Discrete Applied Mathematics 44 (1993) 21-38
North-Holland

Monge and feasibility sequences in general flow problems

Ilan Adler

IEOR Department, University of California, Berkeley, CA 94720, USA

Alan J. Hoffman

Mathematical Sciences Department, IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA

Ron Shamir*

Department of Computer Science, Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, Israel

Received 10 December 1990
Revised 14 February 1992

Abstract

Adler, I., A.J. Hoffman and R. Shamir, Monge and feasibility sequences in general flow problems, Discrete Applied Mathematics 44 (1993) 21-38.

In a feasible transportation problem, there is always an ordering of the arcs such that greedily sending maximal flow on each arc in turn, according to that order, yields a feasible solution. We characterize those transportation graphs for which there exists a *single* order which is good *for all* feasible problems with the same graph. The characterizations are shown to be intimately related to Monge sequences and to totally balanced matrices. We describe efficient algorithms which, for a given graph, construct such order whenever it exists. For a transportation problem with corresponding $m \times n$ bipartite graph with e arcs, we show how to generate such an order in $O(\min(e \log e, mn))$ steps. Using that order, the feasibility question for any given supply and demand vectors can be determined in $O(m+n)$ time.

We also extend the characterization and algorithms to general minimum cost flow problems in which the underlying graph is nonbipartite, and the sources and destinations are not predetermined. We generalize the theory of Monge sequences too to such problems.

Keywords. Network flow, greedy algorithms, Monge sequences, totally balanced matrices, chordal bipartite graphs, transshipment problems.

Correspondence to: Dr. R. Shamir, Department of Computer Science, School of Mathematics, Tel Aviv University, Tel Aviv 69978, Israel. email: shamir@math.tau.ac.il.

* This work was done while this author was a visitor at RUTCOR, Rutgers University, NJ. Supported by AFOSR grants 89-0512 and 90-0008, and by NSF grant STC 88-09648.

1. Introduction

The transportation problem is one of the fundamental problems of combinatorial optimization. It can be stated as follows: A commodity available at certain sources must be shipped to satisfy demands at some destinations. Shipping costs per unit from each source to each destination are known. Given the amount of commodity available at each source and the amount required at each destination, the problem is to determine how much to send from each source to each destination so that all supplies and demands are met and the total shipping cost incurred is minimum. When not every source can ship to every destination, we say that the problem is *restricted* and call the excluded source-destination pairs *inadmissible*. For a recent survey on algorithms for transportation and related network flow problems, see [3].

The greedy algorithm has the obvious advantage that it is very fast (its number of operations is at most linear in the problem dimensions), but in general its output sends the maximum possible amount of flow directly from the source to the destination, and updates the supplies and demands accordingly. The "north-west corner rule" and the "minimum C_{ij} rule" are two well-known examples of such a greedy algorithm (see, e.g., [17]). We assume that the greedy algorithm scans the arcs in a predetermined order which is independent of the supplies and demands. In principle, every permutation of the admissible source-destination pairs can be used by that algorithm.

The greedy algorithm has the obvious advantage that it is very fast (its number of operations is at most linear in the problem dimensions), but in general its output is insufficient. Depending on the permutation and on the supplies and demands, the solution produced by the greedy algorithm may be optimal, feasible, or infeasible. In restricted problems, a permutation may produce an infeasible solution even when the problem is feasible. Our interest here is in those cost matrices for which there exists a *single* permutation which resolves the feasibility issue *for all* possible supplies and demands. Namely, the greedy algorithm with that permutation produces a feasible solution whenever the supplies and demands are such that the problem is feasible. We call such a permutation a *feasibility sequence*. Similarly, if the greedy algorithm using a permutation produces an optimal solution for every feasible problem, we call such permutation an *optimality sequence*. If the sources and destinations can be reordered so that the source-destination pair (i, j) precedes (k, l) in the permutation whenever $i \leq k$ and $j \leq l$, we say that the permutation is *lexicographic*. For example, the permutation used by the "north-west corner rule" is lexicographic.

There are several reasons to study those problems. From the practical point of view, the availability of an optimality sequence is very useful whenever one needs to solve several problems with the same costs, but with varying supplies and demands: Given an optimality sequence, each problem is solvable in linear time. While randomly chosen cost matrices very seldom admit such sequences, "real" problems often tend to have that property (see the ensuing literature survey).

Another theorectical motivation for this study (which can be stated only in hindsight) is the discovery of intimate interconnections between problems which admit feasibility sequences and well-studied objects from graph theory.

Optimality sequences for transportation problems have been studied in the past. Hoffman [19] proved that for the cost matrix of an unrestricted problem, a sequence is an optimality sequence if and only if it satisfies the Monge property. (A definition of this property will be given in the next section.) Dietrich and Shamir [8,27] (see also [28]) generalized the characterization to restricted transportation problems.

Many families of matrices with special structure which implies the Monge property (and hence can be solved greedily) have been studied in the past. Some examples from the operations research literature include the assignment problem arising in the polynomially solved traveling salesman problem studied by Gilmore and Gomory [14], warehousing problems [19], scheduling problems [4,18] and transportation problems [6,21]. In computational geometry and robotics, totally monotone matrices ([1], see also [2] and the references thereof) have recently received a lot of attention, and were shown to be amenable to very efficient search techniques. Matrices which admit lexicographic optimality sequences turn out to be totally monotone. Recently, interest in the question of speeding up dynamic programming has rekindled, due to important applications in molecular biology and string matching. Several studies have shown that in those applications, a certain convexity (or concavity) condition holds in the costs matrix, which allows dynamic programming to be sped up considerably (see, e.g., [9]). In particular, that condition implies the existence of a lexicographic optimality sequence.

The question of constructing an optimality sequence for a given matrix have been studied by Chandrasekaran [7], for the special case where the sequence is lexicographic (see also [15]). Alon, Cosares, Hochbaum and Shamir [5] provided an efficient algorithm which finds an optimality sequence or determines that no such sequence exists for unrestricted problems. Dietrich and Shamir [8,27,28] generalized the algorithm to restricted transportation problems.

The main results of this paper are the following:

- In Section 3, we give two necessary and sufficient conditions under which a permutation is a feasibility sequence for a given transportation network. The conditions are closely related to the Monge condition for optimality sequences.
- Section 4 extends the theory of feasibility and optimality sequences beyond the transportation problem, to any uncapacitated minimum cost flow problem. This requires first an appropriate generalization of the definitions of such sequences in nonbipartite graphs, which is achieved by considering shortest paths rather than direct arcs between sources and destinations. We characterize the flow problems which admit optimality sequences, and those which admit feasibility sequences. The characterizations lead to efficient algorithms for constructing optimality and feasibility sequences in general flow problems.
- In Section 5 we point out some interesting connections between feasibility sequences and certain well-studied graph theoretic concepts. We prove that a bipartite

graph admits a feasibility sequence if and only if it is chordal bipartite (or, equivalently, if and only if its adjacency matrix is totally balanced). Using this relation, we show that a feasibility sequence for a transportation matrix can be constructed in $O(\min(e \log e, mn))$ steps. We also show that whenever a feasibility sequence exists, a lexicographic one exists, and in that case the greedy algorithm can find a solution in *optimum* time. Finally, we generalize these results to the nonbipartite case.

2. Preliminaries

Let $G=(V_1, V_2, E)$ be a bipartite graph whose two vertex sets are V_1 and V_2, where $|V_1|=n$, $|V_2|=m$ and $|E|=e$. Denote by $F=V_1 \times V_2 - E$ the set of *inadmissible arcs*. Let C be an $n \times m$ transportation cost matrix corresponding to the graph. For an arc $(i,j) \in E$, C_{ij} is the finite, nonnegative cost of shipping each unit on that arc. When arc (i,j) is inadmissible, we also set $C_{ij}=\infty$. The pair (G,C) is called a *transportation network*. (Note that the notation (G,C) is redundant: All the information on the graph G is contained in C, since $(i,j) \in E$ if and only if $C_{ij}<\infty$. We include G explicitly since it will be useful later.)

Denote by $P(C,a,b)$ the uncapacitated transportation problem with the cost matrix C and supply and demand vectors a and b respectively. I.e.,

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in E} C_{ij}x_{ij}, \\
\text{s.t.} \quad & \sum_{\{j \mid (i,j)\in E\}} x_{ij} = a_i, \quad i=1,\ldots,n, \\
& \sum_{\{i \mid (i,j)\in E\}} x_{ij} = b_j, \quad j=1,\ldots,m, \qquad (P(C,a,b)) \\
& x_{ij} \geq 0, \qquad (i,j)\in E.
\end{aligned}
$$

We assume throughout that the supplies and demands are nonnegative, that $\sum_{i=1}^{n} a_i = \sum_{j=1}^{m} b_j$, and that $n \leq m$. We also assume that the underlying undirected graph is connected, so in particular $e \geq m$. We say that a and b are *feasible* for the network if $P(C,a,b)$ has a feasible solution. If $F \neq \emptyset$ we say that the problem is *restricted*. An obvious necessary condition for feasibility of $P(C,a,b)$ is that $\sum_{i=1}^{n} a_i = \sum_{j=1}^{m} b_j$. For unrestricted problems, this condition is also sufficient. However, in restricted problems, it is not immediate to determine feasibility.

Let S be a permutation of the admissible arcs. For a problem $P(C,a,b)$, the *greedy algorithm* maximizes each variable in turn, according to the order given in S. A formal description is the following. (Initially, $x_{ij}=0$ for all $(i,j) \in E$.)

Algorithm GREEDY(S);
begin

For $i = 1, \ldots, e$ **do**
begin let $S_i = (r, s)$.
$x_{rs} \leftarrow \min \{a_r, b_s\}$
$a_r \leftarrow a_r - x_{rs}$
$b_s \leftarrow b_s - x_{rs}$
end
end

When the algorithm terminates, a feasible solution has been obtained if and only if all supplies and demands have been reduced to zero. In general, termination without a feasible solution does not imply infeasibility. Obviously, for every feasible problem there exists a permutation S which gives a feasible solution. (In fact, there exists one which also gives an optimal solution.) We focus our attention here on networks for which there is a single permutation which is good for all supply and demand vectors. More precisely, call a permutation S of the admissible arcs an *optimality sequence* (respectively, a *feasibility sequence*) for the network (G, C) if for every feasible pair a, b of supply and demand vectors, GREEDY(S) provides an optimal (respectively, feasible) solution to $P(C, a, b)$. In particular, termination of GREEDY(S) with an infeasible solution implies the infeasibility of the problem. Obviously, every optimality sequence is a feasibility sequence, but the converse is not necessarily true. The *optimality* (respectively, *feasibility*) *sequence problem* is the following: Given a transportation network, construct an optimality (respectively, feasibility) sequence for it or determine that no such sequence exists.

Our questions here are motivated by Hoffman's theory of Monge sequences. A permutation S of the admissible arcs is called a *Monge sequence* for C if the following condition is satisfied:

For every $i, k \in V_1$ and $j, l \in V_2$, such that (i, j), (i, l) and (k, j) are admissible:

(1) If (k, l) is inadmissible then (i, j) does not precede both (i, l) and (k, j) in S.

(2) If (k, l) is admissible and (i, j) precedes (i, l) and (k, j) in S, then $C_{ij} + C_{kl} \leq C_{il} + C_{kj}$.

Condition (1) depends only on the graph structure, and involves 4-tuples i, j, k, l which contain one inadmissible arc. In 4-tuples containing two or more inadmissible arcs, no conditions are imposed. Condition (2) depends on the finite costs, and it involves only 4-tuples in which all arcs are admissible. (Formally, the two conditions could be stated together by requiring that for every 4-tuple i, j, k, l, $C_{ij} + C_{kl} \leq C_{il} + C_{kj}$, where for sums involving infinity one defines $\infty + a = \infty$ for every finite a [28].)

The following theorem summarizes the knowledge on the optimality sequence problem in transportation networks:

Theorem 2.1. (a) (Hoffman [19], Shamir and Dietrich [28]) *S is an optimality sequence for (G, C) if and only if it is a Monge sequence for C.*

(b) (Alon et al. [5], Shamir and Dietrich [28]) *The optimality sequence problem is solvable in* $O(en \log n)$.

3. Feasibility sequences

We first turn to the problem of characterizing feasibility sequences: Given a cost matrix C, define its *skeleton matrix* $\bar{C}$ to be the matrix obtained by replacing all the finite costs in C by ones. I.e., $\bar{C}_{ij} = 1$ if and only if $(i, j) \in E$, and $\bar{C}_{ij} = \infty$ if and only if $(i, j) \in F$. The skeleton matrix of a transportation graph is defined analogously. The following theorem follows from Theorem 2.1(a):

Theorem 3.1. *S is a feasibility sequence for a bipartite graph G if and only if it is a Monge sequence for its skeleton matrix $\bar{C}$.*

Since the existence of a feasibility sequence depends only on the structure of the underlying graph and not on the finite costs (i.e., only on condition (1) in the definition of a Monge sequence), we shall call S a feasibility sequence *for the graph G*. We shall also denote the feasibility problem $P(G, a, b)$ to emphasize its dependence on the graph structure only.

Using Theorem 3.1, one can apply the algorithms of [28] to $\bar{C}$ in order to solve the feasibility sequence for a given transportation graph. With such a sequence, a feasible solution for every feasibile instance of problem $P(G, a, b)$ can be constructed in $O(e)$ steps by the greedy algorithm. We shall improve upon both of these results in Section 5.

Interestingly, even by severely limiting the range of supply and demand vectors for which an optimality (or feasibility) sequence is required to give the right answer, one cannot obtain a larger class of greedily solvable networks. This fact is stated in the following theorem, whose proof is similar to that of Theorem 2.1(a) and is omitted:

Proposition 3.2. *Let (G, C) be a transportation network and let S be a permutation of its admissible arcs. The following are equivalent:*

(a) *S is an optimality sequence.*

(b) *For every feasible* $(0, 1)$ *supply and demand vectors, GREEDY(S) gives an optimal solution.*

(c) *For every feasible* $(0, 1)$ *supply and demand vectors with only two coordinates equal to one in each of them, GREEDY(S) gives an optimal solution.*

One consequence of this proposition is for the perfect matching problem: The bipartite matching problem (see, e.g., [3]) can be viewed as a transportation problem on a bipartite graph, where the supplies and demands are all ones. A perfect matching exists if and only if that transportation problem is feasible. Let us call a sequence S of the admissible arcs a *perfect sequence* for a bipartite graph G if for every induced subgraph on which a perfect matching exists, GREEDY(S) gives a perfect matching. Since assigning supply or demand zero to a vertex is equivalent to removing it from the problem, the above observation implies the following.

Corollary 3.3. *A bipartite graph G admits a perfect sequence if and only if it admits a feasibility sequence. Morever, S is a perfect sequence for G if and only if it is a feasibility sequence for G.*

4. Nonbipartite problems

In the transportation problems discussed above, the underlying graph is bipartite and the vertices are prepartitioned into sources and destinations. An interesting question is whether the notions of optimality and feasibility sequences could be extended to more general flow problems. We show below that indeed this can be done, thereby extending the theory to general minimum cost flow problems. We assume throughout that the problems are uncapacitated.

We first remove the restriction that the graph is bipartite, but assume that the sources and destinations are specified: For a directed graph $G=(V,E)$ with arc costs C, let $V=V^+\cup V^-\cup V^0$ be a partition of the vertices. We also write G_π and V_π to emphasize the underlying partition π. V^+ and V^- are the sources and destinations in V, respectively. Given a pair of nonnegative supply and demand vectors $a=(a_i, i\in V^+)$ and $b=(b_j, j\in V^-)$, with $\sum_{j\in V^+} a_i=\sum_{j\in V^-} b_j$, a minimum cost flow problem is determined, as follows:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} C_{ij}x_{ij}, \\ \text{s.t.} \quad & \sum_{\{j\,|\,(i,j)\in E\}} x_{ij} - \sum_{\{k\,|\,(k,i)\in E\}} x_{ki} = \begin{cases} a_i, & i\in V^+, \\ 0, & i\in V^0, \\ -b_i, & i\in V^-, \end{cases} \qquad (P_\pi(C,a,b)) \\ & x_{ij}\geq 0, \qquad (i,j)\in E. \end{aligned}$$

Transitive closures will play a central role in the sequel. The *transitive closure* of the directed graph G is the grap $G^t=(V,E^t)$ satisfying $(i,j)\in E^t$ if and only if there is a path from i to j in G. We say that an ordered pair (i,j) is *connected* if $(i,j)\in E^t$. For a connected pair (i,j), we denote by C'_{ij} the cost of the cheapest path from i to j in G, with respect to costs C. (We adopt the convention that (i,i) is *not* connected.) If (i,j) is not connected then we set $C'_{ij}=\infty$.

The *greedy algorithm* for the problem $P_\pi(C,a,b)$ repeatedly picks a connected pair (i,j) where $i\in V^+$ and $j\in V^-$, and sends a maximum amount of flow along the cheapest path from i to j. (Note the change in the algorithm, compared to transportation problems: Rather than maximizing flow on a single arc in each step, here the flow along a *complete path* is maximized. Note also that for transportation problems one gets the original algorithm.) An *optimality sequence* for the pair (G_π, C) is a permutation S of the ordered connected pairs in $\{(i,j)\,|\,i\in V^+, j\in V^-\}$ such that for every feasible supply and demand vectors, the greedy algorithm which uses the order S attains an optimal solution. A feasibility sequence for G_π is defined analogously.

Given G_π and C, define a related bipartite graph $G' = G'_\pi = (V^+, V^-, E')$ by $E' = (V^+ \times V^-) \cap E^t$. The costs C' in G' are the cheapest path costs in G, as defined above. Clearly $P(C', a, b)$ is a transportation problem, and it is easy to observe that one can solve it to obtain an optimal solution for the original problem. This observation allows us to reduce the nonbipartite case to the bipartite case and apply to it the results for optimality and feasibility sequences for transportation. Hence we can conclude:

Proposition 4.1. *Given a directed graph G_π (not necessarily bipartite) with arc costs C and specified sets of sources and destinations, S is an optimality sequence for (G_π, C) if and only if it is a Monge sequence for (G'_π, C'). Moreover, S is a feasibility sequence for G_π if and only if it is a Monge sequence for $\bar{C}'$.*

In the following corollaries, denote $N = |V|$, $e = |E|$, $n = |V^+|$, $m = |V^-|$ and $\tilde{e} = |E'|$. Again, without loss of generality we assume $n \le m$. We denote by $\mathrm{SP}(k, l)$ the complexity of solving a single-source shortest path problem on a graph with k vertices and l arcs.

Corollary 4.2. *Under the conditions of Proposition* 4.1, *the optimality sequence problem for (G_π, C) is solvable in $O(n \cdot \mathrm{SP}(N, e) + \tilde{e}n \log m)$ steps.*

Proof. Constructing the graph G' can be done by finding n shortest path trees, one from each of the sources in G. A Monge sequence in (G', C') can subsequently be found in $O(\tilde{e}n \log m)$ steps [28]. □

For example, by using the algorithm of [13], $\mathrm{SP}(N, e) = O(e + N \log N)$, which gives an overall complexity of $O(en + nN \log N + \tilde{e}n \log m)$ steps. (When $n = \Omega(m)$ and G' is very dense, one can improve slightly upon the complexity in the above corollary by using Fredman's all-pairs shortest path algorithm [12] for the preprocessing step.) In particular, when $n + m = N$ and G is not very sparse ($e = \Omega(N \log N)$), this gives $O(en + \tilde{e}n \log m)$ steps. Not surprisingly, the transformation of the graph may be the most costly operation, for dense networks with relatively few supply and demand vertices. However, unless $e = o(\tilde{e})$ this algorithm is still faster than the best known strongly polynomial algorithm for solving a single uncapacitated minimum cost flow problem, with fixed supply and demand vectors, which takes $O(N \log N(e + N \log N))$ steps [24].

Corollary 4.3. *Under the conditions of Proposition* 4.1, *the feasibility sequence problem for G_π is solvable in $O(ne + \tilde{e}n \log m)$ steps.*

Proof. The proof follows from Theorem 3.1, Proposition 4.1 and Corollary 4.2. The slight improvement in complexity is in the preliminary step: Computing the con-

nectivity from each vertex in V^+ can be done in $O(e)$ steps by depth-first search, for a total of $O(ne)$ steps of computing $\bar{C}'$. □

We now consider the more general case, in which there is no fixed partition into sources and destinations. In this case, a directed graph $G=(V,E)$ with arc costs C is given. A supply function d is an assignment of values to the vertices d_v, $v \in V$, where d_v is the supply (demand) at vertex v if $d_v>0$ ($d_v<0$). This determines a minimum cost flow problem $P(C,d)$. A formulation of the problem is the following:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} C_{ij}x_{ij}, \\ \text{s.t.} \quad & \sum_{\{j\,|\,(i,j)\in E\}} x_{ij} - \sum_{\{k\,|\,(k,i)\in E\}} x_{ki} = d_i, \quad i \in V, \qquad (P(C,d)) \\ & x_{ij} \geq 0, \qquad (i,j)\in E. \end{aligned}$$

The *greedy algorithm* for problem $P(C,d)$ repeatedly picks a connected pair (i,j), and checks if i is a source and j is a destination. In case both answers are positive, the algorithm sends a maximum amount of flow along the cheapest path from i to j. For a permutation S of the pairs $\{(i,j)\,|\,(i,j)\in E^t\}$, denote by GREEDY($S$) that algorithm which picks the pairs according to their order in S. S is called an *optimality supersequence* for this problem if for every supply function d for which $P(G,d)$ is feasible, GREEDY(S) terminates with an optimal solution. (We call S a supersequence to distinguish it from the previous cases, and to emphasize that for every supply function only a fraction of the pairs in S is relevant for the greedy algorithm.) A feasibility supersequence is defined analogously.

Recall that C' is the matrix of shortest path costs in G. Define S to be a *Monge supersequence* for (G,C) if it satisfies the following condition:

> For every 4-tuple i,j,k,l of distinct vertices, if each of the pairs (i,j), (i,l) and (k,j) is connected and (i,j) precedes (i,l) and (k,j) in S, then (k,l) is also connected and $C'_{ij}+C'_{kl} \leq C'_{il}+C'_{kj}$.

Note the requirement that all four vertices be distinct. This requirement was not necessary in previous versions of the Monge conditions [19,28], since the two sets of vertices corresponding to the rows and the columns were by definition disjoint. (In fact, since a supersequence S cannot contain pairs (i,i) we could omit the explicit requirement of distinctness.)

Proposition 4.4. *S is an optimality supersequence for (G,C) if and only if it is a Monge supersequence for (G,C).*

Proof. Every supply function d induces a partition π of V into $V_\pi = V^+ \cup V^- \cup V^0$, and a fixed partition problem (G_π, C). Let $\hat{S}$ be the subsequence of S induced by this partition, i.e.,

$$\hat{S} = S \cap \{(i,j)\,|\,i\in V^+,\ j\in V^-,\ (i,j)\in E^t\}.$$

If S is a Monge supersequence for (G, C), then by definition $\hat{S}$ is a Monge sequence for (G'_π, C'), hence by [19,28], it is an optimality sequence for problem (G'_π, C'). In particular, if S is a Monge supersequence then for every feasible supply function, GREEDY(S) terminates with an optimal solution. The proof of the converse is similar to that of the corresponding claims in [19,27]. □

Corollary 4.5. *S is a feasibility supersequence for G if and only if it is a Monge supersequence for $(G, \bar{C})$.*

Corollary 4.6. *Let $G(V,E)$ be a directed graph with arc costs C where $N=|V|$, $e=|E|$, and $e'=|E^t|$. The optimality supersequence problem for (G,C) is solvable in $O(e'N \log N)$ steps. In particular, the feasibility supersequence problem is solvable in the same complexity.*

Proof. The proof is similar to that of Corollaries 4.2 and 4.3. Here one needs to find a Monge sequence for (G', C'), where $G'=(V,V,E')$ satisfies $(i,j)\in E'$ if and only if $i\neq j$ and $(i,j)\in E^t$, and C'_{ij} is the cost of the shortest path from i to j in G. Computing (G', C') requires calculation of all pairs shortest path distances, which can be done, for example, in $O(Ne)$ steps by Floyd's algorithm (see, e.g., [26]). The bottleneck in the complexity is finding a Monge supersequence in (G', C'), which can be done in $O(e'N \log N)$ by a minor adaptation of the algorithm in [27], to take care of the special pairs (i,i). □

For the transportation problem, Alon et al. observed that every $2\times n$ problem admits a Monge sequence [5]. This property is obviously true also for feasibility sequences, and is carried over to the corresponding problems on a nonbipartite graph G_π. When looking for supersequences, where there is no fixed partition anymore, we can recognize two special cases in which a sequence always exists: If G' is a tree then there is always an optimality supersequence, since an arc which is incident on a leaf can always be placed next in a sequence without violating the Monge condition. The other case will be discussed in the next section.

5. Graph theoretic connections

We now give several graph theoretic characterizations of graphs which admit feasibility sequences. Using these characterizations, we will be able to improve upon the complexity of some of the algorithms presented in Sections 3 and 4. First, we need some definitions: A (0, 1) matrix is called *totally balanced* if it contains no $k\times k$ submatrix with $k\geq 3$, which has exactly two ones in each row and each column and has no identical rows [20]. A matrix is called *Γ-free* if it does not contain $\Gamma=\left(\begin{smallmatrix}1 & 1\\ 1 & 0\end{smallmatrix}\right)$ as a submatrix. The following characterization of totally balanced matrices was introduced independently by Lubiw and Farber and used by Hoffman et al.:

Theorem 5.1 [10, 20, 22]. *A matrix is totally balanced if and only if its rows and columns can be permuted in such a way that the resulting matrix is Γ-free.*

Let us now return to bipartite graphs. For a bipartite graph $G=(V_1, V_2, E)$ with $|V_1|=n$, $|V_2|=m$, let $M=M(G)$ be its $n\times m$ vertex-vertex adjacency matrix, i.e., $M_{ij}=1$ or 0 if $(i,j)\in E$ or $(i,j)\in F$, respectively. It follows from the definition that for a bipartite graph G, $M(G)$ is totally balanced if and only if G does not contain an induced chordless cycle of length six or more. (Namely, there is no set of vertices $Y=\{v_1,\ldots,v_k\}$ where $k\geq 6$ such that the subgraph of G induced by Y contains exactly the arcs $(v_1,v_2),(v_2,v_3),\ldots,(v_k,v_1)$.) A bipartite graph satisfying that property is also called *chordal bipartite* [16].

To keep the notation consistent with that of Section 3, let us say that the $(\infty,1)$ skeleton matrix and the $(0,1)$ adjacency matrix *correspond* if they are induced by the same bipartite graph G. This simply means that all ∞ entries in the former are replaced by zeroes in the latter, and vice versa.

Theorem 5.2. *The adjacency matrix of a bipartite graph is totally balanced if and only if the corresponding skeleton matrix admits a Monge sequence.*

Proof. Denote the adjacency and skeleton matrices by M and $\bar{C}$, respectively. Assume first that M is totally balanced. By Theorem 5.1, M can be permuted into a Γ-free form. Assume that M is already in that form. Let S be the permutation obtained from the natural ordering of the arcs $(1,1),(1,2),\ldots,(1,m),(2,1),\ldots,(n,m)$ by omitting all inadmissible arcs. S will be called a *lexicographic permutation* in the sequel. Let arcs $(i,j),(i,s)$ and (r,j) be admissible and assume that (i,j) precedes (r,j) and (i,s) in S. By the lexicographic ordering, this implies that $i<r$ and $j<s$. But since M is Γ-free, (r,s) must also be admissible. Hence S is a Monge sequence for $\bar{C}$, and a feasibility sequence for G.

For the converse, assume that M is not totally balanced. Then M contains a $k\times k$ submatrix $(k\geq 3)$ corresponding to a cycle. It is easy to verify that no element in the corresponding submatrix of $\bar{C}$ can appear first among the elements of the submatrix in a Monge sequence, hence $\bar{C}$ does not admit a Monge sequence. □

In [27], a simple example was given of a problem which does not admit a Monge sequence. (The cost matrix was 3×3 with the diagonal entries ∞ and all the other entries finite.) The above proof implies that only problems of that type (or those which contain subproblems of that type) do not admit a feasibility sequence. More precisely:

Corollary 5.3. *A bipartite graph admits a feasibility sequence if and only if it is chordal bipartite.*

The proof of Theorem 5.2 implies also that every bipartite graph which admits a feasibility sequence has one of a particular simple type:

Corollary 5.4. *If a bipartite graph admits a feasibility sequence, then one can permute its vertices to obtain a lexicographic feasibility sequence.*

In other words, whenever a feasibility sequence exists, there is one which can be represented by row and column permutations. These row and column permutations are exactly those which put M into Γ-free form. Moreover, with respect to that order of rows and columns, every other permutation with the property that (i, j) succeeds $(i, j-1)$ and $(i-1, j)$ in S whenever these arcs are admissible is also a feasibility sequence.

If the matrix is in Γ-free form, then the greedy algorithm, using the lexicographic sequence, scans the matrix from left to right and from top to bottom. In this case, it is possible to avoid scanning entries whose row or column has already been eliminated, and to perform the algorithm in time proportional to the number of variables whose value is positive in the resulting solution. That number is at most $m+n-1$. Hence we can improve upon the results of Section 3 as follows:

Corollary 5.5. *In a bipartite graph G which admits a feasibility sequence, one can find a feasibility sequence which provides a feasible solution or determines infeasibility for every problem $P(G, a, b)$ in $O(m+n)$ steps.*

Note that $O(m+n)$ is optimum for constructing a solution, since a solution consists of $m+n-1$ positive numbers in nondegenerate problems.

Note also that from Corollaries 3.3 and 5.3 we get that for the matching problem, a graph admits a perfect sequence if and only if it is chordal bipartite.

An efficient method to determine if a matrix is totally balanced, and to find its Γ-free form in case it is, is the following: A matrix is totally balanced if and only if the matrix obtained by reverse doubly lexical ordering of it is Γ-free [20]. So, one can first find a reverse doubly lexical ordering of the matrix, and then check if it is Γ-free. Finding doubly lexical ordering can be done in $O(L \log L)$, where $L = m+n+e$, by an algorithm due to Paige and Tarjan [25]. For dense matrices, a recent algorithm of Spinrad reduces the complexity to $O(mn)$ [29]. Given the ordering, checking whether the matrix if Γ-free can be done in $O(L)$ time [23]. Using these results we can improve upon the results of Section 3 and Corollary 4.3 as follows:

Corollary 5.6. *The feasibility sequence problem can be solved in $O(\min(e \log e, mn))$ steps for bipartite graphs, and in $O(ne)$ steps for a nonbipartite graph with fixed source-destination sets.*

Determining if a restricted transportation problem with given supplies and demands is feasible is equivalent to solving a maximum flow problem. The best known general algorithms for the maximum flow problem require $O(em)$ steps or more (see, e.g., [3]). The complexities of our algorithms above are lower. Hence instead of solving a feasibility problem by maximum flow algorithms, one can first

use the faster algorithms above to look for a feasibility sequence, and in case one is found, the greedy algorithm subsequently resolves the feasibility question in linear time. If no sequence is found, one can still use the maximum flow algorithm without loss in the overall complexity.

We now turn to the general case, where there is no fixed partition into sources and destinations. We first prove a lemma which will help simplify the detection of feasibility supersequences. Recall that vertices i and j are called strongly connected in G if each one is reachable from the other, i.e., both $(i,j) \in E^t$ and $(j,i) \in E^t$.

Lemma 5.7. *Let i and j be strongly connected in G, and let G' be the graph obtained from G by contracting i and j into v. G admits a feasibility supersequence if and only if G' does.*

Proof. If G' does not admit a feasibility supersequence then clearly G does not admit one.

Assume that G' admits a feasibility supersequence S'. Construct a sequence S as follows:

```
Algorithm EXPAND(S', v, i, j);
begin
  For every k ≠ v do
  begin
    Replace (v,k) in S' by (i,k),(j,k); Replace (k,v) in S' by (k,i),
    (k,j);
  end
  Set S' ← ((i,j),(j,i),S');
end
```

We claim that the resulting sequence S is a feasibility supersequence for G. To simplify the argument, let us first introduce the following definition: If each of the pairs (k,l), (k,r), (s,l) is connected and (s,r) is not, let us call that 4-tuple of arcs a *blocker* in S if (k,l) precedes (k,r) and (s,l) in S. We also call the pairs k,s and l,r the two *sides* of the blocker. By the definition of a Monge supersequence and Corollary 4.5, S is a feasibility supersequence if and only if it does not contain a blocker.

Suppose S is not a feasibility supersequence. Then there is a blocker in S. It must contain both i and j or else the corresponding 4-tuple in S' would be a blocker in S'. Vertices i and j cannot be on the same side of a blocker, since they have the same set of (in- and out-) neighbors. Hence they must be on opposite sides of a blocking 4-tuple. Out of the four arcs in any such blocker, either (i,j) or (j,i) appears first in S, by the construction of EXPAND. Since i and j are strongly connected, if (i,l), (k,j) are connected then the pair (k,l) is connected too, by transitivity, so (i,j), (j,i) are not contained in any blocker of S. Hence S does not contain a blocker, so it is a feasibility supersequence. □

The *condensation* of a directed graph G is the directed graph obtained from G by contracting each strongly connected component into a single vertex. Using Lemma 5.7 repeatedly we get:

Corollary 5.8. *A graph admits a feasibility supersequence if and only if its condensation does.*

Note that the condensation is always acyclic. In particular, Corollary 5.8 and the discussion in Section 4 imply that it is sufficient to be able to find feasibility supersequences in acyclic graphs which are closed under transitivity. (A graph is closed under transitivity if it is identical to its transitive closure.) We use this fact in the following theorem:

Theorem 5.9. *Let G be a directed acyclic graph which is closed under transitivity, and let $M(G)$ be its adjacency matrix. G admits a feasibility supersequence if and only if $M(G)$ is totally balanced.*

Recall that the adjacency matrix of the directed graph S satisfies $M_{ij}=1$ if and only if $(i,j)\in E$ and 0 otherwise. In particular, $M_{ii}=0$ for all i.

Proof. Sufficiency follows in the same way as in Theorem 5.2: If M is totally balanced then by Theorem 5.1 it has a Γ-free form. With respect to that form, the lexicographic sequence is a feasibility supersequence.

Let us now prove necessity: If M is not totally balanced then it contains a submatrix corresponding to a chordless cycle of $2k$ vertices, $C=(i_0, j_0, i_1, j_1, \ldots, i_{k-1}, j_{k-1}, i_0)$ for some $k\geq 3$. It suffices to show that these vertices must be distinct in order to imply that G does not admit a feasibility sequence. Assume conversely that C contains two identical vertices, r and s. If r and s have the same parity in the cycle, say $r=i_l$ and $s=i_m$, then either (s, j_{l-1}) or (s, j_{l+1}) is a chord (here and throughout, all indices are mod k). If r and s have opposite parity, say $r=i_l$ and $s=j_m$, then since in the adjacency matrix $M_{ii}=0$ for all i, r and s cannot be adjacent in the cycle. Since $(i_m, s)\in E$, $(r, j_l)\in E$ and $(r, j_{l-1})\in E$, and $r=s$, by transitivity both (i_m, j_{l-1}) and (i_m, j_l) must be present in E, and at least one of them is a chord. Hence all vertices in C must be distinct. In such a cycle C, no vertex can appear first in a feasibility supersequence. □

Interestingly, from the above theorem using the terminology of partially ordered sets, we get that G admits a feasibility supersequence if and only if the partially ordered set corresponding to G does not contain a crown of order ≥ 3 (see, e.g., [11]). This interpretation immediately gives another set of graphs on which the existence of a feasibility supersequence is guaranteed: Recall that in a directed acyclic graph two vertices are *comparable* if there is a path from one of them to the other, and an *antichain* is a set of vertices no two of which are comparable.

Corollary 5.10. *Let G be a directed acyclic graph closed under transitivity. If the maximum antichain in G is of size at most two then G admits a feasibility supersequence.*

Using Corollary 5.8 and Theorem 5.9, we can recognize if a graph admits a feasibility supersequence by the following algorithm:

Algorithm FEASIBILITY-SUPERSEQUENCE(G);
begin
Step 1. Contract strong components of G. Call the resulting graph cond(G).
Step 2. Compute D, the transitive closure of cond(G).
Step 3. Compute a reverse doubly lexical ordering of $M(D)$, the adjacency matrix of D. Call the reordered matrix M^*.
Step 4. Check whether M^* is Γ-free. If not, stop, G does not admit a feasibility supersequence. If it is, the lexicographic order in M^* induces a feasibility supersequence S' in D. For each pair i, j which was contracted into v, EXPAND(S', v, i, j).
end

Denote by $\hat{n}$ and $\hat{e}$ the number of vertices and arcs, respectively, in cond(G), and by $\bar{e}$ the number of arcs in D above. For a graph with k vertices and l arcs, denote by $\mathrm{TC}(k,l)$ the complexity of computing its transitive closure, and by $\mathrm{GF}(k,l)$ the complexity of finding a Γ-free permutation of its adjacency matrix, or determining that none exists.

Theorem 5.11. *The above algorithm constructs a feasibility supersequence or determines that no such sequence exists in* $O(e+\mathrm{TC}(\hat{n},\hat{e})+\mathrm{GF}(\hat{n},\bar{e}))$ *steps.*

Proof. Validity follows from Corollary 5.8 and Theorem 5.9. The strong components can be computed in Step 1 in $O(e)$ [30]. Step 2 requires calculating the transitive closure of cond(G), and Steps 3–4 find the Γ-free form. □

As in Corollary 5.6, Steps 3 and 4 require $O(\min(\hat{n}^2, \bar{e}\log\hat{n}))$ and $O(\bar{e})$, respectively. Hence, in particular, using Warshall's algorithm for the transitive closure (see, e.g., [26]) gives an overall complexity of $O(e+\hat{n}\hat{e})$. Note that formally Step 1 is not necessary for the validity of the algorithm. (If we do not perform it, we shall have identical rows and columns in $M(D)$ for vertices in the same strongly connected component.) However, this step may considerably reduce the size of the graph on which Steps 2–4 are performed, and is faster to perform than Step 2. Hence, in practice contracting strong components first may improve the running time when these components form a large part of the graph.

6. Concluding remarks

We have shown how the theory of optimality sequences can be extended from the bipartite case to general minimum cost flow problems. We have also described a parallel theory of feasibility sequences, gave necessary and sufficient conditions for the existence of such sequence, and provided efficient algorithms for constructing optimality and feasibility sequences in general networks. The complexities of these algorithms are comparable to those of the fastest known general algorithms for solving a single feasibility or optimality question for a specific supply and demand function. The complexity results are summarized in Table 1.

We have also shown that the concept of a graph admitting a feasibility sequence is closely related to the concepts of totally balanced matrices and chordal bipartite graphs. Since these concepts have been shown in the past to be very useful for studying integral polyhedra and perfect Gaussian elimination, it may be interesting to investigate these relations further. A deeper understanding of the relations between optimality sequences and totally monotone matrices is also of interest.

If the algorithm of [27] is used to find a Monge sequence (either for the optimality or for the feasibility question), it may terminate with a negative answer after having produced a partial sequence. As already noted by [5, 27], this partial sequence can still be used by the greedy algorithm until exhausted, thereby reducing the problem size considerably. Subsequently, general flow algorithms may be applied on the reduced problem. Since the use of each element in the partial sequence may remove one or two vertices from the problem, a partial sequence may also be useful in practice.

Finally, let us mention several open questions that arise from our study:

(1) Extend the theory of optimality and feasibility sequences to capacitated flow problems.

(2) Given a minimum cost problem which does not admit a feasibility or an optimality sequence, find how to modify the costs "in a minimal way" so that the modified problem will admit such a sequence.

(3) Given a sequence S which is not lexicographic, is there an efficient way to preprocess it in such a way that for every supply function, GREEDY(S) will take $O(m+n)$ steps?

Table 1
Complexity of the optimality and feasibility sequence algorithms in various networks
(Notation: n sources, m destinations, $n \le m$; N vertices, e arcs; $\bar{e}$ arcs in the corresponding bipartite graph for the fixed partition case; e' arcs in the transitive closure; $\hat{n}$ vertices and $\hat{e}$ arcs in the condensation; $\bar{e}$ arcs in the transitive closure of the condensation. $SP(k, l)$ = complexity of the single source shortest path problem in a graph with k vertices, l arcs.)

	Bipartite	Fixed partition	General
Optimality	$O(en \log m)$ [27]	$O(n \cdot SP(N, e) + \bar{e}n \log m)$	$O(e'N \log N)$
Feasibility	$O(\min(e \log e, mn))$	$O(en)$	$O(e + \hat{n}\hat{e})$

References

[1] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor and R. Wilber, Geometric applications of a matrix-search algorithm, Algorithmica 2 (1987) 195–208.

[2] A. Aggarwal, D. Kravets, J.K. Park and S. Sen, Parallel searching in generalized Monge arrays with applications, in: Proceedings 2nd Annual Symposium on Parallel Algorithms and Architectures, to appear.

[3] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, Network flows, in: G.L. Nemhauser, A.H.G. Rinnooy Kan and M.S. Todd, eds., Handbooks in Operations Research and Management Science Vol. I (Elsevier, Amsterdam, 1989) 211–369.

[4] S. Albèrs and P. Brucker, The complexity of one-machine batching problems, Tech. Rep., Osnabrück University (1990).

[5] N. Alon, S. Cosares, D. Hochbaum and R. Shamir, An algorithm for the detection and construction of Monge sequences, Linear Algebra Appl. 114/115 (1989) 669–680.

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

[7] R. Chandrasekaran, Recognition of Gilmore–Gomory traveling salesman problem, Discrete Appl. Math. 14 (1986) 231–238.

[8] B.L. Dietrich, Monge sequences, antimatroids, and the transportation problem with forbidden arcs, Linear Algebra Appl. 139 (1990) 133–145.

[9] D. Eppstein, Z. Galil and R. Giancarlo, Speeding up dynamic programming, in: Proceedings 29th IEEE Symposium on Foundations of Computer Science (1988) 488–496.

[10] M. Farber, Characterizations of strongly chordal graphs, Discrete Math. 43 (1983) 173–189.

[11] P.C. Fishburn, Interval Orders and Interval Graphs (Wiley, New York, 1985).

[12] M.L. Fredman, New bounds on the complexity of the shortest path problem, SIAM J. Comput. 5 (1976) 83–89.

[13] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (1987) 596–615.

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

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

[16] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs (Academic Press, New York, 1980).

[17] G. Hadley, Linear Programming (Addison-Wesley, Reading, MA, 1962).

[18] D.S. Hochbaum and R. Shamir, Minimizing the number of tardy job units under release time constraints, Discrete Appl. Math. 28 (1990) 45–57.

[19] A.J. Hoffman, On simple linear programming problems, in: V. Klee, ed., Convexity: Proceedings of Symposia in Pure Mathematics Vol. 7 (American Mathematical Society, Providence, RI, 1963).

[20] A.J. Hoffman, A.W. Kolen and M. Sakarovitch, Totally balanced and greedy matrices, SIAM J. Algebraic Discrete Methods 6 (1985) 721–730.

[21] B. Lev, A noniterative algorithm for tridiagonal transportation problems and its generalization, Oper. Res. 20 (1972) 109–125.

[22] A. Lubiw, Γ-free matrices, Master's thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ont. (1982).

[23] A. Lubiw, Doubly lexical ordering of matrices, SIAM J. Comput. 16 (1987) 854–879.

[24] J.B. Orlin, A faster strongly polynomial minimum cost flow algorithm, in: Proceedings 20th ACM Symposium on Theory of Computing (1988) 377–387; revised version: Sloan W.P. No. 3060-89-MS, Sloan School of Management, M.I.T. (1989); also: Oper. Res., to appear.

[25] R Paige and R.E. Tarjan, Three partition refinement algorithms, SIAM J. Comput. 16 (1987) 973–989.
[26] R. Sedgewick, Algorithms (Addison-Wesley, Reading, MA, 1988).
[27] R. Shamir, A fast algorithm for constructing Monge sequences in transportation problems with forbidden arcs, Rept. 136/89, Tel Aviv University, Tel Aviv (1989); also: Discrete Math., to appear.
[28] R. Shamir and B. Dietrich, Characterization and algorithms for greedily solvable transportation problems, in: Proceedings of the First ACM/SIAM Symposium on Discrete Algorithms (SIAM, Philadelphia, PA, 1990) 358–366.
[29] J.P. Spinrad, Doubly lexical ordering of dense 0-1 matrices, Rept., Computer Science Department, Vanderbilt University, Nashville, TN (1990); also: Inform. Process. Lett., to appear.
[30] R.E. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (1972) 146–160.