Fast Algorithms with Algebraic Monge Properties[$\star$]

Wolfgang W. Bein[1,$\star\star$], Peter Brucker[2],
Lawrence L. Larmore[1,$\star$], and James K. Park[3$\star\star\star$]

[1] Department of Computer Science, University of Nevada,
Las Vegas, Nevada 89154, USA,
bein@cs.unlv.edu larmore@cs.unlv.edu
[2] Universität Osnabrück, Fachbereich Mathematik/Informatik
D-49069 Osnabrück, Germany,
peter@mathematik.uni-osnabrueck.de
[3] Bremer Associates, Inc.,
215 First Street, Cambridge, Massachusetts 02142, USA,
james.park@bremer-inc.com

Abstract. When restricted to cost arrays possessing the sum Monge property, many combinatorial optimization problems with sum objective functions become significantly easier to solve. The more general algebraic assignment and transportation problems are similarly easier to solve given cost arrays possessing the corresponding algebraic Monge property. We show that Monge-array results for two sum-of-edge-costs shortest-path problems can likewise be extended to a general algebraic setting, provided the problems' ordered commutative semigroup satisfies one additional restriction. In addition to this general result, we also show how our algorithms can be modified to solve certain bottleneck shortest-path problems, even though the ordered commutative semigroup naturally associated with bottleneck problems does not satisfy our additional restriction. We show how our bottleneck shortest-path techniques can be used to obtain fast algorithms for a variant of Hirschberg and Larmore's optimal paragraph formation problem, and a special case of the bottleneck traveling-salesman problem.

1 Introduction

In an algebraic combinatorial optimization problem, we are given a collection $\mathcal{S}$ 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. It is further assumed that the internal composition $*$ is *compatible* with the order relation $\preceq$, i.e. for

$\star$ Conference Version.

$\star\star$ Research of these authors supported by NSF grant CCR-9821009.

$\star\star\star$ This author's work was done while the author was at Sandia National Laboratories, and supported by the U.S. Department of Energy under Contract DE-AC04-76DP00789.

all $a, b, c \in H$, $a \prec b$ implies $c * a \preceq c * b$. Then an algebraic combinatorial problem is given by

$$\min_{S \in \mathcal{S}} \bigotimes_{e \in S} \phi(e)$$

with $\bigotimes_{e \in S} := \phi(e_1) * \phi(e_2) * \cdots * \phi(e_k)$ for $S = \{e_1, e_2, \ldots, e_k\} \subset E$. When the operation "$*$" is the "$+$" operation we have a regular sum objective. Often objectives more general than sum objectives accomodate practical optimization problems more easily. Of particular interest are bottleneck objectives, where the operation "$*$" is replaced by the "max" operation. Many combinatorial optimization problems with sum objectives have efficient algorithms for algebraic objective functions (see Burkard and Zimmermann [7] for a survey of classical results, as well as Burkard [6,8] and Seiffart [16]).

These generalizations are especially elegant for shortest path problems and assignment problems. In this case the set E is a subset of $U \times V$, where $U = \{1, \ldots, m\}$ and $V = \{1, \ldots, n\}$ for some $m, n \in N$. The cost function ϕ is then expressed in terms of a cost array $A = \{a[i,j]\}$ with $\phi(e) = a[i,j]$ if $e = (i,j)$. 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.*

One of the earliest results concerning optimization problems with (sum) Monge arrays goes back to Hoffman [13]. He showed the now classical result that the transportation problem is solved by a greedy $O(n)$ algorithm if the underlying cost array is a Monge array and $m = n$.

For path problems, consider the complete directed acyclic graph $G = (V, E)$, i.e. G has vertices $V = \{1, \ldots, n\}$ and edges $(i,j) \in E$ iff $i < j$. 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. Larmore and Schieber [15] have shown that, for (sum) Monge distances, the unrestricted shortest path problem on a complete directed acyclic graph can be solved in $O(n)$, whereas the k-shortest path problem can be solved in $O(kn)$. Aggarwal, Schieber, and Tokuyama [3] present an alternate $O(n\sqrt{k} \lg n)$ algorithm using parametric search.

In this paper we will derive such results for two path problems with algebraic cost arrays.

Consider the complete directed acyclic graph G as above. Associated with the edges are costs $a[i,j]$, which are drawn from the ordered commutative semigroup $(H, *, \preceq)$. An essential requirement of our algorithms will be that the internal composition $*$ be strictly compatible with the order relation $\preceq$ (the operation $*$ is *strictly compatible* with the order relation $\preceq$ if for all $a, b, c \in H$, $a \prec b$ implies $c * a \prec c * b$).

As mentioned earlier, an important case for practical applications of algebraic objective functions is bottleneck objective functions. Gabow and Tarjan [10] give results concerning bottleneck shortest path problems. In [5], Burkard and Sandholzer 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. Klinz, Rudolf, and Woeginger [14] have developed an algorithm to recognize bottleneck Monge matrices in linear time. We derive here an $O(n)$ algorithms for the unrestricted shortest path bottleneck problem, as well as an $O(kn)$ algorithm and an alternative $O(n^{3/2} \lg^{5/2} n)$ for the k-shortest path bottleneck problems.

Consider the ordered commutative subgroup $(\Re, \max, \leq)$ naturally associated with bottleneck combinatorial optimization problems. Note that the composition max is compatible with the order relation $\leq$ but not strictly compatible with it. (For example, $5 < 7$ but $\max\{8,5\} \not< \max\{8,7\}$.) For an example of an ordered commutative semigroup $(H, *, \preceq)$ whose internal composition $*$ is strictly compatible with its order relation $\preceq$, consider the set T of ordered tuples $(r_1, r_2, \ldots, r_n)$ such that $n \geq 0$, $r_i \in \Re$ for $1 \leq i \leq n$, and $r_1 \geq r_2 \geq \cdots \geq r_n$. Furthermore, suppose we define " $\oplus$" so that

$$(q_1, q_2, \ldots, q_m) \oplus (r_1, r_2, \ldots, r_n) \; = \; (s_1, s_2, \ldots, s_{m+n}) \; ,$$

where $s_1, s_2, \ldots, s_{m+n}$ is the sorted sequence obtained by merging the sequences $(q_1, q_2, \ldots, q_m)$ and $(r_1, r_2, \ldots, r_n)$, and we define $\prec$ so that $(q_1, q_2, \ldots, q_m) \prec (r_1, r_2, \ldots, r_n)$ if and only if there exists an i in the range $1 \leq i \leq m$ such that $q_i < r_i$ and $q_j = r_j$ for $1 \leq j < i$ *or* $m < n$ and $q_j = r_j$ for $1 \leq j \leq m$. It is not hard to see that $(T, \oplus, \preceq)$ is an ordered commutative semigroup and $\oplus$ is strictly compatible with $\preceq$. As we will see later this semigroup can be used to model strict bottleneck Monge conditions.

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 matrix A possesses the algebraic Monge property, then A is totally monotone. Section 3 develops the algorithm for the bottleneck case. As discussed in the previous paragraph, the results for the bottleneck case is more intricate. Section 4 contains an alternate 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 [11] and in Section 6 we obtain a fast algorithms 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 $(H, *, \preceq)$ are significantly easier to solve given edge costs with the algebraic Monge property, provided the internal composition $*$ is strictly compatible. An $m \times n$ array $A = \{a[i,j]\}$ is *totally monotone* if for all $1 \leq i < k \leq m$ and $1 \leq j < \ell \leq n$, either $a[i,j] \preceq a[i,\ell]$ or $a[k,j] \succ a[k,\ell]$. (We say that A is *transpose totally monotone* if the transpose of A is totally monotone; *i.e.*, if the same condition holds with the role of rows and columns reversed.) Strict compatibility is necessary to insure that every array possessing the algebraic Monge property is also totally monotone, the crucial property exploited by our algorithms. The following lemma makes this last claim precise.

Lemma 2.1. *Let $(H, *, \preceq)$ denote an ordered commutative semigroup whose internal composition $*$ is* strictly *compatible with its order relation $\preceq$, and let $A = \{a[i,j]\}$ denote an array whose entries are drawn from $(H, *, \preceq)$. If A possesses the algebraic Monge property, then A is totally monotone and also transpose totally monotone.*

Proof. By contradiction. Suppose that $(H, *, \preceq)$ is an ordered commutative semigroup whose internal composition $*$ is strictly compatible with its order relation $\preceq$, $A = \{a[i,j]\}$ is an algebraic Monge array whose entries are drawn from $(H, *, \preceq)$, and A is not totally monotone. Then for some i, j, k, and ℓ satisfying $1 \leq i < k \leq m$ and $1 \leq j < \ell \leq n$, $a[i,j] \succ a[i,\ell]$ *and* $a[k,j] \preceq a[k,\ell]$. Because the composition $*$ is strictly compatible with $\preceq$, we have that $a[i,j] * a[k,\ell] \succ a[i,\ell] * a[k,\ell]$ and $a[i,\ell] * a[k,\ell] \succeq a[i,\ell] * a[k,j]$. By transitivity of the order relation, we have $a[i,j] * a[k,\ell] \succ a[i,\ell] * a[k,j]$, which contradicts the Monge property of A. The proof that A is transpose totally monotone is similar, as the algebraic Monge property is invariant under tansposition.

Note that if the semigroup's composition $*$ is compatible with its order relation $\preceq$ but not strictly compatible with it, then an array whose entries are drawn from the semigroup may possess the algebraic Monge property without being totally monotone. For example, consider again the ordered commutative subgroup associated with bottleneck combinatorial optimization problems. The array

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

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.

The total monotonicity of arrays possessing the algebraic Monge property allows us to locate these arrays' smallest entries using the array-searching algorithms of Aggarwal et al. [1] (called the SMAWK algorithm) and Larmore and Schieber [15]. However, before we can obtain the desired shortest-path algorithms, we need one more lemma. (Note that this lemma does not require the strict-compatibility assumption.)

Lemma 2.2. *Let $(H, *, \preceq)$ denote an ordered commutative semigroup whose internal composition $*$ is compatible with its order relation $\preceq$, and let $C = \{c[i,j]\}$ denote an array whose entries are drawn from $(H, *, \leq)$. 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 2.3. *Let $(H, *, \preceq)$ denote an ordered commutative semigroup whose internal composition $*$ is strictly compatible with its order relation $\preceq$, 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 $d^{\ell}[i]$ to be the length of the shortest ℓ-edge $1 \hookrightarrow i$ path. Then

$$a^{\ell}[i,j] = \begin{cases} d^{\ell-1}[i] * c[i,j] & \text{if } i < j \\ \infty & \text{otherwise} \end{cases} \quad \text{and} \quad d^{\ell}[j] = \begin{cases} \min_{1 \le i < j} a^{\ell}[i,j] & \text{if } \ell > 1 \\ c[i,j] & \text{if } \ell = 1 \end{cases}$$

where the min operation is performed over the order relation $\preceq$. By Lemma 2.2, 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 $d^{\ell}[1]$ through $d^{\ell}[n]$. Note that $d^{\ell}[1]$ through $d^{\ell}[n]$ 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 algorithm of Aggarwal et al. [1], 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 2.4. *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 2.3 and is given in the journal version.

3 Algorithms for the Bottleneck Shortest-Path Problems

In this section, we show how our two algebraic shortest-path algorithms can be modified to handle the bottleneck shortest-path problems. As mentioned in

Section 1, the ordered commutative subgroup $(\Re, \max, \leq)$ naturally associated with bottleneck combinatorial optimization problems has a composition that is not strictly compatible with its order relation. Thus, we need to model the bottleneck problems with a different semigroup.

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 $\min\{c[i,j], c[k,\ell]\} \leq \min\{c[i,\ell], c[k,j]\}$.

Define the *cost* of the bottleneck shortest-path to be an 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. the largest) entry in the 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 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 $(T, \oplus, \preceq)$ that was defined in Seciton 1, where the set T contains the costs of the bottleneck shortest-paths. To be able to use our results for the algebraic Monge property, we need the following lemma.

Lemma 3.1. *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. 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 $(L_1, L_2) \preceq (R_1, R_2)$.

If C possesses the strict bottleneck Monge property, then for all $i < k$ and $j < \ell$, we have one of two cases

Case 1: $\max\{c[i,j], c[k,\ell]\} < \max\{c[i,\ell], c[k,j]\}$

Case 2: both $\max\{c[i,j], c[k,\ell]\} = \max\{c[i,\ell], c[k,j]\}$ and $\min\{c[i,j], c[k,\ell]\} \leq \min\{c[i,\ell], c[k,j]\}$

In Case 1, we have $L_1 < R_1$, which implies that $(L_1, L_2) \prec (R_1, R_2)$. In Case 2, 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 was discussed in Section 1, the composition $\oplus$ is strictly compatible with the order relation $\preceq$. Thus, given Lemma 3.1, 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 modelled 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 3.2. *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.*

We only give the motivation for the proof of Theorem 3.2 here: Adapting the proof of Theorem 2.3 to our semigroup, recall that we called the SMAWK algorithm to determine the column minima of the array $A^\ell = \{a^\ell[i,j]\}$, where

$$a^\ell[i,j] = \begin{cases} d^{\ell-1}[i] \oplus c_T[i,j] & \text{if } i < j \\ \infty & \text{otherwise} \end{cases} \quad \text{and} \quad d^\ell[j] = \begin{cases} \min_{1 \le i < j} a^\ell[i,j] & \text{if } \ell > 1 \\ c_T[i,j] & \text{if } \ell = 1 \end{cases}$$

In order to improve the running time of Theorem 2.3 on the semigroup $(T, \oplus, \preceq)$, we need to take a close look at the kinds of comparisons performed in the proof of that theorem. All the work in that proof is done by the SMAWK algorithm (Aggarwal et al. [1]). The key idea is that instead of storing and manipulating an entire tuple, we can simply keep first two element of $a^\ell[i,j]$. The resulting matrix (in the ordered semi-group consisting of unordered pairs, instead of k-tuples) is still algebraically Monge. Thus, the times t_a and t_c are reduced to $O(1)$, and using the proof of Theorem 2.3, we can solve the bottleneck k-edge shortest-path problem in $O(kn)$ time.

By analyzing the algorithm of Larmore and Schieber [15], instead of SMAWK, we similarly obtain:

Theorem 3.3. *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.*

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 the following two graphs with the same vertex and edge sets at G, but with different cost functions: a graph $G'_T = (V, E)$ with edge costs

$$c'[i,j] = \begin{cases} 1 & \text{if } c[i,j] \le T \\ +\infty & \text{otherwise} \end{cases}$$

and a graph $G''_T = (V, E)$ with edge costs

$$c''[i,j] = \begin{cases} -1 & \text{if } c[i,j] \le T \\ +\infty & \text{otherwise} \end{cases}$$

Lemma 4.1. *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. The proof is routine and is given in the journal version.

Lemma 4.2. *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 $k' \le k \le k''$.*

Proof. The proof is ommited here and is given in the journal version.

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 Lemmas 4.2, such a path exists if $k' \le k \le k''$, 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 [15], 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 4.1), our query subroutine runs in $O(n)$ time.

Theorem 4.3. *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 $O(n^{3/2} \lg^2 n)$ time (or in $O(n \lg^2 n)$ time if the problem's cost array is also bitonic[1]).*

Proof. We use the result of Agarwal and Sen [2], who show how to find the d-th smallest entry in an $m \times n$ totally monotone array in $O((m+n)\sqrt{n} \lg n)$ time. For our $n \times n$ totally monotone array, this translates into $O(n^{3/2} \lg n)$ 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 the procedure Binary-Search(C_T, 1, n^2). Binary-Search(C_T, i, j) does the following. We use Agarwal and Sen [2] to find the $\lfloor \frac{j+i}{2} \rfloor$-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 answers "yes," then we call Binary-Search(C_T, i, $\lfloor \frac{j+i}{2} \rfloor$); otherwise we 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 4.2. We compute the unrestricted shortest $1 \hookrightarrow n$ paths P and Q, $|P| = k'$, $|Q| = k''$, in the graphs G'_{τ^*} and G''_{τ^*}.

[1] 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.

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 $i < s < j \leq t$ and $|1 \overset{\mathrm{P}}{\hookrightarrow} t| = |1 \overset{\mathrm{P}}{\hookrightarrow} s|$. Let Q' consist of $1 \overset{\mathrm{P}}{\hookrightarrow} i$, followed by the edge (i, j), followed by $j \overset{\mathrm{Q}}{\hookrightarrow} n$; and P' consist of $1 \overset{\mathrm{Q}}{\hookrightarrow} s$, followed by the edge (s, t), followed by $t \overset{\mathrm{P}}{\hookrightarrow} n$. From Lemma 4.2, 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 $O(\lg n^2(n^{3/2} \lg n + n)) = O(n^{3/2} \lg^2 n)$.

In the case of a bitonic cost array, the selection problem is simpler. The selection algorithm of Frederickson and Johnson [9] 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 $O(n \lg n)$ time. Applying Frederickson and Johnson [9], we can compute the d-th smallest entry in an $n \times n$ bitonic array in $O(n \lg n)$ time. Thus, for cost arrays that satisfy the string bottleneck Monge property and are bitonic, the k-edge shortest-path can be found in $O(n \lg^2 n)$ time.

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

Theorem 4.4. *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 $O((m\sqrt{n} \lg m + n) \lg^2 n)$ time (or in $O(m \lg^2 n + n \lg n)$ 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 Theorem 4.3. The 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. 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 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 such matching. 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 $\min\{c[i,j_i],c[s,\ell]\} \le \min\{c[i,\ell],c[s,j_i]\}$. 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 4.3. We use the algorithm of Agarwal and Sen [2] 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 $O(\lg(mn)((m+n)\sqrt{n}\lg n + n)) = O(n^{3/2}\lg^2 n)$. Similarly, if the cost array is bitonic, then the running time is $O(\lg(mn)(m\lg n + n)) = O(m\lg^2 n + n\lg n)$.

5 A Paragraph-Formation Problem

We discuss a variant of Hirschberg and Larmore's optimal-paragraph-formation problem [11]. A slightly simplified version of their problem is to break a sequence of words $w_1,\ldots,w_n$ into lines in order to form a paragraph. Define $c[i,j]$ to be the cost of a line consisting of words w_i through w_{j-1}. 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, the cost function is $c[i,j] = (|w_{i+1}| + |w_{i+2}| + \ldots + |w_j| - 1 - L)^2$. Their objective is to construct a paragraph minimizing the sum of the paragraph's line costs. 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 $(i,j) \in E$ if and only if $i < j$. 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 paragraph's line costs. 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 [17], as well as to Larmore and Schieber [15].)

If we instead seek to minimize the maximum line cost, 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. We call a line cost function $f[i,j]$ *strictly bitonic* if any sequence of costs $f[i_1,j_1], f[i_2,j_2],\ldots,f[i_s,j_s]$, which satisfies the following conditions for every $1 \le \ell < s$:

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 5.1. *Let $F = \{f[i,j]\}$, where $f[i,j]$ is a line cost function that is strictly bitonic. Then F satisfies the strict bottleneck Monge property.*

Proof. The proof is routine.

Lemma 5.2. *Cost function $c[i,j] = (|w_{i+1}| + |w_{i+2}| + \ldots + |w_j| - 1 - L)^2$ is strictly bitonic.*

Proof. The proof is routine and is given in the full paper version.

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 3.3, 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.

We have thus far considered a simplified version of Hirshberg and Larmore cost function. There are three other factors about paragraph formation which they include in their cost. One is the ability to hyphenate a word at a fixed cost per hyphenation. Another is the fact that in addition to penalizing quadratically lines that differ too much from the ideal line length, there may be some lower and upper limits beyond which the line length is simply not allowed. And finally, the last line in the paragraph should not be penalized for being too short. We now attempt to incorporate these factors into our cost function.

The second factor is fairly easy to incorporate. Suppose that that the maximum and the minimum allowed length for a line are *maxlen* and *minlen*. Then define the cost function to be

$$c[i,j] = \begin{cases} (p[i,j] - 1 - L)^2 & \text{if } minlen \leq p[i,j] \leq maxlen \\ M^{i+j} & \text{otherwise} \end{cases}$$

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 last factor, we would need to define a new cost function

$$c'[i,j] = \begin{cases} 0 & \text{if } j = n \text{ and } p[i,j] \leq L \\ (p[i,j] - 1 - L)^2 & \text{if } minlen \leq p[i,j] \leq maxlen \\ M^{i+j} & \text{otherwise} \end{cases}$$

Unfortunately, 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 3.3, 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:

$$d[n] = \min_{1 \le i \le n-1} \left\{ \begin{array}{ll} d[i] & \text{if } p[i,n] \le L \\ \max\{d[i], (p[i,n]-1-L)^2\} & \text{if } L \le p[i,n] \le \mathit{maxlen} \\ \infty & \text{otherwise} \end{array} \right\}$$

Unfortunately, it is not possible to incorporate the first factor 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 cost function is no longer 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

$$\begin{gathered} \max\left\{(p[i,j]-1-L)^2 + B, (p[k.\ell]-1-L)^2\right\} \\ \le \max\left\{(p[k,j]-1-L)^2 + B, (p[i.\ell]-1-L)^2\right\} \end{gathered}$$

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

We also consider one variation of the cost 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

$$c[i,j] = \begin{cases} (p[i,j]-1-L)^2 & \text{if } p[i,j] \le L \\ M^{i+j} & \text{otherwise} \end{cases}$$

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 [5], 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 show that if a graph's edge cost array possesses the strict bottleneck Monge property, then it is possible to find the graph's bottleneck-shortest pyramidal tour in $O(n \lg^2 n)$ time.

Theorem 6.1. *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 $O(n \lg^2 n)$ time.*

Proof. The proof is given in the full paper version. The basic idea is to reduce the problem to two instances of the on-line monotone matrix searching problem, and solve both of them using the algorithm of [15]. The two on-line processors, each using the on-line matrix searching algorithm, run simultaneously, passing information to each other at each step. The total number of queries needed is $O(n)$.

The reduction is not constant cost, however. A query in the reduced problem may require the computation of the cost of a path from i to j which passes through every intermediate point. We can use $O(n \lg n)$ preprocessing time to create a data structure and then satisfy each query in $O(\lg^2 n)$ time, as given in [12].

References

1. 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.
2. 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.
3. A. Aggarwal, B. Schieber, and T. Tokuyama. Finding a minimum weight k-link path in graphs with Monge property and applications. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 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 Comput. Geom.*, 12:263–280, 1994.
5. R. E. Burkard and W. Sandholzer. Efficiently solvable special cases of bottleneck travelling salesman problems. *Discrete Applied Mathematics*, 32(1):61–76, 1991.
6. R. E. Burkard. Remarks on some scheduling problems with algebraic objective functions. *Operations Research Verfahren*, 32:63–77, 1979.
7. 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.

8. R. E. Burkard and B. Klinz and R. Rudolf. Perspectives of monge properties in optimization. *Discrete Applied Mathematics*, 70:95–161, 1996.
9. 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.
10. H. N. Gabow and R. E. Tarjan. Algorithms for two bottleneck optimization problems. *Journal of Algorithms*, 9(3):411–417, 1988.
11. D. S. Hirschberg and L. L. Larmore. The least weight subsequence problem. *SIAM Journal on Computing*, 16(4):628–638, 1987.
12. D. S. Hirschberg and D. J. Volper. Improved update/query algorithms for the interval valuation problem. *Information Processing Letters*, 24:307–310, 1987.
13. A.J. Hoffman. On simple linear programming problems. In *Convexity, Proc. Symposia in Pure Mathematics*, volume 7, pages 317 – 327, Providence, RI, 1961. American Mathematical Society.
14. B. Klinz, R. Rudolf, and G.J. Woeginger. On the recognition of bottleneck monge matrices. *Discrete Applied Mathematics*, 63:43–74, 1995.
15. 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.
16. E. Seiffart. Algebraic transportation and assignment problems with "Monge-property" and "quasi-convexity". *Discrete Applied Mathematics*, 1993.
17. R. Wilber. The concave least-weight subsequence problem revisited. *Journal of Algorithms*, 9(3):418–425, 1988.