper can be adapted to solve such problems although we will not discuss the precise form of the necessary modifications.

Due to space limitations our presentation is somewhat abbreviated. It is thus inevitable that some familiarity with the contents of [1] on the part of the reader is necessary for understanding the mechanism of the algorithm.

2. THE HITCHCOCK PROBLEM

Consider a bipartite graph consisting of two finite sets of nodes S and T with elements denoted i (i=1,...,M) and j (j=1,...,N) respectively, and a set of directed links L with elements denoted (i,j) where $i \in S$ and $j \in T$. We refer to elements of S and T as sources and sinks respectively. Each source i (sink j) has a positive scalar α_i (β_j) associated with it referred to as the supply of i (demand of j). We assume

$$\sum_{i \in S} \alpha_i = \sum_{j \in T} \beta_j. \tag{1}$$

Each link (i,j) has a weight a_{ij} associated with it. We assume that there is at most one link $(i,j) \in L$ for all $i \in S$, $j \in T$. We wish to find a flow $x = \{x_{ij} \mid (i,j) \in L\}$ solving the (primal) transportation problem

$$\text{maximize} \quad \sum_{(i,j) \in L} a_{ij} x_{ij}$$

$$\text{subject to} \quad \sum_{(i,j) \in L} x_{ij} = \alpha_i, \quad \forall\, i \in S$$

$$\sum_{(i,j) \in L} x_{ij} = \beta_j, \quad \forall\, j \in T$$

$$x_{ij} \geq 0, \quad \forall\, (i,j) \in L.$$

Throughout the paper we will assume the following:

a) There exists at least one feasible solution for (PTP).
b) The weights a_{ij}, the supplies α_i, and the demands β_j are all integers.

The corresponding dual transportation problem in the vectors $m = (m_1,\ldots,m_M)$ and $p = (p_1,\ldots,p_N)$ is

$$\text{(DTP)} \quad \text{minimize} \sum_{i\in S} \alpha_i m_i + \sum_{j\in T} \beta_j p_j$$

$$\text{subject to } m_i + p_j \geq a_{ij} \quad , \qquad \forall\ (i,j)\in L \ .$$

The scalars p_j and $(a_{ij} - p_j)$ will be referred to as <u>prices</u> and <u>profit margins</u> respectively.

3. The Algorithm

The algorithm generates a sequence of vectors $\{(m^k, p^k, x^k)\}$ such that for each k the following conditions are satisfied:

$$m_i^k + p_j^k \geq a_{ij} \quad , \qquad \forall\ (i,j)\in L, \tag{2}$$

$$m_i^k + p_j^k = a_{ij} \quad , \qquad \forall\ (i,j)\in L \ \text{ with } x_{ij}^k > 0 \ , \tag{3}$$

$$x_{ij}^k \geq 0 \quad , \qquad \forall\ (i,j)\in L \tag{4}$$

$$\sum_{(i,j)\in L} x_{ij}^k \leq \alpha_i \quad , \qquad \forall\ i\in S, \tag{5}$$

$$\sum_{(i,j)\in L} x_{ij}^k \leq \beta_j \quad , \qquad \forall\ j\in T \ . \tag{6}$$

Thus dual feasibility and complementary slackness are maintained throughout the algorithm [cf. (2), (3)], but primal feasibility may be violated to the extent allowed by (4) - (6). The algorithm, under the preceding assumptions, can be shown to terminate at a flow for which (5) and (6) are satisfied with equality. Since in this case primal and dual feasibility as well as complementary slackness are satisfied, this flow must be optimal.

The overall scheme is similar with the one of the primal-dual method ([2], p.95), in that successive flow augmentations and changes in the dual variables are effected, but there are important differences as discussed in [1]. We now state the algorithm.

The method is initialized with any m^o, p^o that are dual feasible, and with $x_{ij}^o = 0$ for all $(i,j)\in L$.

For $k = 0,1,\ldots$, given (m^k,p^k,x^k) satisfying (2) - (6) we stop if (5) and (6) are satisfied with equality. Otherwise we use the following iteration to generate $(m^{k+1}, p^{k+1}, x^{k+1})$. During the iteration additional intermediate variables, vectors and sets of nodes are generated, and are denoted by $\bar{m}$, $\tilde{m}$, $\bar{T}$, $\tilde{T}$, $\bar{x}$, π, f.

<u>(k+1)st Iteration of the Algorithm</u>

Choose a source $\bar{i}$ such that $\sum_{(i,j)\in L} x_{\bar{i}j}^k < \alpha_{\bar{i}}$.

Set $\bar{m}$ to the maximum profit margin for $\bar{i}$

$$\bar{m} \leftarrow \max \{a_{\bar{i}j} - p_j^k | (\bar{i},j) \in L\}^{\dagger} \quad . \tag{7}$$

Set also

$$\bar{T} \leftarrow \{j \in T | \bar{m} = a_{\bar{i}j} - p_j^k\} \quad , \tag{8}$$

$$\tilde{T} \leftarrow \bar{T} \quad . \tag{9}$$

Step 1 (Preliminary adjustment of flows and dual variables): If $\alpha_{\bar{i}} < \sum_{j \in \bar{T}} \beta_j$ go to Step 2. If $\alpha_{\bar{i}} \geq \sum_{j \in T} \beta_j$ go to Step 1a.

Step 1a: Set

$$\tilde{m} \leftarrow \max\{a_{\bar{i}j} - p_j^k | (\bar{i},j) \in L, \quad j \notin \bar{T}\}. \tag{10}$$

$$\tilde{T} \leftarrow \{j \in T | \tilde{m} = a_{\bar{i}j} - p_j^k\} \tag{11}$$

$$p_j^k \leftarrow \begin{cases} p_j^k & \text{if } j \notin \bar{T} \\ p_j^k + (\bar{m} - \tilde{m}) & \text{if } j \in \bar{T} \end{cases} \tag{12}$$

$$m_i^k \leftarrow \begin{cases} m_i^k & \text{if } i \neq \bar{i} \\ \tilde{m} & \text{if } i = \bar{i} \end{cases} \tag{13}$$

$$x_{ij}^k \leftarrow \begin{cases} x_{ij}^k & \text{if } j \notin \bar{T} \\ \beta_j & \text{if } i = \bar{i} \text{ and } j \in \bar{T} \\ 0 & \text{if } i \neq \bar{i} \text{ and } j \in \bar{T} \end{cases} \quad , \tag{14}$$

If $a_{\bar{i}} = \sum_{j \in T} \beta_j$ set

$$m^{k+1} \leftarrow m^k$$

$$p^{k+1} \leftarrow p^k$$

$$x^{k+1} \leftarrow x^k$$

thereby completing the (k+1)st iteration.

If $\alpha_{\bar{i}} \neq \sum_{j \in \bar{T}} \beta_j$ set

$$\bar{T} \leftarrow \bar{T} \cup \tilde{T}$$

$$\bar{m} \leftarrow \tilde{m}$$

and go to Step 1.

---

† The notation A ← B means that the current value of (the variable, vector, or set) A is changed to the current value of B.

Step 2.

(Note: At this point we have from Step 1 vectors m^k, p^k, x^k and nonempty sets of sinks $\bar{T}$, $\tilde{T}$ with $\tilde{T} \subset \bar{T}$ such that (2) - (6) hold and

$$\sum_{j\in(\bar{T}-\tilde{T})} \beta_j = \sum_{j\in(\bar{T}-\tilde{T})} x^k_{\bar{i}j} \leq \sum_{(i,j)\in L} x^k_{ij} < \alpha_{\bar{i}} < \sum_{j\in\bar{T}} \beta_j \quad , \tag{16}$$

$$x_{\bar{i}j} = \beta_j \quad , \qquad \forall\ j\in(\bar{T}-\tilde{T}), \tag{17}$$

$$m^k_{\bar{i}} + p^k_j = a_{\bar{i}j}, \quad \forall\ j\in\bar{T} \quad , \tag{18}$$

where if $\bar{T} = \tilde{T}$ we interpret sums over $j\in(\bar{T}-\tilde{T})$ to be zero).

If there is no unassigned demand for sinks in $\tilde{T}$, i.e. if

$$\sum_{j\in\tilde{T}} \beta_j = \sum_{j\in\tilde{T}} \sum_{(i,j)\in L} x^k_{ij} \quad ,$$

go to Step 3.

Otherwise increase the flow $x^k_{\bar{i}j}$ along links $(\bar{i},j)$ for sources $j\in\tilde{T}$ for which a portion of their demand is unassigned, (i.e., $\beta_j > \sum_{(i,j)\in L} x^k_{ij}$) until either

a) the unassigned portion of the supply of $\bar{i}$ (i.e. $\alpha_{\bar{i}} - \sum_{(\bar{i},j)\in L} x^k_{\bar{i}j}$) is exhausted,

or

b) The unassigned portion of the demand of sinks in $\tilde{T}$ (i.e. $\sum_{j\in\tilde{T}} \beta_j - \sum_{j\in\tilde{T}} \sum_{(i,j)\in L} x^k_{ij}$) is exhausted.

In case a) set $m^{k+1} \leftarrow m^k$, $p^{k+1} \leftarrow p^k$, and set x^{k+1} to be the new flow. This completes the (k+1)st iteration.

In case b) increase further flows $x^k_{\bar{i}j}$ with $j\in\tilde{T}$ and decrease flows x^k_{ij} with $i \neq \bar{i}$, $j\in\tilde{T}$ as necessary until all unassigned supply of $\bar{i}$ is exhausted. (Note: The choice of flows $x^k_{\bar{i}j}$ which are increased and flows x^k_{ij}, $i\neq\bar{i}$, $j\in\tilde{T}$, which are decreased is arbitrary).

Set x^{k+1} to be the new flow, and set $m^{k+1} \leftarrow m^k$, $p^{k+1} \leftarrow p^k$. This completes the (k+1)st iteration.

Step 3: Increase flows $x^k_{\bar{i}j}$ with $j\in\tilde{T}$ and decrease flows x^k_{ij} with $i \neq \bar{i}$, $j\in\tilde{T}$ as necessary until all unassigned supply of $\bar{i}$ is exhausted. We denote the new set of flows $\bar{x}_{ij}$, $(i,j)\in L$ and refer to $(\bar{x}_{\bar{i}j} - x^k_{\bar{i}j})$, $j\in\tilde{T}$ as the incremental flows and to $(x^k_{ij} - \bar{x}_{ij})$, $j\in\tilde{T}$, $i \neq \bar{i}$ as the displaced flows. Give to $\bar{i}$ the label "$(0,f_{\bar{i}})$" where $f_{\bar{i}} = \alpha_{\bar{i}} - \sum_{(\bar{i},j)\in L} x^k_{\bar{i}j}$ set

$$\pi_j \leftarrow \infty \quad , \forall\ j\in T$$

and go to Step 4.

Step 4 (Labeling): Find a source i with an unscanned label and go to Step 4a, or find a sink j with an unscanned label and $\pi_j = 0$, and go to Step 4b. If no such source or sink can be found go to Step 6.

Step 4a: Scan the label of source i as follows. For each $(i,j) \in L$ for which $m_i^k + p_j^k - \alpha_{ij} < \pi_j$ set $\pi_j = m_i^k + p_j^k - \alpha_{ij}$. If in addition $m_i^k + p_j^k = a_{ij}$ give node j the label "(i,f_j)" where $f_j = \min \{f_i, \beta_j - \bar{x}_{ij}\}$. Return to Step 4.

Step 4b: Scan the label of sink j with $\pi_j = 0$ as follows. If $\sum_{(i,j)\in L} \bar{x}_{ij} < \beta_j$ go to Step 5. Otherwise give to every unlabeled source i with $\bar{x}_{ij} > 0$ the label "(j,f_i)" where $f_i = \min \{f_j, \alpha_i\}$. Return to Step 4.

Step 5 (Augmentation) : An augmenting path has been found that alternates between sources and sinks, originates at source $\bar{i}$ and terminates at the sink j identified in Step 4b. The flow on this path is $f = \min \{f_j, \beta_j - \sum_{(i,j)\in L} \bar{x}_{ij}\}$. Modify $\bar{x}$ by adding or subtracting as appropriate f along the augmenting path. Subtract from $\bar{x}$ a portion of incremental flows totalling f and add to $\bar{x}$ the corresponding displaced flows totalling f. The end result is the new set of flows x^{k+1}. Set $m^{k+1} \leftarrow m^k$, $p^{k+1} \leftarrow p^k$. This completes the (k+1)st iteration.

Step 6 (Change of dual variables): Find

$$\delta = \min\{\pi_j \mid j \in T, \ \pi_j > 0\}.$$

Set

$$m_i^{k+1} \leftarrow \begin{cases} m_i^k - \delta & \text{if i has been labelled} \\ m_i^k & \text{if i has not been labelled,} \end{cases}$$

$$p_j^{k+1} \leftarrow \begin{cases} p_j^k + \delta & \text{if } \pi_j = 0 \\ p_j^k & \text{if } \pi_j > 0 , \end{cases}$$

$$x^{k+1} \leftarrow \bar{x} .$$

This completes the (k+1)st iteration of the algorithm.

The description of the algorithm is quite complicated so we provide some explanatory remarks. The objective of each iteration originating with source $\bar{i}$ is to assign all unassigned supply of $\bar{i}$ while in the process to either change some of the dual variables or assign some of the unassigned demand of certain sinks or both. This is done at the expense, perhaps, of displacing some already assigned supply of other sources. Step 1 is the preparatory stage whereby by modifying flows and dual variables if necessary [cf. (12) - (14)] we identify two sets of sinks $\bar{T}$ and $\tilde{T}$ such

that (16) - (18) are satisfied. The iteration may end in Step 1 or in Step 2 after essentially scanning only source $\bar{i}$. This is the simpler of two possible cases and corresponds to Case 1 of the algorithm of [1]. The more complicated case, corresponding to Case 2 of the algorithm of [1], is when the demand of all sinks in $\tilde{T}$ are fully satisfied and therefore in order to assign even a portion of the remaining unassigned supply $\alpha_{\bar{i}} - \sum_{(\bar{i},j)\in L} x^k_{\bar{i}j}$ of source $\bar{i}$ it is necessary to displace some portion of already assigned supply of other sources. This is done by means of the labeling procedure starting with Step 3. A preliminary assignment of the unsatisfied supply of $\bar{i}$ is made by displacing assigned supply of other sources ($\bar{x}$ replacing x^k in Step 3). We then try to find an augmenting path from $\bar{i}$ to a sink with unsatisfied demand. There are two possibilities. Either such an augmenting path can be found (Step 5) in which case a portion f of the unsatisfied supply of $\bar{i}$ is chanelled through the augmenting path and the corresponding portion f of the displaced supply of other sources is reinstated; or else a change is effected in the dual variables (Step 6). The amount of change is the maximum allowed by the complementary slackness constraint.

Observe that the following hold true similarly as in the algorithm of [1]:

a) The sequences $\{m^k_i\}$ are monotonically nonincreasing and the sequences $\{p^k_j\}$ are monotonically nondecreasing.

b) The sequences of unassigned sink demands $\{\beta_j - \sum_{(i,j)\in L} x^k_{ij}\}$ are monotonically nonincreasing.

c) Dual feasibility and complementary slackness are maintained by the algorithm.

d) At every iteration either the unassigned demand of at least one sink will decrease strictly by an integer amount, or at least one price will increase strictly by an interger amount.

e) In order for the price of a sink to increase it is necessary that its demand is fully assigned.

Based on these observations we can show in a very similar manner as in [1] that the algorithm will terminate at a feasible flow. In view of the fact that complementary slackness and dual feasibility are maintained throughout the algorithm, the flow obtained at termination is optimal.

We have no computational experience with the algorithm with general transportation problems. We except, however, that similarly as for assignment problems [1], some combination with the primal-dual method will turn out to be beneficial. It is hoped that computational experience will be gained in the near future in this regard.

References

[1] D.P. Bertsekas, "A New Algorithm for the Assignment Problem", Lab. for Information and Decision Systems Working Paper, Massachusetts Institute of Technology, July 1979.

[2] L.R. Ford and D.R. Fulkerson, Flows in Networks, Princeton University Press, 1962.

Acknowledgement: This work was supported by Grant NSF ENG-7906332.