The Algebraic Monge Property and Path Problems

Wolfgang Bein

School of Computer Science, University of Nevada, Las Vegas, NV 89154 [1]

Peter Brucker

Fachbereich Mathematik/Informatik, Universität Osnabrück, D-49069 Osnabrück, Germany

Lawrence L. Larmore

School of Computer Science, University of Nevada, Las Vegas, NV 89154 [1]

James K. Park

Sandia National Laboratories, Albuquerque, NM 87185 [2]

Abstract

We give algorithmic results for combinatorial problems with cost arrays possessing certain algebraic Monge properties. We extend Monge-array results for two shortest path problems to a general algebraic setting, with values in an ordered commutative semigroup, if the semigroup operator is strictly compatible with the order relation.

We show how our algorithms can be modified to solve bottleneck shortest path problems, even though strict compatibility does not hold in that case. For example, we give a linear time algorithm for the unrestricted shortest path bottleneck problem on n nodes, also $O(kn)$ and $O(n^{3/2}\log^{5/2} n)$ time algorithms for the k-shortest path bottleneck problem.

Key words: Monge property; Algebraic Monge matrices; Bottleneck shortest path problems; Combinatorial optimization.

[1] Research supported by NSF grants CCR-9821009 and CCR-0312093.
[2] Research supported by the U.S. Department of Energy under Contract DE-AC04-76DP00789.

1 Introduction

When restricted to cost arrays possessing the sum Monge property, many combinatorial optimization problems with sum objective functions become significantly easier to solve. For path problems, consider the complete directed acyclic graph $G = (V, E)$, *i.e.*, G has vertices $V = \{1, \ldots, n\}$ and edges $E = \{(i, j) \mid 1 \leq i < j \leq n\}$. The *unrestricted shortest-path problem* is the problem of finding the shortest path from vertex 1 to vertex n whereas the *k-edge shortest-path problem* is the problem of finding such a path that has exactly k edges. The shortest path problem for a complete directed acyclic graph is also called the *least weight subsequence problem*, where the vertices of the path are thought of as a subsequence of the sequence of all vertices in topological order. In [12], an $O(n \log n)$-time algorithm is given for this problem if the weights satisfy the sum Monge property (the term *concave least weight subsequence problem* is used in that paper), while first Wilber [19] and later Larmore and Schieber [17] showed that the same problem can be solved in $O(n)$ time, and that the k-shortest path problem can be solved in $O(kn)$ time. A substantial improvement in the case of the k-shortest path problem was found by Aggarwal, Schieber, and Tokuyama [3] who present an $O(n\sqrt{k \log n})$ algorithm using parametric search. In this paper we will give a number of efficient algorithms for the more general case of algebraic objectives.

Many combinatorial optimization problems with sum objectives have efficient algorithms for algebraic objective functions. We refer to the work of Burkard and Zimmermann [9] for a survey of classical results. Scheduling problems with algebraic objective functions are considered in Burkard [6]. Seiffart [18] gives results on algebraic transportation problems. In an algebraic combinatorial optimization problem, we are given a collection S of subsets of a finite nonempty set E, as well as a cost function $\phi : E \to H$, where $(H, *, \preceq)$ is an ordered commutative semigroup, and where the semigroup operation $*$ is *compatible* with the order relation $\preceq$, *i.e.*, for all $a, b, c \in H$, $a \prec b$ implies $c * a \preceq c * b$. (Throughout this paper, we require a slightly stronger property, namely that the semigroup operator $*$ be *strictly compatible* with the order relation $\preceq$, *i.e.*, for all $a, b, c \in H$, $a \prec b$ if and only if $c * a \prec c * b$.) The cost of subset $S = \{e_1, e_2, \ldots, e_k\} \subset E$ is $\phi(e_1) * \phi(e_2) * \cdots * \phi(e_k)$, and we are interested in finding a solution $S \in \mathsf{S}$ with minimum value. When the operation is addition, the objective is to minimize the sum, for example. Objectives other than sum often accommodate practical optimization problems more easily. Of particular interest are bottleneck objectives, where the operation $*$ is the "max" operation, *i.e.*, $x * y = \max\{x, y\}$. We mention that Gabow and Tarjan [11] give results concerning bottleneck shortest path problems.

We say that an $m \times n$ array $A = \{a_{i,j}\}$ possesses the *algebraic Monge property* if for all $1 \leq i < k \leq m$ and $1 \leq j < \ell \leq n$, $a_{i,j} * a_{k,\ell} \preceq a_{i,\ell} * a_{k,j}$. If operation

"$*$" is the "$+$" operation we just say that array A has the *Monge property*, if the operation "$*$" is replaced by the "max" operation we say that array A has the *bottleneck Monge property*. Klinz, Rudolf, and Woeginger [15] have developed an algorithm to recognize bottleneck Monge matrices in linear time. Burkard and Sandholzer [8] identify several families of cost arrays in which the bottleneck traveling-salesman problem can be solved in polynomial time; their results include bottleneck Monge arrays as an important special case. Finally we mention that Burkhard, Klinz, and Rudolf give an excellent survey article on Monge properties [7]. We note that their article refers to a number of results which appear in an earlier unpublished manuscript of ours [5].

Many of the result presented here rely on the fact that the Monge property implies total monotonicity. A 2×2 array $A = \{a_{i,j}\}$ is *monotone* if $a_{1,1} \leq a_{1,2}$ implies $a_{2,1} \leq a_{2,2}$. A partial array (*i.e.*, some of the entries can be undefined) is said to be *totally monotone* if every 2×2 subarray where all entries are defined is monotone. Similarly, a partial array is *transpose totally monotone* if its transpose (defined by reversing the role of rows and columns) is totally monotone. If an array is totally monotone (or transpose totally monotone), all row (or column) minima can be found by the so-called "SMAWK" algorithm of Aggarwal et al.[2], using at most linearly many queries to the array and linearly many comparisons. The *online monotone matrix searching algorithm* of [17], essentially an online version of SMAWK, solves a similar problem with an online restriction: Given a totally monotone partial matrix $A = \{a_{i,j}\}_{1 \leq j < i \leq n}$, all row minima of A can be found using $O(n)$ queries to A and $O(n)$ comparisons, subject to the online condition that no query to the k^{th} column of A can be made until the minimum of the k^{th} row is found. Similarly, given a transpose totally monotone partial matrix $B = \{b_{i,j}\}_{1 \leq i < j \leq n}$, all column minima of B can be found using $O(n)$ queries to B and $O(n)$ comparisons, subject to the online condition that no query to the k^{th} row of B can be made until the minimum of the k^{th} column is found.

Our paper is organized as follows: In Section 2 we derive the general algorithm for algebraic shortest path problems with the Monge property. The results follows fairly directly from the fact that if a matrix A possesses the algebraic Monge property, then A is totally monotone. Section 3 develops the algorithm for the bottleneck case. It might seem that the bottleneck results follow routinely, but, in fact as we shall see, the bottleneck case is more intricate because strict compatibility does not hold. We derive an $O(n)$ time algorithm for the unrestricted shortest path bottleneck problem, as well as an $O(kn)$ time algorithm the k-shortest path bottleneck problem. Section 4 contains an alternate $O(n^{3/2} \log^{5/2} n)$ algorithm that in some sense generalizes Aggarwal, Schieber, and Tokuyama's [3] algorithm. In Section 5 we apply our results to a variant of Hirschberg and Larmore's optimal-paragraph-formation problem [12] and in Section 6 we obtain a fast algorithm for a special case of the bottleneck traveling-salesman problem.

2 Algorithms for Algebraic Shortest-Path Problems

We will now show that both the unrestricted and the k-edge variants of the algebraic shortest-path problem for an ordered commutative semigroup $(H, *, \preceq)$, where costs satisfy the algebraic Monge property, are significantly easier to solve, if the operation $*$ is strictly compatible with the order relation. The strict compatibility condition of the ordered semigroup guarantees that an array possessing the algebraic Monge property is also totally monotone, the crucial property exploited by the algorithms given here, as well as others, including the classic SMAWK algorithm mentioned earlier.

We note that the results of this section and the next section can be obtained in a straightforward way by applying techniques known in the literature on shortest path problems in graphs with Monge weights and using the fact that those matrices are totally monotone. We include most of the short proofs in the interest of making the paper self-contained. The following lemma, which we state without proof, makes the claim in the previous paragraph about monotonicity precise.

Lemma 1 *Let $(H, *, \preceq)$ denote an ordered commutative semigroup whose operation is strictly compatible with its order relation, and let $A = \{a_{i,j}\}$ denote an array whose entries are drawn from that semigroup. If A possesses the algebraic Monge property, then A is totally monotone and also transpose totally monotone.*

Note that if the semigroup operation $*$ is compatible with its order relation $\preceq$ but not strictly compatible with it, then an array whose entries are drawn from the commutative semigroup may possess the algebraic Monge property without being totally monotone. For example, consider again the ordered commutative subgroup $(\mathbb{R}, \max, \leq)$ associated with bottleneck combinatorial optimization problems. The array $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ satisfies the inequality $\max\{a_{i,j}, a_{k,\ell}\} \leq \max\{a_{i,\ell}, a_{k,j}\}$ for all $i < k$ and $j < \ell$, but it is not totally monotone.

Before we can obtain the desired shortest-path algorithms, we need one more lemma.

Lemma 2 *Let $(H, *, \preceq)$ denote an ordered commutative semigroup whose operation is compatible with its order relation, and let $C = \{c_{i,j}\}$ denote an array whose entries are drawn from $(H, *, \preceq)$. Furthermore, let $B = \{b_i\}$ denote any vector, and let $A = \{a_{i,j}\}$ denote the array given by $a_{i,j} = b_i * c_{i,j}$. If C possesses the algebraic Monge property, then so does A.*

Proof: If C is algebraic Monge, then for all $i < k$ and $j < \ell$, $c_{i,j} * c_{k,\ell} \preceq c_{i,\ell} * c_{k,j}$. Since the order relation is compatible, and the composition $*$ is commutative, we have

$$\begin{aligned} c_{i,j} * c_{k,\ell} * b_i * b_k &\preceq c_{i,\ell} * c_{k,j} * b_i * b_k \\ b_i * c_{i,j} * b_k * c_{k,\ell} &\preceq b_i * c_{i,\ell} * b_k * c_{k,j} \\ a_{i,j} * a_{k,\ell} &\preceq a_{i,\ell} * a_{k,j} \ . \end{aligned}$$

□

Theorem 1 *Let $(H, *, \preceq)$ denote an ordered commutative semigroup whose operation is strictly compatible with its order relation, and let G denote a complete directed acyclic graph on vertices $1, \ldots, n$ whose edge costs $C = \{c_{i,j}\}$ are drawn from H. If C possesses the algebraic Monge property, then the algebraic k-edge shortest-path problem for G can be solved in $O((t_a + t_c)kn)$ time, where t_a is the worst-case time required for performing a composition $*$ and t_c is the worst-case time required for comparing two elements of H.*

Proof: Let $1 \hookrightarrow j$ denote a path from vertex 1 to vertex j. Define $a^\ell_{i,j}$ to be the length of the shortest ℓ-edge $1 \hookrightarrow j$ path that contains the edge (i, j), and d^ℓ_i to be the length of the shortest ℓ-edge $1 \hookrightarrow i$ path. Then

$$a^\ell_{i,j} = \begin{cases} d^{\ell-1}_i * c_{i,j} & \text{if } i < j \\ \infty & \text{otherwise} \end{cases} \quad \text{and} \quad d^\ell_j = \begin{cases} \min_{1 \le i < j} a^\ell_{i,j} & \text{if } \ell > 1 \\ c_{i,j} & \text{if } \ell = 1 \end{cases} \ .$$

By Lemma 2, for $2 \le \ell \le k$, the array $A^\ell = \{a^\ell_{i,j}\}$ is algebraic Monge.

Our algorithm contains k phases. In phase ℓ, we use $d^{\ell-1}_1$ through $d^{\ell-1}_n$ to compute d^ℓ_1 through d^ℓ_n. Note that d^ℓ_1 through d^ℓ_n are simply the minima of column 1 through column n of array A^ℓ. In phase ℓ, any entry $a^\ell_{i,j}$ of array A^ℓ can be computed in t_a time, since $d^{\ell-1}_i$ is already known. Thus, using the SMAWK algorithm [2], the column minima of A^ℓ can be found in $O((t_a+t_c)n)$ time. Since our algorithm has k phases, the total running time is $O((t_a+t_c)kn)$. □

Theorem 2 *Let $(H, *, \preceq)$ denote an ordered commutative semigroup whose internal composition $*$ is strictly compatible with its order relation $\le$, and let G denote a complete directed acyclic graph on vertices $1, \ldots, n$ whose edge costs are drawn from H. If G's edge costs possess the algebraic Monge property, then the algebraic unrestricted shortest-path problem for G can be solved in $O((t_a + t_c)n)$ time, where t_a is the worst-case time required for computing $d_i * c_{i,j}$ and t_c is the worst-case time required for comparing two entries of A.*

Proof: This proof is very similar to the proof of Theorem 1. Define $a_{i,j}$ to be the length of the shortest $1 \hookrightarrow j$ path that contains the edge (i, j), and d_i to be the length of the shortest $1 \hookrightarrow i$ path. Then

$$a_{i,j} = \begin{cases} d_i * c_{i,j} \text{ if } i < j \\ \infty \qquad \text{otherwise} \end{cases} \qquad \text{and} \qquad d_j = \begin{cases} \min_{1 \le i < j} a_{i,j} \text{ if } j > 1 \\ e \qquad\qquad\quad \text{if } j = 1 \end{cases},$$

where the min operation is performed over the order relation $\preceq$ and e is the identity element for the operation $*$. By Lemma 2, the array $A = \{a_{i,j}\}$ is algebraic Monge.

Note that d_1 through d_n are the minima of column 1 through column n of array A. An entry $a_{i,j}$ can be computed in t_a time if d_i is already known. Thus, the array $\{a_{i,j}\}$ fits the hypotheses of the online monotone matrix searching algorithm of Larmore and Schieber [17]. Since A is transpose totally monotone in our application, this algorithm computes the column minima of A in $O(n)$ steps, which requires $O((t_a + t_c)n)$ time. □

3 Algorithms for Bottleneck Shortest-Path Problems

In this section, we show how the two algebraic shortest-path algorithms considered before can be modified to handle the bottleneck shortest-path problems. Consider the ordered commutative subgroup $(\mathbb{R}, \max, \le)$ naturally associated with bottleneck combinatorial optimization problems, where $\mathbb{R}$ is the real numbers. In this case the operation is compatible with the order relation $\le$ but not strictly compatible with it. For example, $5 < 7$ but $\max\{8, 5\} \not< \max\{8, 7\}$. We will overcome this limitation by "extending" the semigroup to $(T, \oplus, \preceq)$, where T is the set of all finite multisets of real numbers, $\oplus$ is union, and $\le$ is lexical comparison, starting with the largest element. We shall call this the *semigroup of ordered lists* because we think of each multiset as a list sorted in non-decreasing order, and union as merge. $(T, \oplus, \preceq)$ is an ordered commutative semigroup and $\oplus$ is strictly compatible with $\preceq$.

To obtain our results we assume that the cost array possesses what we call the *strict bottleneck Monge property*, which requires that for all $i < k$ and $j < \ell$, either $\max\{c_{i,j}, c_{k,\ell}\} < \max\{c_{i,\ell}, c_{k,j}\}$ or both $\max\{c_{i,j}, c_{k,\ell}\} = \max\{c_{i,\ell}, c_{k,j}\}$ and $\min\{c_{i,j}, c_{k,\ell}\} \le \min\{c_{i,\ell}, c_{k,j}\}$.

Define the *full cost* of the bottleneck shortest-path to be the ordered tuple containing the costs of all the edges on this path sorted into non-increasing order. Define the *bottleneck cost* of the bottleneck shortest-path to be the first (*i.e.*, largest) entry in the full cost of the bottleneck shortest-path. For

example, if the bottleneck shortest $1 \hookrightarrow j$ path consists of edges $(1, i_1)$, (i_1, i_2), (i_2, i_3), (i_3, j) with the costs $c_{1,i_1} = 6$, $c_{i_1,i_2} = 3$, $c_{i_2,i_3} = 9$, $c_{i_3,j} = 5$, then the full cost of this path is $(9, 6, 5, 3)$ and its bottleneck cost is 9. We model the bottleneck shortest-path problems using the ordered commutative semigroup $(T, \oplus, \preceq)$. To be able to use our results for the algebraic Monge property, we need the following lemma.

Lemma 3 *Let $C = \{c_{i,j}\}$ be the array of edge costs. Let $C^T = \{c^T_{i,j}\}$ denote an array where each entry $c^T_{i,j}$ is a tuple consisting of a single element $c_{i,j}$. If C possesses the strict bottleneck Monge property, then C^T possesses the algebraic Monge property under $(T, \oplus, \preceq)$.*

Proof: Suppose $i < k$ and $j < \ell$. We need to prove that $(c^T_{i,j}) \oplus (c^T_{k,\ell}) \preceq (c^T_{i,\ell}) \oplus (c^T_{k,j})$. Let $L_1 = \max\{c_{i,j}, c_{k,\ell}\}$, $L_2 = \min\{c_{i,j}, c_{k,\ell}\}$, $R_1 = \max\{c_{i,\ell}, c_{k,j}\}$ and $R_2 = \min\{c_{i,\ell}, c_{k,j}\}$. Using this notation, $(c^T_{i,j}) \oplus (c^T_{k,\ell}) = (L_1, L_2)$ and $(c^T_{i,\ell}) \oplus (c^T_{k,j}) = (R_1, R_2)$. Thus, we need to prove that $(L_1, L_2) \preceq (R_1, R_2)$.

By the strict bottleneck Monge property, we have one of two cases: In the first case, when $\max\{c_{i,j}, c_{k,\ell}\} < \max\{c_{i,\ell}, c_{k,j}\}$, we have $L_1 < R_1$, which implies that $(L_1, L_2) \prec (R_1, R_2)$. Case 2 is that both $\max\{c_{i,j}, c_{k,\ell}\} = \max\{c_{i,\ell}, c_{k,j}\}$ and $\min\{c_{i,j}, c_{k,\ell}\} \leq \min\{c_{i,\ell}, c_{k,j}\}$ Then we have $L_1 = R_1$ and $L_2 \leq R_2$, which implies that $(L_1, L_2) \preceq (R_1, R_2)$. □

As stated in Section 1, the composition $\oplus$ is strictly compatible with the order relation $\preceq$. Thus, given Lemma 3, if we use array C^T as our cost array, all the lemmas and theorems of Section 2 apply to the bottleneck shortest-path problems modeled by $(T, \oplus, \preceq)$. Note that once the algorithms in Section 2 return their answers for the cost of the shortest-path, the bottleneck-cost of the shortest-path can be determined by taking the largest element in the cost tuple. We now state our results:

Theorem 3 *The bottleneck k-edge shortest-path problem for an n-vertex directed acyclic graph whose edge costs possess the strict bottleneck Monge property can be solved in $O(kn)$ time.*

Proof: This theorem appears to be a simple corollary of Theorem 1, but the operations in the semigroup of ordered lists take $O(n)$ time each. We resolve this problem by ignoring all but the largest two elements of each ordered list $a_{i,j}$. When the minimum of a column d_j is found, only the largest element of this list is retained for the next step of the algorithm. As a result, each query and each comparison takes only constant time. The resulting path obtained may differ from the one obtained by using the full lists, but that path nevertheless has minimum bottleneck cost. □

Similarly we obtain the result for the unrestricted case:

Theorem 4 *The bottleneck unrestricted shortest-path problem for an n-vertex directed acyclic graph whose edge costs possess the strict bottleneck Monge property can be solved in $O(n)$ time.*

Proof: This theorem would appear to be a simple corollary of Theorem 2, using the online monotone matrix searching algorithm [17], but once again we are concerned with the time to execute the operations of the semigroup. This difficulty is resolved just as in the proof of Theorem 3. Retain only the largest item in the sorted list for d_i, and only the largest two items in the sorted list for $a_{i,j}$. Each operation will take $O(1)$ time, and a perhaps different path will be found, but it will still be of minimum bottleneck cost. □

4 An Alternate Algorithm for the Bottleneck k-edge Shortest-Path

In this section we present a second algorithm for the bottleneck k-edge shortest-path problem that in some sense generalizes Aggarwal, Schieber, and Tokuyama's [4] algorithm. Our algorithm is based on a $O(n)$-time query subroutine for determining whether the graph contains a k-edge $1 \hookrightarrow n$ path using only edges whose costs are less than or equal to some threshold T. To create the query subroutine, we need two technical lemmas. We say that a path *satisfies the threshold* if every edge on the path is less than or equal to T.

Define two graphs $G'_T = (V, E)$ and $G''_T = (V, E)$ with the same vertex and edge sets as G but with different cost functions $\{c'_{i,j}\}$ and $\{c''_{i,j}\}$, where

$$c'_{i,j} = \begin{cases} 1 & \text{if } c_{i,j} \le T \\ +\infty & \text{otherwise} \end{cases} \quad \text{and} \quad c''_{i,j} = \begin{cases} -1 & \text{if } c_{i,j} \le T \\ +\infty & \text{otherwise} \end{cases} .$$

Lemma 4 *If the cost array $C = \{c_{i,j}\}$ is bottleneck Monge, then the arrays $C' = \{c'_{i,j}\}$ and $C'' = \{c''_{i,j}\}$ are Monge.*

Proof: We prove the lemma for the array $C' = \{c'_{i,j}\}$. The proof for C'' is identical. We prove by contradiction. Since all the entries in C' are either 1 or $+\infty$, there are only two possible ways that C' could fail to be Monge. In the first case, for some i, j, k, ℓ, both $c'_{i,\ell}$ and $c'_{j,k}$ are 1 and either $c'_{i,j}$ or $c'_{k,\ell}$ is $+\infty$. In the second case, for some i, j, k, ℓ, one of $c'_{i,\ell}$ or $c'_{j,k}$ is $+\infty$ and both $c'_{i,j}$ and $c'_{k,\ell}$ are $+\infty$.

Case 1 implies that in C, $c_{i,\ell}, c_{j,k} \le T$ and either $c_{i,j}$ or $c_{k,\ell}$ is greater than T. This means that $\max\{c_{i,j}, c_{k,\ell}\} > \max\{c_{i,\ell}, c_{j,k}\}$, which violates the bottleneck Monge property of C. Case 2 implies that in C, either $c_{i,\ell}$ or $c_{j,k}$ is greater than T and both $c_{i,j}, c_{k,\ell} > T$. This means that either $\max\{c_{i,j}, c_{k,\ell}\} > \max\{c_{i,\ell}, c_{j,k}\}$, which violates the bottleneck Monge property of C; or that

both $\max\{c_{i,j}, c_{k,\ell}\} = \max\{c_{i,\ell}, c_{j,k}\}$ and $\min\{c_{i,j}, c_{k,\ell}\} > \min\{c_{i,\ell}, c_{j,k}\}$, which also violates the bottleneck Monge property of C. □

Lemma 5 *Suppose the unrestricted shortest $1 \hookrightarrow n$ path P in G'_T contains k' edges and has length that is less than $+\infty$. Also, suppose the unrestricted shortest $1 \hookrightarrow n$ path Q in G''_T contains k'' edges and has length that is less than $+\infty$. Then there exists a k-edge $1 \hookrightarrow n$ path in G that satisfies the threshold if $k' \leq k \leq k''$.*

Proof: Note that because of the edge costs in G'_T and G''_T, $k' \leq k''$. Also, if either $k = k'$ or $k = k''$, we are done. Thus, assume that $k' < k < k''$. Our approach is to take the two paths P and Q and use them to create two new $1 \hookrightarrow n$ paths P' and Q' that contain $k'+1$ and $k''-1$ edge respectively, and satisfy the threshold. By repeating this procedure, we can create a $1 \hookrightarrow n$ path of any length k for $k' < k < k''$ that satisfies the threshold, thus proving the lemma.

Let $i \overset{\text{P}}{\hookrightarrow} j$ be the subpath of P from vertex i to vertex j and let $|i \overset{\text{P}}{\hookrightarrow} j|$ be the number of edges on that subpath. To create P' and Q', notice that by the pigeon-hole principle, there is an edge (i,t) in P and an edge (s,j) in Q such that $i < s < j \leq t$ and $|1 \overset{\text{P}}{\hookrightarrow} t| = |1 \overset{\text{P}}{\hookrightarrow} s|$. Let Q' consist of $1 \overset{\text{P}}{\hookrightarrow} i$, followed by the edge (i,j), followed by $j \overset{\text{Q}}{\hookrightarrow} n$; and P' consist of $1 \overset{\text{Q}}{\hookrightarrow} s$, followed by the edge (s,t), followed by $t \overset{\text{P}}{\hookrightarrow} n$. Clearly, P' and Q' are both $1 \hookrightarrow n$ paths. Since $|1 \overset{\text{P}}{\hookrightarrow} i| = |1 \overset{\text{P}}{\hookrightarrow} s| - 1$, we conclude that $|1 \overset{\text{P'}}{\hookrightarrow} n| = k'+1$ and $|1 \overset{\text{Q'}}{\hookrightarrow} n| = k''-1$. The only thing that remains to be shown is that both P' and Q' satisfy the threshold. If $j = t$, then the set of edges that appear in P' and Q' are exactly the same as the set of edges that appear in P and Q, and therefore P' and Q' satisfy the threshold. Thus, we assume that $j < t$. The set of edges that appear in P' and Q' are the same as the set of edges that appear in P and Q, except that the edges (i,t) and (s,j) are replaced by edges (i,j) and (s,t). By the bottleneck Monge property, we know that $\max\{c_{i,j}, c_{s,t}\} < \max\{c_{i,t}, c_{s,j}\}$ or both $\max\{c_{i,j}, c_{s,t}\} = \max\{c_{i,t}, c_{s,j}\}$ and $\min\{c_{i,j}, c_{s,t}\} \leq \min\{c_{i,t}, c_{s,j}\}$. Since the edges (i,t) and (s,j) are on paths P and Q, we know that they both cost $\leq T$. By the bottleneck Monge property, $c_{i,j}, c_{s,t} \leq T$, and thus P' and Q' satisfy the threshold. □

We are now ready to design the query subroutine for determining whether the given graph contains a k-edge $1 \hookrightarrow n$ path that uses only edges whose costs are less than or equal to some threshold T. Given Lemma 5, such a path exists if $k' \leq k \leq k''$, where k' is the length of the unrestricted shortest $1 \hookrightarrow n$ path in G'_T and k'' is the length of the unrestricted shortest $1 \hookrightarrow n$ path in G'''_T. Using the algorithm of Larmore and Schieber [17], we can determine the length of the unrestricted shortest $1 \hookrightarrow n$ path in a graph with Monge property in $O(n)$ time. Since the cost arrays of both G'_T and G''_T satisfy the Monge property

(Lemma 4), our query subroutine runs in $O(n)$ time.

Theorem 5 *The bottleneck k-edge shortest-path problem for an n-vertex graph whose edge costs possess the strict bottleneck Monge property can be solved in $O(n^{3/2}\log^2 n)$ time (or in $O(n\log^2 n)$ time if the problem's cost array is also bitonic[3]).*

Proof: We use the result of Agarwal and Sen [1], who show how to find the d^{th} smallest entry in an $m\times n$ totally monotone array in $O((m+n)\sqrt{n}\log n)$ time. For our $n\times n$ totally monotone array, this translates into $O(n^{3/2}\log n)$ time. There are n^2 entries in the $n\times n$ cost array C^T of the bottleneck shortest-path problem. We perform a binary search on these n^2 entries by calling a procedure which we call BINARY-SEARCH. For $i<j$ BINARY-SEARCH(C^T, i, j) is recursively defined as follows. First, use the algorithm of Agarwal and Sen [1] to find the $\lfloor\frac{j+i}{2}\rfloor^{\text{th}}$ smallest entry in C^T, which we call τ. We use our query algorithm to test if the graph contains a k-edge $1\hookrightarrow n$ path that uses only edges whose costs are less than or equal to τ. If the query returns "yes," then call BINARY-SEARCH(C^T, i, $\lfloor\frac{j+i}{2}\rfloor$); otherwise call BINARY-SEARCH(C^T, $\lfloor\frac{j+i}{2}\rfloor$, j). The binary search returns the smallest entry τ^* in C^T for which there exists a k-edge $1\hookrightarrow n$ path that uses only edges whose costs are less than or equal to τ^*. Thus, τ^* is the bottleneck-cost of the bottleneck k-edge shortest-path.

To find the actual path, we consider Lemma 5. We compute the unrestricted shortest $1\hookrightarrow n$ paths P and Q, $|P|=k'$, $|Q|=k''$, in the graphs G'_{τ^*} and G''_{τ^*}. Scanning vertices in increasing order starting at 1, we then look for an edge (i,t) in P and an edge (s,j) in Q that satisfy $i<s<j\le t$ and $|1\overset{\text{P}}{\hookrightarrow}t|=|1\overset{\text{P}}{\hookrightarrow}s|$. Let Q' consist of $1\overset{\text{P}}{\hookrightarrow}i$, followed by the edge (i,j), followed by $j\overset{\text{Q}}{\hookrightarrow}n$; and P' consist of $1\overset{\text{Q}}{\hookrightarrow}s$, followed by the edge (s,t), followed by $t\overset{\text{P}}{\hookrightarrow}n$. From Lemma 5, we know that P' and Q' are both $1\hookrightarrow n$ paths that satisfy τ^* and that $|P'|=k'+1$ and $|Q'|=k''-1$. If $k=k'+1$ or $k=k''-1$, we are done. Otherwise, we repeat this procedure, except that we start our search for the new edges where we want to switch the paths not from vertex 1, but from vertices i and s. Thus, the total running time to find the path given τ^* is $O(n+k)=O(n)$. The total running time to find the bottleneck k-edge shortest-path is $O(\log n^2(n^{3/2}\log n+n))=O(n^{3/2}\log^2 n)$.

In the case of a bitonic cost array, the selection problem is simpler. The *selection algorithm* of Frederickson and Johnson [10] computes the n largest elements overall in $O(n)$ sorted lists in $O(n)$ time. When the array is bitonic, we can easily decompose it into $2n$ sorted lists in $O(n\log n)$ time. Applying

[3] An n-entry vector $B=\{b_i\}$ is called bitonic if there exists an i satisfying $1\le i\le n$ such that $b_1\ge\cdots\ge b_{i-1}\ge b_i\le b_{i+1}\le\cdots\le b_n$. We call a 2-dimensional array bitonic if its rows or its columns are bitonic.

the selection algorithm, we can compute the d^{th} smallest entry in an $n \times n$ bitonic array in $O(n \log n)$ time. Thus, for cost arrays that satisfy the string bottleneck Monge property and are bitonic, the k-edge shortest-path can be found in $O(n \log^2 n)$ time. □

Using a similar query technique, we can also obtain the following result for the unbalanced assignment problem.

Theorem 6 *The bottleneck assignment problem for an $m \times n$ bipartite graph, where $m \leq n$ and the edge costs possess the strict bottleneck Monge property, can be solved in $O((m\sqrt{n \log m} + n) \log^2 n)$ time (or in $O(m \log^2 n + n \log n)$ time if the problem's cost array is also bitonic).*

Proof: We say that a matching *satisfies the threshold* T if all the edges of the matching have weight T or less. First, we design a query algorithm analogous to the one used in the proof of Theorem 5. This query algorithm, given a bipartite graph G and a threshold T, determines if there is a perfect matching of G that satisfies the threshold. The query algorithm is greedy, and it works by finding the minimum value j_1 such that $w_{1,j_1} \leq T$, then the minimum value j_2 such that $w_{2,j_2} \leq T$ and $j_2 > j_1$, then the minimum value j_3 such that $w_{3,j_3} \leq T$ and $j_3 > j_2$, and so forth. This algorithm takes $O(n)$ time. If this algorithm produces a perfect matching, then clearly this matching satisfies the threshold. Furthermore, it is not difficult to see that if there exists a perfect matching that satisfies the threshold, then our algorithm finds one. To see this, consider any perfect matching M that satisfies the threshold. Consider the first vertex i such that $(i, j_i) \notin M$. We show how to get another perfect matching M' in which the first vertex i' such that $(i', j_{i'}) \notin M'$ is greater than i. Suppose that i is matched to ℓ in M. Our computation of j_i guarantees that $\ell > j_i$. If j_i is unmatched in M, then we can simply remove (i, ℓ) and add (i, j_i) to get another perfect matching M' that satisfies the threshold and contains (i, j_i). Hence, assume that j_i is matched in M to s, $s > i$. By the strict bottleneck Monge property, we have that $\max\{c_{i,j_i}, c_{s,\ell}\} < \max\{c_{i,\ell}, c_{s,j_i}\}$ or both $\max\{c_{i,j_i}, c_{s,\ell}\} = \max\{c_{i,\ell}, c_{s,j_i}\}$ and $\min\{c_{i,j_i}, c_{s,\ell}\} \leq \min\{c_{i,\ell}, c_{s,j_i}\}$. Thus, if M satisfies the threshold, then $M' = M - \{(i, \ell), (s, j_i)\} + \{(i, j_i), (s, \ell)\}$ is another perfect matching and it contains (i, j_i).

To complete the proof, we use the query algorithm just as in the proof of Theorem 5. We use the algorithm of Agarwal and Sen [1] in our binary search, and for each d^{th} smallest value T in the $m \times n$ cost array, we query to see if there exists a perfect matching that satisfies the threshold T. The total running time is $O(\log(mn)((m+n)\sqrt{n} \log n + n)) = O(n^{3/2} \log^2 n)$. Similarly, if the cost array is bitonic, then the running time is $O(\log(mn)(m \log n + n)) = O(m \log^2 n + n \log n)$. □

5 A Paragraph-Formation Problem

One practical application of the shortest path problem in a complete directed acyclic graph is optimal paragraph formation. Suppose that a paragraph consists of n words $w_1, \ldots w_n$, of varying length, and a paragraph of text must be set using those words, with the rule that no word may be broken between lines. (If a word can be broken into syllables on separate lines, we call each syllable a "word.") We are given a *penalty matrix* C, where c_{ij} is defined to be the *cost* ("penalty") of a line consisting of the words $w_i \ldots w_{j-1}$. In [12], the optimal paragraph is defined to be a paragraph with the smallest possible total penalty. If the penalty matrix satisfies the sum Monge condition, the optimal paragraph can be found in $O(n)$ time using the online monotone matrix searching algorithm of [17].

Let L be the optimal line width and $|w_i|$ be the length of word w_i plus one for the cost of the space after word w_i. Then, following the ideas of Hirschberg and Larmore [12], the penalty function could be defined as $c_{i,j} = (|w_{i+1}| + |w_{i+2}| + \ldots + |w_j| - 1 - L)^2$. This problem is easily transformed into an instance of the sum unrestricted shortest-path problem. Consider a directed acyclic graph $G = (V, E)$, where the vertices are numbered 0 through n and $E = \{(i, j) \mid 0 \leq i < j \leq n\}$. The cost of edge (i, j) is $c_{i,j}$ defined above. A $0 \hookrightarrow n$ path $p = \langle (0, i_1), (i_1, i_2), \ldots, (i_s, n) \rangle$ corresponds to putting words w_1 through w_{i_1} into the first line, words w_{i_1+1} through w_{i_2} into the second line, and so forth, with words w_{i_s} through w_n forming the last line. The shortest $0 \hookrightarrow n$ path in this graph corresponds to the minimum sum of the line costs of the paragraph. Hirschberg and Larmore prove that the above cost function satisfies the sum Monge property, and thus it can be solved in $O(n)$ time. (Credit for the linear-time algorithm belongs to Wilber [19] and to Larmore and Schieber [17].)

If we instead seek to minimize the maximum line penalty, we obtain an instance of the bottleneck unrestricted shortest-path problem. The following two lemmas prove that the edge costs in this problem possess the strict bottleneck Monge property, providing the penalty functions satisfies a condition we call "strictly bitonic," which we claim holds for many practical applications.

We call a penalty function $f_{i,j}$ *strictly bitonic* if for any sequence of penalties $f_{i_1,j_1}, f_{i_2,j_2}, \ldots, f_{i_s,j_s}$, which satisfies the following conditions for every $1 \leq \ell < s$:

(1) either $i_\ell = i_{\ell+1}$ and $j_\ell < j_{\ell+1}$,
(2) or $j_\ell = j_{\ell+1}$ and $i_\ell > i_{\ell+1}$

is strictly decreasing then strictly increasing. Note that either the strictly decreasing subsequence or the strictly increasing subsequence may have length zero.

Lemma 6 *Let $F = \{f_{i,j}\}$, where $f_{i,j}$ is a penalty function that is strictly bitonic. Then F satisfies the strict bottleneck Monge property.*

Proof: Let $i < k < j < \ell$ and consider the two sequences $f_{k,j}, f_{k,\ell}, f_{i,\ell}$ and $f_{k,j}, f_{i,j}, f_{i,\ell}$. Both of these sequences must be bitonic. For the first sequence, this implies that it is not possible that $f_{k,j} < f_{k,\ell}$ and $f_{i,\ell} < f_{k,\ell}$, *i.e.* either $f_{k,j} > f_{k,\ell}$ or $f_{i,\ell} > f_{k,\ell}$. Similarly, either $f_{k,j} > f_{i,j}$ or $f_{i,\ell} > f_{i,j}$. Together, these two statements imply that $\max\{f_{i,j}, f_{k,\ell}\} < \max\{f_{i,\ell}, f_{k,j}\}$ and thus F satisfies the strict bottleneck Monge property. □

Lemma 7 *The penalty function $c_{i,j} = (|w_{i+1}| + |w_{i+2}| + \ldots + |w_j| - 1 - L)^2$ is strictly bitonic.*

Proof: Let $p_{i,j} = |w_{i+1}| + |w_{i+2}| + \ldots + |w_j|$. Notice that any sequence p_{i_1,j_1}, $p_{i_2,j_2}, \ldots, p_{i_s,j_s}$, which for every $1 \leq \ell < s$ satisfies:

(1) either $i_\ell = i_{\ell+1}$ and $j_\ell < j_{\ell+1}$,
(2) or $j_\ell = j_{\ell+1}$ and $i_\ell > i_{\ell+1}$

is strictly increasing. Now consider any sequence $(p_{i_1,j_1} - 1 - L)^2, (p_{i_2,j_2} - 1 - L)^2, \ldots, (p_{i_s,j_s} - 1 - L)^2$, which for every $1 \leq \ell < s$ satisfies:

(1) either $i_\ell = i_{\ell+1}$ and $j_\ell < j_{\ell+1}$,
(2) or $j_\ell = j_{\ell+1}$ and $i_\ell > i_{\ell+1}$.

This sequence is a quadratic function and thus is strictly increasing and then strictly increasing. □

From these two lemmas, we conclude that any strictly bitonic line cost function $f(i, j)$ satisfies the strict bottleneck Monge property, and thus, by Theorem 4, a variant of Hirschberg and Larmore's problem which uses $f(i, j)$ can also be solved in $O(n)$ time. In particular, the variant with cost function $c_{i,j}$ can be solved in $O(n)$ time.

Suppose that we wish to modify the penalty function by increasing the penalty for any line that ends with a hyphen. We also would like to impose absolute upper and lower bounds on the length of a line, and would like to not charge a penalty for the last line of a paragraph being too short. We now consider each of these modifications separately.

Absolute bounds on the line length are handled as follows. Suppose that the minimum and the maximum allowed length for a line are α and β, respectively. Then define the cost function to be

$$c_{i,j} = \begin{cases} (p_{i,j} - 1 - L)^2 \text{ if } \alpha \leq p_{i,j} \leq \beta \text{ and } \ j < n \\ M^{i+j} \qquad\qquad \text{otherwise} \end{cases},$$

where M is a very large number (say the sum of all the word lengths). It is not hard to see that the proof that $c_{i,j}$ is bitonic holds with this new definition of the cost function.

To take into account the elimination of a penalty for the last line being too short, we need to define a new penalty function

$$c'_{i,j} = \begin{cases} 0 & \text{if } j = n \text{ and } p_{i,j} \leq L \\ (p_{i,j} - 1 - L)^2 & \text{if } \alpha \leq p_{i,j} \leq \beta \\ M^{i+j} & \text{otherwise} \end{cases} .$$

This new cost function $c'_{i,j}$ is not strictly bitonic and furthermore not even bottleneck Monge. Thus, we stick to $c_{i,j}$ as our cost function, but we modify the algorithm. Instead of computing the $1 \hookrightarrow n$ bottleneck shortest path in Theorem 4, we compute the bottleneck cost of the $1 \hookrightarrow n-1$ bottleneck shortest path. The algorithm in the proof of this theorem actually computes all d_i's for $1 \leq i \leq n-1$, where d_i is the bottleneck cost of the bottleneck shortest $1 \hookrightarrow i$ path. To find the bottleneck shortest $1 \hookrightarrow n$ path, we evaluate the following, which takes $O(n)$ time:

$$d_n = \min_{1 \leq i \leq n-1} \left\{ \begin{array}{ll} d_i & \text{if } p_{i,n} \leq L \\ \max\{d_i, (p_{i,n} - 1 - L)^2\} & \text{if } L \leq p_{i,n} \leq \beta \\ \infty & \text{otherwise} \end{array} \right\} .$$

Unfortunately, it is not possible to incorporate the hyphenation penalty into the bottleneck framework. If we are to assign a fixed penalty function B for breaking a word in the middle and hyphenating it, the penalty function may no longer be bottleneck Monge. More specifically, we assume that each w_i is now a syllable. Suppose we have $i < k < j < \ell$ and j^{th} syllable is not the last syllable of its word, but the ℓ^{th} syllable is. Then for the cost function to be bottleneck Monge, we would need to have

$$\begin{aligned} \max\left\{(p_{i,j} - 1 - L)^2 + B, (p_{k,\ell} - 1 - L)^2\right\} \leq \\ \max\left\{(p_{k,j} - 1 - L)^2 + B, (p_{i,\ell} - 1 - L)^2\right\} . \end{aligned}$$

It is easy to come up with a numerical example when this does not hold.

We also consider another variation of the penalty function. In some circumstances, a more accurate portrayal of a typical text formatting application would be that instead of having an ideal line length and penalizing for both

running under and running over this ideal length, the application has L as the available line width. In this case, there is a very large penalty for running over this available line width and a quadratic penalty for running under this width. The cost function in this case is

$$c_{i,j} = \begin{cases} (p_{i,j} - 1 - L)^2 \text{ if } p_{i,j} \leq L \\ M^{i+j} \qquad\qquad \text{otherwise} \end{cases},$$

where M is a very large number (say the sum of all the word lengths). It is not hard to see that this $c_{i,j}$ is strictly bitonic.

6 A Special Case of the Bottleneck Traveling-Salesman Problem

For our final application, we consider a special case of the bottleneck traveling-salesman problem. Given a complete directed graph G on vertices $\{1, \ldots, n\}$ and a cost array $C = \{c_{i,j}\}$ assigning cost $c_{i,j}$ to the edge (i, j), we seek a tour of G that visits every vertex of G exactly once and minimizes the maximum of the tour's edges' costs. We call such a tour the *bottleneck shortest-tour*. In [8], Burkard and Sandholzer identified several families of cost arrays corresponding to graphs containing at least one bottleneck-optimal tour that is pyramidal. A tour T is called *pyramidal* if (1) the vertices on the path T starting from vertex n and ending at vertex 1 have monotonically decreasing labels, and (2) the vertices on the path T starting from vertex 1 and ending at vertex n have monotonically increasing labels. For example, a tour $T = \langle 4, 2, 1, 3, 6, 8, 7, 5, 4 \rangle$ is pyramidal, but a tour $T = \langle 4, 2, 1, 6, 3, 8, 7, 5, 4 \rangle$ is not. Thus, since there is a simple $O(n^2)$-time dynamic-programming algorithm for computing a pyramidal tour whose maximum edge cost is minimum among all pyramidal tours, the bottleneck traveling-salesman problem for any graph whose cost array is a member of one of Burkard and Sandholzer's families can be solved in $O(n^2)$ time.

We now show that if the edge cost array of a graph possesses the strict bottleneck Monge property, then it is possible to find the bottleneck-shortest pyramidal tour in that graph in $O(n)$ time. We make use of a technique by Komlos for answering interval-minimum queries in constant amortized time [16].

Theorem 7 *Given a graph G whose cost array C satisfies the strict bottleneck Monge property, the bottleneck pyramidal shortest-tour of G can be found in $O(n)$ time.*

Proof: We reduce the problem to an instance of the on-line monotone matrix

searching problem, and apply the algorithm of [17], which takes $O(n)$ comparisons. We then show that each comparison can be done in constant amortized time.

We now describe the reduction. Define d_i to be the bottleneck cost of the shortest bitonic path from i to $i+1$ which uses every vertex in the range 0 to $i+1$ exactly once. Such a path can be thought of as the union of an increasing path from 0 to i and an increasing path from 0 to $i+1$, where each vertex in the range 1 to $i+1$ is in exactly one of the two paths. Let $e_{i,j}$ be the full cost of the path from i to j which passes through every intermediate point; that is, $e_{i,j} = c_{i,i+1} * \cdots * c_{j-1,j}$, a member of the semigroup of ordered lists $\mathsf{S} = (T, \oplus, \preceq)$. Note that $e_{i,i}$ is the empty list, which is the minimum element of that semigroup. For $i < j$, let $a_{i,j} = d_i * c_{i,j+1} * e_{i+1,j}$, also a member of S. Then, we observe that d_j is the largest element of the ordered list $\min_{i<j} \{a_{i,j}\}$. Thus, the online matrix searching algorithm of [17] can be used to find all column minima, and hence finally $\max\{d_{n-1}, c_{n-1,n}\}$, the desired bottleneck cost, since $\{a_{i,j}\}$ is transpose totally monotone. Unfortunately, a comparison in the semigroup S takes $O(n)$ time in the worst case. However, the time can be reduced to $O(1)$ for our application, as we see below.

The comparisons required are always whether $a_{i,j} \leq a_{i',j}$ for some $i < i' < j$. Both sides of the inequality are multisets of real numbers, and the set $e_{i'+1,j}$ is in the intersection. Deletion of those common elements does not effect the results of the comparison. Thus, it is possible to reduce the comparison to the question of whether $d_i * c_{i,j+1} * e_{i+1,i'+1} \leq d_{i'} * c_{i',j+1}$. In order to answer this question, at most the largest two items of the multiset $e_{i+1,i'+1}$ need be examined. Similarly, after determining that the minimum of the j^{th} column is $a_{i,j}$, the value of d_j is the maximum item in the multiset $a_{i,j} = d_i * c_{i,j+1} * e_{i+1,j}$. Thus, only the largest item in the multiset $e_{i+1,j}$ need be examined. We conclude that a query to $e_{i,j}$ need only return the two largest items in that multiset.

We can spend $O(n)$ preprocessing time to create a data structure which allows us to find the minimum item of any subinterval of a list of items in $O(1)$ amortized time, using the techniques of Komlos [16]. The smallest two elements in a subinterval can be found using three such queries; one to find the smallest item, and the other two to find the minima of the sublists to the left and to the right of that minimum. Thus, the smallest two elements of any $e_{i,j}$ can be found in $O(1)$ amortized time. The overall running time of our algorithm is thus $O(n)$. □

Acknowledgements

We thank Seth Pettie for helpful conversations.

References

[1] P. K. Agarwal and S. Sen. Selection in monotone matrices and computing k^{th} nearest neighbors. In *Proceedings of the 4th Scandinavian Workshop on Algorithm Theory*, 1994.

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

[3] A. Aggarwal, B. Schieber, and T. Tokuyama. Finding a minimum weight k-link path in graphs with Monge property and applications. In *Proceedings* 9^{th} *Annual ACM Symposium on Computational Geometry*, pages 189–197, 1993.

[4] A. Aggarwal, B. Schieber, and T. Tokuyama. Finding a minimum-weight k-link path in graphs with the concave Monge property and applications. *Discrete & Computational Geometry*, 12:263–280, 1994.

[5] W. W. Bein, P. Brucker, and P. K. Park. Applications of an algebraic Monge property. Unpublished manuscript. Presented at the 3^{rd} Twente Workshop on Graphs and Combinatorial Optimization, Enschede, The Netherlands, 1993.

[6] R. E. Burkard. Remarks on some scheduling problems with algebraic objective functions. *Operations Research Verfahren*, 32:63–77, 1979.

[7] R. E. Burkard, B. Klinz and R. Rudolf. Perspectives of monge properties in optimization. *Discrete Applied Mathematics*, 70:95–161, 1996.

[8] R. E. Burkard and W. Sandholzer. Efficiently solvable special cases of bottleneck traveling salesman problems. *Discrete Applied Mathematics*, 32(1):61–76, 1991.

[9] R. E. Burkard and U. Zimmermann. Combinatorial optimization in linearly ordered semimodules: A survey. In B. Korte, editor, *Modern Applied Mathematics: Optimization and Operations Research*, pages 391–436. North-Holland Publishing Company, Amsterdam, Holland, 1982.

[10] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in X + Y and matrices with sorted columns. *Journal of Computer and System Sciences*, 24(4):197–208, 1982.

[11] H. N. Gabow and R. E. Tarjan. Algorithms for two bottleneck optimization problems. *Journal of Algorithms*, 9(3):411–417, 1988.

[12] D. S. Hirschberg and L. L. Larmore. The least weight subsequence problem. *SIAM Journal on Computing*, 16(4):628–638, 1987.

[13] D. S. Hirschberg and D. J. Volper. Improved update/query algorithms for the interval valuation problem. *Information Processing Letters*, 24:307–310, 1987.

[14] A.J. Hoffman. On simple linear programming problems. In *Convexity, Proceedings of Symposia in Pure Mathematics*, Volume 7, American Mathematical Society, pages 317 – 327, Providence, RI, 1961.

[15] B. Klinz, R. Rudolf, and G.J. Woeginger. On the recognition of bottleneck monge matrices. *Discrete Applied Mathematics*, 63:43–74, 1995.

[16] J. Komlos. Linear verification for spanning trees. *Combinatorica*, 5(1): 57–65, 1985.

[17] L. L. Larmore and B. Schieber. On-line dynamic programming with applications to the prediction of RNA secondary structure. *Journal of Algorithms*, 12(3):490–515, 1991.

[18] E. Seiffart. Algebraic transportation and assignment problems with "Monge-property" and "quasi-convexity". *Unpublished*, 1993.

[19] R. Wilber. The concave least-weight subsequence problem revisited. *Journal of Algorithms*, 9(3):418–425, 1988.