アルゴリズム 52−8
(1996. 7. 24)

完全２部有向グラフ上の最短路問題とその応用

Xin He[†] 陳 致中[‡]

[†] ニューヨーク州立大学バッファロー校 [‡] 東京電機大学理工学部数理学科

完全２部有向グラフ上の最短路問題 (SPCB) を次のように定義する：重み付き完全２部有向グラフ $G = (X = \{x_0, ..., x_n\}, Y = \{y_0, ..., y_m\}, E)$ が与えられたとき、G における x_0 から x_n までの最短路を計算せよ。重みを指定する行列に何も制限がないとき、SPCB を解くのに $\Omega(nm)$ 時間が必要である。本稿では、重みを指定する行列が concave であるとき、SPCB が $O(n + m\log n)$ 時間で解ける、ことを示す。

また、SPCB の応用として、凸多角形上の n 点に対する TSP 問題と直線上の n 点に対する最小潜伏ツアー問題を論ずる。この二つの問題を解く既知のアルゴリズムの計算時間は $O(n^2)$ であった。SPCB を利用して、この二つの問題を解く $O(n\log n)$- 時間のアルゴリズムを提案する。

Shortest Path in Complete Bipartite Digraph Problem and its Applications

Xin He[†] and Zhi-Zhong Chen[‡]

[†]Department of Computer Science, State University of New York at Buffalo
[‡]Department of Mathematical Sciences, Tokyo Denki University

We introduce the *shortest path in complete bipartite digraph* (SPCB) problem: Given a 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 a shortest path from x_0 to x_n in G. For arbitrary weight matrices, the problem needs at least $O(nm)$ time to solve since all edges of G must be examined. We show that if the weight matrices are *concave*, then the problem can be solved in $O(n + m\log n)$ time by using a modified version of Wilber's algorithm for solving the *least weight subsequence* problem.

As applications of the SPCB problem, we discuss the Traveling Salesman problem for n points on a convex polygon and the Minimum Latency Tour problem for n points on a straight line. The known algorithms for solving both problems require $O(n^2)$ time. Using our SPCB algorithm, we show both problems can be solved in $O(n\log n)$ time.

1 Introduction

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 $x_i \rightarrow y_j$ and $y_j \rightarrow x_i$ to denote the edges. Let $A[0..n, 0..m]$ be the matrix with $A[i,j] = w(x_i \rightarrow y_j)$ and $B[0..m, 0..n]$ be the matrix with $B[i,j] = w(y_i \rightarrow x_j)$. In applications, matrices A and B are not explicitly stored. Rather, an entry is computed in constant time when it is needed. The weight of a (directed) path P in G is defined to be $w(P) = \sum_{e \in P} w(e)$.

The *shortest path in complete bipartite digraph* (SPCB) problem is: given such a digraph G, find a path P in G from x_0 to x_n such that $w(P)$ is minimumized. (We require that G contains no negative cycles, since otherwise the shortest path of G is not well-defined.) For arbitrary weight matrices, we need at least $O(nm)$ time to solve the problem since all edges of G must be examined. A matrix $M[0..n, 0..m]$ is called *concave* if the following hold:

$$M[i_1, j_1] + M[i_2, j_2] \leq M[i_2, j_1] + M[i_1, j_2] \quad \text{for} \quad 0 \leq i_1 \leq i_2 \leq n \quad \text{and} \quad 0 \leq j_1 \leq j_2 \leq m \tag{1}$$

Concave matrices were first discussed in [10] and have been very successfully used in solving various problems. In this paper, we show that if both A and B are concave, the SPCB problem can be solved in $O((n+m)\log n)$ time.

The following *least weight subsequence* (LWS) problem was introduced in [7]. Given a sequence $\{x_0, x_1, \ldots, x_n\}$ and a real-valued weight function $w(x_i, x_j)$ defined for indices $0 \leq i < j \leq n$, find an integer $k \geq 1$ and a sequence $S = \{0 = i_0 < i_1 < \ldots < i_{k-1} < i_k = n\}$ 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: Let G be a digraph with vertex set $V = \{x_0, x_1, \ldots, v_n\}$, the edge set $\{x_i \rightarrow x_j \mid 0 \leq i < j \leq n\}$ 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 $O(n^2)$ time to solve. The weight function w is *concave* if the following hold:

$$w(x_{i_1}, x_{j_1}) + w(x_{i_2}, x_{j_2}) \leq w(x_{i_2}, x_{j_1}) + w(x_{i_1}, x_{j_2}) \quad \text{for} \quad 0 \leq i_1 \leq i_2 \leq j_1 \leq j_2 \leq n \tag{2}$$

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 [7]. Similar algorithms were also developed in [5, 6]. Wilber discovered an elegant linear time algorithm for solving this problem [9]. All these algorithms assume each entry $w(i,j)$ can be computed in constant time. In this paper we only consider the concave LWS problem. ¿From now on, the phrase "LWS problem" always means the concave LWS problem.

We will show that an instance of the SPCB problem defined by concave matrices A and B can be reduced to an instance of an *enhanced* version of the LWS problem. However, in the reduced problem, the weight matrix w is the product matrix $A \times B$ (with operators min and +). Thus an entry $w(x_i, x_j)$ cannot be evaluated in $O(1)$ time. So when using Wilber's algorithm to solve our problem, it is necessary to modify the algorithm and its analysis.

We also discuss two applications of the SPCB problem. The first one is the *Traveling Salesman* problem (TSP) for points on an n-vertex convex polygon C. Given two points x and y on C, we want to find a Hamiltonian path P connecting all points of C from x to y such that the total weight of P is minimized, where the distance between any two points is the Euclidean distance between them. This problem can be solved in $O(n^2)$ time by dynamic programming [8]. It was posed in [8] as an open problem whether there exists an $o(n^2)$ algorithm for solving the problem. We show the TSP problem for this case can be reduced to the SPCB problem and solved in $O(n \log n)$ time.

The second application we consider is the *Minimum Latency Tour* (MLT) problem discussed in [4]. Given a set S of n points, a symmetric distance matrix w, and a tour T which visits the points of S in some order, the *latency* of a point p is the length of the tour from the staring point to p. The *total latency* $w(T)$ of T is the sum of the latencies of all points. We wish to find a tour T such that $w(T)$ is minimized. Of particular interest is the case when the distance matrix satisfies the triangle inequality. This problem is also known as the *delivery-man* problem or the *traveling repairman* problem in the literature (see [4] for more discussions and references). Although it looks similar to the TSP problem, the MLT problem is very different from the TSP problem in nature [4]. For general case, the MLT problem is NP-complete [4]. Even for points on a tree or on a convex polygon, it is not known whether the MLT problem is in P or NP-complete [4]. The case where points are on a straight line was considered in [1, 4]. 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 point 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 MLT problem for this special case can be solved in $O(n^2)$ time by dynamic programming [1, 4]. We show the MLT problem for this case can be reduced to the SPCB problem and solved in $O(n \log n)$ time.

2 Background

Given two matrices $A[0..n, 0..m]$ and $B[0..m, 0..n]$, $C[0..n, 0..n] = A \times B$ is defined by:

$$C[i,j] = \min_{0 \leq k \leq m} (A[i,k] + B[k,j]) \tag{3}$$

Lemma 2.1 *[10]. If both A and B are concave, so is C.*

For $0 \leq i \leq n$ and $0 \leq j \leq m$, let $I(i,j)$ denote the smallest index k that realizes the minimum value in definition (3).

Lemma 2.2 *[10]. For any i, j $(0 \leq i < n,\ 0 \leq j < m)$, we have: $I(i,j) \leq I(i,j+1) \leq I(i+1,j+1)$.*

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 2.2, $(i,j) \prec (i',j')$ implies $I(i,j) \leq I(i',j')$.

Lemma 2.3 *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.*

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 each entry of M can be evaluated in $O(1)$ time [3]. Following [7], we will refer to the algorithm in [3] as SMAWK algorithm.

3 The Enhanced Least Weight Subsequence Problem

In this section, we introduce the *enhanced* LWS problem, and show the SPCB problem can be reduced to it. 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** $0 \leq i,j \leq n$ such that $w(x_i, x_i) \geq 0$ for all $0 \leq i \leq n$. 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), 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, we are 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 $e = x_i \rightarrow x_j$ be an edge of G. If $i < j$, e is called a *forward* edge. If $i = j$, e is called a *self loop*. If $i > j$, e is called a *backward* edge. We require the weight of self loops of G are non-negative since otherwise the weight of the shortest path in G would be $-\infty$.

Lemma 3.1 *For any instance of the enhanced LWS problem, there exists a shortest x_0 to x_n path consisting of only forward edges.*

Lemma 3.1 implies that there are no negative cycles in any instance of the enhanced LWS problem. It also implies we can ignore all backward edges and self-loops when solving the enhanced LWS problem.

Consider an instance of the SPCB problem defined by a complete bipartite digraph $G = (X, Y, E)$ and concave weight matrices A and B. Let $G' = (X, E')$ be the complete digraph on X with 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 $P' = \{0 = i_0 < i_1 < \ldots < i_k = n\}$ 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 $P = \{x_0 = x_{i_0} \to y_{j_1} \to x_{i_1} \to y_{j_2} \to \ldots \to y_{j_k} \to x_{i_k} = x_n\}$ is a path in G from x_0 to x_n. Let $w(P')$ denote the weight of P' in G'. Let $w(P)$ denote the weight of P in G. Clearly, $w(P) = w(P')$. We will show $w(P)$ is minimum among all x_0 to x_n paths 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 $w(Q)$. So, without loss of generality, we can assume $j'_l = I(i'_{l-1}, i'_l)$ for all $1 \leq l \leq p$. Hence the weight of Q_l is $w[i'_{l-1}, i'_l]$. Thus Q corresponds to an x_0 to x_n path $Q' = \{0 = i'_0, i'_1, \ldots, i'_p = n\}$ in G' with $w(Q') = w(Q)$. Since the weight of P' is minimum among all such paths in G', we have $w(P) = w(P') \leq w(Q') = w(Q)$. So P is a shortest x_0 to x_n path in G.

Lemma 3.2 *Let A and B be two concave matrices such that all main diagonal entries of the product matrix $w = A \times B$ are non-negative. If the enhanced LWS problem defined by the matrix w can be solved in $T(n, m)$ time, then the SPCB problem defined by A and B can be solved in $O(T(n, m) + m \log n)$ time.*

We would like to use Wilber's algorithm in [9] to solve our enhanced LWS problem. We have to overcome two difficulties, however. 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 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 following two sections, respectively.

4 Wilber's Algorithm

In this subsection, we briefly describe Wilber's algorithm for solving the LWS problem. We assume the readers are familiar with [9]. Then we show that Wilber's algorithm can be used to solve the enhanced LWS problem without any change.

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 the last element is x_i. (That is, the lowest weight subsequence of the form $0 = l_0 < l_1 < \ldots < l_{k-1} = i < l_k = j$). Then we have:

$$\begin{cases} f(j) = \min_{0 \leq i < j} g(i, j) & \text{for } 1 \leq j \leq n \\ g(i, j) = f(i) + w(x_i, x_j) & \text{for } 0 \leq i < j \leq n \end{cases} \tag{4}$$

Adding $f(i_1) + f(i_2)$ to both sides of inequality (2) and apply definition (4), we get:

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

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 computation). Our goal is to determine the row index of the minimum value in each column of g. So we would like to simply apply SMAWK algorithm. But we cannot, because for $i < j$, the value of $g(i, j)$ depends on $f(i)$ which depends on all values of $g(l, i)$ for $0 \leq l < i$. So we cannot compute the value of g in $O(1)$ time as required by SMAWK algorithm.

Wilber's algorithm starts in the upper left corner of g and work rightwards and downwards, at each iteration learning enough new values for 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 value of f and g. The currently computed value for $f(j)$ is denoted by $F(j)$, and will sometimes be incorrect. The currently computed value of $g(i,j)$ is denoted by $G[i,j]$, and is always computed as $F[i] + w(i,j)$. So $G[i,j] = g(i,j)$ 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. First, set $F[0] \leftarrow c \leftarrow r \leftarrow 0$. Then, **while** $(c < n)$, perform the following 5 steps:

1. $p \leftarrow \min\{2c - r + 1, n\}$.
2. Apply 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 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** $F[j_0] \leftarrow H[j_0]$; $r \leftarrow c + 1$; $c \leftarrow j_0$.

Each time we are at the beginning of the loop, the following invariants hold:
(1) $r \geq 0$ and $c \geq r$.
(2) For each $j \in [0, c]$, $F[j] = f(j)$.
(3) All minima in columns $c + 1$ through n of g are in rows $\geq r$.
These invariants are clearly satisfied at the start when $r = c = 0$.

Invariant (2) implies that $G[i,j] = g(i,j)$ for all j and all $i \in [0, c]$. So the entries of 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 subcolumn $g(r, c; j)$. Also, since S contains all 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 for if we add $F[i_1] + F[i_2]$ to both sides of (2), we get $G[i_1, j_1] + G[i_2, j_2] \leq G[i_1, j_2] + G[i_2, j_1]$. Thus 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 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 verify that $F[j] = f(j)$ for $j \in [c+2, p]$ (this is the case if $H[j] \geq F[j]$ for all $j \in [c+2, p]$); or 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 entries in L by $+\infty$. Then w' defines an instance of the (ordinary) LWS problem. By Lemma 3.1, 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 SPCB 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 distroy some properties of w that are crucial for obtaining a fast algorithm. Fortunately, we will show 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 effects on the computation of Wilber's algorithm. The only place where Wilber's algorithm needs the entries in L is step 3, where 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 does not affect the computation. In the following we show case (c) cannot occur. Toward a contradiction, assume 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: $i = j$. Then $H[j] = G[j,j] = F[j] + w(j,j) \geq F[j]$. This is impossible.

Case 2: $i > j$. In this case, $H[j] = G[i,j] = F[i] + w(i,j)$. Recall that $F[i]$ is the minimum value of the subcolumn $G[r,c;i]$. Suppose $F[i] = G[t,i] = F[t] + w(t,i)$ 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 4.1 *Wilber's algorithm solves the enhanced LWS problem without changing the weight matrix.*

5 Implementation and Time Analysis

In this section, we discuss how to use Wilber's algorithm to solve an instance of the enhanced LWS problem that is derived from an instance of the SPCB problem. Namely, the enhanced LWS problem is defined by a product matrix $C = A \times B$ where $C[i,i] \geq 0$ for all i.

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

Lemma 5.1 *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 $k_2 = I(p,p)$.*

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, $r_{i+1} = c_i + 1$, $c_{i+1} = j_0$ $(c_i + 2 \leq j_0 \leq p_i)$; 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 the 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 the left and the right end of the ith span, respectively. Note that for a type 1a or 1b iteration, the left end is not changed, the right end increases. For a type 2a or 2b iteration, the left end increases, the right end may increase, decrease, or remain unchanged. For an interval $[t, t+1]$ $(0 \leq t < n)$, we say 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. Once the left end of a span is $\geq t+1$, $[t, t+1]$ will never be covered by subsequent spans. First we make the following obvious observations.

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

(2) If the $(i+1)$th 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+twice the length of the ith span.

(3) Suppose the $(i+1)$th iteration is of type 2a or 2b. Since $p_i \leq 2c_i - r_i + 1$, we have $c_i \geq (p_i + r_i - 1)/2$. Hence: $r_{i+1} = c_i + 1 \geq (p_i + r_i - 1)/2 + 1$.

(4) Suppose an interval $[t, t+1]$ is covered by the ith span $[r_i, p_i]$. If the $(i+1)$th iteration is of type 1a or 1b, and the $(i+2)$th 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 $[r_{i+2}, p_{i+2}]$ and subsequent spans.

Lemma 5.2 *Any interval $[t, t+1]$ $(0 \le t < n)$ is covered by at most $2\log n + 2$ spans.*

Theorem 5.3 *Given two concave matrices A and B such that the main diagonal entries of $A \times B$ are non-negative, the SPCB problem defined by A and B can be solved in $O(n + m\log n)$ time.*

6 TSP Problems for Points on a Convex Polygon

Let C be a convex polygon. For any two points x and y on C, let $d(x,y)$ be the Euclidean distance between x and y. Let P be a path connecting points on C. The weight of P is $w(P) = \sum_{e \in P} d(e)$. Given two points x, y of C, we wish to find a Hamiltonian path P connecting all points of C from x to y such that $w(P)$ is minimized. One can show that no two edges of P cross each other.

Let $x = x_0, x_1, \ldots, x_n = y$ be the points of C from x to y in clockwise order. Let C_X denote this portion of C. Let $x = y_0, y_1, \ldots, y_m = y$ the the points of C from x to y in counterclockwise order. Let C_Y denote this portion of C. 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 $x_n = y_m$. Depending on whether the first and the last edge of P is in C_X or in C_Y, there are four possibilities. We assume both the first and the last edge of P are in C_Y. Then P must be of the following form.

$$x_{i_0} = x_0 = y_0 \xrightarrow{Y} y_{j_1} \rightarrow x_1 \xrightarrow{X} x_{i_1} \rightarrow y_{j_1+1} \xrightarrow{Y} y_{j_2} \rightarrow x_{i_1+1} \xrightarrow{X} \ldots \xrightarrow{X} x_{i_t} = x_{n-1} \rightarrow y_{j_t+1} \xrightarrow{Y} y_m = x_n$$

for some $0 < j_1 < j_2 < \ldots < j_t < m-1$ and $0 = i_0 < i_1 < i_2 < \ldots < i_t = n-1$. We use the following *dummy path* P' to represent P:

$$P' = \{0 = i_0 \rightarrow j_1 \rightarrow i_1 \rightarrow j_2 \rightarrow i_2 \rightarrow \ldots \rightarrow j_{t-1} \rightarrow i_{t-1} \rightarrow j_t \rightarrow i_t = n-1\}$$

Each edge $i_{l-1} \rightarrow j_l$ and $j_l \rightarrow i_l$ in P' is called a *dummy edge*. P is completely specified by P'. Define the weights of dummy edges as follows.

$$A[x_i, y_j] = w(x_i \rightarrow y_j) = d(x_{i+1}, y_j) - d(x_i, x_{i+1}) \text{ and } B[y_j, x_i] = w(y_j \rightarrow x_i) = d(y_{j+1}, x_i) - d(y_j, y_{j+1})$$

Note that $A[x_0, y_0] = B[y_0, x_0] = 0$. Let $S_X = \sum_{i=1}^{n-1} d(x_{i-1}, x_i)$ and $S_Y = \sum_{j=1}^{m} d(y_{j-1}, y_j)$. Then,

$$w(P) = S_X + S_Y + \sum_{l=1}^{t} A[x_{i_{l-1}}, y_{j_l}] + \sum_{l=1}^{t} B[y_{j_l}, x_{i_l}] \tag{5}$$

Since the term $S_X + S_Y$ in equation (5) is fixed for any P, in order to minimize $w(P)$, we only need to minimize the *reduced weight* $w(P')$ defined by $w(P') = \sum_{l=1}^{t} A[x_{i_{l-1}}, y_{j_l}] + \sum_{l=1}^{t} B[y_{j_l}, x_{i_l}]$

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}. Also, A and B are concave.

Theorem 6.1 *TSP for points on an N-point convex polygon needs $O(N\log N)$ time.*

7 Minimum Latency Problem 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 $p \in S$ on T is the length of the tour from the staring point to the first occurrence of p. More precisely, suppose 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 points: $w(T) = \sum_{i=1}^{n} w(p_i)$.

We wish to find a tour T with minimum $w(T)$. In this section, we show that the MLT problem for points on a straight line can be reduced to the SPCB 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: $w(T_X) = \sum_{k=1}^{n}(x_k - x_{k-1})(n - k + 1)$ and $w(T_Y) = \sum_{k=1}^{m}(y_k - y_{k-1})(m - k + 1)$. $w(T_X)$ 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 edge of T is to the left or to the right, there are four possibilities. We assume the first edge is to the right and the last edge is to the left. Then T must be of the following form:

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

for some $0 = j_0 < j_1 < j_2 < \ldots j_{t-1} < j_t = m$ and $0 = i_0 < i_1 < \ldots i_{t-1} < i_t = n$. (The notation $x_i \overset{\Delta}{\rightarrow} y_j$ denotes a path from x_i to y_j consisting of several edges). We use the following *dummy tour* T' to represent T:

$$T' = \{0 = i_0 \rightarrow j_1 \rightarrow i_1 \rightarrow \ldots \rightarrow i_{t-1} \rightarrow j_t = m \rightarrow i_t = n\}$$

T is completely specified by T'. We can show the following:

$$w(T) = w(T_X) + w(T_Y) + \sum_{l=0}^{t} y_{j_l}[2n + 2m - 2i_{l-1} - 2j_l] + \sum_{l=0}^{t} x_{i_l}[2n + 2m - 2i_l - 2j_l].$$

Define the *reduced weight* of the dummy tour T' to be:

$$w(T') = \sum_{l=0}^{t} y_{j_l}[n + m - i_{l-1} - j_l] + \sum_{l=0}^{t} x_{i_l}[n + m - i_l - j_l].$$

Then, we have: $w(T) = w(T_X) + w(T_Y) + 2w(T')$.

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:

$$A[i,j] = w(x_i \rightarrow y_j) = y_j(n + m - i - j) \text{ and } B[j,i] = w(y_j \rightarrow x_i) = x_i(n + m - i - j).$$

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. We can also show that both A and B are concave. Thus, we have:

Theorem 7.1 *The MLT problem for a set of N points on straight line needs $O(N \log N)$ time.*

References

[1] F. Afrati, *et al.*, The Complexity of the Traveling Repairman Problem, Informatique Theorique et Applications (Theoretical Informatics and Applications) 20 (1986) 79-87.

[2] A. Aggarwal and J. Park, Notes on Searching in Multidimensional Monotone Arrays, *FOCS'88*.

[3] A. Aggarwal, *et al.*, Geometric Applications of a Matrix ..., Algorithmica 2 (1987).

[4] A. Blum, *et al.*, The Minimum Latency Problem, *STOC'94*.

[5] D. Eppstein, Sequence Comparison with Mixed Convex and Concave Costs, J. Alg. 11 (1990).

[6] Z. Galil and R. Giancarlo, Speeding-up Dynamic Programming with Applications ..., TCS 64 (1989).

[7] D. S. Hirschberg and L. L. Larmore, The Least Weight Subsequence Problem, SIAMJC 16 (1987).

[8] O. Marcotte and S. Suri, Fast Matching Algorithms for Points on a Polygon, SIAMJC 20 (1991).

[9] R. Wilber, The Concave Least-Weight Subsequence Problem Revisited, J. Alg. 9 (1988).

[10] F. F. Yao, Efficient Dynamic Programming Using Quadrangle Inequalities, *STOC'80*.