SIAM J. COMPUT.
Vol. 29, No. 1, pp. 65–80

AN ALGORITHM FOR SHORTEST PATHS IN BIPARTITE DIGRAPHS WITH CONCAVE WEIGHT MATRICES AND ITS APPLICATIONS*

XIN HE† AND ZHI-ZHONG CHEN‡

Abstract. The traveling salesman problem on an n-point convex polygon and the minimum latency tour problem for n points on a straight line are two basic problems in graph theory and have been studied in the past. Previously, it was known that both problems can be solved in $O(n^2)$ time. However, whether they can be solved in $o(n^2)$ time was left open by Marcotte and Suri [*SIAM J. Comput.*, 20 (1991), pp. 405–422] and Afrati et al. [*Informatique Theorique Appl.*, 20 (1986), pp. 79–87], respectively.

In this paper we show that both problems can be solved in $O(n \log n)$ time by reducing them to the following problem: Given an edge-weighted complete bipartite digraph $G = (X, Y, E)$ with $X = \{x_0, \ldots, x_n\}$ and $Y = \{y_0, \ldots, y_m\}$, we wish to find the shortest path from x_0 to x_n in G. This new problem requires $\Omega(nm)$ time to solve in general, but we show that it can be solved in $O(n + m \log n)$ time if the weight matrices A and B of G are both concave, where for $0 \le i \le n$ and $0 \le j \le m$, $A[i, j]$ and $B[j, i]$ are the weights of the edges (x_i, y_j) and (y_j, x_i) in G, respectively. As demonstrated in this paper, the new problem is a powerful tool and we believe that it can be used to solve more problems.

Key words. graph algorithm, shortest path, traveling salesman problem, minimum latency tour problem, concave matrix

AMS subject classifications. 68Q25, 68R10

PII. S0097539797322255

1. Introduction.

The traveling salesman problem (TSP) is a classical problem of combinatorial optimization. It has been the testing ground of many new algorithmic ideas during the past half-century: dynamic programming, linear programming, genetic algorithms, etc. The TSP is NP-hard and even nonapproximatable. This has motivated researchers to look at its special cases. It turns out that various special cases of the TSP remain NP-hard but are approximatable. Among the special cases solvable in polynomial time, the TSP for points on a convex polygon is well known. In this special case, we are given a set S of n points on the boundary of a convex polygon C and two points x and y in S and are requested to compute a shortest tour starting at x, visiting all the points in $S - \{x, y\}$, and ending at y. Here, the distance between two points in S is the Euclidean distance between them. This special case can be solved in $O(n^2)$ time via dynamic programming [10], but whether it can be solved in $o(n^2)$ time was an open question (posed in [10]). In this paper, we give an affirmative answer to this open question. More specifically, we show that the TSP for points on a convex polygon can be solved in $O(n \log n)$ time.

The minimum latency problem (MLP) is as follows: We are given a metric space M on n points $\{x_1, \ldots, x_n\}$ and are requested to compute a tour in M starting at x_1

*Received by the editors June 4, 1997; accepted for publication (in revised form) January 28, 1998; published electronically September 14, 1999. A preliminary version of this work was presented as *Shortest path in complete bipartite digraph problem and its applications*, 8th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 1997.

http://www.siam.org/journals/sicomp/29-1/32225.html

†Department of Computer Science, State University of New York at Buffalo, Buffalo, NY 14260 (xinhe@cse.buffalo.edu). The research of this author was supported in part by NSF grant CCR-9205982.

‡Department of Mathematical Sciences, Tokyo Denki University, Hatoyama, Saitama 350-03, Japan.

which minimizes the sum of the arrival times at the n points. More precisely, if T is a tour starting at x_1 we say that the *latency* of x_i with respect to T is the distance traveled in T before reaching x_i; and the latency of T is the sum of the latencies of the n points. The goal is then to find a tour of minimum latency. The MLP is a well-studied problem in the operations research literature, where it is also known as the "delivery-man problem" and the "traveling repairman problem" (see [5] for more discussions and references). Although it looks similar to the TSP, the MLP is very different from the TSP in nature [5]. Generally, the MLP is NP-complete [5]. Even for points on a tree or on a convex polygon, it is not known whether the MLP is in P or NP-complete [5]. The case where points are on a straight line was considered in [1, 5]. This case is interesting since it is exactly the following *disk head scheduling problem*: A disk head moves along a straight line L. The head must visit a set of n points on L in order to satisfy disk access requests. The time needed to travel is proportional to the distance being traveled. Once the head reaches a point, the disk access time can be ignored (since the disk rotating speed is much higher than the head moving speed). We want to find a tour of the head such that the average delay (or equivalently, the total delay) of all requests is minimized. The MLP for this special case can be solved in $O(n^2)$ time via dynamic programming [1, 5]. However, whether it can be solved in $o(n^2)$ time was an open question [1]. In this paper, we answer this question in the affirmative by giving an $O(n \log n)$-time algorithm for it.

We obtain the two results mentioned above by efficient reductions to a *single* problem called the *shortest path in bipartite digraph* (SPBD) problem, which is defined as follows. Let $G = (X, Y, E)$ be a complete bipartite digraph with $X = \{x_0, x_1, \ldots, x_n\}$ and $Y = \{y_0, y_1, \ldots, y_m\}$. Each edge $e \in E$ is associated with a real-valued weight $w(e)$. We use $x_i \to y_j$ and $y_j \to x_i$ to denote the edges. Let $A[0..n, 0..m]$ be the matrix with $A[i, j] = w(x_i \to y_j)$ and $B[0..m, 0..n]$ be the matrix with $B[i, j] = w(y_i \to x_j)$. The weight of a (directed) path P in G is defined as $w(P) = \sum_{e \in P} w(e)$. Given such a digraph G, the SPBD problem is to find a path P in G from x_0 to x_n such that $w(P)$ is minimized. For arbitrary weight matrices, we must examine all the edges of G in order to find the shortest path. Thus we need at least $\Omega(nm)$ time to solve the problem. A matrix $M[0..n, 0..m]$ is called *concave* if the following hold:

$$M[i_1, j_1] + M[i_2, j_2] \le M[i_2, j_1] + M[i_1, j_2] \quad \text{for} \quad 0 \le i_1 \le i_2 \le n \quad \text{and} \quad 0 \le j_1 \le j_2 \le m. \tag{1.1}$$

Concave matrices were first discussed in [12] and have been very successfully used in solving various problems (see [2, 3, 4, 6, 7, 8, 9, 11, 12, 13] and the references cited within). Given two matrices $A[0..n, 0..m]$ and $B[0..m, 0..n]$, the product matrix $W[0..n, 0..n] = A \times B$ is defined by

$$W[i, j] = \min_{0 \le k \le m} (A[i, k] + B[k, j]). \tag{1.2}$$

For the SPBD problem we require that G contains no negative cycles since otherwise the shortest path of G is not well defined. If both A and B are concave, as we will prove later, this requirement is satisfied if all the entries on the main diagonal of the product matrix $W = A \times B$ are nonnegative. In section 4 we will prove the following theorem.

THEOREM 1.1. *Given two concave matrices A and B such that all the entries on the main diagonal of the product matrix $W = A \times B$ are nonnegative, the SPBD problem defined by A and B can be solved in $O(n + m \log n)$ time.*

Even for this special case, no algorithm with $o(nm)$ running time was previously known. In this theorem we assume that the matrices A and B are not explicitly given. Rather, an entry is computed in constant time when it is needed. This is true when we apply our SPBD algorithm to solve the TSP and the MLP.

The rest of this paper is organized as follows. In section 2 we present the reduction from the TSP for points on a convex polygon to the SPBD problem. In section 3 we reduce the MLP for points on a straight line to the SPBD problem. In section 4 we prove Theorem 1 by giving an $O(n + m \log n)$-time algorithm for the SPBD problem. Section 5 concludes the paper.

2. The TSP for points on a convex polygon. Let C be a convex polygon and Z be the set of the corner vertices of C. The members of Z will be called *points*. For two points z_1 and z_2 in Z, let $d(z_1, z_2)$ denote the Euclidean distance between z_1 and z_2. The points in Z induce an edge-weighted complete graph G_Z, where the weight on each edge $\{z_1, z_2\}$ is $d(z_1, z_2)$. We identify each edge $\{z_1, z_2\}$ with the line segment whose endpoints are z_1 and z_2. The weight of a path P in G_Z is the sum of the weights on the edges in P and is denoted by $w(P)$. Fix two points x and y in Z. Hereafter, a Hamiltonian path in G_Z always means one from x to y. Our goal is to compute an optimal Hamiltonian path in G_Z, i.e., a Hamiltonian path in G_Z of minimum weight. An easy geometric argument shows that every optimal path P must be simple, i.e., no two edges of P cross each other.

Let $x = x_0, x_1, \ldots, x_n = y$ be the points in Z that lie on the boundary of C from x to y in the clockwise order. Let C_X be the portion of the boundary of C which starts at x, includes $x_1, \ldots, x_{n-1}$, and ends at y. Similarly, let $x = y_0, y_1, \ldots, y_m = y$ be the points in Z that lie on the boundary of C from x to y in the counterclockwise order. Let C_Y be the portion of the boundary of C which starts at x, includes $y_1, \ldots, y_{m-1}$, and ends at y. For $0 \le i < j \le n$ let $x_i \xrightarrow{X} x_j$ denote the portion of C_X from x_i to x_j. For $0 \le i < j \le m$ let $y_i \xrightarrow{Y} y_j$ denote the portion of C_Y from y_i to y_j.

Let P be an optimal Hamiltonian path from $x_0 = y_0$ to $x_n = y_m$ in G_Z. Depending on whether the first and the last edges of P are in C_X or in C_Y, there are four possibilities. We assume that both the first and the last edges of P are in C_Y. Then P must be of the following form:

$$\begin{aligned} x_{i_0} = x_0 = y_0 \xrightarrow{Y} y_{j_1} \to x_1 \xrightarrow{X} x_{i_1} \to y_{j_1+1} \xrightarrow{Y} y_{j_2} \to x_{i_1+1} \xrightarrow{X} \cdots \xrightarrow{X} x_{i_t} \\ = x_{n-1} \to y_{j_t+1} \xrightarrow{Y} y_m = x_n \end{aligned}$$

for some $0 < j_1 < j_2 < \cdots < j_t < m - 1$ and $0 = i_0 < i_1 < i_2 < \cdots < i_t = n - 1$. (See Figure 2.1. In Figure 2.1(b) the points in C_X and C_Y are drawn on two vertical lines for the sake of clarity.) We use the following *dummy path* P' to represent P (the edges of P' are shown as dashed lines in Figure 2.1(b)):

$$P' : \ x_{i_0}(= x_0) \to y_{j_1} \to x_{i_1} \to y_{j_2} \to x_{i_2} \to \cdots \to y_{j_{t-1}} \to x_{i_{t-1}} \to y_{j_t} \to x_{i_t}(= x_{n-1}).$$

The edges $x_{i_{l-1}} \to y_{j_l}$ and $y_{j_l} \to x_{i_l}$ in P' are called *dummy edges*. P is completely specified by P'.

For each dummy edge $x_{i_{l-1}} \to y_{j_l}$ in P', the edge $x_{i_{l-1}} \to x_{i_{l-1}+1}$ is not in P, while the edge $y_{j_l} \to x_{i_{l-1}+1}$ is in P. For each dummy edge $y_{j_l} \to x_{i_l}$ in P', the edge $y_{j_l} \to y_{j_l+1}$ is not in P, while the edge $x_{i_l} \to y_{j_l+1}$ is in P. This motivates the following definition of the weights of dummy edges $x_i \to y_j$ and $y_j \to x_i$ given in the matrices $A[0..n-1, 0..m-1]$ and $B[0..m-1, 0..n-1]$:

$$A[i, j] = w(x_i \to y_j) = d(x_{i+1}, y_j) - d(x_i, x_{i+1}), \tag{2.1}$$

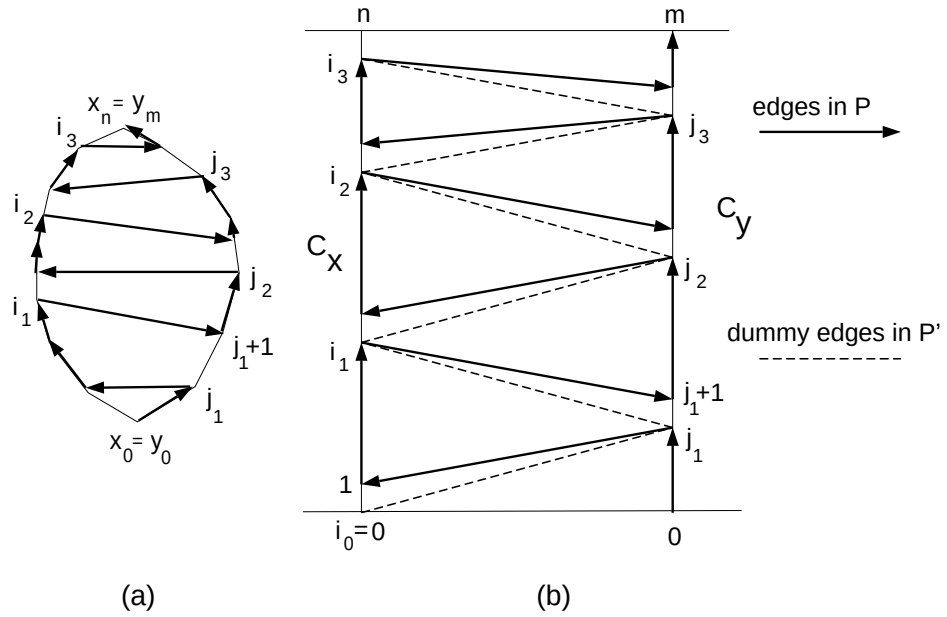

FIG. 2.1. (a) *An optimal path P in a convex polygon;* (b) *a simplified representation of P.*

$$B[j,i] = w(y_j \to x_i) = d(y_{j+1}, x_i) - d(y_j, y_{j+1}). \tag{2.2}$$

Note that $A[0,0] = B[0,0] = 0$. Let $S_X = \sum_{i=1}^{n-1} d(x_{i-1}, x_i)$ and $S_Y = \sum_{j=1}^{m} d(y_{j-1}, y_j)$. It is easy to verify that the total weight of P is

$$w(P) = S_X + S_Y + \sum_{l=1}^{t} A[i_{l-1}, j_l] + \sum_{l=1}^{t} B[j_l, i_l]. \tag{2.3}$$

Although the above discussion is carried out by assuming that the first and the last edges of P are in C_Y, it also applies to other cases. For example, if the first edge of P is in C_X, we let $j_1 = 0$. If the last edge of P is in C_X, we let $j_t = m - 1$. It is easy to verify that (2.3) is valid for these cases, too.

Since the term $S_X + S_Y$ in (2.3) is fixed for any P, to minimize $w(P)$ we need only to minimize the *reduced weight* $w(P')$ defined as follows:

$$w(P') = \sum_{l=1}^{t} A[i_{l-1}, j_l] + \sum_{l=1}^{t} B[j_l, i_l]. \tag{2.4}$$

Let $G = (X, Y, E)$ be the complete bipartite digraph with $X = \{x_0, x_1, \ldots, x_{n-1}\}$ and $Y = \{y_0, y_1, \ldots, y_{m-1}\}$ and the weight matrices A and B. Then a dummy path P' with minimum reduced weight $w(P')$ is exactly a shortest path in G from x_0 to x_{n-1}.

For $0 \le i < i' \le n-1$ and $0 \le j < j' \le m-1$, by the definition of A and the fact that C is a convex polygon, we have $A[i,j] + A[i',j'] - A[i,j'] - A[i',j] = d(x_{i+1}, y_j) + d(x_{i'+1}, y_{j'}) - d(x_{i+1}, y_{j'}) - d(x_{i'+1}, y_j) < 0$.

Thus A is concave. Similarly, we can show that B is also concave. Let $W = A \times B$. Then

$$W[i,i] = \min_{0 \le j \le m-1} [d(x_{i+1}, y_j) - d(x_i, x_{i+1}) + d(y_{j+1}, x_i) - d(y_j, y_{j+1})].$$

By the quadrangle inequality the expression within the min sign is > 0 for each j. Thus $W[i,i] > 0$ for all $0 \le i \le n-1$. By Theorem 1.1 we have the following theorem.

THEOREM 2.1. *The traveling salesman problem for points on an N-point convex polygon can be solved in $O(N \log N)$ time.*

3. The MLP for points on a straight line. Consider a set S of $n+1$ points, a symmetric distance matrix $d[0..n, 0..n]$, and a tour T which visits the points of S in some order. The *latency* of a point $p \in S$ with respect to T is the length of the tour from the starting point to the first occurrence of p. More precisely, suppose that T visits the points in S in the order $p_0, p_1, \ldots, p_n$ starting at p_0. Let $d(p_{i-1}, p_i)$ be the distance traveled along T between p_{i-1} and p_i. Then the latency of p_i is $w(p_i) = \sum_{j=1}^{i} d(p_{j-1}, p_j)$. The *total latency* $w(T)$ of T is the sum of the latencies of all the points in S: $w(T) = \sum_{i=1}^{n} w(p_i)$. Or, equivalently,

$$w(T) = \sum_{k=1}^{n} d(p_{k-1}, p_k)(n-k+1). \tag{3.1}$$

We wish to find a tour T with minimum $w(T)$. In this section we show that the MLP for points on a straight line can be reduced to the SPBD problem and solved in $O(n \log n)$ time.

Let $S = \{x_n, x_{n-1}, \ldots, x_1, x_0 = y_0 = 0, y_1, y_2, \ldots, y_m\}$ be a set of $N = n+m+1$ distinct points on the real line from left to right. We overload x_i (and y_j) to denote both a point and the distance from it to the origin. The tour starts at the point 0. Define

$$w(T_X) = \sum_{k=1}^{n} (x_k - x_{k-1})(n-k+1),$$
$$w(T_Y) = \sum_{k=1}^{m} (y_k - y_{k-1})(m-k+1).$$

$w(T_X)$ is the total latency of the tour T_X that starts at $x_0 = 0$ and travels the points $x_1, x_2, \ldots, x_n$ in this order. $w(T_Y)$ is the total latency of the tour T_Y that starts at $y_0 = 0$ and travels the points $y_1, y_2, \ldots, y_m$ in this order.

Consider an optimal tour T for S. Depending on whether the first and the last edges of T are to the left or to the right, there are four possibilities. If, for example, the first edge is to the right and the last edge is to the left, then T must be of the following form (see Figure 3.1):

$$x_{i_0} = y_{j_0} = x_0 = y_0 \overset{\Delta}{\to} y_{j_1} \overset{\Delta}{\to} x_{i_1} \overset{\Delta}{\to} y_{j_2} \overset{\Delta}{\to} x_{i_2} \overset{\Delta}{\to} \cdots \overset{\Delta}{\to} x_{i_{t-1}} \overset{\Delta}{\to} y_{j_t} = y_m \overset{\Delta}{\to} x_{i_t} = x_n$$

for some $0 = j_0 < j_1 < j_2 < \cdots < j_{t-1} < j_t = m$ and $0 = i_0 < i_1 < \cdots < i_{t-1} < i_t = n$. (The notation $x_i \overset{\Delta}{\to} y_j$ denotes the straight line segment whose end points are x_i and y_j consisting of several edges.) We use the following *dummy tour* T' to represent T:

$$T' : \ x_{i_0}(= x_0) \to y_{j_1} \to x_{i_1} \to \cdots \to x_{i_{t-1}} \to y_{j_t}(= y_m) \to x_{i_t}(= x_n).$$

T is completely specified by T'. Define the *reduced weight* of the dummy tour T' to be

$$w(T') = \sum_{l=1}^{t} y_{j_l}[n+m-j_l-i_{l-1}] + \sum_{l=1}^{t} x_{i_l}[n+m-j_l-i_l]. \tag{3.2}$$

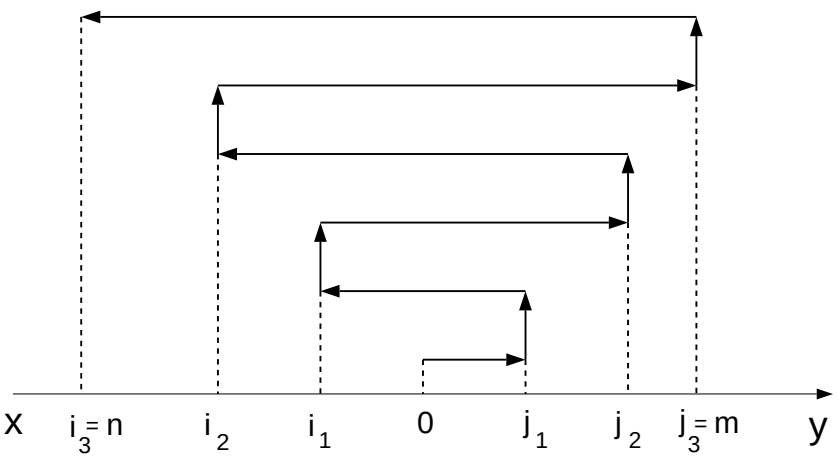

FIG. 3.1. *Optimal tour for points on a straight line.*

LEMMA 3.1. *The weight of T and the reduced weight of T' satisfies*

$$w(T) = w(T_X) + w(T_Y) + 2w(T'). \tag{3.3}$$

Proof. First we note that $w(T_X)+w(T_y)$ is the sum of the shortest-path distances from $x_0 = y_0$ to the points x_i $(1 \le i \le n)$ and y_j $(1 \le j \le m)$. In the tour T, the latency of each point will be the shortest-path distance plus some additional delay (caused by the zigzag-shaped detour). We need to compute this additional delay. Each "loop" of the form "from y_0 to y_{j_l} and back" contributes a delay of $2y_{j_l}$ to each point that is as yet unvisited when this loop is traversed; there are $(n+m-j_l-i_{l-1})$ such points. So the additional delay contributed by this loop is

$$2y_{j_l}(n+m-j_l-i_{l-1}).$$

Similarly, the additional delay contributed by a loop of the form "from x_0 to x_{i_l} and back" is

$$2x_{i_l}(n+m-j_l-i_l).$$

Summing up all these additional delay terms, we have $w(T) = w(T_X) + w(T_Y) + \sum_{l=1}^{t} 2y_{j_l}[n+m-j_l-i_{l-1}] + \sum_{l=1}^{t} 2x_{i_l}[n+m-j_l-i_l] = w(T_X) + w(T_Y) + 2w(T')$. □

Although Lemma 3.1 is proved by assuming that the first edge of T is to the right and the last edge of T is to the left, it also applies to other cases. For example, if the first edge of T is to the left, we can let $j_1 = 0$. If the last edge of T is to the right, we can let $i_{t-1} = n$ and delete from T the subpath from y_m to x_n. It can be verified that (3.3) is valid for those cases, too. Since the term $w(T_X) + w(T_Y)$ is fixed for all T, in order to minimize $w(T)$ we need only to minimize $w(T')$.

Let $G = (X, Y, E)$ be the complete bipartite digraph with $X = \{x_0, x_1, \ldots, x_n\}$, $Y = \{y_0, y_1, \ldots, y_m\}$, and the weight matrices $A[0..n, 0..m]$ and $B[0..m, 0..n]$ defined

as follows:

$$A[i,j] = w(x_i \to y_j) = y_j(n+m-i-j),$$
$$B[j,i] = w(y_j \to x_i) = x_i(n+m-i-j).$$

Note that $A[0,0] = B[0,0] = 0$. It is easy to check that a dummy tour T' with minimum reduced weight $w(T')$ is exactly a shortest path in G from x_0 to x_n. By the definition of A, for $0 \le i < i' \le n$ and $0 \le j < j' \le m$, we have $(A[i,j] + A[i',j']) - (A[i,j'] + A[i',j]) = (i'-i)(y_j - y_{j'}) < 0$. Thus A is concave. Similarly, we can show that B is also concave. Since all the entries of A and B are nonnegative, all the entries of $W = A \times B$ are nonnegative. Thus by Theorem 1.1 we have the following.

THEOREM 3.2. *The minimum latency problem for a set of N points on straight line can be solved in $O(N \log N)$ time.*

4. Solving the SPBD problem. Our algorithm for solving the SPBD problem is a reduction to an enhanced version of the *least weight subsequence* (LWS) problem. In section 4.1 we describe the LWS problem and its enhanced version. The reduction from the SPBD problem to the enhanced LWS problem is discussed in section 4.2. An algorithm for solving the enhanced LWS problem is given in section 4.3. In section 4.4 we give a complete description of our algorithm for the SPBD problem and analyze its time complexity.

4.1. The LWS problem and the enhanced LWS problem. The following LWS problem was introduced in [8]. Given a sequence $\{x_0, x_1, \ldots, x_n\}$ and a real-valued weight function $w(x_i, x_j)$ defined for indices $0 \le i < j \le n$, find an integer $k \ge 1$ and a sequence $S = \{0 = i_0 < i_1 < \cdots < i_{k-1} < i_k = n\}$ such that the total weight $w(S) = \sum_{l=1}^{k} w(x_{i_{l-1}}, x_{i_l})$ is minimized. The LWS problem can also be formulated as a graph problem: Given an acyclic digraph G with vertex set $V = \{x_0, \ldots, x_n\}$, the edge set $E = \{x_i \to x_j \mid 0 \le i < j \le n\}$, and the weight function w, we wish to find a shortest path in G from x_0 to x_n. For an arbitrary weight function w, the LWS problem requires $\Omega(n^2)$ time to solve. The weight function w is *concave* if the following hold:

$$w(x_{i_1}, x_{j_1}) + w(x_{i_2}, x_{j_2}) \le w(x_{i_2}, x_{j_1}) + w(x_{i_1}, x_{j_2}) \quad \text{for } 0 \le i_1 \le i_2 \le j_1 \le j_2 \le n. \tag{4.1}$$

If the weight function is concave, then we have an instance of the concave LWS problem. Hirschberg and Larmore showed that the concave LWS problem can be solved in $O(n \log n)$ time [8]. Similar algorithms were developed in [6, 7]. Wilber discovered an elegant linear-time algorithm for solving this problem [11]. All these algorithms assume that each entry $w(i,j)$ can be computed in constant time. In this paper we consider only the concave LWS problem. From now on the phrase "LWS problem" always means the concave LWS problem.

The *enhanced* version of the LWS problem is defined as follows. An instance of the enhanced LWS problem is a sequence $\{x_0, x_1, \ldots, x_n\}$ and a real-valued concave weight function $w(x_i, x_j)$ defined on *all* $0 \le i, j \le n$ such that $w(x_i, x_i) \ge 0$ for all $0 \le i \le n$. We want to find a sequence $S = \{0 = i_0, i_1, \ldots, i_k = n\}$ ($i_0, i_1, \ldots, i_k$ are not necessarily in increasing order, as in the ordinary LWS problem), such that the total weight $w(S) = \sum_{l=1}^{k} w(x_{i_{l-1}}, x_{i_l})$ is minimized. In terms of the graph formulation, given a complete digraph G with vertex set $V = \{x_0, x_1, \ldots, x_n\}$ and a weight function w, we wish to find a shortest x_0 to x_n path in G. Let $e = x_i \to x_j$

be an edge of G. If $i < j$, e is called a *forward* edge. If $i = j$, e is called a *selfloop*. If $i > j$, e is called a *backward* edge. We require that the weight of the selfloops of G be nonnegative since otherwise the weight of the shortest path in G would be $-\infty$.

LEMMA 4.1. *For any instance of the enhanced LWS problem there exists a shortest x_0 to x_n path consisting of only forward edges.*

Proof. Let P be a shortest path from x_0 to x_n in G such that the number of edges in P is minimum. Since $w(x_i, x_i) \geq 0$ for all i, we may assume that P contains no selfloops. Toward a contradiction, suppose that P contains a backward edge. Let $x_{i_l} \to x_{i_{l+1}}$ be the first backward edge of P. Then $i_l > i_{l+1}$ and $i_l > i_{l-1}$. By the concavity of w and the assumption that $w(x_i, x_i) \geq 0$ for all x_i, we have $w(x_{i_{l-1}}, x_{i_{l+1}}) \leq w(x_{i_{l-1}}, x_{i_{l+1}}) + w(x_{i_l}, x_{i_l}) \leq w(x_{i_{l-1}}, x_{i_l}) + w(x_{i_l}, x_{i_{l+1}})$. Thus replacing the two edges $x_{i_{l-1}} \to x_{i_l} \to x_{i_{l+1}}$ in P by a single edge $x_{i_{l-1}} \to x_{i_{l+1}}$, we get a path P' such that $w(P') \leq w(P)$ and the number of edges in P' is one less than that in P. This contradicts the choice of P. □

Lemma 4.1, together with the concavity of w, implies that there are no negative cycles in any instance of the enhanced LWS problem. It also implies that we can ignore all the backward edges and selfloops when solving the enhanced LWS problem.

4.2. Reduction. In this section we show that the SPBD problem can be reduced to the enhanced LWS problem. First we need several technical lemmas. The following lemma was proved in [12].

LEMMA 4.2. *If both A and B are concave, then the product matrix $W = A \times B$ is also concave.*

For $0 \leq i \leq n$ and $0 \leq j \leq n$, let $I(i, j)$ denote the smallest index k that realizes the minimum value in definition (1.2). Namely, $I(i, j)$ is the smallest index such that $W[i, j] = A[i, I(i, j)] + B[I(i, j), j]$. The following lemma was proved in [12].

LEMMA 4.3. *For any i, j $(0 \leq i < n,\ 0 \leq j < n)$, we have $I(i, j) \leq I(i, j + 1) \leq I(i + 1, j + 1)$.*

Remark. The definitions of concavity and the matrix product in [12] are slightly different from the definitions used here. In [12] a concave matrix is an upper triangular matrix such that the condition (1.1) is true for $i_1 \leq i_2 \leq j_1 \leq j_2$. In the matrix product definition (1.2) the minimum is taken over $i \leq k \leq j$. Under these definitions, Yao proved Lemmas 4.2 and 4.3. Under our definitions, Lemmas 4.2 and 4.3 can be proved via similar methods.

Let (i, j) and (i', j') be two pairs of indices. If $i \leq i'$ and $j \leq j'$, we write $(i, j) \prec (i', j')$. By Lemma 4.3, $(i, j) \prec (i', j')$ implies $I(i, j) \leq I(i', j')$.

LEMMA 4.4. *Let $(i_1, j_1), (i_2, j_2), \ldots, (i_p, j_p)$ be p pairs of indices such that $(i_l, j_l) \prec (i_{l+1}, j_{l+1})$ for all $1 \leq l < p$. Then $I(i_1, j_1), I(i_2, j_2), \ldots, I(i_p, j_p)$ can be computed in $O(m \log p)$ time.*

Proof. This can be done in a binary search fashion. More specific, we find $I_{p/2} = I(i_{p/2}, j_{p/2})$ in the first stage, find $I(i_{p/4}, j_{p/4})$ (by searching the range $0..I_{p/2}$) and $I(i_{3p/4}, j_{3p/4})$ (by searching the range $I_{p/2}..m$) in the second stage, and so on. In total, there are $\log p$ stages. Since each stage can be done in $O(m)$ time, the lemma holds. □

Consider an instance of the SPBD problem defined by a complete bipartite digraph $G = (X, Y, E)$ and two concave weight matrices A and B. Let $G' = (X, E')$ be the complete digraph on X with the concave weight matrix $w = A \times B$. If $w(x_i, x_i) \geq 0$ for all $0 \leq i \leq n$, then G' and w define an instance of the enhanced LWS problem.

Let $P' :\ x_0 = x_{i_0} \to x_{i_1} \to \cdots \to x_{i_k} = x_n$ be a shortest path in G' from x_0 to x_n. For each l $(0 \leq l \leq k)$, let $j_l = I(i_{l-1}, i_l)$. Then $P :\ x_0 = x_{i_0} \to y_{j_1} \to x_{i_1} \to$

$y_{j_2} \to \cdots \to y_{j_k} \to x_{i_k} = x_n$ is a path in G from x_0 to x_n. Let $w(P')$ denote the weight of P' in G' and $w(P)$ the weight of P in G. Clearly, $w(P) = w(P')$. We will show that $w(P)$ is minimum among all paths from x_0 to x_n in G.

Let Q be a shortest path in G from x_0 to x_n. Since G is bipartite, Q is a concatenation of subpaths $Q_1, Q_2, \ldots, Q_p$ for some $p \geq 1$, where each Q_l $(1 \leq l \leq p)$ consists of two edges $x_{i'_{l-1}} \to y_{j'_l} \to x_{i'_l}$ ($i'_0 = 0$ and $i'_p = n$). For each $1 \leq l \leq p$, if $j'_l \neq I(i'_{l-1}, i'_l)$, we can replace Q_l by the subpath $x_{i'_{l-1}} \to y_{I(i'_{l-1},i'_l)} \to x_{i'_l}$ without increasing the total weight $w(Q)$. Therefore, without loss of generality, we may assume that $j'_l = I(i'_{l-1}, i'_l)$ for all $1 \leq l \leq p$. Hence the weight of Q_l is $w[i'_{l-1}, i'_l]$. Thus Q corresponds to a path $Q' = \{x_0 = x_{i'_0} \to x_{i'_1} \to \cdots \to x_{i'_p} = x_n\}$ from x_0 to x_n in G' with $w(Q') = w(Q)$. Since the weight of P' is minimum among all such paths in G', we have $w(P) = w(P') \leq w(Q') = w(Q)$. Thus P is a shortest path in G from x_0 to x_n.

LEMMA 4.5. *Let A and B be two concave matrices such that all the entries on the main diagonal of the product matrix $w = A \times B$ are nonnegative. If the enhanced LWS problem defined by the matrix w can be solved in $T(n, m)$ time, then the SPBD problem defined by A and B can be solved in $O(T(n, m) + m \log n)$ time.*

Proof. In order to solve the SPBD problem defined by matrices A and B, we first solve the enhanced LWS problem defined by the matrix $w = A \times B$. Let $P' : x_0 = x_{i_0} \to x_{i_1} \to \cdots \to x_{i_k} = x_n$ be the solution path found. We compute $j_1, j_2, \ldots, j_k$, where $j_l = I(i_{l-1}, i_l)$. Since $(i_0, i_1) \prec (i_1, i_2) \prec \cdots \prec (i_{k-1}, i_k)$, this can be done in $O(m \log n)$ time by Lemma 4.4. The path $x_0 = x_{i_0} \to y_{j_1} \to x_{i_1} \to \cdots \to y_{j_k} \to x_{i_k} = x_n$ is the solution for the SPBD problem. □

We want to use Wilber's algorithm in [11] to solve our enhanced LWS problem. In order to do this, however, we have to overcome two difficulties. First, Wilber's algorithm is for solving the (ordinary) LWS problem which is defined by an upper triangle matrix while our problem is defined by a full matrix. Second, Wilber's algorithm assumes that each entry $w(i, j)$ can be evaluated in $O(1)$ time. In our case, an entry of the matrix $w = A \times B$ may need $\Theta(m)$ time to evaluate. We will address these two issues in the next section.

4.3. An algorithm for the enhanced LWS problem. Our algorithm for the enhanced LWS problem is a modification of Wilber's algorithm for the LWS problem. First, we briefly review Wilber's algorithm. (We assume that the reader is familiar with [11].) Then we show how to modify Wilber's algorithm to solve our problem.

Consider an instance of the LWS problem with the sequence $\{x_0, x_1, \ldots, x_n\}$ and the weight matrix $w(x_i, x_j)$. Recall that w is an $(n + 1) \times (n + 1)$ upper triangular matrix. Let $f(0) = 0$ and, for $1 \leq j \leq n$, let $f(j)$ be the weight of the lowest weight subsequence between x_0 and x_j. For $0 \leq i < j \leq n$ let $g(i, j)$ be the weight of the lowest-weight subsequence between x_0 and x_j whose next-to-last element is x_i. Then we have

$$\begin{cases} f(j) = \min_{0 \leq i < j} g(i, j) & \text{for } 1 \leq j \leq n, \\ g(i, j) = f(i) + w(x_i, x_j) & \text{for } 0 \leq i < j \leq n. \end{cases} \tag{4.2}$$

To solve the LWS problem it is enough to compute $f(1), f(2), \ldots, f(n)$. Adding $f(i_1) + f(i_2)$ to both sides of inequality (4.1) and applying definition (4.2), we get

$$g(i_1, j_1) + g(i_2, j_2) \leq g(i_1, j_2) + g(i_2, j_1) \quad \text{for } 0 \leq i_1 \leq i_2 \leq j_1 \leq j_2 \leq n.$$

Consider a matrix $M[0..n, 0..m]$. For each column index $0 \leq j \leq m$ let $i(j)$ be the smallest row index such that $M(i(j), j)$ equals the minimum value in the jth

column of M. The *column minima searching* problem for M is to find the $i(j)$'s for all $0 \leq j \leq m$. M is called *monotone* if $i(j_1) \leq i(j_2)$ for all $0 \leq j_1 < j_2 \leq m$. M is *totally monotone* if every 2×2 submatrix of M is monotone [3]. If M is concave, then it is easy to check that M is totally monotone. (The reverse is not necessarily true.) For a totally monotone matrix M, the column minima searching problem for M can be solved in $O(n+m)$ time, assuming that each entry of M can be evaluated in $O(1)$ time [3]. Following [8], we will refer to the algorithm in [3] as the SMAWK algorithm.

We extend the definition of g by setting $g(i,j) = +\infty$ for $0 \leq j \leq i \leq n$. Then g becomes a full $(n+1) \times (n+1)$ matrix. It is easy to verify that the extended matrix g is totally monotone. (The only role of the $+\infty$ entries is to make g a full matrix for convenience. These entries otherwise have no effect on the computation.) Our goal is to determine the row index of the minimum value in each column of g (which gives $f(1), \ldots, f(n)$). One might simply want to apply the SMAWK algorithm to g. But we cannot, because for $i < j$, the value of $g(i,j)$ depends on $f(i)$ and $f(i)$ depends on $g(0,i), g(1,i), \ldots, g(i-1,i)$. Thus we cannot compute the value of g in $O(1)$ time as required by the SMAWK algorithm.

Wilber's algorithm starts in the upper left corner of g and works rightward and downward, at each iteration learning enough new values of f to be able to compute enough new values of g to continue with the next iteration. Actually, during one step of each iteration, the algorithm might "pretend" to know values of f that it really does not have. At the end of the iteration, the assumed value of f is checked for validity.

We use $f(j)$ and $g(i,j)$ to refer to the correct values of f and g, respectively. The currently computed value of $f(j)$ is denoted by $F(j)$ and sometimes will be incorrect. The currently computed value of $g(i,j)$ is denoted by $G[i,j]$ and is always computed as $F[i] + w(i,j)$. Therefore $G[i,j] = g(i,j)$ iff $F(i) = f(i)$. The algorithm does not explicitly store the matrices w, g, G. Rather, their entries are calculated when needed. Let $G[i_1, i_2; j_1, j_2]$ denote the submatrix of G consisting of the intersection of rows i_1 through i_2 and columns j_1 through j_2. $G[i_1, i_2; j]$ denotes the intersection of rows i_1 through i_2 with column j. The rows of G are indexed from 0 and the columns are indexed from 1. Wilber's algorithm is as follows.

Wilber's Algorithm.

$F[0] \leftarrow c \leftarrow r \leftarrow 0$.

while $(c < n)$ do

1. $p \leftarrow \min\{2c - r + 1, n\}$.
2. Apply the SMAWK algorithm to find the minimum in each column of submatrix $S = G[r, c; c+1, p]$. For $j \in [c+1, p]$, let $F[j]$ = the minimum value found in $G[r, c; j]$.
3. Apply the SMAWK algorithm to find the minimum in each column of the submatrix $T = G[c+1, p-1; c+2, p]$. For $j \in [c+2, p]$, let $H[j]$ = the minimum value found in $G[c+1, p-1; j]$.
4. If there is an integer $j \in [c+2, p]$ such that $H[j] < F[j]$, then set j_0 to the smallest such integer. Otherwise set $j_0 \leftarrow p+1$.
5. **if** $(j_0 = p+1)$ **then** $c \leftarrow p$;
   else $F[j_0] \leftarrow H[j_0]$; $r \leftarrow c+1$; $c \leftarrow j_0$.

end.

Figure 4.1 shows the submatrices S and T during a typical iteration of the algorithm. (This figure is taken from [11].) Each time we are at the beginning of the

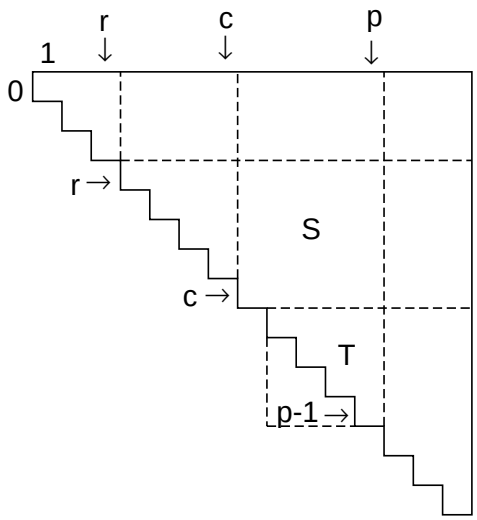

FIG. 4.1. *A typical iteration of Wilber's algorithm.*

loop, the following invariants hold:

(a) $r \geq 0$ and $c \geq r$.

(b) For each $j \in [0, c]$, $F[j] = f(j)$.

(c) All the minima in columns $c+1$ through n of g are in rows $\geq r$.

These invariants are clearly satisfied at the beginning when $r = c = 0$.

Invariant (b) implies that $G[i, j] = g(i, j)$ for all j and all $i \in [0, c]$. So the entries of the submatrix S are the same as the corresponding entries of g. Therefore S is totally monotone, and for each $j \in [c+1, p]$ step 2 sets $F[j]$ to the minimum value of the subcolumn $g(r, c; j)$. Also, since S contains all the finite-valued cells in column $c+1$ of g that are in rows $\geq r$, we have $F[c+1] = f(c+1)$ at the end of step 2. On the other hand, we do not necessarily have $F[j] = f(j)$ for any $j \in [c+2, p]$, since g has finite-valued cells in those columns that are in rows $\geq r$ and not in S.

In step 3 we proceed as if $F[j] = f(j)$ for all $j \in [c+1, p-1]$. Since this may be false, some of the values in T may be bogus. However, T is always totally monotone because if we add $F[i_1] + F[i_2]$ to both sides of (4.1) we get $G[i_1, j_1] + G[i_2, j_2] \leq G[i_1, j_2] + G[i_2, j_1]$. Thus the SMAWK algorithm works correctly and $H[j]$ is set to the minimum value of the subcolumn $G[c+1, p-1; j]$ (which is not necessarily the same as the minimum value of the subcolumn $g(c+1, p-1; j)$). Note that since all the entries on and below the main diagonal of g are $+\infty$, they cannot be $H[j]$ for any j and hence have no effect on the computation.

In step 4 we either verify that $F[j] = f(j)$ for all $j \in [c+2, p]$ (this is the case if $H[j] \geq F[j]$ for all $j \in [c+2, p]$) or we find the smallest j where this condition fails (this is the case when there exists $j \in [c+2, p]$ such that $H[j] < F[j]$). In either case, the values of c and r are set accordingly at step 5 so that the loop invariants hold. This completes the description of Wilber's algorithm.

Next we discuss how to use Wilber's algorithm to solve the enhanced LWS problem. Let $w[0..n, 0..n]$ be the weight matrix of an instance of the enhanced LWS problem. Let L denote the portion of w consisting of the entries on and below the main diagonal of w. Let w' be the matrix obtained from w by replacing all the entries in L by $+\infty$. Then w' defines an instance of the (ordinary) LWS problem. By

Lemma 4.1, the solution for the problem defined by w' is identical to the solution for the problem defined by w. If each entry of w can be computed in $O(1)$ time, we can use Wilber's algorithm on w' to solve the problem. However, if the enhanced LWS problem is derived from an instance of the SPBD problem, the entries of the matrix $w = A \times B$ cannot be computed in $O(1)$ time. In this case we cannot afford to change w to w' since doing so will destroy some properties of w that are crucial for obtaining a fast algorithm. Fortunately, we will show that Wilber's algorithm can be applied directly to w to solve the enhanced LWS problem.

It is enough to show that the entries in L have no effect on the computation of Wilber's algorithm. The only place where Wilber's algorithm needs the entries in L is step 3, where the SMAWK algorithm is applied to the submatrix T. For each $j \in [c+2, p]$ let $F[j]$ and $H[j]$ be the minimum value of column j in S and T, respectively. There are three cases:

(a) $F[j] \leq H[j]$.

(b) $F[j] > H[j]$ and $H[j]$ is not in L. Namely, $H[j] = G[i, j]$ for some $i < j$.

(c) $F[j] > H[j]$ and $H[j]$ is in L. Namely, $H[j] = G[i, j]$ for some $i \geq j$.

In cases (a) and (b) the values in L do not affect the computation. In the following we show that case(c) cannot occur. Toward a contradiction, assume that there exist indices $j \in [c+2, p]$ and i such that $i \geq j$ and $H[j] = G[i, j] < F[j]$.

Case 1: If $i = j$, then $H[j] = G[j, j] = F[j] + w(j, j) \geq F[j]$. This is impossible.

Case 2: If $i > j$, then $H[j] = G[i, j] = F[i] + w(i, j)$. Recall that $F[i]$ is the minimum value of the subcolumn $G[r, c; i]$. Suppose that $F[i] = G[t, i] = F[t] + w(t, i)$ for some $r \leq t \leq c$. Note that $t \leq c < i$ and $j < i$. By the concavity of w we have $w(t, j) + w(i, i) \leq w(t, i) + w(i, j)$. Since $w(i, i) \geq 0$ for all i, we have $H[j] = F[i]+w(i, j) = F[t]+w(t, i)+w(i, j) \geq F[t]+w(t, j)+w(i, i) \geq F[t]+w(t, j) = G[t, j] \geq F[j]$. This contradicts the assumption that $H[j] < F[j]$.

Since case (c) cannot occur, the entries in L do not affect the computation of Wilber's algorithm, regardless of whether they are changed to $+\infty$ or not. Hence we have proved the following lemma.

LEMMA 4.6. *An instance of the enhanced LWS problem defined by a (full) concave matrix w can be correctly solved by applying Wilber's algorithm to the matrix w.*

Next we address the second difficulty mentioned at the end of section 4.2. If the instance of the enhanced LWS problem is derived from an instance of the SPBD problem (defined by matrices A and B), the weight matrix w of the enhanced LWS problem is the product matrix $w = A \times B$. Therefore the key assumption of Wilber's algorithm that each entry $w[i, j]$ can be evaluated in $O(1)$ time is not valid.

During each stage of Wilber's algorithm (steps 2 and 3) we need to find column minima of the submatrices S and T. Both S and T have the form $C'[r, c; q, p]$ where $C'[i, j] = F[i] + w[i, j]$ for some known value $F[i]$. Since the values $C'[i, j]$ cannot be computed in $O(1)$ time, we cannot use the SMAWK algorithm directly. Instead, we use the algorithm given in the following lemma.

LEMMA 4.7. *The column minima searching problem for the submatrix $C'[r, c; q, p]$ with $r \leq q$ and $c \leq p$ can be solved in $O((c - r) + (p - q) + (k_2 - k_1))$ time, where $k_1 = I(r, r)$ and $k_2 = I(p, p)$.*

Proof. By Lemma 4.3, for each $i \in [r, c]$ and $j \in [q, p]$, $w[i, j] = \min_{0 \leq k \leq m}(A[i, k]+ B[k, j])$ can be computed by searching k in the range $k \in [k_1, k_2]$. For $j \in [q, p]$ let $d(j)$ denote the column minimum of $C'[r, c; j]$. Then $d(j) = \min_{r \leq i \leq c}\{F[i] + w[i, j]\} = \min_{r \leq i \leq c}\{F[i]+\min_{k_1 \leq k \leq k_2}(A[i, k]+B[k, j])\} = \min_{k_1 \leq k \leq k_2}\{B[k, j]+\min_{r \leq i \leq c}(F[i]+ A[i, k])\}$.

For $i \in [r, c]$ and $k \in [k_1, k_2]$ let $A'[i, k] = F[i] + A[i, k]$. Then A' is totally monotone. For each $k \in [k_1, k_2]$ let $J[k]$ be the minimum of the subcolumn $A'[r, c; k]$.

For $k \in [k_1, k_2]$ and $j \in [q, p]$ let $B'[k, j] = B[k, j] + J[k]$. Then B' is totally monotone. Clearly, $d(j)$ is the minimum of the subcolumn $B'[k_1, k_2; j]$. Thus the column minima $d(j)$'s of $C'[r, c; q, p]$ can be found by two applications of the SMAWK algorithm, once on A' and once on B'. Each entry of A' and B' can be evaluated in $O(1)$ time. Thus the total time needed is $O((c-r)+(k_2-k_1))+O((k_2-k_1)+(p-q)) = O((c-r)+(p-q)+(k_2-k_1))$. □

4.4. The SPBD algorithm and its complexity. The following is the complete description of our SPBD algorithm.

SPBD Algorithm.

Input: An instance of the SPBD problem defined by two concave matrices A and B.

1. Compute $I(0,0), I(1,1), \ldots, I(n,n)$ (cf. Lemma 4.4).
2. Solve the enhanced LWS problem defined by the matrix $w = A \times B$ by applying Wilber's algorithm on w. But instead of using the SMAWK algorithm we use the algorithm given in Lemma 4.7 to search the column minima of the submatrix S and T during the execution.
3. Using the method described in Lemma 4.5, convert the solution of the enhanced LWS problem defined by w to the solution of the original SPBD problem.

end.

The correctness of the algorithm follows from the discussion of the last subsection. Next we analyze the running time of the SPBD algorithm. We concentrate on step 2 since this is the nontrivial part of the algorithm. Each iteration of Wilber's algorithm is completely specified by three parameters: r, c, p. Let r_i, c_i, p_i be the values of these parameters in the ith iteration. The parameters for the next iteration are calculated in step 5 as follows:

Case 1: "then" part of step 5 is executed. In this case, $r_{i+1} = r_i$, $c_{i+1} = p_i$, and

Case 1a: $p_{i+1} = 2c_{i+1} - r_{i+1} + 1$, if it is $\leq n$, or

Case 1b: $p_{i+1} = n$, otherwise.

Case 2: "else" part is executed. In this case, $r_{i+1} = c_i + 1$, $c_{i+1} = j_0$ (where $c_i + 2 \leq j_0 \leq p_i$), and

Case 2a: $p_{i+1} = 2c_{i+1} - r_{i+1} + 1$, if it is $\leq n$, or

Case 2b: $p_{i+1} = n$, otherwise.

If Case 1a (or 1b, 2a, 2b, respectively) applies to the ith iteration, we call it a type 1a (or 1b, 2a, 2b, respectively) iteration. We call $[r_i, p_i]$ the ith *span*; r_i and p_i are the left and the right ends of the ith span, respectively. Note that after a type 1a or 1b iteration, the left end is not changed and the right end increases. After a type 2a or 2b iteration, the left end increases and the right end may increase, decrease, or remain unchanged. For an interval $[t, t+1]$ $(0 \leq t < n)$ we say that a span $[r_i, p_i]$ *covers* $[t, t+1]$, written as $[t, t+1] \in [r_i, p_i]$, if $r_i \leq t$ and $t+1 \leq p_i$. Since the left end of spans never decreases, the spans "move" from left to right during the execution of the algorithm. Once the left end of a span is $\geq t+1$, $[t, t+1]$ will never be covered by any subsequent spans. First we make the following obvious observations:

(a) If a type 1a or 1b iteration follows a type 1b or 2b iteration, the algorithm terminates immediately.

(b) If the ith iteration is of type 1a, then $p_{i+1} - r_{i+1} = (2c_{i+1} - r_{i+1} + 1) - r_{i+1} = 2(p_i - r_i) + 1$. Namely, the length of the $(i+1)$th span is $1 + 2 \times$ (the length of the ith span).

(c) Suppose that the ith iteration is of type 2a or 2b. Since $p_i \le 2c_i - r_i + 1$, we have $c_i \ge (p_i + r_i - 1)/2$. Hence $r_{i+1} = c_i + 1 \ge (p_i + r_i - 1)/2 + 1$.

(d) Suppose that an interval $[t, t+1]$ is covered by the ith span $[r_i, p_i]$. If the ith iteration is of type 1a or 1b, and the $(i+1)$st iteration is of type 2a or 2b, then $r_{i+2} = c_{i+1} + 1 = p_i + 1 > t + 1$. Hence $[t, t+1]$ is not covered by $[r_{i+2}, p_{i+2}]$ nor by any subsequent spans.

The following lemma gives an upper bound on the number of times an interval $[t, t+1]$ can be covered by spans. This bound is needed in the analysis of our algorithm.

LEMMA 4.8. *Any interval* $[t, t+1]$ $(0 \le t < n)$ *is covered by at most* $2 \log n + 2$ *spans.*

Proof. Let $[r_{i_1}, p_{i_1}], [r_{i_2}, p_{i_2}], \ldots, [r_{i_k}, p_{i_k}]$ be all the spans covering $[t, t+1]$, where $i_1 < i_2 < \cdots < i_k$. Then $r_{i_l} \le t$ and $t + 1 \le p_{i_l}$ for all $1 \le l \le k$.

Let l be the first index such that the i_lth iteration is of type 1a or type 1b. (If no such l exists, let $l = k$.) We first show that $k - l \le \log n + 2$.

Case 1: The i_lth iteration is of type 1b. If the $(i_l + 1)$st iteration is of type 1a or 1b, then the algorithm terminates by observation (a). If the $(i_l + 1)$st iteration is of type 2a or 2b, then by observation (d) $[t, t+1]$ is not covered by $[r_{i_l+2}, p_{i_l+2}]$ nor by any subsequent spans.

Case 2: The i_lth iteration is of type 1a. Let s (possibly $s = 0$) be the largest integer such that the iterations $i_l, i_l + 1, \ldots, i_l + s$ are all of type 1a. Clearly, $[t, t+1]$ is covered by the spans $[r_{i_l+1}, p_{i_l+1}], \ldots, [r_{i_l+s}, p_{i_l+s}]$. By observation (b) each type 1a iteration doubles the length of the span. Since the length of a span is at most n, we have $s \le \log n$. The $(i_l + s + 1)$st iteration is of type 1b, 2a, or 2b. If it is of type 2a or 2b, then by observation (d) $[t, t+1]$ is not covered by the $(i_l + s + 2)$nd span nor by any subsequent spans. If the $(i_l + s + 1)$st iteration is of type 1b, then, similar to Case 1, either the algorithm terminates at the $(i_l + s + 2)$nd iteration, or $[t, t+1]$ is not covered by the $(i_l + s + 2)$nd span nor by any subsequent spans.

In either case, the number of spans following the i_lth iteration that cover $[t, t+1]$ is at most $\log n + 2$. So $k - l \le \log n + 2$. Next we show $l \le \log n$ and this will complete the proof of the lemma.

For each $1 \le h < l$ the i_hth iteration is of type 2a or 2b. Fix an index h. For each $j \ge i_h$ let $L_j = (t+1) - r_j$. Note that if $L_j \le 0$, then the span $[r_j, p_j]$ cannot cover the interval $[t, t+1]$. By the fact that $t + 1 \le p_{i_h}$ and observation (c), we have

$$\begin{aligned} L_{i_h+1} &= (t+1) - r_{i_h+1} \le (t+1) - ((p_{i_h} + r_{i_h} - 1)/2 + 1) \\ &= (2t - p_{i_h} - r_{i_h} + 1)/2 \le (t - r_{i_h})/2 < L_{i_h}/2. \end{aligned}$$

Since the left end of the spans never decreases, we now have that $L_{i_{h+1}} \le L_{i_h+1} < L_{i_h}/2$. This is true for all $1 \le h < l$. Hence $L_{i_l} < L_{i_1}/2^l$. If $l > \log n$, then L_{i_l} becomes 0 and the interval $[t, t+1]$ is not covered by $[r_{i_l}, p_{i_l}]$ nor by any subsequent spans. So we must have $l \le \log n$. This establishes the lemma. □

We are now ready to prove Theorem 1.1.

Proof of Theorem 1.1. Step 1 of the SPBD algorithm takes $O(m \log n)$ time by Lemma 4.4. In step 2 we use Wilber's algorithm to solve the enhanced LWS problem defined by the matrix $w = A \times B$. But instead of using the SMAWK algorithm we use the subroutine in Lemma 4.7 for finding column minima in S and T. Note that all the other steps of Wilber's algorithm take $O(n + m)$ time. Thus if we can show that the time needed by these subroutine calls is bounded by $O(n + m \log n)$, the theorem will follow from Lemma 4.5.

Consider the ith iteration. We need to find the column minima of the submatrices $S_i = G[r_i, c_i; c_i + 1, p_i]$ and $T_i = G[c_i + 1, p_i - 1; c_i + 2, p_i]$. Let $k_1 = I(r_i, r_i)$, $k_2 = I(p_i, p_i)$, and $k_3 = I(c_i + 1, c_i + 1)$. Since $r_i < c_i + 1 \le p_i$ we have $k_1 \le k_3 \le k_2$ by Lemma 4.3.

By Lemma 4.7 the searching of S_i needs $O((c_i - r_i) + (p_i - c_i - 1) + (k_2 - k_1)) = O((p_i - r_i) + (k_2 - k_1))$ time. The searching of T_i needs $O((p_i - 1 - c_i - 1) + (p_i - c_i - 2) + (k_2 - k_3))$ time. Since $p_i \le 2c_i - r_i + 1$ and $k_3 \ge k_1$, this is bounded by $O((p_i - r_i) + (k_2 - k_1))$. Thus the total time needed to search S_i and T_i in all the iterations is $\sum_{i=1}^{K} O((p_i - r_i) + (I(p_i, p_i) - I(r_i, r_i)))$, where K is the total number of iterations of the algorithm. Since Wilber's original algorithm takes $O(n)$ time, the term $\sum_{i=1}^{K} O(p_i - r_i)$ is bounded by $O(n)$. On the other hand,

$$\begin{aligned}\sum_{i=1}^{K}(I(p_i, p_i) - I(r_i, r_i)) &= \sum_{i=1}^{K}\sum_{t=r_i}^{p_i-1}(I(t+1, t+1) - I(t, t)) \\ &= \sum_{t,i \text{ where } [t,t+1]\in[r_i,p_i]} (I(t+1, t+1) - I(t, t)). \qquad (4.3)\end{aligned}$$

By Lemma 4.8 each interval $[t, t + 1]$ is covered by at most $2 \log n + 2$ spans. Thus the above sum is bounded by $O(\log n \sum_{t=0}^{n-1}(I(t + 1, t + 1) - I(t, t))) = O(m \log n)$ as to be shown. □

5. Conclusion. We introduced the SPBD problem and showed that if the weight matrices are concave, then the SPBD problem can be reduced to the enhanced LWS problem and solved in $O(n+m \log n)$ time. As applications, we showed that the MLP for points on a straight line and the TSP for points on a convex polygon can be reduced to the SPBD problem and solved in $O(n \log n)$ time, which substantially improves the previously known $O(n^2)$-time algorithms. The setting of the SPBD problem is quite general. It is interesting to find other applications of the SPBD problem.

We tried (but failed) to use this technique to solve the MLP for points on a convex polygon. (To our knowledge, the MLP for this special case is not known to be in P.) In the two applications discussed in this paper, the optimal paths are simple (i.e., no two edges of the path cross). Unfortunately, the optimal tour in the MLP for points on a convex polygon does not have this crucial property. It would be interesting to find a polynomial-time algorithm for solving the MLP for this case.

Another open problem is to solve the MLP for an *r-arm star graph* G, which has a center vertex c and r "arms" connected to c, and each arm is a straight line with several vertices on it. (Thus the straight line discussed in section 3 is a two-arm star graph.) If each arm of G contains at most k vertices, then the MLP problem for G can be solved in $O(r \times k^r)$ time by using dynamic programming, which is not a polynomial in terms the number of vertices of G. Is it possible to solve this problem in polynomial time using ideas similar to the SPBD algorithm? This would seem to require, at the very least, solving r-partite generalization of the SPBD problem.

Acknowledgments. The authors would like to thank the referees for pointing out the much simplified proof of Lemma 3.1, for mentioning the last open problem in section 5, and for suggestions that considerably improved the exposition.

REFERENCES

[1] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, and N. Papkostantinou, *The complexity of the traveling repairman problem*, Informatique Theorique Appl. (Theoret. Informatics Appl.) 20 (1986), pp. 79–87.

[2] A. Aggarwal and J. Park, *Notes on searching in multidimensional monotone arrays*, in Proc. 29th IEEE Foundations of Computer Science, White Plains, NY, 1988, pp. 497–512.

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

[4] M. J. Atallah, S. R. Kosaraju, L. L. Larmore, G. L. Miller, and S.-H. Teng, *Constructing trees in parallel*, in Proc. ACM Symposium on Parallel Algorithms and Architectures, Santa Fe, NM, 1989, pp. 421–431.

[5] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, in Proc. 26th ACM Symposium on the Theory of Computing, Montreal, Quebec, 1994, pp. 163–171.

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

[7] Z. Galil and R. Giancarlo, *Speeding-up dynamic programming with applications to molecular biology*, Theoret. Comput. Sci., 64 (1989), pp. 107–118.

[8] D. S. Hirschberg and L. L. Larmore, *The least weight subsequence problem*, SIAM J. Comput., 16 (1987), pp. 628–638.

[9] M. M. Klawe and D. J. Kleitman, An almost linear time algorithm for generalized matrix searching, SIAM J. Discrete Math., 3 (1990), pp. 81–97.

[10] O. Marcotte and S. Suri, *Fast matching algorithms for points on a polygon*, SIAM J. Comput., 20 (1991), pp. 405–422.

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

[12] F. F. Yao, *Efficient dynamic programming using quadrangle inequalities*, in Proc. 12th ACM Symposium on the Theory of Computing, 1980, pp. 429–435.

[13] F. F. Yao, *Speed-up in dynamic programming*, SIAM J. Alg. Discrete Methods, 3 (1982), pp. 532–540.