

Network Optimization Problems, pp. 1-22
Eds. D.-Z. Du and P.M. Pardalos
©1993 World Scientific Publishing Co.

Greedily Solvable Transportation Networks and Edge-Guided Vertex Elimination

Ilan Adler

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

Ron Shamir

*Department of Computer Science, Sackler Faculty of Exact Sciences, Tel Aviv
University, Tel-Aviv 69978, ISRAEL.*¹

Abstract

The greedy algorithm for the transportation problem repeatedly picks an edge, maximizes flow on it and updates the supplies and demands. If, with the same order of edges, the greedy algorithm gives an optimal (resp., feasible) solution for every feasible supply and demand functions, we call that order an optimality (resp., feasibility) sequence. We show that with a feasibility sequence, one can guarantee a feasible solution which is also a spanning tree, or give a certificate on infeasibility. Furthermore, with an optimality sequence, one can guarantee an optimal basic solution for every feasible problem. We also show how to obtain a spanning tree with a given signature or a dual feasible basis with a given signature, using feasibility and optimality sequences. Our results build on some interesting properties of vertex elimination algorithms which are guided by edge orders.

1 Introduction

The transportation problem can be stated as follows: A commodity which is available at certain sources is demanded at some destinations. Shipping costs from each source

¹Supported by AFOSR grants 89-0512 and 90-0008, and by NSF grant STC88-09648.

to each destination are known. Given the amount available at each source and the amount demanded at each destination, the problem is to determine how much to send directly 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*. The transportation problem is one of the fundamental problems in combinatorial optimization, and has been studied intensively over the last fifty years. The excellent surveys [2] and [11] describe several algorithms for transportation and related network flow problems.

One natural approach for solving the transportation problem is by a greedy algorithm, guided by a predetermined order of the edges. This algorithm repeatedly picks the next edge in that order, sends the maximum possible amount of flow along it and updates the supplies and demands accordingly. In general, this approach does not guarantee an optimal solution, but is often used to generate an initial feasible solution for more elaborate methods in unrestricted problems. The “north-west corner rule” and the “minimum C_{ij} rule” are two well-known examples of this approach (see, e.g., [13].)

The greedy algorithm has the obvious advantage that it is very fast, hence the interest in identifying when it guarantees an optimal solution. Research has focused on cases when there exists a *single* order of the edges which - when used by the greedy algorithm - is guaranteed to produce an optimal solution *for all* feasible problems on that network. Such order of the edges is called an *optimality sequence*. The knowledge of an optimality sequence is useful in particular if 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. Similarly, if the greedy algorithm guided by some edge-order produces a feasible solution for every feasible problem, that order is called a *feasibility sequence*.

Hoffman [15] gave a characterization of bipartite networks which admit an optimality sequence, in terms of the Monge property. (The property will be defined in the next section.) Dietrich and Shamir [8, 18] (see also [19]) generalized the characterization to restricted transportation problems. Many families of problems which fit those characterizations or closely related ones have been studied in operations research, computational geometry and molecular biology applications. (See [1] for a list of references.) Alon et. al. [3] provided an efficient algorithm which finds an optimality sequence or determines that no such sequence exists for unrestricted problems. Dietrich and Shamir [8, 18, 19]) generalized the algorithm to restricted transportation problems. Adler, Hoffman and Shamir [1] characterized the bipartite graphs which admit a feasibility sequence in graph-theoretic terms and gave very efficient algorithms for recognizing them. They also extended the characterizations and algorithms for feasibility and optimality sequences to general, non-bipartite networks.

In this paper we introduce a requirement that the solution obtained from the greedy algorithm will also be *basic*. This requirement has several motivations: In many linear programming situations, the optimal solution of a problem is also required

to be basic. This is needed when dual information is required, or for post-optimality analysis. Such complete information on the problem is not always available from an optimal solution, unless it is also a dual feasible basis. For example, a solution to the dual problem may not be immediately available in that situation. In case the problem is infeasible, it may also be important to obtain a proof of infeasibility by pointing out a “bottleneck” which caused the infeasibility in the original problem. For example, methods using linear programming solvers as subroutines (or “oracles”), require a separating hyperplane in case the problem is infeasible (cf. [10]). How much then do we have to sacrifice in terms of complexity, and in terms of the size of the class of greedily solvable problems, in order to guarantee that we get a basis? As it turns out, by modest modifications of the algorithms, we can guarantee obtaining basic information without increasing the complexity and without decreasing the size of the class.

It is well known in linear programming theory that proper perturbation of the right-hand side and/or the objective function generates a problem in which the only optimal solution is also an optimal basis. However, it is interesting to show how to achieve an optimal basis directly, by equipping an optimization algorithm with an appropriate mechanism (typically some sort of tie breaker). For example, Bland [7] showed how to prevent cycling in the simplex method, by endowing the simplex algorithm with some combinatorial properties, instead of using the geometric ‘trick’ of perturbation. Similarly, for the greedy algorithm in the transportation problem, we construct here edge selection rules which guarantee *independently* that the final selected set is a basis, primal feasible and dual feasible. One immediate consequence of this setup is that imposing the intersection of these selection rules guarantees that the final set of edges is an optimal basis.

The “modular” approach outlined above yields additional results: Since these selection rules are independent, one can combine a subset of these rules with other rules so that the final selected set of edges will satisfy additional desired properties. For example, an additional rule which enforces a given signature leads the greedy algorithm (using an optimality sequence) to produce a dual feasible spanning tree with a prescribed signature. This type of analysis is used to obtain several additional results. Our main results are the following:

- In section 3 we define a very simple generic vertex elimination algorithm, which scans the edges in predetermined order and eliminates one vertex in each step. We show that with any edge order and on any graph, that algorithm generates a spanning forest. These results do not require the transportation and greedy algorithm setting, and may be of interest by themselves. Next we show that for bipartite graphs, if the edge order forms a feasibility sequence, then the generic algorithm forms a spanning tree and therefore a basis. Moreover, if the order is an optimality sequence, then the resulting basis is dual feasible.
- In section 4 we slightly modify the greedy algorithm so that it will have the

properties of the vertex elimination algorithm defined in section 3. We show that with an optimality sequence this new algorithm guarantees an optimal basis for every feasible problem.

- In section 5 we concentrate on the consequences on negative termination in the greedy algorithm: Whenever the algorithm uses a feasibility sequence and determines that a problem is infeasible, we can identify a “bottleneck” set of sources (or destinations) whose total supply exceeds the total demand of all their neighbors in the original problem.
- As a by-product, we show in section 6 that the tools developed here yield some interesting results on signatures of transportation problems. Recall that the signature of a tree on a bipartite graph is the vector of the degrees of the sources in that tree. We give an algorithm which when guided by a feasibility sequence, produces a spanning tree with a given signature or determines that the signature is invalid. The same algorithm, when guided by an optimality sequence, produces a dual feasible basis with that signature, if it is valid. Finally, we give new characterizations for feasibility and optimality sequences using signatures.

2 Preliminaries

Let $G = (I, J; E)$ be a bipartite graph whose two vertex sets (or *sides*) are I and J , where $E \subseteq I \times J$, $|I| = m$, $|J| = n$ and $|E| = p$. The vertices in I and J are numbered $1, \dots, m$ and $m + 1, \dots, m + n$, respectively. We assume throughout that graphs are undirected and connected. For convenience, ij and ji are used interchangeably to denote the edge between i and j . Denote $V = I \cup J$. The set $N(v) = \{j \in V | vj \in E\}$ is called the *neighborhood* of v , and $d_v = |N(v)|$ is called the *degree* of vertex v . v is called an *end-vertex* if $d_v = 1$.

For each edge $ij \in E$, a *cost* $C_{ij} \geq 0$ is assigned. Again, we use both C_{ij} and C_{ji} for the cost of the edge between i and j . Edge costs may also be represented by an $m \times n$ matrix, in which case we define $C_{ij} = \infty$ for each $ij \notin E$. The graph together with its edge-costs are called a *network* and are denoted $N = (G, C)$. Finally, for each vertex $v \in V$, a non-negative *excess* e_v is given. The bipartite network together with an excess vector specify the input to a transportation problem. The problem is called *restricted* if the underlying bipartite graph is incomplete.

The origin of the model is in planning of shipping, where the sets I and J correspond to sources and destinations, respectively. A commodity available at the sources must be shipped to satisfy demands at the destinations. C_{ij} is the cost of shipping each unit of the commodity on edge ij . e_v is the supply available at source v if $v \in I$, or the demand at destination v if $v \in J$. The *transportation problem* is to determine the amount x_{ij} to be shipped each edge ij , so that all supplied and demands are met

at minimum overall cost, i.e.,

$$\begin{aligned} \min \quad & \sum_{ij \in E} C_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{\{j|ij \in E\}} x_{ij} = e_i \quad i \in V \\ & x_{ij} \geq 0 \quad ij \in E \end{aligned} \quad (P(N, e))$$

We say that the excess vector e is *feasible* if $P(N, e)$ has a feasible solution, i.e., a solution in which all supplies and demands are satisfied. An obvious necessary condition for feasibility is that $\sum_{i \in I} e_i = \sum_{j \in J} e_j$. Given a spanning tree T in G , the (unique) solution of $\sum_{ij \in T} x_{ij} = e_i \quad i \in V, x_{ij} = 0 \quad ij \in E - T$ is called the *the primal basic solution associated with T* . (A spanning tree is sometimes called a *basis*, since the set of columns corresponding to its edges forms a maximal independent set in the coefficient matrix of the equations in $P(N, e)$.) If the primal basic solution is non-negative then it is called a feasible solution, and T is called *primal feasible*.

The dual of the transportation problem is defined as :

$$\begin{aligned} \max \quad & \sum_{i \in I} e_i u_i + \sum_{j \in J} e_j v_j \\ \text{s.t.} \quad & u_i + v_j \leq C_{ij} \quad ij \in E \end{aligned} \quad (D(N, e))$$

Given a spanning tree T in G , the (unique) solution of the system $u_i + v_j = C_{ij} \quad ij \in T$ is called the *the dual basic solution associated with T* . If the dual basic solution satisfies the rest of the inequalities in $D(N, e)$ then it is called dual feasible solution, and T is called *dual feasible*. A tree which is both primal and dual feasible is called *optimal*.

Let $S = (S_1, S_2, \dots, S_p)$ be a permutation of the edges. For a problem $P(N, e)$, the *greedy algorithm* maximizes each variable in turn, according to the order given in S . A formal description is the following. (We assume that initially $x_{ij} = 0$ for all $ij \in E$).

```

algorithm GREEDY( $S$ );
begin
  For  $i = 1, \dots, p$  do :
    begin pick the next edge  $S_i = rs$ .
       $x_{rs} \leftarrow \min\{e_r, e_s\}$ 
       $e_r \leftarrow e_r - x_{rs}$ 
       $e_s \leftarrow e_s - x_{rs}$ 
    end
    if  $e \neq 0$  then output "infeasible" else output  $x$ 
end

```

A permutation S of the edges is a *feasibility sequence* for the graph G if for every feasible excess vector e , GREEDY(S) provides a feasible solution to $P(N, e)$. S is

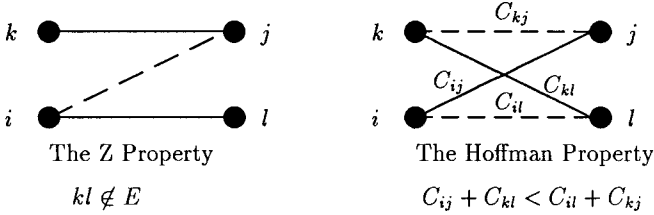


Figure 1: demonstration of the Z and Hoffman properties.

A broken edge cannot appear first in S among the edges in its figure.

called an *optimality sequence* for the network N if for every feasible excess vector e , algorithm GREEDY(S) provides an optimal solution to $P(N, e)$. For an order S of the edges, write $ij \overset{S}{\prec} kl$ if ij precedes kl in S . To simplify notation, we shall use $ij \prec kl$ when there is no ambiguity about the order. We say that S has the *Z property* for the graph G if the following holds:

The Z Property:

For every $i, k \in I$ and $j, l \in J$, if $ij \overset{S}{\prec} il$ and $ij \overset{S}{\prec} kj$ then $kl \in E$.

Note that the condition implicitly requires that ij , il and kj are edges, by the fact that they are included in S . An equivalent statement to that above is that if $kl \notin E$ then the order $li \overset{S}{\succ} ij \overset{S}{\prec} jk$ is impossible (see figure 1).

We say that S has the *Hoffman property* for the network N if the following property holds (see figure 1):

The Hoffman Property:

For every $i, k \in I$ and $j, l \in J$, such that ij , il , kj and kl are all edges in E , if $ij \overset{S}{\prec} il$ and $ij \overset{S}{\prec} kj$ then $C_{ij} + C_{kl} \leq C_{il} + C_{kj}$.

A sequence is called a *Monge sequence* for the network N if it satisfies both the Z property and the Hoffman property. Note that in a complete bipartite graph every permutation trivially satisfies the Z property, hence the Hoffman property and the Monge property are equivalent on unrestricted transportation networks.

The following theorem summarizes some of the known characterizations and complexity results for optimality and feasibility sequences in bipartite networks. (Recall that a bipartite graph is called chordal bipartite if it does not contain an induced cycle of length six or more [12].) For the complexity results we assume $m \leq n$:

Theorem 2.1

(a) [15, 19] S is an optimality sequence for a network if and only if it is a Monge sequence.

- (b) [19, 1] S is a feasibility sequence for G if and only if it has the Z property.
- (c) [1] A bipartite graph admits a feasibility sequence if and only if it is chordal bipartite.
- (d) [3, 19] In a bipartite network, one can construct an optimality sequence or determine that no such sequence exists in $O(pm \log n)$ steps.
- (e) [1] In a bipartite graph, one can construct a feasibility sequence or determine that no such sequence exists in $O(p \log n)$ steps.

3 Vertex Elimination and Spanning Trees

First, we consider vertex elimination in a general setting. A basic step which will be used in several algorithms in the sequel is eliminating a vertex together with all the edges incident on it from the graph $G(V, E)$. A formal description is the following:

```

procedure ELIMINATE( $v$ );
begin
     $V \leftarrow V - \{v\}$ 
     $E \leftarrow E - \{vj \in E \mid j \in N(v)\}$ 
end

```

In all the algorithm we shall discuss, we call a vertex *active* at a certain stage if it is in the current set of vertices V . An edge is called *active* if it is in the current set E , i. e., if both of its endpoints are active.

The generic algorithm described below builds a set of edges B which under certain conditions will be shown to form a spanning tree. Initially, B is empty. Part of the input of the algorithm is a permutation of the edges $S = (S_1, S_2, \dots, S_p)$, which is used to guide the algorithm: The algorithm examines the edges one at a time, according to their order in S . Whenever the examined edge is active, it is added to the set B and one of the vertices incident on it is eliminated. If exactly one vertex incident on the examined edge is an end-vertex, that vertex is eliminated. Otherwise, the eliminated vertex is chosen arbitrarily. This algorithm is described below:

```

algorithm GENERATE-B( $S$ );
begin
     $B \leftarrow \emptyset$ ;
    For  $i = 1, \dots, p$  do :
        begin pick the next edge  $S_i = rs$ .
            if  $rs$  is active then
                begin  $B \leftarrow B \cup \{rs\}$ 
                    if exactly one endpoint of  $rs$  is an end-vertex then call it  $v$ .
                    else pick either  $v \leftarrow r$  or  $v \leftarrow s$  endif
                    ELIMINATE( $v$ )
                end
            end
        end

```

end
end

This algorithm satisfies two elementary properties, which we call the *vertex elimination properties*:

- VE.1: For each edge added to B , one of its endpoints is eliminated and no other edge incident on that endpoint may be added later into B .
- VE.2: If an edge added to B is incident on at least one end-vertex, then one of its endpoints which is an end-vertex is eliminated.

The freedom to choose the permutation S and the eliminated vertex is quite large. We shall show in the sequel several ways of exploiting this freedom, in designing algorithms which carry out additional tasks while maintaining these vertex elimination properties.

Vertex and edge elimination schemes have been studied in the past, beginning with the work of Rose [17], in conjunction with efficient Gaussian elimination in sparse matrices (see [12, ch. 3 and 12] and the references thereof). In that context, vertex elimination procedures use a permutation of the *vertices*, and in each step one vertex is eliminated. In edge elimination, an ordering of a *subset of the edges* in sought, and *both endpoints* are always eliminated with the current edge. The vertex elimination properties defined here require that the algorithm eliminate *one* endpoint of a selected edge in each step, but be guided by an ordering of *all* the edges. The reason for this tighter definition will become clear in the following sections.

We first note some useful properties of the algorithm which are true for any ordering S of the edges.

Lemma 3.1 *Let B be a set of edges generated by algorithm GENERATE- B . If $(i_1i_2, i_2i_3, \dots, i_{k-1}i_k)$ is a path of edges in B , then either $i_1i_2 \prec i_r i_{r+1}$ for $r = 2, \dots, k-1$, or $i_{k-1}i_k \prec i_r i_{r+1}$ for $r = 1, \dots, k-2$.*

Proof. Assume that $i_t i_{t+1}$ is the first among the edges in the path to appear in S , and $1 < t < k-1$. Then on the same step when that edge was introduced into B , by property VE.1, either i_t or i_{t+1} became inactive. In the former case, $i_{t-1}i_t$ could not be added later to B . In the latter, $i_{t+1}i_{t+2}$ could not be added later to B . In both cases we obtain a contradiction. ■

Corollary 3.2 *If B was generated by algorithm GENERATE- B , then for every path of edges in B , the order of appearance of these edges in S is unimodal. That is, if the path is $(i_1i_1, i_2i_3, \dots, i_{k-1}i_k)$, then for some r , $1 \leq r \leq k-1$,*

$$i_1i_2 \prec i_2i_3 \prec \dots \prec i_{r-1}i_r \prec i_r i_{r+1} \succ i_{r+1}i_{r+2} \succ \dots \succ i_{k-1}i_k.$$

Proof. Apply Lemma 3.1 to subpaths of that path. ■

Corollary 3.3 *The set B generated by the algorithm is acyclic. ■*

Note that for the above only the first vertex elimination property (VE.1) was required. Note also that in every stage of the algorithm, every connected component of B contains exactly one active vertex.

So far we did not require the bipartiteness of the graph. We shall use the results above for graphs of the transportation problems, which are bipartite. Generating a spanning forest or a spanning tree in any graph is an easy, well-studied problem. It is the additional requirements from the spanning tree (e.g., dual feasibility, optimality or correspondence to a given signature) which make the problem of interest to us.

In general, the final set B generated by the above algorithm may contain several connected components. We now show that if S is a feasibility sequence, the final set B will be a spanning tree. The proof relies on the following lemma:

Lemma 3.4 *Let S be a feasibility sequence for G . Algorithm GENERATE- $B(S)$ maintains the following invariants: (i) the set E is connected, (ii) the set $B \cup E$ is connected, and (iii) the set $B \cup E$ spans the original graph.*

Proof. The fact that in each step $B \cup E$ spans the original graph is immediate from the algorithm. The proof of (i) and (ii) is inductive. We give here the proof of (i) only, since the proof of (ii) is similar. Originally G is connected. Assume that E is connected before the insertion of edge rs into B . Denote the set of active edges after that step by E' . We need to prove that E' is connected. If $d_r = 1$ or $d_s = 1$, by property VE.2, $E' = E - \{rs\}$, so the statement is clearly true. Otherwise the degrees of both endpoints are at least two.

Let $q \neq s$ and $t \neq r$ be such that $rq \in E$ and $ts \in E$. Since $rs \prec rq$, $rs \prec ts$, by the Z property of sequence S , $tq \in E$. More generally, by the Z property, the set $N(r) \cup N(s)$ induces a complete bipartite graph in E . Without loss of generality suppose r is eliminated. Then $N(s) - \{r\} \neq \emptyset$, and the set $N(r) \cup N(s) - \{r\}$ induces a complete bipartite graph in E' . If E' is not connected, then since E was connected, and since all the edges which were inactivated in that step were incident on r , each component of E' must contain a vertex in $N(r)$. But any two neighbors of r are also neighbors of t , so they remain connected in E' , a contradiction. ■

Proposition 3.5 *If S is a feasibility sequence, then the final set B produced by algorithm GENERATE- $B(S)$ is a spanning tree.*

Proof. When the algorithm terminates, $E = \emptyset$. Hence by Lemma 3.4, B is connected and spans the original graph. By Corollary 3.3 B is cycle-free. Hence it is a spanning tree. ■

Let us now show that in a sense, the converse of Proposition 3.5 also holds: Call a permutation S a *spanning tree sequence* for G if for every induced connected subgraph $G'(V', E')$ of G , algorithm GENERATE- $B(S)$ operating on G' generates a spanning

tree. (The algorithm can be viewed as operating on G while considering all vertices which are not in V' as inactive from the beginning. An equivalent interpretation is that the induced subsequence of S which is obtained by restricting S to edges in E' is used on G' .)

Proposition 3.6 *S is a spanning tree sequence for G if and only if it is a feasibility sequence for G .*

Proof. If S is a feasibility sequence then for every induced subgraph G' , the corresponding induced subsequence of S is also a feasibility sequence on G' , by the Z property. Hence by Proposition 3.5, S is a spanning tree sequence. If S is not a feasibility sequence, then there exist some i, j, k, l such that $kl \notin E$ and $kj \succ ji \prec il$ (cf. figure 1). On the subgraph induced by those four vertices, S does not form a spanning tree. ■

We now show that when S is a Monge sequence, the spanning tree generated by the algorithm is also dual feasible:

Theorem 3.7 *If S is a Monge sequence for N , GENERATE- $B(S)$ produces a dual feasible spanning tree.*

Proof. Since every Monge sequence is a feasibility sequence, by Proposition 3.5, B is a spanning tree. To show it is dual feasible, it suffices to prove that no cycle which is formed by adding a non-basic edge to B is a negative cost cycle (see, e.g., [2]). That is, if $i_k i_{k+1} \in B$ $k = 1, \dots, 2l - 1$, and $i_{2l} i_1 \in E - B$,

$$C_{i_1 i_2} + C_{i_3 i_4} + \dots + C_{i_{2l-1} i_{2l}} \leq C_{i_2 i_3} + C_{i_4 i_5} + \dots + C_{i_{2l} i_1}. \quad (3.1)$$

The proof is by induction on l : For $l = 2$, let the cycle be (st, tu, ur, rs) where rs is the non-tree edge. Applying lemma 3.1 to the path (st, tu, ur) we get that tu cannot precede both st and ur in S . Clearly, rs cannot precede both st and ur , since otherwise rs would have been in B . Hence by the Hoffman property, $C_{st} + C_{ur} \leq C_{rs} + C_{tu}$, and equation (3.1) is satisfied.

Assume now that the equation (3.1) is true for all cycles of length up to $2l - 2$, and let $(i_1 i_2, i_2 i_3, \dots, i_{2l-1} i_{2l}, i_{2l} i_1)$ be a cycle of $2l$ edges, where only $i_{2l} i_1 \notin B$ (see figure 2). Using Lemma 3.1 again, we can assume without loss of generality that $i_1 i_2$ appeared first in S among the cycle edges. By property Z of the Monge sequence this implies that $i_{2l} i_3 \in E$. Moreover, by the Hoffman property,

$$C_{i_1 i_2} + C_{i_{2l} i_3} \leq C_{i_2 i_3} + C_{i_{2l} i_1}. \quad (3.2)$$

Using the induction hypothesis for the cycle $(i_3 i_4, i_4 i_5, \dots, i_{2l-1} i_{2l}, i_{2l} i_3)$ whose length is $2l - 2$, we get that

$$C_{i_3 i_4} + C_{i_5 i_6} + \dots + C_{i_{2l-1} i_{2l}} \leq C_{i_4 i_5} + C_{i_6 i_7} + \dots + C_{i_{2l} i_3}. \quad (3.3)$$

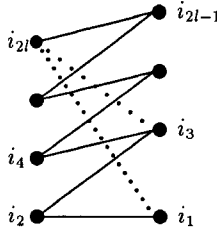


Figure 2: diagram for the proof of Theorem 3.7.
Solid lines: edges in B . Dotted lines: edges in $E - B$.

Summing equations (3.2) and (3.3) together, we obtain the desired proof of (3.1) for l . ■

We can again show that the above condition leads to a new characterization of Monge sequences, in a fashion similar to Proposition 3.6: Call a sequence a *dual feasibility sequence* for N if GENERATE-B(S) produces a dual feasible tree on every induced connected subnetwork.

Proposition 3.8 S is a dual feasibility sequence for N if and only if it is an optimality sequence for G .

Proof. Analogous to the proof of Proposition 3.6. ■

4 Generating an Optimal Spanning Tree

The algorithm above used only the graph structure and the order S to select a subset of the edges which forms a spanning tree. We now describe an algorithm which also receives as input the excesses at the vertices, and attempts to generate a feasible or optimal solution to the transportation problem. The algorithm greedily maximizes the flow on each active edge in turn, according to the order S , and eliminates one of its two endpoints. If one endpoint has strictly smaller excess than that vertex is eliminated. If both have equal excess and one is an end-vertex, then that vertex is eliminated. Otherwise, (if excesses at the two endpoints are equal, and either none is an end-vertex or both are,) we can choose the eliminated vertex arbitrarily. Infeasibility is detected whenever a vertex with positive excess becomes isolated. The algorithm is described below:

```

algorithm GREEDY*( $S$ );
begin
   $B \leftarrow \emptyset$ ;  $k \leftarrow 0$ ;

```

```

For  $i = 1, \dots, p$  do :
begin pick the next edge  $S_i = rs$ .
  if  $rs$  is inactive then return
  else begin  $k \leftarrow k + 1$  /* step  $k$  begins */
    if one endpoint has less excess, say  $e_r < e_s$  then
begin ELIMINATE( $r$ ); if  $d_s = 0$  then output "infeasibility" and stop end
    else ( $e_r = e_s$ )
begin if  $d_r = 1$  then ELIMINATE( $r$ ) else ELIMINATE( $s$ ) end
       $B \leftarrow B \cup \{rs\}$ 
       $x_{rs} \leftarrow \min\{e_r, e_s\}$ 
       $e_r \leftarrow e_r - x_{rs}$ 
       $e_s \leftarrow e_s - x_{rs}$ 
    end
  end
end
end

```

Successful termination occurs when step $k = m + n - 1$ is completed without detecting infeasibility. In that case a single vertex remains active and has excess zero.

The above algorithm is a specialization of algorithm GREEDY which was described in section 2, modified so that it will also have the vertex elimination properties. In other words, it has both the properties of the greedy algorithm and of algorithm GENERATE-B. An immediate consequence of this, using Theorem 2.1(b) and Proposition 3.5, is the following:

Proposition 4.1 *If the sequence S satisfies the Z property, then for every feasible problem, algorithm GREEDY*(S) produces a primal feasible basic solution and its associated spanning tree B . ■*

From Proposition 4.1 and Theorem 3.7 we get the final result of this section:

Theorem 4.2 *If S is a Monge sequence, then for feasible problems, algorithm GREEDY*(S) terminates with an optimal basic solution and an optimal spanning tree B . ■*

As mentioned in section 2, given the optimal tree B , one can also construct the corresponding dual basic solution, by solving a system of linear equations. Note that the converse of Proposition 4.1 and Theorem 4.2 are already known to be true, by [1] and [19], respectively.

5 Gale Certificates

In this section we discuss the consequences of terminating the algorithm of Section 4 with the output 'infeasible'. For a set of vertices $S \in V$, let $e(S) = \sum_{v \in S} e_v$ be its

total excess. If there exists a set of sources (or destinations) whose total supply (resp., demand) exceeds the total demand (resp., supply) of all their neighbors then the problem is obviously infeasible. Gale [9] has shown that the converse is also true, i.e., whenever the transportation problem is infeasible, there exists a set $S \in I$ (or $S \in J$) such that $e(S) > e(N(S))$. (This is a generalization of the celebrated Hall's theorem for the assignment problem[14].) Therefore, finding such a set S can be viewed as a proof of infeasibility. Let us call such a set S a *Gale certificate* of infeasibility. We shall show that whenever the greedy algorithm is guided by a feasibility sequence and terminates with the answer "infeasible", a Gale certificate can be pointed out.

For the following proposition we need some notation which refers to intermediate steps of the algorithm. Let $G^k = (V^k, E^k)$ be the graph after step k of the algorithm, and let e_v^k and d_v^k be the excess and the degree, respectively, of vertex v in that stage. $G = G^0$ is the original graph. We can view the problem $P^k = ((G^k, C), e^k)$ as a new transportation problem starting with the modified data after step k . Let B^k be the final set B produced by an algorithm operating on problem P^k as the initial problem. Note that for each k , step i of the algorithm on P^k is identical to step $k + i$ on the original problem. In particular, if a^k is the edge introduced into B in step k by the algorithm operating on the original problem P^0 , then $B^k = B^{k+1} \cup \{a^k\}$. Moreover, infeasibility is detected at the same vertex in all these nested problems. Suppose the algorithm terminated after determining infeasibility at vertex w , in step l of the original problem. That is, $d_w^l = 0$ and $e_w^l > 0$. Without loss of generality assume $w \in I$. Define

$$R^k = \{u \in I \mid \exists \text{ a path in } B^k \text{ from } w \text{ to } u\}$$

$$L^k = \{u \in J \mid \exists v \in R^k \text{ such that } vu \in E^k\}$$

In other words, R^k is the set of vertices in I which are in the same connected component of B^k with w , and L^k is the set of their neighbors in G^k . Finally, let $D^k = R^k \cup L^k$.

Proposition 5.1 *Suppose S is a feasibility sequence for G . Then with the definitions above, for $k = 0, \dots, l$:*

- (i) R^k is a Gale certificate for P^k .
- (ii) For every $v \in D^k$, there exists a path from v to w containing only edges from B^k .

Proof. The proof is by backwards induction on $k = l, l-1, \dots, 0$. For the induction basis, note that after the last step l , $d_w^l = 0$ and $e_w^l > 0$. Hence $R^l = \{w\}$ is a Gale certificate, and $D^l = \{w\}$, so (ii) is trivially true.

We now show the inductive step. Suppose (i) and (ii) hold for k , and let us prove them for $k-1$. Denote the edge introduced into B in step k by $a^k = rt$, where r is the vertex eliminated in step k . Recall that we assumed that $w \in I$. We distinguish four cases (compare figure 3):

Case I: $r \in I$, $t \in L^k$. Since $t \in L^k$, by the induction hypothesis there exists $p \in R^k$ such that $pt \in B^k$, so rt belongs to the same component as w in B^{k-1} . Is it possible that in G^{k-1} , vertex r has a neighbor which is not in L^k ? Suppose $rj \in E^{k-1}$.

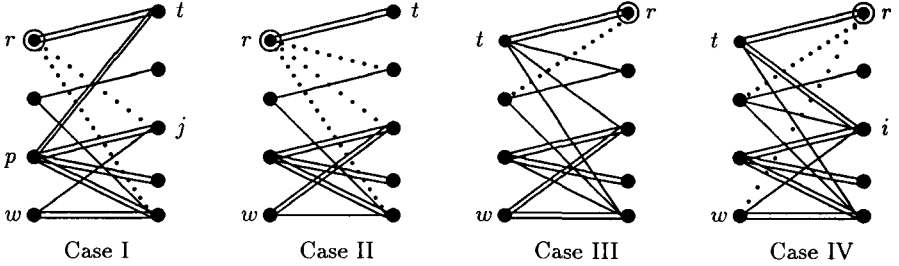


Figure 3: diagrams for the proof of the existence of a Gale certificate.

In all cases r is the eliminated vertex, and dotted edges are eliminated with it. Edges included in the final set B are marked by double lines. Only vertices which are used in the proof are labeled.

Since j is active in step $k-1$, we get $rj \succ rt \prec pt$, so by the Z property of the sequence S , $pj \in E^{k-1}$. But then also $pj \in E^k$, so $j \in L^k$. Hence all neighbors of r in G^{k-1} are also present in G^k , which implies $D^{k-1} = D^k \cup \{rt\}$. This immediately implies (ii). (i) follows since

$$e^{k-1}(R^{k-1}) = e^k(R^k) + e_r^{k-1} > e^k(L^k) + e_r^{k-1} = e^{k-1}(L^{k-1}), \quad (5.1)$$

where the inequality follows from the inductive hypothesis.

Case II: $r \in I$, $t \notin L^k$. Here we want to show that $R^k = R^{k-1}$, i.e., rt is not in the connected component of w in B^{k-1} . This will imply (i) and (ii). Suppose there exists a path in B^{k-1} from r to w . Such path must contain an edge $it \in B^{k-1}$ such that i is connected to w in B^{k-1} , since r is eliminated in step $k-1$. But then $it \in B^k$ and $i \in R^k$, so $t \in L^k$, a contradiction.

Case III: $r \in J$, $t \notin R^k$. In that case $R^k = R^{k-1}$, and $r \notin L^k$, since $r \notin V^k$. (i) and (ii) are thus immediate by the inductive hypothesis.

Case IV: $r \in J$, $t \in R^k$. By the inductive hypothesis, t belongs to the same component as w in B^k . Hence in B^{k-1} , both r and t belong to that component, which implies that there exists an edge ti on the path from t to w in B^{k-1} , and that edge is also present in B^k . Since every other neighbor of t (except r) is also in V^k , by the inductive hypothesis it is also in L^k . Hence (ii) is true. (i) now follows by an argument identical to the chain of inequalities (5.1) in case I. ■

Applying the proposition with $k=0$ we get the desired result:

Theorem 5.2 *If S is a feasibility sequence and algorithm GREEDY* detects infeasibility at vertex w , then the vertices in w 's connected component of B which are on the same side as w form a Gale certificate of infeasibility. ■*

6 Signatures

Let B be a subset of the edges in the bipartite graph $G = (I, J; E)$. In this section it will be more convenient to denote edges by ordered pairs of vertices, i.e., for the edge $(r, s) \in E$, $r \in I$ and $s \in J$. The (row) signature of B is defined as an integer vector $\sigma^B = (\sigma_1^B, \dots, \sigma_m^B)$ such that

$$\sigma_i^B = |\{k | (i, k) \in B\}| \quad i = 1, \dots, m. \quad (6.1)$$

The column signature is defined analogously. We shall discuss row signatures here, but the same results apply to column signatures, by interchanging the roles of I and J .

For a bipartite graph an integer vector $\sigma = (\sigma_1, \dots, \sigma_m)$ is called a *valid signature* if there exists a set B which forms a spanning tree and $\sigma = \sigma^B$, i.e., for each vertex $i \in I$ its degree in the subgraph (V, B) is σ_i . In particular, only connected graphs have valid signatures, and every valid signature vector satisfies $\sigma > 0$ and $\sum_{i=1}^m \sigma_i = m + n - 1$. (For complete bipartite graphs, these conditions are sufficient for validity.) Balinsky and Russakoff have shown that in unrestricted problems, for every valid signature vector σ , there exists at least one dual feasible spanning tree B with that signature [6] (see also [4, 5]). Indeed, under dual non-degeneracy, they showed that there exists one to one correspondence between dual feasible bases and valid signatures. In this section, we shall show how to use vertex elimination and feasibility sequences in order to determine quickly if a given signature is valid. Subsequently, we shall show that with a Monge sequence, one can also find a dual feasible spanning tree with that signature, if one exists. We shall also provide new characterizations based on signatures for feasibility and optimality sequences.

A preliminary question is whether one can efficiently decide if a given signature is valid. More precisely, suppose we are given a bipartite graph and a signature σ , and asked whether there exist a spanning tree with that signature. We now show that this problem is polynomial, and also give a polynomial algorithm which constructs a spanning tree with the given signature if it is valid. Clearly, the conditions $\sigma > 0$ and $\sum_{i=1}^m \sigma_i = m + n - 1$ are necessary. For complete bipartite graphs these conditions are also sufficient, so the question is trivial. For an arbitrary bipartite graph, having verified the necessary conditions, we can rephrase the question:

Problem 1:

Given σ , does there exist an independent (cycle-free) set of edges D with $\sigma^D = \sigma$?

Let us first discuss a slightly weaker question:

Problem 2:

Given σ , find a set of edges D of maximum cardinality satisfying (a) D is independent, and (b) $\sigma^D \leq \sigma$.

It is easy to verify that the collection of edge-sets which satisfies (a) forms a matroid, and so does the collections of edge-sets which satisfies (b). Hence problem 2 is that of finding a maximum cardinality set in the intersection of two matroids. This problem is known to be polynomially solvable (see, e.g., [16]). A solution D to problem 2 gives also the answer to problem 1: If $\sigma^D = \sigma$, the answer is positive. Otherwise, it is impossible to achieve a valid signature, since its edge-set must also satisfy (a) and (b), and have a larger cardinality than D .

We first describe an algorithm which receives as input a signature vector σ in addition to the graph and a permutation S of the edges, and tries to construct a spanning tree B with that signature. We assume that the input signature is positive, otherwise it is trivially invalid.

```

algorithm SIGNATURE-B( $\sigma, S$ );
begin  $B \leftarrow \emptyset$ ;  $\sigma_i^B \leftarrow 0, i = 1, \dots, m$ ;
      For  $i = 1, \dots, p$  do :
        begin pick the next edge  $S_i = (r, s)$ .
          if  $(r, s)$  is active then
            begin  $B \leftarrow B \cup \{(r, s)\}$ ;  $\sigma_r^B \leftarrow \sigma_r^B + 1$ ;
              if  $d_s = 1$  then ELIMINATE( $s$ )
              else if  $d_r = 1$  or  $\sigma_r^B = \sigma_r$  then ELIMINATE( $r$ )
              else ( $d_r > 1, d_s > 1, \sigma_r^B < \sigma_r$ ) ELIMINATE( $s$ )
            end
          end
        end
      if  $\sigma \neq \sigma^B$  then output "failure" else output  $B$ .
end

```

The algorithm terminates either with a set of edges B corresponding to the given signature, or with the answer "failure". Note that termination with failure can be done earlier, by stopping the algorithm when a vertex r is eliminated because $d_r = 1$ but still $\sigma_r^B < \sigma_r$.

Observe that the algorithm eliminates one vertex in every step, and it chooses to eliminate an end-vertex if such a vertex is incident on the currently examined edge. This immediately implies:

Lemma 6.1 *Algorithm SIGNATURE-B satisfies the vertex elimination properties VE.1 and VE.2. ■*

Theorem 6.2 *If S is a feasibility sequence then algorithm SIGNATURE-B(σ, S) determines if the given vector σ is a valid signature, and constructs a spanning tree B with that signature if one exists.*

Proof. The fact that the algorithm constructs a spanning tree follows from Lemma 6.1 and Proposition 3.5. If the algorithm terminates successfully, it follows from the algorithm that B has the correct signature. It remains to show that the algorithm never

terminates with failure for a valid signature. To prove this, it suffices to show that by performing the first step of the algorithm we do not lose the validity of the signature. Suppose G has a spanning tree T with signature σ , and $a = (r, s)$ is the first edge in S . In the first step of the algorithm, one of the endpoints of a was eliminated. Let G' be the graph after the first step. We claim that there exists a set of edges B' in G' so that $B' \cup \{a\}$ is a spanning tree on G with the required signature. To show that, let $\sigma'_r = \sigma_r - 1$, $\sigma'_i = \sigma_i, i \neq r$ be the modified signature prescribed for G' . We shall prove that σ' is indeed valid for G' .

If $d_r = 1$ or $d_s = 1$ then clearly $T - \{a\}$ is a spanning tree on G' with signature σ' , so the claim follows. Suppose $r \in I$ was the eliminated vertex in that step, $d_r > 1$ and $d_s > 1$. This can happen only if $\sigma_r = 1$, and in that case again $T - \{a\}$ has the desired properties. The only remaining case is when $s \in J$ was the eliminated vertex, $\sigma_s > 1$, $d_r > 1$ and $d_s > 1$.

Let us first show that there exists a spanning tree T' in G with signature σ which includes (r, s) . If $(r, s) \notin T$ then $T \cup \{(r, s)\}$ contains a unique cycle. In that cycle there exists an edge (r, t) , $t \neq s$. The set $T' = T \cup \{(r, s)\} - \{(r, t)\}$ has the same signature as T , it contains (r, s) and is also a spanning tree for G .

Next, suppose $(r, s) \in T$ and the spanning tree T contains additional edges incident on the vertex s . In that case, since the algorithm removes all those edges during the first step, we should show that there is another spanning tree with the same signature but without those edges. Let $(j, s) \in T$ be such an edge. Since $\sigma_r > 1$, there is some other edge $(r, k) \in T, r \neq s$. Since $(r, s) \prec (r, k)$ and $(r, s) \prec (j, s)$, by property Z of the sequence S , $(j, k) \in E$. Moreover, $(j, k) \notin T$, or else the four edges would have formed a cycle in T . Let $T^* = T \cup \{(j, k)\} - \{(j, s)\}$. The set T^* has the same signature as T , is still a spanning tree for G and has one less edge incident on s . By repeating the argument (with T^* replacing T) if necessary, we conclude that there exists a spanning tree T with signature σ such that (r, s) is the only edge it contains which is incident on s . Therefore, $T - \{(r, s)\}$ is a spanning tree in G' with signature σ' . ■

To prove the converse, we need the following lemma which shows that it is possible to extend a valid signature by “going backwards in time” and adding rather than eliminating vertices. Define the *residual signature* at any stage of the algorithm to be $\sigma - \sigma^B$. Namely, this is the signature that must be completed on the remaining subgraph at that stage in order to satisfy the original signature requirement.

Lemma 6.3 *Let G be a bipartite graph, and let S be any permutation of its edges where $S_1 = (i, j)$. Let G' be the subgraph induced by eliminating either i or j . If G' admits a valid signature σ' , then there exists a valid signature σ on G such that after the first step of SIGNATURE-B(σ, S), the residual signature on G' is σ' .*

Proof. By the assumptions of the lemma there exists a spanning tree T' in G' with the valid signature σ' . If i is eliminated, define $\sigma_i = 1$ and $\sigma_k = \sigma'_k$ for $k \neq i$. If

j is eliminated, define $\sigma_i = \sigma'_i + 1$ and $\sigma_k = \sigma'_k$ for $k \neq i$. In both cases, the set $T = T' \cup \{(i, j)\}$ forms a spanning tree in G , which satisfies the requirements of the lemma. ■

Theorem 6.4 *Let S be such that for every valid signature σ , SIGNATURE-B(σ, S) generates a spanning tree with that signature. Then S is a feasibility sequence.*

Proof. Suppose S is not a feasibility sequence. Let $(i, j) \in E$ be the first edge in the sequence S which violates the Z property. In particular, there are vertices k, l such that $(i, l) \in E$, $(k, j) \in E$, $(i, j) \stackrel{S}{\prec} (i, l)$ and $(i, j) \stackrel{S}{\prec} (k, j)$ and $(k, l) \notin E$.

If (i, j) is the first edge in S , define $R_k = \{(k, s) | s \in N(k)\}$, $\delta_k = |R_k|$, $R_i = \{(i, s) | s \in N(i) - N(k)\}$, $\delta_i = |R_i| + 1$. The set $R = R_k \cup R_i$ is cycle free, hence it can be extended to a spanning tree with a valid signature σ , satisfying $\sigma_i = \delta_i$, $\sigma_k = \delta_k$. Moreover, any such spanning tree T must contain all the edges in R_k . If in addition $(i, j) \in T$, then no other edge (i, s) with $s \in N(k)$ can be included in T , since it would form a cycle. Hence every spanning tree T which satisfies $\sigma_i = \delta_i$, $\sigma_k = \delta_k$, and $(i, j) \in T$ must contain all edges in R_k and R_i . Since in the first step the algorithm introduces (i, j) into the spanning tree and eliminates one of its endpoints, it is impossible that both $(i, l) \in R_i$ and $(k, j) \in R_k$ will be included later in T , a contradiction.

Assume now that (i, j) is the $(t+1)$ -st edge in the sequence S , $t \geq 1$. Suppose we use the algorithm GENERATE-B on the subsequence S_1, \dots, S_t , temporarily ignoring signature requirements, but making sure that none of the vertices i, j, k, l is eliminated. This can be guaranteed since none of those vertices is an end vertex incident on an edge in S_r for $r \leq t$. Let E^t be the set of active edges produced by the algorithm after step t . Since the subsequence S_1, \dots, S_t satisfies property Z, by Lemma 3.4, E^t is connected. Hence the same argument as in the first part of this proof implies that in the active graph after step t , there is a valid signature σ^t for which the algorithm fails. Lemma 6.3 implies that σ^t can be extended to a valid signature σ in the original graph, so that algorithm SIGNATURE-B(σ, S) leads to the residual signature σ^t after stage t . Hence the algorithm fails for the valid signature σ . ■

Theorem 6.5 *If S is a Monge sequence and σ is a valid signature, then the set B generated by SIGNATURE-B(σ, S) is a dual feasible spanning tree with signature σ .*

Proof. By Lemma 6.1, SIGNATURE-B is a special case of algorithm GENERATE-B. The result thus follows from Theorem 6.2 and Theorem 3.7. ■

Let us now show that the converse of Theorem 6.5 is also true:

Theorem 6.6 *Let S be such that for every valid signature σ , SIGNATURE-B(σ, S) generates a dual feasible spanning tree with that signature. Then S is a Monge sequence.*

Proof. By Theorem 6.4, S must be a feasibility sequence. Suppose it does not have the Hoffman property. Then there exist edges $(i, j), (i, l), (k, j), (k, l)$ such that

$$(i, j) \prec (i, l), (i, j) \prec (k, l), (i, j) \prec (k, j) \quad (6.2)$$

and

$$C_{ij} + C_{kl} > C_{il} + C_{kj}. \quad (6.3)$$

Let (i, j) be the first edge in the order S which satisfies conditions (6.2) and (6.3). Define \overline{G} to be the same graph without edge (k, l) . Note that S does not satisfy the Z property on \overline{G} , by (6.2). Let σ be the signature generated on \overline{G} by the process described in the proof of Theorem 6.4. Since σ is valid for \overline{G} , it is also a valid signature for the original graph G . By the proof of Theorem 6.4, algorithm SIGNATURE-B(σ, S) must fail on \overline{G} .

Consider the operation of algorithm SIGNATURE-B(σ, S) on G . If the algorithm fails, then we get the desired contradiction. Suppose the algorithm succeeds. When edge (i, j) is examined by the algorithm, it is still active and thus is included in B . Moreover, edge (k, l) must be also included in B , or else the final set B will form a spanning tree to \overline{G} with signature σ , contradicting Theorem 6.4. Let $(\overline{u}, \overline{v})$ be the dual solution vector corresponding to B . Since $(i, j) \in B$ and $(k, l) \in B$, this solution must satisfy:

$$\overline{u}_i + \overline{v}_j = C_{ij} \quad \overline{u}_k + \overline{v}_l = C_{kl}$$

Substituting the above equations into (6.3), we obtain:

$$\overline{u}_i + \overline{v}_j + \overline{u}_k + \overline{v}_l > C_{il} + C_{kj}$$

or, equivalently,

$$(\overline{u}_i + \overline{v}_l - C_{il}) + (\overline{u}_k + \overline{v}_j - C_{kj}) > 0.$$

Hence, at least one of the values in parentheses must be strictly positive, in contradiction to the dual feasibility of $(\overline{u}, \overline{v})$. ■

Theorems 6.5 and 6.6 together give another characterization to Monge sequences, analogous to the primal one: Call S a *dual feasibility sequence* for C if for every feasible signature σ , SIGNATURE-B(σ, S) generates a dual feasible basis with that signature.

Corollary 6.7 S is a Monge sequence for C if and only if it is a dual feasibility sequence for C . ■

7 Summary

This paper brings together two previously studied topics: vertex elimination, and greedily solvable bipartite flow problems. A concept which, somewhat surprisingly,

turns out to be relevant to both, is the feasibility sequence. We have defined a simple vertex elimination algorithm guided by an edge order, and have shown that with a feasibility sequence, the algorithm generates a spanning tree. Moreover, only a feasibility sequence guarantees that the algorithm will generate a spanning tree for every induced subgraph. When we consider also the edge costs, then with a Monge sequence the algorithm generates a dual feasible spanning tree, and again, Monge sequences are the only sequences that guarantee this for every induced subgraph.

When supplies and demands are also introduced, we have a complete transportation problem. By combining the properties of the greedy algorithm and the vertex elimination algorithm, we have proved that with a Monge sequence, the hybrid algorithm generates an optimal basic solution and its corresponding optimal spanning tree, for every feasible supply and demand function. When that function is infeasible, the algorithm produces a Hall-type certificate of infeasibility.

Finally, we have considered spanning trees with given signatures. By adjusting the vertex elimination algorithm to take into consideration a given signature, we have shown that with a feasibility sequence, the algorithm generates a spanning tree with the input signature whenever that signature is valid (i.e., corresponds to a spanning tree). Moreover, with any other sequence, this is not the case. As we noted, building a spanning tree with a given signature can be done polynomially even without a feasibility sequence, using matroid intersection.

Several interesting problems arise from our discussions. First, given a signature, how can one find a dual feasible spanning tree which corresponds to it, or determine that none exist? We have shown how to do this only in case a Monge sequence exists. Second, if the signature algorithm terminates with infeasibility, can one give a certificate that shows that the signature is invalid, in a fashion similar to what was done in section 5? Finally, can the results of this paper be generalized to non-bipartite problems?

References

- [1] I. Adler, A. J. Hoffman, and R. Shamir, Monge and Feasibility Sequences in General Flow Problems, Technical report, RUTCOR, Rutgers University, (1990). To appear in *Discrete Mathematics*.
- [2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Network Flows, *Handbooks in Operations Research and Management Science*, Vol. I, G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. S. Todd (editors), (Elsevier, Amsterdam, 1989) p. 211–369.
- [3] N. Alon, S. Cosares, D. S. Hochbaum and R. Shamir, An algorithm for the detection and construction of Monge sequences, *Linear Algebra and Its Applications*, **114/115** (1989) 669–680.

- [4] M. L. Balinsky, The Hirsch conjecture for dual transportation polyhedra, *Mathematics of Operations Research*, **9(4)** (1984) 629–633.
- [5] M. L. Balinsky, Signature methods for the assignment problem, *Operations Research*, **33(3)** (1985) 527–536.
- [6] M. L. Balinsky and A. Russakoff, Faces of dual transportation polyhedra, *Mathematical Programming Study*, **22** (1984) 1–8.
- [7] R. G. Bland, New finite pivoting rules for the simplex method, *Mathematics of Operations Research* **2(2)**, (1977) 103–107.
- [8] B.L. Dietrich, Monge sequences, antimatroids, and the transportation problem with forbidden arcs, *Linear Algebra and Its Applications*, **139** (1990) 133–145.
- [9] D. Gale, A theorem on flows in networks, *Pacific Journal of Mathematics*, **7** (1957) 1073–1082.
- [10] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization* (Springer, Berlin, 1988).
- [11] A. V. Goldberg, É. Tardos, and R. E. Tarjan, Network Flow Algorithms, *Paths, Flows and VLSI-Layout*, B. Korte, L. Lovász and A. Schrijver (editors), (Springer-Verlag, Berlin 1990) p. 101–164.
- [12] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. (Academic Press, New York, 1980).
- [13] G. Hadley, *Linear Programming*. (Addison-Wesley, Reading, Massachusetts, 1962).
- [14] P. Hall, On representatives of subsets, *J. London Math. Soc.*, **10** (1935) 26–30.
- [15] A.J. Hoffman, On simple linear programming problems, in *Convexity: Proceedings of symposia in Pure Mathematics*, Vol 7, (V. Klee, Ed.), (American Mathematical Society, 1963).
- [16] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*. (Holt, Reinhart and Winston, New York, 1976).
- [17] D. J. Rose, Triangulated graphs and the elimination process, *J. Math. Anal. Appl.*, **32** (1970) 597–609.
- [18] R. Shamir, A fast algorithm for constructing Monge sequences in transportation problems with forbidden arcs, Report 136/89 (1989), Tel Aviv University. To appear in *Discrete Mathematics*.

- [19] R. Shamir and B. Dietrich, Characterization and algorithms for greedily solvable transportation problems, *Proceedings of the first ACM/SIAM Symposium on Discrete Algorithms*, (SIAM, Philadelphia, 1990) p. 358–366.