Mathematical Programming 21 (1981) 152–171
North-Holland Publishing Company

A NEW ALGORITHM FOR THE ASSIGNMENT PROBLEM*

Dimitri P. BERTSEKAS

Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

Received 15 August 1979
Revised manuscript received 6 October 1980

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 average computational complexity of an efficient implementation of the algorithm seems to be considerably better than the one of the Hungarian method. 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 the problem dimension N and reaches an order of magnitude for N equal to several hundreds.

Key words: Assignment Problems, Network Flows, Hungarian Method, Computational Complexity.

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 important in itself, but is also suitable for development of 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.

It seems that the currently most popular solution methods for the assignment problem are specialized forms of the simplex method [3–5] and versions of Kuhn's Hungarian method [6–8]. There has been some controversy regarding the relative merits of simplex codes and primal–dual (i.e. Hungarian) codes [6, 9]. A recent computational study [7] finds simplex and primal–dual codes roughly comparable, while another study [8] reports that a primal–dual code based on the

* Work supported by Grant NSF ENG-7906332.

work of Edmonds and Karp [10] outperforms simplex-like algorithms by a considerable margin. The development of specialized codes for the assignment problem using sophisticated programming techniques is an active research area so it is difficult to assess the current state of the art.

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, $N \times N$ assignment problems is $O(N^3)$ [1, p. 205]. There is no simplex type method with complexity as good as $O(N^3)$ and there are no average complexity estimates for either Hungarian methods or simplex methods to the extent of our knowledge.

The purpose of this paper is to propose a new method for solving the assignment problem. We show in Section 3 that the worst case complexity of the pure form of the method for full dense, all integer, $N \times N$ problems with the elements of the assignment matrix taking values in the interval $[0, R]$ is $O(N^3) + O(RN^2)$. It appears, however, that for large values of R (say $R > 100$) the method performs at its best when it is combined with the Hungarian method. One such combination is described in Section 4 and the worst case complexity of the resulting method is $O(N^3)$. Its average complexity, however, seems to be substantially better than 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 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. 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.

We finally note that we 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. One such algorithm for the Hitchcock transportation problem is reported in [11].

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 $(i, j) \notin L$. 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 nonnegative 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.

Assumption (b) can be replaced by the more general assumption that a_{ij} are all integers (not necessarily nonnegative) and for some integer $R > 0$ we have

$$\max_{j\in T} a_{ij} - \min_{j\in T} a_{ij} \leq R, \quad \forall i \in S.$$

To see this note that if for any source $i \in S$ we subtract the constant $\min_{j\in T} a_{ij}$ from the weights a_{ij}, $\forall j \in T$ then the value of all possible assignments of cardinality N changes by the same constant. Assumption (c) involves no loss of generality, since if for some source i there is only one link $(i, j) \in L$, 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

$$\begin{aligned} &\text{maximize} && \sum_{(i,j)\in L} a_{ij}x_{ij}, \\ &\text{subject to} && \sum_{(i,j)\in L} x_{ij} = 1, \quad \forall i = 1, \dots, N, \\ & && \sum_{(i,j)\in L} x_{ij} = 1, \quad \forall j = 1, \dots, N, \\ & && x_{ij} \geq 0, \quad \forall i, j = 1, \dots, N. \end{aligned} \tag{1}$$

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

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{N} m_i + \sum_{j=1}^{N} p_j, \\ &\text{subject to} && m_i + p_j \geq a_{ij}, \quad \forall (i, j) \in L. \end{aligned} \tag{2}$$

The scalars p_j and $(a_{ij} - p_j)$ will be referred to as *prices* and *profit margins* 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

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

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 3.

The method is initialized with any integer vectors m^0, p^0 that are dual feasible and with X^0 being the empty assignment.

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

$$m_i^k + p_j^k = a_{ij} \quad \text{if } (i,j) \in X^k,$$
$$m_i^k + p_n^k \geq a_{in} \quad \forall (i,n) \in L,$$

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 $i \in S$

$$m_i^{k+1} + p_j^{k+1} = a_{ij} \quad \text{if } (i,j) \in X^{k+1},$$
$$m_i^{k+1} + p_n^{k+1} \geq a_{in} \quad \forall (i,n) \in L.$$

$(k+1)^{\text{st}}$ 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

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

Compute also the 'second maximum' profit margin

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

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

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

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

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

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

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.,

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

This completes the $(k+1)^{\text{st}}$ iteration of the algorithm.

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

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

$$m_{\bar{i}}^k \leftarrow \bar{m},$$
$$\pi_j = \infty, \quad j = 1, \ldots, N, \tag{9}$$

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 $m_j^k + p_j^k - a_{ij} < \pi_j$ give node j the label 'i' (replacing any existing label) and set $\pi_j \leftarrow m_i^k + p_j^k - a_{ij}$. 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 $m^{k+1} = m^k$, $p^{k+1} = p^k$. This completes the $(k+1)^{st}$ iteration of the algorithm.

Step 3 (*Change of dual variables*): Find

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

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 cf. (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.,

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

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 i which was assigned under X^k becomes unassigned under X^{k+1}.

A useful conceptual aid in understanding the algorithm is to think of it in terms of an *auction*. Let us imagine that sinks are items for sale in an auction

and each source i is a person for whom each item j is worth a_{ij}. After the k^{th} iteration of the algorithm some of the items have been temporarily assigned to persons that have bid up their prices to levels p_j^k. If Case 1 holds at the $(k+1)^{\text{st}}$ iteration, the (unassigned) person $\bar{i}$ selects the item $\bar{j}$ that offers maximum profit margin and bids up its price by the amount $(\bar{m} - \hat{m})$—the maximum amount for which $\bar{j}$ will still offer maximum profit margin. The item $\bar{j}$ is then assigned to the individual $\bar{i}$ in place of any other person $\hat{i}$ that may have bid earlier for $\bar{j}$. In these terms *the algorithm may be viewed as a process whereby persons compete for items by trying to outbid each other.* One can interpret Case 2, Step 3 similarly except that this interpretation involves the (admittedly less intuitive) idea of a *cooperative bid* whereby several persons simultaneously increase the prices of corresponding items. During actual computation Case 1 occurs far more frequently than Case 2, so, should someone wish to give a name to the algorithm, we suggest calling it the *auction algorithm.*

Unfortunately the description of the algorithm in Case 2 is quite complicated. For this reason some explanatory remarks are in order. 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 Fig. 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 Fig. 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

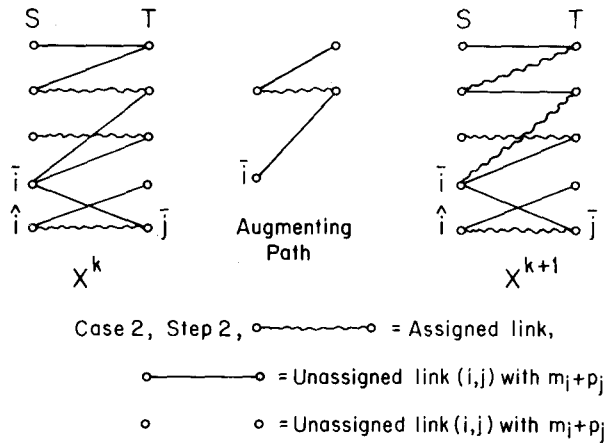

Fig. 1.

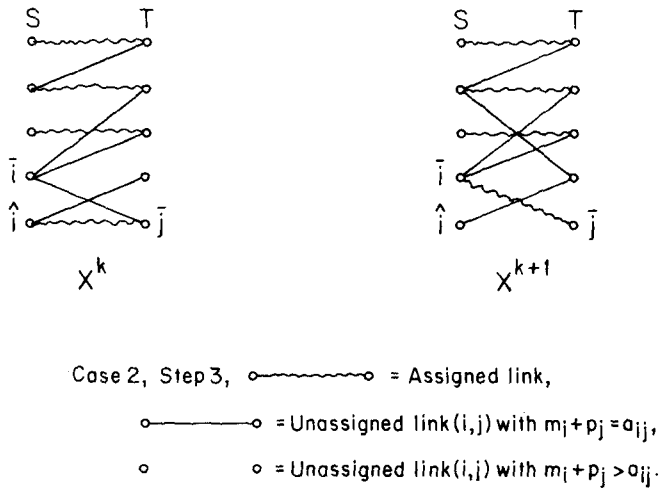

Fig. 2.

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.

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

	p_j	0	0	0	0
j	m_i				
	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 Fig 3.

1st *iteration*: we choose the unassigned source 2. 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 obtained

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

After 1st iteration

		p_j	0	0	3	0
j	m_i					
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

		p_j	0	1	3	0
j	m_i					
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

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

After 4th iteration

		p_j	0	2	4	0
j	m_i					
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

		p_j	0	2	4	0
j	m_i					
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

Fig. 3.

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 $\tilde{m} = 5$, $\tilde{m} = 4$, $\bar{j} = 2$. 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 $\tilde{m} = \tilde{m} = 2$. 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 Fig. 3.

5th *iteration*: We choose the unassigned source 2. Here $\tilde{m} = \tilde{m} = 3$. 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 $\tilde{m} = \tilde{m} = 3$. 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^k, then it remains assigned under X^k for all $k > \bar{k}$.

(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

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

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

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

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

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

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 $(i, j) \in L$ such that $i \in S_\infty$ and $j \notin T_\infty$. Thus we have

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

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

3. Computational complexity analysis

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

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

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

$$B_1 \le p_j^0 \le B_2, \quad \forall j \in T. \tag{13}$$

For each k there must be at least one unassigned sink, say $\bar{n}_k$ and we must have $p_{\bar{n}_k}^k = p_{\bar{n}_k}^0 \le B_2$. It follows from (12) and observation (a) that

$$B_1 \le p_j^0 \le p_j^k \le R + B_2, \quad \forall k = 0, 1, \ldots, \text{and } j \in T. \tag{14}$$

It is easy to see that there is an integer γ such that the k^{th} iteration of the algorithm requires at most $\gamma z_k N$ computer operations where

$$z_k = \begin{cases} 1, & \text{in Case 1,} \\ \text{number of labeled sources,} & \text{in Case 2,} \end{cases}$$

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, z_k prices will be increased by at least unity. It follows from $z_k \leq N$ 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

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

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 sovling 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 $N \times N$ full dense problem specified by the assignment matrix

$$\begin{bmatrix}
N & & & & & & & \\
N & N-1 & & & & 0 & & \\
N & N-1 & N-2 & & & & & \\
\cdot & \cdot & \cdot & \cdot\ \cdot\ \cdot & & 3 & & \\
\cdot & \cdot & \cdot & \cdot\ \cdot\ \cdot & & 2 & 2 & \\
\cdot & \cdot & \cdot & \cdot\ \cdot\ \cdot & & 3 & 2 & 1 \\
N & N-1 & N-2 & & & & &
\end{bmatrix}$$

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 k^{th} 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 $[N, N, \cdots N]$ 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 labeled 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 last two or three sinks. Predictably in view of the complexity estimate (15) this typically occured for large values of R (over 100). 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 some form of the Hungarian method so that if the algorithm does not make sufficiently fast progress a switch is made to the Hungarian method. One of the

many possible such combined methods is 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 the Hungarian method and (for $R > 100$) to the pure form of the algorithm as well. Significantly, the factor of improvement over the Hungarian method increases with problem dimension, thereby suggesting a better average computational complexity.

4. Combination with the Hungarian method—computational results

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

The initial assignment X^0 is taken to be the empty assignment and the initial dual variables are chosen according to the usual scheme

$$m_i^0 = \max\{a_{ij} \mid j \in T\}, \quad i = 1, \ldots, N,$$

$$p_j^0 = \max\{a_{ij} - m_i^0 \mid i \in S\}, \quad j = 1, \ldots, N.$$

Notice that the equations above imply

$$m_i^0 + p_j^0 \geq a_{ij}, \quad \forall (i, j) \in L.$$

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

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

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

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 labeling procedure.

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, go to Step 3.

Step 1a: Scan the label of source i as follows. For each $(i, j) \in L$ for which

[1] Actually throughout the algorithm the stronger condition $m_i^k = \max\{a_{in} - p_n^k \mid (i, n) \in L\}$ 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.

$m_i^k + p_j^k - a_{ij} < \pi_j$ give node j the label 'i' (replacing any existing label) and set $\pi_j \leftarrow m_i^k + p_j^k - a_{ij}$. 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

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

Set

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

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

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

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. It can be shown that the worst case computational complexity of this algorithm is $O(N^3)$.

As already discussed, for $R > 100$ 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. There are several other possibilities along similar lines.

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, $\bar{j}$ 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 labeled sources being more than a fixed prespecified number (10 in our experiments)[2]. 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 $X^0 =$ empty, and $p_j^0 = 0$, $m_i^0 = R$ for all i and j. However at the end of the first cycle (i.e. at the end of the N^{th} iteration) the prices of all unassigned sinks j are changed from $p_j^N = 0$ to

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

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 given earlier. 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 1 and 2 show the results of our computational experiments with randomly generated full dense, $N \times N$ 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 1), or from a normal distribution $N(0, 1)$ and subsequent multiplication by Σ (Table 2). 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 370/168 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

[2] 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.

Table 1
Top entry in each cell = time in secs on IBM 370. Bottom entry = number of sources scanned. Average over five $N \times N$ 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 \ R	30	100	1 000	100 000	30	100	1 000	100 000	30	100	1 000	100 000
50	0.052 291	0.052 289	0.062 325	0.064 330	0.023 133	0.023 136	0.027 149	0.027 147	0.022 130	0.030 174	0.062 388	0.056 373
100	0.262 814	0.319 1017	0.406 1293	0.459 1396	0.091 285	0.103 319	0.119 382	0.130 388	0.091 285	0.103 317	0.311 1023	0.300 1107
150	0.578 1175	1.09 2442	1.32 2953	1.56 3356	0.184 402	0.288 593	0.338 735	0.348 712	0.184 402	0.288 593	0.665 1469	0.942 2217
200	0.822 1208	2.30 4021	2.81 4986	3.59 5663	0.276 467	0.526 861	0.637 1072	0.644 994	0.276 467	0.526 861	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 $N \times N$ full dense problems with weights chosen by normal distribution $N(0, 1)$ and subsequent multiplication by Σ 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	0.079 440	0.085 449	0.089 458	0.024 135	0.025 136	0.026 140	0.033 192	0.037 233	0.038 238
100	0.419 1317	0.455 1383	0.487 1447	0.091 285	0.091 288	0.094 283	0.103 326	0.118 388	0.113 395
150	1.25 2868	1.40 3128	1.53 3265	0.260 570	0.267 599	0.292 601	0.342 728	0.425 929	0.420 1024
200	2.78 4975	3.07 5395	3.43 5700	0.492 808	0.486 819	0.533 852	0.603 975	1.00 1607	0.800 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

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 for $R > 100$ 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. For $R \leq 100$ the performance of the pure and the combined forms of the algorithm are nearly identical indicating that in most cases a switch to the Hungarian method was never made and when it was it did not result to any substantial improvement.

While we have not developed a sparse problem code, we did test the algorithms on sparse problems with the full dense code by employing the device of setting the weights of a fixed percentage of links ranging from 50% to 90% to (essentially) $-\infty$ and choosing the weights of the remaining links via a uniform distribution as in Table 1. The results were qualitatively similar to those of Table 1. The number of rows scanned was decreased by roughly 10% on the average for the pure and combined forms of the new algorithm and by roughly 5% for

the Hungarian method. Computation times were not recorded as these are meaningless in the absence of special data structure techniques exploiting sparsity. However, when such techniques are implemented the comparison of computation times should favor our algorithm even more since Step 3 (Case 2) of our algorithm (which is relatively time consuming for sparse problems) is executed far less frequently than the corresponding Step 3 of the Hungarian method.

As a final comparison with existing methodology it is worth observing that the computation time of Table 1 for the combined method and 200×200 problems with weights in the range $[0, 100]$ is 0.526 seconds. There are five 200×200 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 range from 0.96 to 1.68 secs on a CDC 6600 [3, 7] and 0.38 to 0.90 secs on an IBM 370/168 [12]. Making an adjustment for the advantage in speed of the IBM 370 over the CDC 6600 we conclude that our time is comparable ([14] gives an advantage in speed of 5 to 6 for the IBM 370 over the CDC 6600 for network problems although there has been some question on the accuracy of this figure). Yet the NETGEN problems are only 3 – 12% dense while our time corresponds to 100% dense problems. Since existing codes are constantly improved, these figures do not constitute definite proof that our algorithm is superior to other algorithms based on simplex or primal–dual methods. They do, however, suggest that our algorithm can provide, with the aid of sophisticated programming techniques, the basis for improved codes for assignment.

Acknowledgment

The assistance of Eli Gafni with the computational experiments as well as helpful discussions are gratefully acknowledged.

References

[1] E. Lawler, *Combinatorial optimization: networks and matroids* (Holt, Rinehart and Winston, 1976).

[2] H.W. Kuhn, "The Hungarian method for the assignment problem", *Naval Research Logistics Quarterly* 2 (1955) 83–97.

[3] R.S. Barr, F. Glover and D. Klingman, "The alternating basis algorithm for assignment problems", *Mathematical Programming* 13 (1977) 1.

[4] G.H. Bradley, G.G. Brown and G.W. Graves, 'Design and implementation of large scale primal transhipment algorithms", *Management Science* 24 (1977) 1.

[5] R.V. Helgason and J.L. Kennigton, "NETFLOW program documentation", technical report IEOR 76011, Department of Industrial Engineering and Operations Research, Southern Methodist University (1976).

[6] R.S. Hatch, "Bench marks comparing transportation codes based on primal simplex and primal-dual algorithms", *Operations Research* 23 (1975) 1167.

[7] L.F. McGinnis, "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] A. Weintraub, and F. Barahona, "A dual algorithm for the assignment problem", Departmento de Industrias Report No. 2, Universidad de Chile-Sede Occidente (April 1979).
[9] F. Glover and D. Klingman, "Comment on a note by Hatch on network algorithms" *Operations Research* 26 (1978) 370.
[10] J. Edmonds and R. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems", *Journal of the Association for Computing Machinery* 19 (1972) 248–264.
[11] D.P. Bertsekas, "An algorithm for the Hitchcock transportation problem", *Proceedings of the* 18^{th} *Allerton conference on communication, control and computing*, Allerton Park, 11 Oct. 1979.
[12] M.D. Grigoriadis, Talk at Mathematical Programming Symposium, Montreal, August 1979 (also private communication).