An Algorithm for the Detection and Construction of Monge Sequences

Noga Alon*
School of Mathematical Sciences
Sackler Faculty of Exact Sciences
Tel Aviv University
Tel Aviv, Israel

Steven Cosares
Bellcor
3 Corporate Place
Piscataway, N.J. 08854

Dorit S. Hochbaum†
School of Business Administration and IEOR Department
University of California, Berkeley
Berkeley, California 94720

and

Ron Shamir‡
School of Mathematical Sciences
Sackler Faculty of Exact Sciences
Tel Aviv University
Tel Aviv, Israel

Dedicated to Alan J. Hoffman on the occasion of his 65th birthday.

Submitted by Uriel G. Rothblum

ABSTRACT

We give an efficient algorithm which determines whether a condition due to Hoffman (1963) is satisfied by the cost matrix of a transportation problem. In case the condition is satisfied, our algorithm generates a permutation of the matrix entries

*Supported in part by Allon Fellowship and by a Bat Sheva de Rothschild grant.

†Supported in part by the National Science Foundation under grant ECS-85-01988 and the Office of Naval Research under grant N000 14-88-K-0377.

‡Supported in part by Allon Fellowship.

(called a Monge sequence), which allows for the solution of any problem with that cost matrix in linear time, by way of a "greedy" algorithm. This is the first polynomial algorithm for this problem. The running time of our algorithm is better than that of the best known algorithms for the transportation problem, and thus it can be used as a preliminary step in solving such problems without an increase in the overall complexity.

1. INTRODUCTION

Hoffman [10] proved a necessary and sufficient condition for a transportation problem to be solvable by a greedy algorithm. The greedy algorithm is given a permutation (called a Monge sequence) of the decision variables for the problem and maximizes each variable in turn. It is much faster than standard algorithms for the transportation problem. The applicability of Hoffman's condition has since been established for various families of problems [2, 9, 10, 15] in order to exploit the faster algorithm. Hoffman's condition, though, is not constructive, since it requires prior knowledge (or guessing) of the Monge sequence. Until now, no polynomial algorithm has been known for checking whether a transportation problem satisfies Hoffman's condition, and for constructing a Monge sequence for it in case it does [12]. Previously, these questions had to be answered separately for every family of transportation matrices.

In this note we give an algorithm which tests whether a Monge sequence exists for a given transportation cost matrix, and constructs such a sequence if one exists. A straightforward implementation of the algorithm requires $O(m^2n^2)$ time and $O(mn)$ space for an $m \times n$ transportation matrix. A more sophisticated implementation reduces the running time to $O(m^2 n \log n)$ at the cost of increasing the space requirement to $O(m^2n)$, where m can be chosen as the smaller dimension of the matrix. The running time of our algorithm is better than that of the best known strongly polynomial algorithm for the transportation problem by a factor of at least $n/m \geqslant 1$, and thus it can be used as a preliminary step towards solving such problems without an increase in the overall complexity. We also make some observations as to when such a sequence exists, and discuss the applicability of the algorithm to practical situations.

2. PRELIMINARIES AND BACKGROUND

Given that some commodity must be shipped from m different "source" locations to satisfy the demand at n "destination" locations, the transportation problem is defined as the task of finding the appropriate amounts to send from each source i directly to each destination j, in a way that minimizes the total shipping cost. Formulated as a linear program, the transportation problem is to find values for x_{ij}, $i = 1, \dots, m$, $j = 1, \dots, n$, where

$$\sum_{j=1}^{n} x_{ij} = a_i, \qquad \sum_{i=1}^{m} x_{ij} = b_j,$$

$$x_{ij} \geq 0, \qquad i = 1, \dots, m, \quad j = 1, \dots, n,$$

such that $\sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} x_{ij}$ is minimized. Here, a_i is the amount of the commodity available at source i, and b_j is the amount required at destination j, where $\sum_{j=1}^{m} a_i = \sum_{i=1}^{n} b_j$ is assumed. C is a matrix with m rows and n columns, whose (i, j)th element, c_{ij}, is the cost per unit of shipping from source i to destination j.

This problem was first described by Hitchcock [8] and Kantorovich [13] and has been the subject of intensive study (see, e.g. [4, 14, 19]). Since it can be represented as a minimum cost flow problem, there exist many efficient solution techniques for the problem. Most of the algorithms solving it are based either on simplex method operations or on flow augmentations in the corresponding digraph. The currently fastest strongly polynomial algorithm for the transportation problem is due to Orlin [18] and requires $O(n \log n)$ solutions of shortest path problems on graphs with $n + m$ vertices and nm edges. The fastest (weakly) polynomial algorithm for the problem is due to Ahuja, Orlin, Stein, and Tarjan [1] and has running time $O(n^2 m \log nC)$, where C is the largest cost in the matrix, [17].

It is well known that the special structure of the problem makes it possible to obtain an initial (basic) feasible solution by taking the indices of the variables in any order, say $((i_1, j_1), (i_2, j_2), \dots, (i_{mn}, j_{mn}))$, and performing the following linear time algorithm:

```
Let $\hat{a}_i = a_i$, $i = 1, \dots, m$, $\hat{b}_j = b_j$, $j = 1, \dots, n$.
For $k = 1, \dots, mn$ do:
    Set $x_{i_k j_k} \leftarrow \min(\hat{a}_{i_k}, \hat{b}_{j_k})$
    $\hat{a}_{i_k} \leftarrow \hat{a}_{i_k} - x_{i_k j_k}$
    $\hat{b}_{j_k} \leftarrow \hat{b}_{j_k} - x_{i_k j_k}$
end
```

(i.e., each variable, in its turn, takes on the largest feasible value possible). Examples of this "greedy" technique include the "northwest corner" method and the "minimum c_{ij}" rule (see, e.g., [7]). While a sophisticated variable ordering rule may be useful in determining a "good" solution, it cannot, in general, guarantee an optimal solution.

In [10], the author provides necessary and sufficient conditions for an ordering of the variables which guarantees that the greedy technique provides an optimal solution to the transportation problem, for any values of $(a_1, \dots, a_m)$ and $(b_1, \dots, b_n)$. Namely, the permutation $S = ((i_1, j_1), (i_2, j_2), \dots, (i_{mn}, j_{mn}))$ of the indices of the cost matrix must satisfy

For every $1 \leqslant i, r \leqslant m$, $1 \leqslant j, s \leqslant n$, whenever (i, j) precedes both (i, s) and (r, j), the corresponding entries in matrix C are such that $c_{ij} + c_{rs} \leqslant c_{is} + c_{rj}$. (C1)

An ordering satisfying condition (C1) is called a *Monge sequence* for the matrix C, named after the mathematician who made similar observations about greedy solution techniques as long ago as 1781 [16]. (We have deviated slightly from Hoffman's terminology for the sake of brevity.)

Given a transportation problem with cost matrix C, we would like to find a Monge sequence for it, if one exists. By Hoffman's theorem, obtaining such a sequence reduces the time complexity of any subsequent solution of the problem to linear. Therefore, the sequence would be useful in many practical applications of the transportation problem. It is particularly attractive in cases where the cost matrix is fixed but the supply and demand vectors vary over time (e.g. planning a daily transportation program where the demands and supplies change daily, but the per-unit shipping costs remain fixed for a long period). In these situations, identification of the sequence greatly accelerates and simplifies repeated solutions.

In this paper we study the problem of constructing a Monge sequence for a transportation matrix or showing that none exists. We describe two polynomial algorithms for that problem. While this general problem has not been solved before [12], certain special cases involving assignment problems have been studied, in connection with polynomially solvable special cases of the traveling salesperson problem. In that context, Gilmore and Gomory [5] addressed $n \times n$ assignment matrices D (cf. [4]), satisfying (among other conditions)

$$d_{ij} + d_{rs} \leqslant d_{is} + d_{rj} \qquad \text{for every} \quad i \leqslant r \text{ and } j \leqslant s.$$

Chandrasekaran [3] observed that such matrices can be recognized in polynomial time even if the rows and the columns of the matrix have been permuted, and the two n-element permutations can be polynomially reconstructed. Gilmore et al. [6] made a similar observation earlier, for the case where only the columns have been permuted. (In their case there are additional conditions of monotonicity along each row, which further simplify the problem.) The solution techniques developed for constructing the n-element permutations in the assignment problem are, however, tailored for that special case. They are not suitable for our more general problem, in which one wishes to construct an nm-long permutation for a transportation problem.

3. OBSERVATIONS

A natural question that arises is whether every matrix C gives rise to a Monge sequence. Clearly, such a sequence exists for every matrix of dimension $1 \times n$. It turns out that every $2 \times n$ matrix also has such a Monge sequence, but not every larger matrix does:

OBSERVATION 3.1. *There is a Monge sequence for every* $2 \times n$ *cost matrix.*

Proof. Renumber the columns of the matrix so that $c_{11} - c_{21} \leqslant c_{12} - c_{22} \leqslant \cdots \leqslant c_{1n} - c_{2n}$. The ordering $((1,1),(1,2),\dots,(1,n),(2,1),\dots,(2,n))$ is a Monge sequence for C, because $c_{1i} + c_{2j} \leqslant c_{1j} + c_{2i}$ for all $i < j$, implying that condition (C1) holds for every item. ∎

NOTE. A somewhat surprising fact is that in the $2 \times n$ case, the reversed sequence $((2,n),(2,n-1),\dots(1,1))$ is also a Monge sequence. This, of course, does not hold in general.

OBSERVATION 3.2. *For every* m, n *such that* $\min(m,n) \geqslant 3$, *there exist* $m \times n$ *cost matrices which do not have a corresponding Monge sequence.*

Proof. Consider the 3×3 identity matrix. A simple check shows that none of the nine elements in the matrix can appear before the remaining eight in a Monge sequence, without violating condition (C1). Consequently, for $\min(m,n) \geqslant 3$, no $m \times n$ matrix containing a 3×3 identity matrix as a submatrix has a Monge sequence. ∎

4. A SIMPLE ALGORITHM

In this section we describe a new simple polynomial algorithm for the detection and construction of a Monge sequence in a transportation matrix. First, we need some notation: When $i \neq r$ and $j \neq s$ in condition (C1), the four entries (i,j), (i,s), (r,j), (r,s) determine a 2×2 matrix which we call a *quadruple*. This matrix has two diagonals, each containing two entries, namely $((i, j), (r, s))$ and $((i, s), (r, j))$. If $c_{ij} + c_{rs} \leqslant c_{is} + c_{rj}$, then $((i, j), (r, s))$ is called a *small diagonal* of that quadruple. Both diagonals will be called small if equality holds.

THE ALGORITHM.

Initialization:

1. Build a graph whose nodes correspond to the matrix entries. Two nodes are connected by an edge if they are on the strictly larger diagonal in the quadruple which they share. (Nodes which correspond to entries in the same row or column are not connected by an edge.)

The iterative step:

2. If there is a node (i, j) of degree zero (an isolated node) in the graph, place it next in the sequence, and eliminate from the graph all the edges which connect an element from row i with an element from column j. [In every quadruple which includes (i, j), the Monge condition is already satisfied; hence these edges are unnecessary.]
3. If there is no isolated node, stop. No Monge sequence exists.

Successful termination occurs when all the nodes have been sequenced in the iterative step.

PROPOSITION 4.1. *The algorithm generates a Monge sequence if and only if one exists.*

Proof. The algorithm generates a sequence $(R_1, \ldots, R_l)$ where every R_i is a distinct pair of matrix indices. The sequence preserves the following invariant conditions:

For every $j = 1, \ldots, l$, in every quadruple which includes R_j but does not include $R_1, \ldots, R_{j-1}$, R_j is on the small diagonal. (C2)

But this is equivalent to the Monge condition (C1). Hence if a full sequence $(R_1, \ldots, R_{mn})$ has been generated, then it is a Monge sequence.

We now want to prove that the failure of the algorithm implies the nonexistence of a Monge sequence. Assume to the contrary that a Monge sequence $(S_1, \ldots, S_{mn})$ exists, but the algorithm stopped after the construction of the subsequence $R = (R_1, \ldots, R_k)$. Let t be the smallest index such that

$S_t \notin \{R_1, \ldots, R_k\}$. Since both S and R satisfy (C2), S_t can be appended to R while maintaining (C2). This is a contradiction to the maximality of R. ■

NOTE. We have actually proved a somewhat stronger result. Define a *Monge subsequence* as any sequence $(R_1, \ldots, R_l)$ (possibly with $l < mn$) satisfying (C2). We have proved that the algorithm generates a maximum length Monge subsequence. We shall return to this point later.

PROPOSITION 4.2. *The algorithm can be implemented in quadratic time and linear space.*

Proof. The initialization phase requires scanning all quadruples, i.e. $O((mn)^2)$ steps. The elimination of edges after an iteration which placed (i, j) in the sequence requires scanning all quadruples containing an element from row i and column j, and hence can be done in $O(mn)$ steps per iteration. The identification of an isolated node can also be done in $O(mn)$ steps per iteration: Maintain a degree variable for each node which is originally zero, and is updated whenever the degree increases (in the initialization phase) or decreases (in the iterative phase). The additional work is constant for each edge update. Finding a zero degree node can thus be done by scanning the degree variables of all nodes which requires linear time. (This part can indeed be done in constant time per node, by maintaining a queue of the nodes of degree zero, but the improvement will not change the overall complexity.) Since the total work per iteration is $O(mn)$ steps and the number of iterations is $O(mn)$, we get the quadratic time bound.

In a straightforward implementation of the algorithm one would keep the whole graph (vertices and edges), which requires quadratic space. Instead, we can keep only the degree variables for each node without keeping the edges explicitly, thereby reducing the space requirement to linear. The updating of the degrees can be done after (i, j) has been sequenced by recomputing for each pair $((i, k_1),(k_2, j))$ whether an edge should have connected them before (i, j) was placed and changing their degrees accordingly. The recomputing requires constant time for each quadruple containing (i, j) and hence linear time per iteration, and does not change the overall time complexity. ■

Even if the algorithm terminates with a Monge subsequence only, this information may be quite useful in practice: As easily follows from Hoffman's arguments [10], using the greedy algorithm with such subsequence produces values for the corresponding variables which occur in some optimal solution. Hence, their values can be fixed, and the resulting lower-dimensional problem can be solved by standard techniques. That smaller problem cannot be solved greedily, but its size may have been drastically reduced from the original problem size (since every element in the Monge sequence may eliminate up to a full row or column from the problem).

5. A FASTER ALGORITHM

Henceforth, we assume that the cost matrix C is such that $m \leqslant n$.

For every pair of row indices, i and r, we construct a *precedence vector*, $P_{ir} = \{p_1, \dots, p_n\}$, which represents a permutation of the column indices satisfying

$$c_{ip_1} - c_{rp_1} \leqslant c_{ip_2} - c_{rp_2} \leqslant \cdots \leqslant c_{ip_n} - c_{rp_n}.$$

Let $\hat{c}(i, r) = c_{ip_1} - c_{rp_1}$.

Notice that $P_{ir} = (p_1, \dots, p_n)$ implies that $P_{ri} = (p_n, \dots, p_1)$, so these vectors can be created by sorting $\binom{m}{2}$ lists, each having size n, which can be accomplished in $O(m^2 n \log n)$ time.

From the proof in Observation 3.1, we see that P_{ir} gives an ordering of the columns which provides a Monge sequence for C, when it is restricted to rows i and r. In particular, it gives the order of elements in row i. The following observation follows:

LEMMA 5.1. *The element (i, j) can be placed first in a Monge sequence for C if and only if column index j can be first in the precedence vectors P_{ir} for every $r \neq i$.*

[Note: It is possible that, for some r, $\hat{c}(i, r) = c_{is} - c_{rs}$ for more than one value of s, so there can be more than one possible first element in P_{ir}.]

The following data structures are to be used to make efficient the search for an appropriate item to enter the sequence. For each row index i, create a $(0, 1)$ matrix B_i, having m rows and n columns. These matrices will have the property that $B_i(r, s)$ is 1 if and only if, according to the order constraints implied by rows i and r, (i, s) can be next in a Monge sequence. The matrices B_i are initialized as follows:

$$\begin{array}{ll} B_i(i, j) = 1 & \text{for } \; j = 1, \dots, n, \\ B_i(r, s) = 1 & \text{for } r \neq i, \text{ and for all } s \text{ such that } c_{is} - c_{rs} = \hat{c}(i, r), \\ B_i(r, s) = 0 & \text{otherwise.} \end{array}$$

In particular, for each B_i, row r initially contains a 1 in the column(s) corresponding to each possible first element in P_{ir}, $r \neq i$. (Row i is set for convenience.)

In addition, create a matrix, DEGREE, with m rows and n columns, such that DEGREE(i, j) is the number of 1's in column j of the matrix B_i. DEGREE(i,s) will therefore be m if and only if, according to the order constraints implied by all rows, (i, s) can be placed next. That is,

CLAIM 5.2. *An element, say (i, j), can be placed first in a Monge sequence for C if and only if* DEGREE$(i, j) = m$. *So if* DEGREE $< m$ *for every element, no Monge sequence exists.*

The initialization of each row r in B_i, $r \neq i$, requires finding column indices j such that $c_{ij} - c_{rj} = \hat{c}(i, r)$, which can be accomplished in $O(n)$ time, so the total complexity of building $B_1, \ldots, B_m$ is $O(m^2 n)$. Notice, however, that work can be saved if the precedence vectors are referred to. The precedence vectors will also be needed for the subsequent iterations of the algorithm. Assigning DEGREE and finding a pair (i, j) with value m can be done as a by-product of building and later updating the matrices B_i, $i = 1, \ldots, m$, by maintaining a queue of the entries (i, j) with DEGREE$(i, j) = m$.

In the iterative phase of the algorithm, after having placed the pair (i, j) in the sequence, we must insure that the set of conditions (C1) which are associated with items (i, s), $s \neq j$, and (r, j), $r \neq i$, which have yet to be placed in the sequence, no longer involve (i, j). This is done as follows:

For each B_r, $r \neq i$, if $B_r(i, j) = 0$, set it to 1 and update DEGREE(r, j).

In each of the precedence vectors P_{ir}, $r \neq i$, remove element j. If this results in an update to $\hat{c}(i, r)$, i.e., if t, the first element in the revised vector, is such that $c_{it} - c_{rt}$ is greater than the previous $\hat{c}(i, r)$, then for t and each other column index s such that $c_{is} - c_{rs} = \hat{c}(i, r)$, make the following changes: If $B_i(r, s) = 0$, set it to 1 and update DEGREE(i, s).

LEMMA 5.3. *An element, say (i, j), can be placed next in a Monge sequence for C if and only if* DEGREE$(i, j) = m$. *So if* DEGREE $< m$ *for every element, there is no Monge sequence.*

Proof. Notice that, by the construction of B_i, if $B_i(r, j) = 1$ for some $r \neq i$, then either (r, j) is already in the sequence, or $c_{ij} - c_{rj} = \hat{c}(i, r) \leqslant c_{is} - c_{rs}$ for all $s \neq j$ which have not been removed from the precedence vector P_{ir} [i.e., (i, s) which have not yet been placed in the sequence]. So if DEGREE$(i, j) = m$, then for every (r, j) and (i, s) not yet in the sequence, $r \neq i$, $s \neq j$, we have $c_{ij} - c_{rj} \leqslant c_{is} - c_{rs}$, which is equivalent to condition (C1).

If, on the other hand, DEGREE$(i, j) < m$, there is an $r \neq i$ such that $B_i(r, j) = 0$ [which indicates that (r, j) is not yet in the sequence]. Since row r in B_i is representative of the precedence vector P_{ir}, we know that j is not a possible first element in P_{ir} [or $B_i(r, j)$ would be 1]. So there is some other column index, s, such that (i, s) and (r, s) have not yet been placed in the sequence, and $c_{is} - c_{rs} = \hat{c}(i, r) < c_{ij} - c_{rj}$ indicating that condition (C1) has been violated; item (i, j) cannot be next in the sequence. [Note that (r, s) indeed has not yet been placed, since otherwise condition (C1) would have been violated before.] ■

By the time the iterative phase of the algorithm is complete, the following operations have been performed. If C is such that a Monge sequence exists, every one of the $m \times n$ items have been placed, each calling for a set of updates to the precedence vectors P_{ir} and the matrices B_i.

When item (i, j) is added to the sequence, each B_r, $r \neq i$, has no more than one of its entries changed. So no more than $O(m^2 n)$ operations are required, in total, for such updates. Updating B_i, however, requires consulting each of $P_{i1}, P_{i2}, \dots, P_{im}$.

If for $P_{i\hat{r}}$, say, the removal of column index j does not result in a change to $\hat{c}(i, \hat{r})$ then no update to row $\hat{r}$ in B_i is necessary. Note that this can be checked in $O(1)$ time by maintaining for each i and r the number of indices j satisfying $c_{ij} - c_{rj} = \hat{c}(i, r)$. If, on the other hand, there is a new value for $\hat{c}(i, \hat{r})$, then for each of the column indices t satisfying, $c_{it} - c_{\hat{r}t} = \hat{c}(i, \hat{r})$, an update to $B_i(\hat{r}, t)$ is necessary. If there are k such indices, they can be found and treated in $O(k)$ time. As in the matrices B_i no 1 is ever changed to 0, the total time for operations of this kind is bounded by the total number of entries in $B_1, \dots, B_m$, which is $O(m^2 n)$.

Hence, since the initialization phase requires $O(m^2 n \log n)$ operations, the total running time of the algorithm is $O(m^2 n \log n)$. We have thus proved the following:

THEOREM 5.4. *The above algorithm generates a Monge sequence for an $m \times n$ transportation cost matrix if and only if one exists. Its running time is $O(\bar{m}^2 \bar{n} \log \bar{n})$ and its space requirement is $O(\bar{m}^2 \bar{n})$, where $\bar{m} = \min(m, n)$ and $\bar{n} = \max(m, n)$.*

Note that for every fixed m, our algorithm works in time $O(n \log n)$. It is unlikely (in fact, not possible, under the comparison model) that this can be improved, as one can easily produce, in time and space $O(n)$, a sorting of the n numbers $a_1, \dots, a_n$ from any Monge sequence for the $2 \times n$ matrix

$$\begin{bmatrix} a_1 & \cdots & a_n \\ -a_1 & \cdots & -a_n \end{bmatrix}.$$

6. CONCLUDING REMARKS AND OPEN PROBLEMS

The above algorithm which detects and constructs Monge sequences can be applied for three purposes:

(1) To help identify new families of greedily solvable transportation problems. Various families of transportation problems for which Monge sequences exist have been identified so far [2, 9, 10, 15]. The main difficulty in the identification of new such families of problems has been the discovery of the Monge sequences, whereas the verification that these are indeed Monge sequences is usually straightforward. Applying the above algorithm makes the identification of new families a much easier task.

(2) To help solve—or partially solve—particular problems where the costs are fixed but repeated solutions are required with varying supply and demand vectors. This was elaborated on in Section 4.

(3) As a preliminary step towards solving any transportation problem. Since the time complexity of the algorithm is better than that of the most efficient solution techniques known to date (see [18]), its benefits come at (essentially) no additional cost.

Once a Monge sequence has been identified by the algorithm, all the subsequent computations with different supply and demand vectors can be performed faster than by any standard transportation algorithm, using the greedy algorithm with that sequence. In fact, the time for a subsequent solution with a given pair of supply and demand vectors is optimal, since reading the input will already require linear time by any algorithm which relates to each instance separately. We suspect that our algorithm may detect new, previously unnoticed Monge sequences in practical problems. This is because such problems usually have highly structured cost matrices, and such structure may allow for a "hidden"Monge sequence.

A question arises as to whether the generation of the solution, when the Monge sequence is already known, can be done in sublinear time, with appropriate efficient preparations. Reading the cost matrix requires mn steps, but when repeated solutions are required with the same cost matrix and only the supply and demand vectors vary, one can argue that the "fresh" input size is $m + n$ only. Since there exist optimal solutions which give nonzero values to at most $m + n$ variables, it is conceivable that an algorithm which does not require the scanning of the whole Monge sequence exists. This issue remains at this point an open question.

Helpful comments from Alan Hoffman and two anonymous referees are gratefully acknowledged. In particular, we thank one of the referees for bringing to our attention References [6] and [3].

REFERENCES

1 R. K. Ahuja, J. B. Orlin, C. Stein, and R. E. Tarjan, unpublished report.

2 E. R. Barnes and A. J. Hoffman, On transportation problems with upper bounds on leading rectangles, *SIAM J. Algebraic Discrete Methods* 6(3):487–496 (1985).

3 R. Chandrasekaran, Recognition of Gilmore-Gomory traveling salesman problem, *Discrete Appl. Math.* 14:231–238 (1986).

4 G. Dantzig, *Linear Programming and Extensions*, Princeton U.P., Princeton, N.J., 1963.

5 P. C. Gilmore and R. E. Gomory, Sequencing a one-state variable machine: A solvable case of the traveling salesman problem, *Oper. Res.* 11:655–679 (1964).

6 P. C. Gilmore, E. L. Lawler, and D. B. Shmoys, Well-solved special cases, in *The Traveling Salesman Problem* (E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, Eds.), Wiley, Chichester, 1985, pp. 87–144.

7 G. Hadley, *Linear Programming*, Addison-Wesley, Reading, Mass., 1962.

8 F. L. Hitchcock, The distribution of a product from several sources to numerous localities, *J. Math. and Phys.* 20:224–230 (1941).

9 D. S. Hochbaum and R. Shamir, Strongly Polynomial Algorithms for the High Multiplicity Scheduling Problem, Tech. Rep. 100/88, Inst. of Computer Sciences, Tel Aviv Univ., Apr. 1988.

10 A. J. Hoffman, On simple transportation problems, in *Convexity; Proceedings of Symposia in Pure Mathematics*, Vol. 7 (V. Klee, Ed.), Amer. Math. Soc., 1963, pp. 317–327.

11 A. J. Hoffman, Some Greedy Ideas, Report, IBM T. J. Watson Research Center, 1979.

12 A. Hoffman, private communication, Aug. 1988.

13 L. Kantorovitch, On the translocation of masses, *C. R. (Dokl.) Acad. Sci. URSS* 37(7–8):199–201 (1942).

14 E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Reinhart and Winston, 1976.

15 B. Lev, A noniterative algorithm for tridiagonal transportation problems and its generalization, *Oper. Res.* 20:109–125 (1972).

16 G. Monge, *Déblai et Remblai*, Mémoires de l'Académie des Sciences, 1781.

17 J. B. Orlin, private communication, June 1988.

18 J. B. Orlin, A faster strongly polynomial minimum cost flow algorithm, in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ACM Press, New York, 1988, pp. 377–387.

19 C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, N.J., 1982.

Received 10 June 1988; final manuscript accepted 3 October 1988