Information Processing Letters 40 (1991) 247–254
North-Holland

13 December 1991

A special case of the n-vertex traveling-salesman problem that can be solved in $O(n)$ time

James K. Park *

Sandia National Laboratories, Division 1423 (Theoretical Computer Science), P.O. Box 5800, Albuquerque, NM 87185-5800, USA

Communicated by M.J. Atallah
Received 15 April 1991
Revised 4 October 1991

Abstract

Park, J.K., A special case of the n-vertex traveling-salesman problem that can be solved in $O(n)$ time, Information Processing Letters 40 (1991) 247–254.

The traveling-salesman problem, though in general NP-hard, possesses several special cases that can be solved in polynomial time. In particular, when the cost array C associated with an n-vertex traveling-salesman problem satisfies what are known as the Demidenko conditions, then a minimum-cost traveling-salesman tour can be computed in $O(n^2)$ time using a simple dynamic-programming algorithm. In this paper, we identify a subset Λ of the set of all cost arrays satisfying the Demidenko conditions, such that for any $C \in \Lambda$, the running time of the aforementioned dynamic-programming algorithm can be reduced to $O(n)$. We obtain this speedup using recently developed techniques for on-line searching in Monge arrays.

Keywords: Design of algorithms, analysis of algorithms, combinatorial problems, the traveling-salesman problem, dynamic programming, Monge arrays

1. Introduction

Given an n-vertex complete directed graph G whose vertices are labeled $1,\ldots,n$ and an $n \times n$ cost array $C = \{c[i, j]\}$ such that the cost of traversing arc (i, j) of G is $c[i, j]$, the *traveling-salesman problem* is that of computing a minimum-cost tour of G that visits each vertex exactly once. Though this famous problem is NP-hard for arbitrary C, there exist several special cases of the traveling-salesman problem, corresponding to restricted sets of cost arrays, that can be solved in polynomial time. Many of these special cases are listed in a survey article written by Gilmore, Lawler, and Shmoys [8].

In this paper, we will focus on one of the special cases described by Gilmore, Lawler, and Shmoys. This special case was first considered by V.M. Demidenko; he identified a set Δ of cost arrays, such that for any $C \in \Delta$, a minimum-cost traveling-salesman tour through the directed graph corresponding to C can be computed in $O(n^2)$ time [1]. The set Δ consists of all cost arrays

* This work was done while the author was a member of MIT's Laboratory for Computer Science and was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-87-K-0825 and the Office of Naval Research under Contract N00014-86-K-0593.

[1] As I do not have easy access to Demidenko's 1979 Russian-language paper describing his result (see [8] for the reference) nor do I read Russian, this paper is based solely on Gilmore, Lawler, and Shmoys's presentation of the result.

satisfying the following conditions: if $1 \leqslant i < j$ and $j + 1 < k \leqslant n$, then

$$c[i, j] + c[j, j+1] + c[j+1, k] \leqslant c[i, j+1] + c[j+1, j] + c[j, k],$$

$$c[j, i] + c[j+1, j] + c[k, j+1] \leqslant c[j+1, i] + c[j, j+1] + c[k, j],$$

$$c[i, j] + c[k, j+1] \leqslant c[i, j+1] + c[k, j],$$

$$c[j, i] + c[j+1, k] \leqslant c[j+1, i] + c[j, k].$$

These conditions, which Gilmore, Lawler, and Shmoys call the *Demidenko conditions*, are depicted graphically in Fig. 1.

To explain why the Demidenko conditions are relevant to the traveling-salesman problem, we first need to introduce the notion of a *pyramidal* traveling-salesman tour. A traveling-salesman tour T of the graph G is said to be pyramidal if (1) the vertices on the path T follows from vertex n to vertex 1 have monotonically decreasing labels, and (2) the vertices on the path T follows from vertex 1 to vertex n have monotonically increasing labels. For example, if G has five vertices labeled 1 through 5, then the tours $5 \to 2 \to 1 \to 3 \to 4 \to 5$ and $5 \to 4 \to 3 \to 2 \to 1 \to 5$ are pyramidal, but the tour $5 \to 2 \to 3 \to 1 \to 4 \to 5$ is not.

Pyramidal tours are interesting because a minimum-cost pyramidal tour through G can always be computed in $O(n^2)$ time using dynamic programming. Gilmore, Lawler, and Shmoys obtain this result as follows. For $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n$, and $i \neq j$, let $E(i, j)$ denote the cost of a minimum-cost pyramidal path from vertex i to vertex j that passes through each vertex in $\{1, \ldots, \max\{i, j\}\}$ exactly once. (A pyramidal path, by analogy with a pyramidal tour, is a path P that can be decomposed into two subpaths P_1 and P_2 such that (1) the vertices on P_1 have monotonically decreasing labels, and (2) the vertices on P_2 have monotonically increasing labels.) Clearly, $E(1, 2) = c[1, 2]$, $E(2, 1) = c[2, 1]$, and the cost of a minimum-cost pyramidal tour is

$$\min\{E(n-1, n) + c[n, n-1], \; E(n, n-1) + c[n-1, n]\}.$$

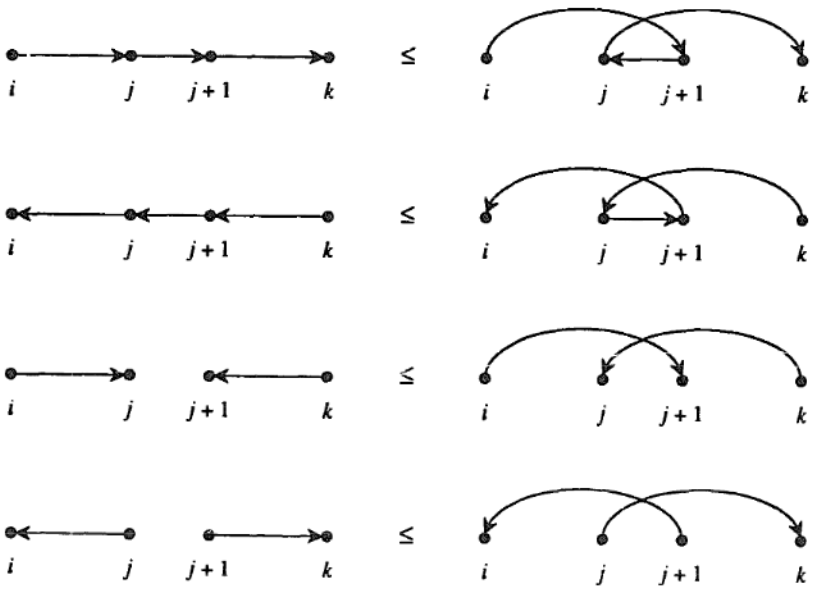

Fig. 1. The Demidenko conditions require that if $1 \leqslant i < j$ and $j + 1 < k \leqslant n$, then in each of the four "comparisons" depicted above, the total cost of the arcs on the left is at most the total cost of the corresponding arcs on the right.

Furthermore, it is not difficult to see that for $i \neq j$ and $\max\{i, j\} > 2$,

$$E(i, j) = \begin{cases} E(i, j-1) + c[j-1, j] \\ \quad \text{if } i < j-1, \\ \min\limits_{1 \leq k < i} \{E(i, k) + c[k, j]\} \\ \quad \text{if } i = j-1, \\ \min\limits_{1 \leq k < j} \{E(k, j) + c[i, k]\} \\ \quad \text{if } i = j+1, \\ E(i-1, j) + c[i, i-1] \\ \quad \text{if } i > j+1. \end{cases}$$

This recurrence can be used to compute all the $E(i, j)$ (and hence the cost of a minimum-cost pyramidal tour) in $O(n^2)$ time; moreover, a minimum-cost tour (and not just its cost) is easily extracted from this computation.

Minimum-cost pyramidal tours and the Demidenko conditions are related by the following theorem, which Gilmore, Lawler, and Shmoys attribute to Demidenko.

Theorem 1.1 (Demidenko). *Let C denote an $n \times n$ cost array, and let G denote the n-vertex complete directed graph corresponding to C. If $C \in \Delta$ (i.e., it satisfies the Demidenko conditions), then some minimum-cost traveling-salesman tour through G is pyramidal.* □

This theorem, together with the aforementioned dynamic-programming algorithm for computing a minimum-cost pyramidal tour, gives an $O(n^2)$-time algorithm for any instance of the traveling-salesman problem whose cost array C is a member of Δ.

In this paper, we identify another set of cost arrays, denoted Γ, such that for any $n \times n$ cost array C in Γ, a minimum-cost pyramidal tour through the n-vertex graph G corresponding to C can be computed in $O(n)$ time. We obtain this result using recently developed algorithms for computing minimal entries in what are known as *Monge* arrays. Our use of Monge-array algorithms to speed up dynamic programming is not a new idea (for example, see [2–4, 6, 10–12]), but our application of these Monge-array techniques to computing minimum-cost pyramidal tours is. An immediate consequence of our pyramidal-tour result is an $O(n)$-time algorithm for any instance of the n-vertex traveling-salesman problem whose cost array C is a member of $\Lambda = \Gamma \cap \Delta$. We remark that this intersection Λ includes all square (i.e., $s \times s$ for some integer s) Monge arrays.

The remainder of this paper is organized as follows. Section 2 defines a Monge array, relates Monge arrays to the Demidenko conditions, and discusses several algorithms for searching in Monge arrays. Section 3 then describes our $O(n)$-time algorithm for computing a minimum-cost pyramidal tour in a graph whose cost array C is a member of Γ. Finally, Section 4 contains a few concluding remarks.

2. Searching in Monge arrays

This section describes previous work relevant to our use of Monge arrays in speeding up the $O(n^2)$-time dynamic-programming algorithm for computing minimum-cost pyramidal tours. We begin by defining a Monge array and proving that every square Monge array satisfies the Demidenko conditions. We then discuss several recently developed algorithms for computing minimal entries in Monge arrays and show how these algorithms can be used in a dynamic-programming context.

An $m \times n$ array $A = \{a[i, j]\}$ is called *Monge* if it satisfies the following condition: if $1 \leq i < k \leq m$ and $1 \leq j < l \leq n$, then

$$a[i, j] + a[k, l] \leq a[i, l] + a[k, j].$$

(Equivalently, A is Monge if for all i and j such that $1 \leq i < m$ and $1 \leq j < n$, we have

$$a[i, j] + a[i+1, j+1] \leq a[i, j+1] + a[i+1, j];$$

the equivalence follows from a simple inductive argument.) Note that every square Monge array satisfies the Demidenko conditions, as the following lemma shows.

Lemma 2.1. *If an $n \times n$ array A is Monge, then it satisfies the Demidenko conditions.*

Proof. Consider any i, j, and k such that $1 \leqslant i < j$ and $j + 1 < k \leqslant n$. The third and fourth Demidenko conditions follow immediately from the definition of a Monge array. As for the first Demidenko conditions, A's Mongité implies

$$a[j, j] + a[j+1, j+1] \leqslant a[j, j+1] + a[j+1, j],$$

$$a[i, j] + a[j, j+1] \leqslant a[i, j+1] + a[j, j],$$

and

$$a[j, j+1] + a[j+1, k] \leqslant a[j, k] + a[j+1, j+1].$$

Summing these three inequalities and canceling yields the first Demidenko condition:

$$a[i, j] + a[j, j+1] + a[j+1, k] \leqslant a[i, j+1] + a[j+1, j] + a[j, k].$$

The second Demidenko condition follows in a similar fashion. □

Monge arrays have several important properties. First, every subarray of a Monge array A (corresponding to a subset of A's rows and columns) is also Monge. Furthermore, the *row minima* of a Monge array A are highly structured: if we let $c(i)$ denote the column of A containing the leftmost minimum entry in row i of A (which we call A's ith row minimum), then $c(1) \leqslant c(2) \leqslant \cdots \leqslant c(m)$. Similarly, if we let $r(j)$ denote the row of A containing the uppermost minimum entry in column j of A (which we call A's jth column minimum), then $r(1) \leqslant r(2) \leqslant \cdots \leqslant r(n)$.

These properties allow us to find the row and column minima of a Monge array quite efficiently. In particular, we need only examine $O(m + n)$ of the mn entries in an $m \times n$ Monge array A to locate a minimum entry in each of A's rows (or columns). More generally, we can find A's row (or column) minima in $O(m + n)$ time, provided any particular entry of A can be looked up (or computed) in constant time. This result is due to Aggarwal, Klawe, Moran, Shor, and Wilber [1], who considered a slight variant of this array-searching problem in the context of several problems from computational geometry and VLSI river routing. We call this array-searching problem an *off-line* problem, since every entry of A is available at any time.

In developing a linear-time algorithm for the concave least-weight subsequence problem, Wilber [12] extended the algorithm of Aggarwal et al. to a dynamic-programming setting. Specifically, he gave an algorithm for the following *on-line* variant of the Monge-array column-minima problem. Let $W = \{w[i, j]\}$ denote an $n \times n$ Monge array, where any entry of W can be computed in constant time. Furthermore, let $A = \{a[i, j]\}$ denote the $n \times n$ array defined by

$$a[i, j] = \begin{cases} E(i) + w[i, j] & \text{if } i < j, \\ +\infty & \text{if } i \geqslant j, \end{cases}$$

where $E(1)$ is given and for $1 < i \leqslant n$, $E(i)$ is some function that can be computed in constant time from the ith column minimum of A. Using the Mongité of A, which follows from its definition, Wilber showed that the column minima of A (and hence $E(2), \ldots, E(n)$) can be computed in $O(n)$ time. This problem is called an on-line problem because certain inputs to the problem (i.e., entries of A) are available only after certain outputs of the problem (i.e., column minima of A) have been calculated.

In a subsequent paper dealing with the modified string-editing problem, Eppstein [4] generalized Wilber's result, showing that the column minima of A can be computed in $O(n)$ time even when his algorithm is restricted to computing $E(1), \ldots, E(n)$ *in order*, i.e., $E(i)$ can be computed only after $E(1), \ldots, E(j-1)$ have been computed. This result is significant in that it allows the computation of $E(1), \ldots, E(n)$ to be interleaved with the computation of some other sequence $F(1), \ldots, F(n)$ such that $E(j)$ depends on $F(1), \ldots, F(j-1)$ and $F(j)$ depends on $E(1), \ldots, E(j)$.

Finally, though Eppstein's algorithm is sufficient for the purposes of this paper, we note that Galil and K. Park [6], Klawe [10], and Larmore and Schieber [11] independently extended Eppstein's result a step further, showing that as long as $a[i, j]$ can be computed in constant time once the first through ith column minima of A are known, the column minima of A can still be computed in $O(n)$ time.

3. Finding minimum-cost pyramidal tours

In this section, we identify a set Γ of cost arrays such that for any $n \times n$ cost array $C \in \Gamma$, we can find a minimum-cost pyramidal tour through the directed graph corresponding to C in $O(n)$ time. This result implies that for any cost array C in $\Lambda = \Gamma \cap \Delta$, we find a minimum-cost traveling-salesman tour through the graph corresponding to C in $O(n)$ time.

The set Γ consists of all $n \times n$ cost arrays $C = \{c[i, j]\}$ satisfying the following condition: if $1 \leqslant i < n$, $1 \leqslant j < n$, and either $i \leqslant j - 3$ or $i \geqslant j + 3$, then

$$c[i, j] + c[i + 1, j + 1] \leqslant c[i, j + 1] + c[i + 1, j].$$

Note that Γ is a superset of the set of all square Monge arrays, since an $n \times n$ Monge array C satisfies the above inequality for *all* i and j satisfying $1 \leqslant i < n$ and $1 \leqslant j < n$, including those i and j such that $j - 2 \leqslant i \leqslant j + 2$.

Our algorithm for computing a minimum-cost pyramidal tour is based on a slight variation of Gilmore, Lawler, and Shmoys's dynamic-programming recurrence for the problem. For $1 \leqslant j < n$, let $F(j)$ denote the cost of a minimum-cost pyramidal path from vertex j to vertex $j + 1$ that passes through each vertex in $\{1, \ldots, j + 1\}$ exactly once. (In terms of Gilmore, Lawler, and Shmoys's notation, $F(j) = E(j, j + 1)$.) Similarly, for $1 \leqslant j < n$, let $G(j)$ denote the cost of a minimum-cost pyramidal path from vertex $j + 1$ to vertex j that again passes through each vertex in $\{1, \ldots, j + 1\}$ exactly once. (In terms of Gilmore, Lawler, and Shmoys's notation, $G(j) = E(j + 1, j)$.) Clearly, $F(1) = c[1, 2]$, $G(1) = c[2, 1]$, and the cost of a minimum-cost pyramidal tour through G is

$$\min\{F(n - 1) + c[n, n - 1], G(n - 1) + c[n - 1, n]\}.$$

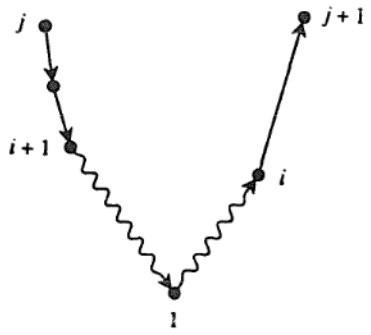

Fig. 2. The shortest pyramidal path from vertex j to vertex $j + 1$ passing through vertices $1, \ldots, j + 1$ can be decomposed into three parts: (1) an edge $(i, j + 1)$ such that $1 \leqslant i < j$, (2) a path from vertex j to vertex $i + 1$ passing through vertices $i + 1, \ldots, j$ *in strictly descending order*, and (3) the shortest pyramidal path from vertex $i + 1$ to vertex i passing through vertices $1, \ldots, i + 1$.

Now consider any pyramidal path P from j to $j + 1$ that achieves $F(j)$, and let $(i, j + 1)$ denote the last arc traversed by P. We must have $1 \leqslant i < j$, as suggested in Fig. 2. Moreover, if $i + 1 < j$, then, since P is pyramidal, the first $j - (i + 1)$ arcs traversed by P must be $(j, j - 1), (j - 1, j - 2), \ldots, (i + 2, i + 1)$. Thus,

$$F(j) = \min_{1 \leqslant i < j} \left\{ G(i) + c[i, j + 1] + \sum_{l=i+1}^{j-1} c[l + 1, l] \right\}.$$

By a similar argument, we must also have

$$G(j) = \min_{1 \leqslant i < j} \left\{ F(i) + c[j + 1, i] + \sum_{l=i+1}^{j-1} c[l, l + 1] \right\}.$$

Using this recurrence to compute $F(n - 1)$ and $G(n - 1)$ in the naive fashion takes $O(n^2)$

time. However, if the cost array C is a member of Γ, then we can apply the on-line array-searching techniques mentioned in the previous section. To see why these techniques are applicable, consider the $(n-1)\times(n-1)$ array $A=\{a[i,j]\}$ where

$$a[i,j]=\begin{cases} G(i)+c[i,j+1]+\sum_{l=i+1}^{j-1} c[l+1,l] \\ \qquad \text{if } i<j, \\ +\infty \quad \text{if } i\geqslant j, \end{cases}$$

and the $(n-1)\times(n-1)$ array $B=\{b[i,j]\}$ where

$$b[i,j]=\begin{cases} F(i)+c[j+1,i]+\sum_{l=i+1}^{j-1} c[l,l+1] \\ \qquad \text{if } i<j, \\ +\infty \quad \text{if } i\geqslant j. \end{cases}$$

Clearly,

$$F(j)=\min_{1\leqslant i\leqslant n-1} a[i,j],$$

i.e., $F(j)$ is the jth column minimum of A, and

$$G(j)=\min_{1\leqslant i\leqslant n-1} b[i,j],$$

i.e., $G(j)$ is the jth column minimum of B. Moreover, $C\in\Gamma$ implies both A and B are Monge, as the following lemma shows.

Lemma 3.1. *Both A and B are Monge if and only if C is a member of Γ.*

Proof. We begin by showing that A is Monge if and only if

$$c[i,j]+c[i+1,j+1] \leqslant c[i,j+1]+c[i+1,j]$$

for all i and j such that $1\leqslant i<j-2<n-2$.

From the definition of a Monge array, A is Monge if and only if

$$a[i,j]+a[i+1,j+1] \leqslant a[i,j+1]+a[i+1,j] \qquad (1)$$

for all i and j such that $1\leqslant i<n-1$ and $1\leqslant j<n-1$. Furthermore, if $i+1\geqslant j$, then $a[i+1,j]=+\infty$, which immediately implies (1). Thus, A is Monge if and only if (1) holds for all i and j such that $1\leqslant i<j-1<n-2$.

Now consider any i and j such that $1\leqslant i<j-1<n-2$. From the definition of A, we have

$$\begin{aligned} &(a[i,j]+a[i+1,j+1]) \\ &-(a[i,j+1]+a[i+1,j]) \\ &\quad = G(i)+c[i,j+1]+\sum_{l=i+1}^{j-1} c[l+1,l] \\ &\qquad +G(i+1)+c[i+1,j+2] \\ &\qquad +\sum_{l=i+2}^{j} c[l+1,l] \\ &\qquad -G(i)-c[i,j+2]-\sum_{l=i+1}^{j} c[l+1,l] \\ &\qquad -G(i+1)-c[i+1,j+1] \\ &\qquad -\sum_{l=i+2}^{j-1} c[l+1,l] \\ &\quad = (c[i,j+1]+c[i+1,j+2]) \\ &\qquad -(c[i,j+2]+c[i+1,j+1]). \end{aligned}$$

Thus, A is Monge if and only if

$$c[i,j]+c[i+1,j+1] \leqslant c[i,j+1]+c[i+1,j]$$

for all i and j such that $1\leqslant i<j-2<n-2$.

In a similar fashion, we can show that B is Monge if and only if

$$c[i,j]+c[i+1,j+1] \leqslant c[i,j+1]+c[i+1,j]$$

for all i and j such that $1\leqslant j<i-2<n-2$. Thus, both A and B are Monge if and only if C is a member of G. □

Now suppose we precompute

$$\sum_{l=1}^{j-1} c[l+1,l]$$

and

$$\sum_{l=1}^{j-1} c[l, l+1]$$

for all j in the range $2 \leqslant j < n$. This preprocessing requires $O(n)$ time. Moreover, it allows any entry $a[i, j]$ of A to be computed in constant time from $G(i)$, the ith column minimum of B, and any entry $b[i, j]$ of B to be computed in constant time from $F(i)$, the ith column minimum of A. Thus, by interleaving the computation of A's column minima with the computation of B's column minima, as discussed in Section 2, we can use Eppstein's on-line array-searching algorithm to compute $F(2), \ldots, F(n-1)$ and $G(2), \ldots, G(n-1)$ in $O(n)$ time.

Since a minimum-cost pyramidal tour (and not just its cost) is easily extracted from the computation of $F(2), \ldots, F(n-1)$ and $G(2), \ldots, G(n-1)$, we have the following theorem and corollary.

Theorem 3.2. *Let C denote an $n \times n$ cost array, and let G denote the n-vertex complete directed graph corresponding to C. If $C \in \Gamma$, then a minimum-cost pyramidal tour through G can be computed in $O(n)$ time.* □

Corollary 3.3. *Let C denote an $n \times n$ cost array, and let G denote the n-vertex complete directed graph corresponding to C. If $C \in \Gamma \cap \Delta$, then a minimum-cost traveling-salesman tour through G can be computed in $O(n)$ time.* □

4. Concluding remarks

In this paper, we have described an $O(n)$-time algorithm for computing a minimum-cost pyramidal tour through any directed graph whose cost array is a member of Γ. An immediate consequence of this result is an $O(n)$-time algorithm for computing a minimum-cost traveling-salesman tour through any directed graph whose cost array is a member of $\Lambda = \Gamma \cap \Delta$.

As Λ contains all square Monge arrays, the latter result is in some sense analogous to a result Hoffman [9] obtained for the transportation problem. Hoffman showed (among other things) that if the $m \times n$ cost array associated with an instance of the transportation problem is Monge, then a simple greedy algorithm solves the transportation problem in $O(m+n)$ time. We have shown that Monge arrays give rise to a similarly "easy" special case of the traveling-salesman problem.

We conclude by mentioning two applications of this paper's $O(n)$-time algorithm for computing pyramidal tours. These applications are somewhat unsatisfying, as there are simpler algorithms for both problems (see below), but they do provide examples of interesting cost arrays in Λ.

The first application comes from computational geometry (and provides an example of an interesting non-Monge array in C). Consider a convex polygon P in the plane with vertices $v_1, \ldots, v_n$ in clockwise order. Corresponding to P is a complete directed graph G on P's vertices with cost array $C = \{c[i, j]\}$, where $c[i, j]$ is the Euclidean distance between vertices v_i and v_j of P. It is not hard to verify that C satisfies both the Demidenko conditions and the conditions defining Γ, i.e., $C \in \Lambda$. Thus, the traveling-salesman problem corresponding to P can be solved in $O(n)$ time. However, for cost arrays of this form, an $O(n)$-time algorithm is not particularly impressive, as $1 \to 2 \to \cdots \to n \to 1$ (i.e., the tour traversing the perimeter of P in clockwise order) is always a minimum-cost traveling-salesman tour for G.

The second application of this paper's pyramidal-tour algorithm involves a special case of the Gilmore–Gomory traveling-salesman problem. In [7], Gilmore and Gomory considered instances of the traveling-salesman problem characterized by cost arrays $C = \{c[i, j]\}$ of the following form: for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant n$,

$$c[i, j] = \begin{cases} \int_{b[i]}^{a[j]} f(x)\, dx & \text{if } a[j] \geqslant b[i], \\ \int_{a[j]}^{b[i]} g(x)\, dx & \text{if } a[j] < b[i], \end{cases}$$

where the vectors $A = \{a[j]\}$ and $B = \{b[i]\}$ are arbitrary and the functions $f(\cdot)$ and $g(\cdot)$ satisfy

$f(x) + g(x) \geqslant 0$ for all x. (Without loss of generality, we assume $b[1] \leqslant b[2] \leqslant \cdots \leqslant b[n]$.) Gilmore and Gomory showed that, given a permutation φ such that $a[\varphi(1)] \leqslant a[\varphi(2)] \leqslant \cdots \leqslant a[\varphi(n)]$, a minimum-cost traveling-salesman tour through their graph can be computed in $O(n + T_{MST}(n))$ time, where $T_{MST}(n)$ is the maximum over all m in the range $1 \leqslant m \leqslant n$ of the time required to find a minimum-cost spanning tree for a (weighted) undirected graph with m vertices and n edges. As the best bound known for the n-edge minimum-spanning-tree problem is $T_{MST}(n) = O(n \lg^* n)$ [5], Gilmore and Gomory's algorithm for their special case of the traveling-salesman problem runs in (every so slightly) superlinear time.

Now suppose φ is the identity permutation I (i.e., $a[1] \leqslant a[2] \leqslant \cdots \leqslant a[n]$). It is not difficult to verify that this assumption implies C is Monge; hence, this paper's pyramidal-tour algorithm gives a linear-time algorithm for the $\varphi = I$ special case of Gilmore and Gomory's problem. However, a closer examination of Gilmore and Gomory's algorithm reveals that the minimum-spanning-tree problem that they must solve for the $\varphi = I$ special case is trivial; thus, their approach ends up yielding a simpler $O(n)$-time algorithm for the problem.

Acknowledgment

For helpful comments on an earlier version of this paper, I would like to thank Eric Schwabe, Ron Shamir, Cliff Stein, Joel Wein, and the two anonymous referees who reviewed this paper for *Information Processing Letters*.

References

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

[2] A. Aggarwal and J.K. Park, Sequential searching in multidimensional monotone arrays, Research Rept. RC 15128, IBM T.J. Watson Research Center, November 1989; Submitted to *J. Algorithms*; Portions of this paper appear in: *Proc. 29th Ann. IEEE Symp. on Foundations of Computer Science* (1988) 497–512.

[3] A. Aggarwal and J.K. Park, Improved algorithms for economic lot-size problems, *Oper. Res.* (1991), to appear; An earlier version of this paper appears as Research Rept. RC 15626, IBM T.J. Watson Research Center, March 1990.

[4] D. Eppstein, Sequence comparison with mixed convex and concave costs, *J. Algorithms* **11** (1) (1990) 85–101.

[5] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM* **34** (3) (1987) 596–615.

[6] Z. Galil and K. Park, A linear-time algorithm for concave one-dimensional dynamic programming, *Inform. Process. Lett.* **33** (6) (1990) 309–311.

[7] P.C. Gilmore and R.E. Gomory, Sequencing a one state-variable machine: A solvable case of the traveling salesman problem, *Oper. Res.* **12** (5) (1964) 655–679.

[8] P.C. Gilmore, E.L. Lawler and D.B. Shmoys, Well-solved special cases, in: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, eds., *The Traveling Salesman Problem* (Wiley, New York, 1985) 87–143.

[9] A.J. Hoffman, On simple linear programming problems, in: V. Klee, ed., *Convexity: Proc. Seventh Symp. in Pure Mathematics of the AMS*, Proceedings of Symposia in Pure Mathematics **7** (American Mathematical Society, Providence, RI, 1963) 317–327.

[10] M.M. Klawe, A simple linear time algorithm for concave one-dimensional dynamic programming, Tech. Rept. 89-16, University of British Columbia, Vancouver, 1989.

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

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