Discrete Comput Geom 12:263–280 (1994)

Discrete & Computational
Geometry

Finding a Minimum-Weight k-Link Path in Graphs with the Concave Monge Property and Applications

A. Aggarwal,[1] B. Schieber,[1] and T. Tokuyama[1,2]

[1] Research Division, IBM, T. J. Watson Research Center,
P.O. Box 218, Yorktown Heights, NY 10598, USA
aggarwa@watson.ibm.com
sbar@watson.ibm.com

[2] Research Division, IBM, Tokyo Research Laboratory,
1623-14 Shimo-tsuruma, Yamato, Kanagawa 242, Japan
ttoku@trl.vnet.ibm.com

Abstract. Let G be a weighted, complete, directed acyclic graph (DAG) whose edge weights obey the concave Monge condition. We give an efficient algorithm for finding the minimum-weight k-link path between a given pair of vertices for any given k. The time complexity of our algorithm is $O(n\sqrt{k \log n} + n \log n)$. Our algorithm uses some properties of DAGs with the concave Monge property together with the parametric search technique. We apply our algorithm to get efficient solutions for the following problems, improving on previous results: (1) Finding the largest k-gon contained in a given convex polygon. (2) Finding the smallest k-gon that is the intersection of k half-planes out of n half-planes defining a convex n-gon. (3) Computing maximum k-cliques of an interval graph. (4) Computing length-limited Huffman codes. (5) Computing optimal discrete quantization.

1. Introduction

Let $G = (V, E)$ be a weighted, complete, directed acyclic graph (DAG) with the vertex set $V = \{v_1, v_2, \ldots, v_n\}$. (For convenience, we sometimes represent v_i by i.) For $1 \leq i < j \leq n$, let $w(i, j)$ denote the weight associated with the edge (i, j). (See Fig. 1.)

An edge in a path in G is called a *link* of the path. We call a path in G a *k-link* path if the path contains exactly k links. For any two vertices, i and j, we call a path from i to j a *minimum* k-link path if it contains exactly k links and among all such paths it has the minimum-weight. A weighted DAG, G, satisfies the

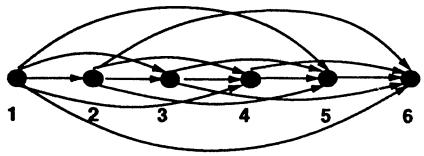

Fig. 1. Complete DAG.

concave Monge property if the inequality $w(i,j) + w(i+1,j+1) \leq w(i,j+1) + w(i+1,j)$ holds for all $1 < i+1 < j < n$.

In this paper we are interested in computing the minimum k-link path from 1 to n in concave Monge DAGs, i.e., weighted DAGs whose weights satisfy the concave Monge property.

Using the results of Aggarwal *et al.* [1] and Aggarwal and Park [2], it is easy to show that the minimum k-link path can be computed in $O(nk)$ time for a concave Monge DAG. The main result of this paper is an $O(n\sqrt{k \log n} + n \log n)$-time algorithm for computing the minimum k-link path. Note that this algorithm is superior to the $O(nk)$-time algorithm when $k = \Omega(\log n)$.

We solve the problem using Megiddo's parametric search technique [17], [9]—a powerful technique for designing algorithms, especially in computational geometry [8]. The original parametric search runs a (sequential version of a) generic parallel p-processor algorithm (called the *guide algorithm*) without knowing the key parameter τ, and calls a *decision algorithm* $\log p$ times at each stage in order to compute the comparisons that involve the unknown parameter.

For our problem, a naive application of the parametric search would not suffice since the known parallel algorithms for it are not efficient enough. Therefore, we design a new guide algorithm in a relaxed model. This guide algorithm has *sequential* steps and *parallel* steps, and the property that all the comparisons that involve the unknown parameter are done in the parallel steps. It is not difficult to see that this guide algorithm can be used for the parametric search. Let t_S be the number of sequential steps, let t_P be the number of parallel steps, and let p be the number of processors in the guide algorithm. Applying parametric search, the resulting algorithm has a time complexity of $O(t_S + t_P \cdot p + t_P \cdot t_D \log p)$, where t_D is the time complexity of the decision algorithm.

It is known that parametric search can sometimes be sped up by refining the parallel guide algorithm [9], [10]; in fact, Frederickson [10] pointed out that the essential requirement for parametric search is to have a "nice" partial order of computation. Our approach is similar in philosophy, although we construct a new guide algorithm instead of refining an existing parallel algorithm. Finally, we note that although it is almost trivial that a guide algorithm in which only the comparisons that involve the unknown parameter are done in parallel is sufficient for parametric search, this paper, to the best of our knowledge, presents the first attempt to use this fact to obtain a more efficient parametric search.

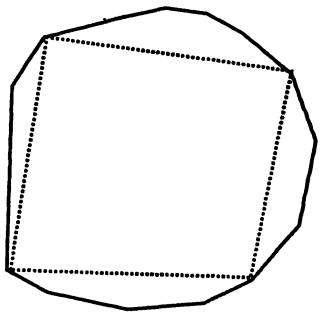

Fig. 2. Max-area inscribed polygon.

1.1. Applications

The minimum k-link path in a concave Monge DAG has several applications. Given below are five such applications to geometric path finding (Applications I and II), interval graphs (Application III), data optimization (Application IV), and data compression (Application V):

Application I. Computing the maximum area k-gon and the maximum perimeter k-gon that are contained in a given convex n-gon. (See Fig. 2.) For this problem Boyce *et al.* [6] provided an $O(nk \log n)$-time algorithm that was later improved by Aggarwal *et al.* [1] to $O(nk + n \log n)$ time. By incorporating the main result of this paper, this problem can now be solved by an algorithm that takes $O(n\sqrt{k \log n} + n \log n)$ time.

Application II. Computing the minimum area k-gon that is the intersection of k half-planes out of n half-planes defining a given convex n-gon. In other words, computing the minimum area circumscribing polygon touching edge-to-edge. (See Fig. 3.) This problem can also be solved in $O(n\sqrt{k \log n} + n \log n)$ time.

Application III. Let H be an interval graph generated by m weighted intervals on n terminals. Given k, find k cliques of H so that the sum of the weights of intervals in the union of the cliques is maximized. (See Fig. 4.) We give an $O(m + n(\sqrt{k \log n} + \log n) \log \log n)$-time algorithm, thereby, improving a previous result of [4].

Application IV. Given a weighted alphabet of size n, we want to find an optimal prefix-free binary code for the alphabet with the restriction that no code string be longer than length k. Larmore and Hirschberg [14] gave an $O(nk)$-time algorithm

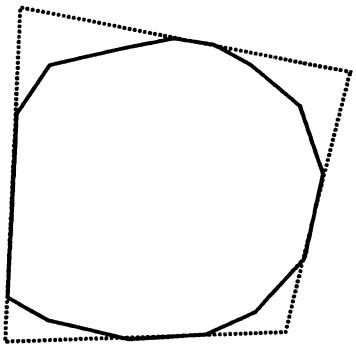

Fig. 3. Min-area inscribed polygon with edge-to-edge contact.

for this problem. Using the reduction of this problem to the min-weight k-link path problem [15], we solve it in $O(n\sqrt{k \log n} + n \log n)$ time.

Application V. Let $f: \{x_1, x_2, \ldots, x_n\} \to \mathscr{R}$ be a real-valued function, where $\mathscr{R}$ is the set of the real numbers and $x_1 \leq x_2 \leq \cdots \leq x_n$ are real numbers. Fix k and consider a sorted set of real numbers $Z = \{z_1, z_2, \ldots, z_k\}$ and a mapping $\psi: \{1, 2, \ldots, n\} \to \{1, 2, \ldots, k\}$. The pair (Z, ψ) is called a quantization, and the sum $\sum_{i=1}^{n} f(x_i)(x_i - z_{\psi(i)})^2$ the error of the quantization. Optimal quantization is the one which minimizes the error. It is easy to see that $\psi^{-1}(j)$ becomes an interval for each $j = 1, 2, \ldots, k$. Quantization can be regarded as a data compression of n data items into k items, as illustrated in Fig. 5. Wu [19] showed that computing optimal quantization can be reduced to finding a minimum-weight k-link path. Hence it can be solved in $O(n\sqrt{k \log n} + n \log n)$ time by applying our algorithm.

The remainder of the paper is organized as follows. Section 2 shows how parametric search can be applied to our problem; as a by-product we show how a simple binary-search of the unknown parameter yields an $O(n \log U)$-time

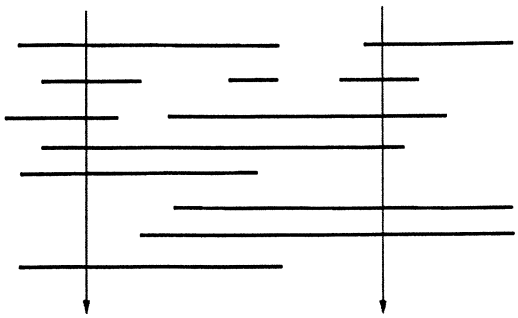

Fig. 4. k maximum weight cliques of an interval graph ($k = 2$).

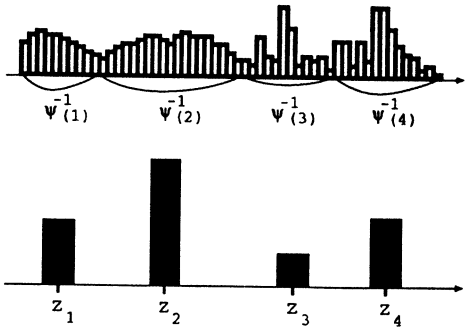

Fig. 5. Quantization ($k = 4$).

algorithm, in case all the edge weights are integral and U is the maximum absolute value of the weights. Section 3 describes our efficient guide algorithm and analyzes the complexity of the resulting parametric search. Section 4 briefly describes the applications, and Section 5 concludes with a few open problems.

2. The Parametric Search

We show how parametric search can be applied to our problem. Let G be our weighted DAG. For a real number τ, define $G(\tau)$ to be the weighted DAG with the same sets of edges and vertices as G, in which each edge $e \in E$ has the weight $w(e) + \tau$ (where $w(e)$ is the weight of e in G). Note that if G has the concave Monge property, then $G(\tau)$ also has this property. Define a *diameter* path in G to be a path from 1 to n.

The following three lemmas are the basis of the parametric search.

Lemma 1. *If, for some τ, the minimum-weight diameter path in $G(\tau)$ has k links, then this path is the minimum k-link diameter path in G.*

Lemma 2. *For any $1 \leq k \leq n - 1$, a real number τ exists such that a minimum-weight diameter path of $G(\tau)$ has k links.*

Lemma 3. *If a minimum-weight diameter path in $G(\tau)$ has k links, then, for every $\xi < \tau$, any minimum-weight diameter path in $G(\xi)$ has at least k links.*

Both Lemmas 1 and 3 hold for any DAG and do no depend on the fact the G has the concave Monge property. Lemma 1 is obvious.

Proof of Lemma 3. Let P and Q be minimum-weight paths in $G(\tau)$ and $G(\xi)$, respectively. Suppose that P has k links, and Q has l links. Let $W_\tau(P)$

denote the weight of P in $G(\tau)$. Then $W_\tau(Q) - W_\tau(P) \geq 0$ and $W_\xi(Q) - W_\xi(P) \leq 0$. Thus,

$$l(\tau - \xi) = W_\tau(Q) - W_\xi(Q) \geq W_\tau(P) - W_\xi(P) = k(\tau - \xi).$$

Since $\tau - \xi > 0$, we have that $l \geq k$. □

We now prove Lemma 2.

Definition 1. An edge (i_1, j_1) *covers* another edge (i_2, j_2) if $i_1 \leq i_2 < j_2 \leq j_1$ and $(i_1, j_1) \neq (i_2, j_2)$.

Let P_1 and P_2 be paths in G. Suppose that a link (i_1, j_1) of P_1 and a link (i_2, j_2) of P_2 exist such that (i_1, j_1) covers (i_2, j_2). We define a *path-swap* operation with respect to this pair of edges. This operation creates two new paths Q_1 and Q_2. Path Q_1 is given by connecting the prefix of P_1 ending at i_1 with the suffix of P_2 starting at j_2 by edge (i_1, j_2). Path Q_2 is given by connecting the prefix of P_2 ending at i_2 with the suffix of P_1 starting at j_1 by edge (i_2, j_1).

Lemma 4. *Let Q_1, Q_2 be paths obtained from P_1 and P_2 by a path-swap operation with respect to (i_1, j_1) and (i_2, j_2). The sum of the weights of paths Q_1 and Q_2 is at most the sum of the weights of paths P_1 and P_2. In particular, if P_1 and P_2 are minimum-weight paths so are Q_1 and Q_2.*

Proof. In case $i_1 = i_2$ or $j_1 = j_2$, clearly, $W(Q_1) + W(Q_2) = W(P_1) + W(P_2)$. Otherwise, $i_1 < i_2 < j_2 < j_1$, and we have

$$\begin{aligned} W(Q_1) + W(Q_2) &= W(P_1) + W(P_2) - w(i_1, j_1) - w(i_2, j_2) + w(i_1, j_2) + w(i_2, j_1) \\ &\leq W(P_1) + W(P_2). \end{aligned}$$

The inequality follows from the concave Monge property of the edge weights. □

For $a \leq b$, let P_a and P_b be paths from v_1 to v_a and from v_1 to v_b, respectively. Suppose that P_a has k_a links, P_b has k_b links, and $k_a > k_b$.

Lemma 5. *For any $0 \leq x \leq k_a - k_b$ there are links $e_a = (i_a, j_a)$ of P_a and $e_b = (i_b, j_b)$ of P_b with the following two properties.*

1. *Edge e_b covers edge e_a.*
2. *The prefix of p_a ending at i_a has x more links than the prefix of P_b ending at i_b.*

Proof. Let $e = (i_b, j_b)$ be the leftmost link of P_b that covers some link of P_a. Such a link must exist since $b \geq a$ and $k_a > k_b$. Suppose that e covers c links of P_a, and let $f = (i_a, j_a)$ be the leftmost such link. Let d be the difference between the length of the prefix of P_a ending at i_a and the length of the prefix of P_b ending at i_b. It

follows from our selection of e that this difference is less than or equal to zero. Observe that for any $d \leq 0 \leq k < c + d$ we can set e_b to e and e_a to one of the links of P_a covered by e and have the two properties of the lemma satisfied. In case $k_a - k_b < d + c$ we are done. Otherwise, there must be another link of P_b that covers some link of P_a. Let $e' = (i'_b, j'_b)$ be the leftmost such link. Again, suppose that e' covers c' links of P_a, and let $f' = (i'_a, j'_a)$ be the leftmost such link. Note that the difference d' between the length of the prefix of P_a ending at i'_a and the length of the prefix of P_b ending at i'_b is less than or equal to $d + c$. Hence, for any $d' \leq d + c \leq k < d' + c'$ we can set e_b to e' and e_a to one of the links of P_a covered by e' and have the two properties of the lemma satisfied. If $k_a - k_b < d' + c'$ we are done. Otherwise, we continue in the same manner. □

Lemma 6. *Let a, b, P_a, P_b, k_a, and k_b be as above. For any k in the range $[k_b, k_a]$, there are paths Q_a with k links from v_1 to v_a and Q_b with $k_a + k_b - k$ links from v_1 to v_b such that the sum of the weights of paths Q_a and Q_b is at most the sum of the weights of paths P_a and P_b. In particular, if P_a and P_b are minimum-weight paths so are Q_a and Q_b.*

Proof. Fix some k in the range $[k_b, k_a]$. By Lemma 5 there are links $e_a = (i_a, j_a)$ in P_a and $e_b = (i_b, j_b)$ in P_b such that edge e_b covers edge e_a, and the prefix of P_a ending at i_a has $k_a - k$ more links than the prefix of P_b ending at i_b. Perform a path swap with respect to e_a and e_b to obtain two paths Q_a and Q_b from v_1 to v_a and v_b, respectively. Since Q_a is created by connecting the prefix of P_b ending at i_b with the suffix of P_a starting at j_a, the length of Q_a is $k_a - (k_a - k) = k$. Similarly, the length of Q_b is $k_a + k_b - k$. Lemma 4 implies that the sum of the weights of paths Q_a and Q_b is at most the sum of the weights of paths P_a and P_b.

Definition 2. Denote the weight of the minimum-weight l-link diameter path in G by $P(l)$.

Corollary 7. *For $1 < l < n - 1$, $2P(l) \leq P(l - 1) + P(l + 1)$.*

Proof of Lemma 2. The number of links of the minimum-weight diameter path goes to 1 and $n - 1$ if τ goes to ∞ and $-\infty$, respectively. Fix some $1 < k < n - 1$. By Corollary 7 $P(k) - P(k + 1) \leq P(k - 1) - P(k)$. We claim that for any $P(k) - P(k + 1) \leq \tau \leq P(k - 1) - P(k)$ there is a minimum-weight diameter path in $G(\tau)$ with k links. Consider some $k < l \leq n$. Applying Lemma 3 it is easy to verify that for all $\tau \geq (P(k) - P(l))/(l - k)$, a minimum-weight k-link path in $G(\tau)$ is no larger than a minimum-weight l-link path. By Corollary 7

$$\begin{aligned} P(k) - P(l) &= P(k) - P(k + 1) + P(k + 1) - P(k + 2) + \cdots + P(l - 1) - P(l) \\ &\leq (l - k)(P(k) - P(k + 1)). \end{aligned}$$

We get that, for all $\tau \geq P(k) - P(k + 1)$, a minimum-weight k-link path in $G(\tau)$ is no larger than a minimum-weight l-link path. Similarly, for $1 \leq l < k$, for all

$\tau \leq P(k-1) - P(k)$, a minimum-weight k-link path in $G(\tau)$ is no larger than a minimum-weight l-link path. □

In the parametric search we search for a value τ_{opt} such that $G(\tau_{opt})$ has a minimum-weight diameter path with k links. Suppose that such a τ_{opt} is found. To compute a minimum-weight k-link path of G (or, equivalently, a minimum-weight diameter path of $G(\tau_{opt})$ with k links) we do the following. Apply the linear-time algorithm for finding a minimum-weight diameter paths in DAGs with the concave Monge property, to find two minimum-weight diameter paths in $G(\tau_{opt})$: P_a with the maximum number of links and P_b with the minimum number of links. It is easy to see that both known linear-time algorithms for this problem, the one given by Wilber [18] and the one by Klawe [11]), can be used to find these paths. Then, to find a minimum-weight diameter path with k links, we apply Lemma 6. It is easy to see that finding the required links e_a and e_b in the proof of Lemma 6 and performing the path swap can be done in time proportional to the length of P_a; that is, $O(n)$ time.

One way to compute τ_{opt} is by binary search. Assume that all edge weights are integral, and let U be the maximum absolute value of the weights.

Lemma 8. *The number of links of the minimum-weight diameter path of* $G(-3U)$ *is* $n-1$, *while that of* $G(3U)$ *is one.*

Proof. Consider $G(-3U)$. Let P be a diameter path with $k < n-1$ links. Then there is a link $e = (v_i, v_j)$ of P such that $j > i+1$. We replace e by the pair of edges (v_i, v_{i+1}) and (v_{i+1}, v_j) to obtain a path with $k+1$ links. It is easy to see that this path has smaller weight than P. Hence, the minimum-weight diameter path of $G(-3U)$ must have $n-1$ links. The statement for $G(3U)$ can be shown similarly. □

Since the weights are integral, the differences between weights are also integral. Recall that any $P(k) - P(k+1) \leq \tau \leq P(k-1) - P(k)$ can be taken as τ_{opt}. Hence, an integral τ_{opt} in the range $[-3U, 3U]$ exists. We can use binary search to find it. Initially, set $\tau_L = -3U$ and $\tau_R = 3U$. Iteratively, compute the minimum-weight path in $G(\lfloor(\tau_L + \tau_R)/2\rfloor)$. If it has k links, then we are done: $\tau_{opt} = \lfloor(\tau_L + \tau_R)/2\rfloor$. Else, if it has more than k links, set $\tau_L = \lfloor(\tau_L + \tau_R)/2\rfloor + 1$, otherwise, set $\tau_R = \lfloor(\tau_L + \tau_R)/2\rfloor - 1$. It is easy to see that τ_{opt} is found after $O(\log U)$ iterations.

Theorem 9. *The binary-search algorithm finds the minimum* k*-link path in* $O(n \log U)$ *time.*

Proof. The algorithm computes the minimum-weight path in concave DAGs $O(\log U)$ times, and each minimum-weight path finding is solved in $O(n)$ time using [11] and [12]. □

We have a weakly polynomial $O(n \log U)$-time algorithm for the minimum k-link path problem. (Bein *et al.* [5] discovered the above weakly polynomial algorithms independently.) This algorithm is faster than the known $O(nk)$-time algorithm when $k = \Omega(\log U)$. Practically, it often suffices to obtain a solution of an approximated system where weights are rounded so that the precision U is bounded by a polynomial of n. The algorithm finds such an approximate solution in $O(n \log n)$ time.

From the theoretical point of view, it is important to design an efficient strongly polynomial algorithm. We can design a strongly polynomial algorithm based on the above binary-search algorithm by using the parametric search paradigm [17]. Assume that there is a parallel algorithm (*guide algorithm*) that computes the minimum-weight path in $G(\tau)$ in t_P time using p processors. Also assume that there is a sequential algorithm (*decision algorithm*) that computes it in t_D time. Then the parametric search scheme [17] finds the minimum-weight k-link path of G in $O(t_P \cdot p + t_P \cdot t_D \log p)$ time.

Unfortunately, no polylogarithmic-time parallel algorithm that uses $O(n \text{ polylog}(n))$ processors is known for the minimum-weight path problem. The known polylogarithmic-time algorithms require $O(n^2)$ processors; hence, they are not suitable for our use. The best-known algorithm that uses $O(n)$ processors requires $O(\sqrt{n} \log n)$ time [15]. Thus, we have the following:

Theorem 10. *The minimum k-link path problem can be solved in $O(n\sqrt{n} \log^2 n)$ time.*

The above time complexity is far from satisfactory, since, for $k < \sqrt{n} \log^2 n$, it is worse than the $O(nk)$-time algorithm given by using [1] and [2]. In the next section we give a better algorithm by using a more sophisticated parametric-search technique. In this algorithm we use as a guide algorithm an algorithm with $O(n\sqrt{k \log n})$ sequential steps and $O(\sqrt{k/\log n})$ parallel steps; this algorithm performs a total of $O(n\sqrt{k \log n})$ work, and the property that all the comparisons that involve the unknown parameter are done in the parallel steps. Applying parametric search, we get an algorithm that runs in $O(n\sqrt{k \log n})$ time.

3. The Guide Algorithm

The outline of the algorithm is very simple. Like other parametric search algorithms, the algorithm maintains an interval (τ_L, τ_R) of the parameter containing τ_{opt}. Whenever the decision algorithm is called this interval is updated, so that every comparison executed so far in the algorithm is independent of the choice of τ provided that $\tau \in (\tau_L, \tau_R)$. We explain in detail how to update the interval later. Initially, $\tau_L = -\infty$ and $\tau_R = \infty$. We fix an integer l, which will be set appropriately in the analysis. For convenience, we assume that both l and n/l are integers. The algorithm has n/l stages. In the tth stage, for $t = 0, \ldots, n/l - 1$, we compute the

minimum-weight paths in $G(\tau_{opt})$ with the minimum number of links and the maximum number of links (from v_1) to v_j, for $tl < j \leq (t+1)l$. If, in the course of this computation, when we update the interval (τ_L, τ_R), we also find τ_{opt}, then we are done. Otherwise, we continue to the next stage.

At the end of stage n/l, we have the minimum-weight diameter paths in $G(\tau_{opt})$ with the minimum number of links and with the maximum number of links. Then we find the one with k links as described above.

Before we describe stage i of the algorithm we need a few definitions.

Definition 3. Let $K(i)$ be the number of links in a minimum-weight path from v_1 to v_i in $G(\tau_{opt})$ with the minimum number of links.

Definition 4. Let P be a path from v_1 to v_j. The left endpoint of the last link of P is called the anchor of P. If the anchor of a path P is in an interval I, we say that the path P has its anchor in I.

We now describe how to compute the minimum-weight paths from v_1 to $v_{tl+1}, \ldots, v_{(t+1)l}$ in $G(\tau_{opt})$. Recall that this computation is done without knowing the value of τ_{opt}. The input to this stage is the current interval (τ_L, τ_R) of the parameter, and weights (as linear functions of the parameter τ) of minimum-weight paths in $G(\tau_{opt})$ with the minimum number of links and the maximum number of links to v_j, for $1 \leq j \leq tl$. We describe only how to find minimum-weight paths with the minimum number of links. The ones with maximum number of links are found similarly. From now on, whenever we refer to a minimum-weight path we refer to one with the minimum number of links.

Lemma 11. *For all $1 \leq i < j \leq n$, the following inequality holds: $K(i) \leq K(j)$.*

Proof. Straightforward from Lemma 6. □

Lemma 12. *For any $j > tl$, the minimum-weight path in $G(\tau_{opt})$ from v_1 to v_j that is anchored in $[1, tl]$ has either $K(tl)$ or $K(tl) + 1$ links.*

Proof. Let P be a minimum-weight path from v_1 to v_j anchored in $[1, tl]$, and let v_x be the anchor of P. Since $x \leq tl$, it follows from Lemma 11 that $K(x) \leq K(tl)$, and thus the length of P is bounded from above by $K(tl) + 1$. Suppose that the length of P is less than $K(tl)$. This implies that the length of the minimum-weight path from v_1 to v_x is less than $K(tl) - 1$. Let Q be a minimum-weight path from v_1 to v_{tl} with $K(tl)$ links, and let v_s be the anchor of Q. Since the length of the minimum-weight path from v_1 to v_s is $K(tl) - 1$, $x < s$. Thus, $e = (v_x, v_j)$ covers $f = (v_s, v_{tl})$. By executing path swap with respect to e and f, we obtain a pair of paths P' and Q'. Path P' is from v_1 to v_j and path Q' is from v_1 to v_{tl}. Because of Lemma 4, Q' is a minimum-weight path. However, Q' has less than $K(tl)$ links; a contradiction.

For $tl < j \leq (t+1)l$, define a *candidate h-link path* from v_1 to v_j to be a minimum-weight h-link path among the paths whose prefix is a minimum-weight

path anchored in $[1, tl]$ in $G(\tau_{opt})$, and whose suffix consists of some (possibly zero) links in $[tl + 1, (t + 1)l]$. Note that a minimum-weight h-link path from v_1 to v_j in $G = G(0)$ need not be a candidate h-link path. However, it is easy to see that *if some minimum-weight path from v_1 to v_j in $G(\tau_{opt})$ has h links, then any candidate h-link path is a minimum-weight path from v_1 to v_j in $G(\tau_{opt})$.*

Stage i consists of four steps:

Step 1. For all $tl < j \leq (t + 1)l$, compute the minimum-weight path in $G(\tau_{opt})$ from v_1 to v_j anchored in $[1, tl]$.

Step 2. Find an integer m satisfying

$$K((t + 1)l) - K(tl) \leq m \leq 2(K((t + 1)l) - K(tl).$$

Step 3. For all $tl < j \leq (t + 1)l$, compute candidate h-link paths in $G(\tau_{opt})$ from v_1 to v_j, for all $h = K(tl) + 1, \ldots, K(tl) + m$.

Step 4. For each $tl < j \leq (t + 1)l$, find the minimum-weight path in $G(\tau_{opt})$ among the candidate *h-link paths from v_1* to v_j found in steps 1 and 3.

It is clear that the final path computed in the step 4 is the minimum-weight path from v_1 to v_j in $G(\tau_{opt})$.

We now describe each of the steps in detail.

Step 1. By Lemma 12 each of the minimum-weight paths in $G(\tau_{opt})$ from v_1 to v_j anchored in $[1, tl]$ has either $K(tl)$ or $K(tl) + 1$ links. For each $tl < j \leq (t + 1)l$, we first find a minimum-weight path from v_1 to v_j anchored in $[1, tl]$ with $K(tl)$ links. Then we find such a path with $K(tl) + 1$ links. This computation is independent of τ since we compare between paths with the same number of links. Finally, we compare the two paths by applying parametric search.

We show how to find the minimum-weight paths with $K(tl)$ links. Recall that we already computed the minimum-weight paths in $G(\tau)$ from v_1 to all vertices in $[1, tl]$. Since we are interested in minimum-weight paths anchored in $[1, tl]$ with $K(tl)$ links, we may consider only the vertices v_j in $[1, tl]$, such that the minimum-weight path from v_1 to v_j has $K(tl) - 1$ links. Suppose that there are n' such vertices. Consider the $l \times n'$ matrix in which the (j, i)th entry is the length of the minimum-weight path from v_1 to the ith such vertex plus the weight of the edge connecting this vertex to v_j. It is not difficult to see that:

(i) This matrix has the concave Monge property.

(ii) The minimum entry in row j corresponds to the minimum-weight path from v_1 to v_j with $K(tl)$ links.

Hence, all the minimum-weight paths can be found in $O(n)$ time by applying the matrix-search algorithm of [1]. Note that the matrix need not be stored explicitly. Instead, each entry can be computed upon demand. The minimum-weight paths with $K(tl) + 1$ links are found similarly.

Now, for each $tl < j \leq (t + 1)l$, we have the minimum-weight paths with $K(tl)$ links and $K(tl) + 1$ links. To compare them we apply the parametric-search

paradigm. When two paths are compared, we compute the critical value ξ of the parameter, such that for all $\tau > \xi$ the path with $K(tl)$ links is of smaller weight in $G(\tau)$, and for all $\tau < \xi$ the path with $K(tl) + 1$ links is of smaller weight in $G(\tau)$. If this critical value is not in the interval (τ_L, τ_R), then the comparison is independent of τ. Otherwise, we execute the sequential decision algorithm to find minimum-weight diameter paths with the minimum number of links and the maximum number of links in $G(\xi)$. If $G(\xi)$ has a minimum-weight diameter path with k links, we can report ξ as τ_{opt} and find such a path. If the minimum-weight diameter path with the minimum number of links has more than k links, then $\tau_{\text{opt}} > \xi$, and we can set τ_L to ξ. Similarly, if the minimum-weight diameter path with the maximum number of links has less than k links, then $\tau_{\text{opt}} < \xi$, and we can set τ_R to ξ. The parametric-search paradigm uses a parallel algorithm to reduce the number of calls to the decision algorithm. All the l comparisons can be done with $O(l)$ processors in $O(1)$ time if the parameter τ is given. Hence, the associated parametric-search algorithm runs in $O(n \log l)$ time. (Recall that the decision algorithm is the $O(n)$ minimum-weight path algorithm.)

We postpone the description of step 2. Assume that m has already been computed. Next, we describe steps 3 and 4.

Step 3. The key fact is that the computation executed in this substep is independent of the parameter τ. Let A be the matrix of the edge weights between the vertices $v_{tl+1}, \ldots, v_{(t+1)l}$. Let $\mathbf{x}$ be the vector of weights of the minimum-weight paths from v_1 to $v_{tl+1}, \ldots, v_{(t+1)l}$ anchored in $[1, tl]$ in $G(\tau)$, for any $\tau \in (\tau_L, \tau_R)$. This vector is computed in previous stages. For $tl < j \leq (t+1)l$, let $\mathbf{x}_1$ be the vector whose jth entry is the weight of the minimum-weight path from v_1 to v_j that contains $K(tl)$ links, if such a path exists, and infinity otherwise. Let $\mathbf{x}_2$ be the vector whose jth entry is the weight of the minimum-weight path from v_1 to v_j that contains $K(tl) + 1$ links, if such a path exists, and infinity otherwise. From Lemma 11 it follows that the finite entries of $\mathbf{x}_1$ and $\mathbf{x}_2$ are contiguous. From Lemma 12 it follows that the paths considered in computing vector $\mathbf{x}$ have either $K(tl)$ or $K(tl) + 1$ links. Thus, the vector $\mathbf{x}$ is the entry-wise minimum of $\mathbf{x}_1$ and $\mathbf{x}_2$.

Consider the semiring defined over the reals with the operations $\{\min, +\}$. For an $l \times l$ matrix A and an l vector $\mathbf{z}$, the product $\mathbf{w} = A\mathbf{z}$ is defined as $w_s = \min_{i=1,2,\ldots,l}\{A_{s,i} + z_i\}$. The following proposition is obvious from the definition:

Proposition 13. *For a given h, all candidate h-link paths from v_1 to $v_{tl+1}, \ldots, v_{(t+1)l}$ are obtained by computing* $\min\{A^{h-K(tl)}\mathbf{x}_1, A^{h-K(tl)-1}\mathbf{x}_2\}$.

The above operation is done independently of the key parameter τ, since we compare paths with the same number of links. Hence, for any given m, we can find all candidate h-link paths for $h = K(tl), K(tl) + 1, \ldots, K(tl) + m$ in $2m$ multiplications of an $l \times l$ matrix by l vectors. (Note that the computation can be done as a sequence of matrix–vector multiplications so that no multiplication between matrices is done.) Since the matrices in all these products have the concave Monge property, each product can be computed in $O(l)$ time using the matrix-search algorithm of [1].

Step 4. In this step we have to compare the candidate h-link paths from v_1 to v_j, for $K(tl) \leq h \leq K((t+1)l)$ and $tl < j \leq (t+1)l$, and find the minimum-weight path among them at $\tau = \tau_{\text{opt}}$. These comparisons depend on the parameter τ. Here, we apply the parametric-search paradigm. To do the computation efficiently we use the unimodality of the weights of the candidate paths as given in the following lemma.

Lemma 14. *For $1 < h < n$, and for $tl < j \leq (t+1)l$, let $W_h(j)$ be the weight of the candidate h-link path from v_1 to v_j. Then $W_h(j) \leq \max\{W_{h+1}(j), W_{h-1}(j)\}$.*

Proof. Let P be an $(h-1)$-link candidate path and let Q be an $(h+1)$-link candidate path from v_1 to v_j. By Lemma 5 there is a link $e = (v_x, v_y)$ in P and a link $f = (v_a, v_b)$ in Q such that e covers f, and the number of links in the prefix of P ending at v_y, denoted P_y, is one less than the number of links in the prefix of Q ending at v_b, denoted Q_b. Note that $x \leq a < b \leq y$. We have three different cases:

Case 1: $a \leq tl$. In this case both P_y and Q_b are anchored in $[1, tl]$. From the definition of a candidate path it follows that both P_y and Q_b are minimum-weight anchored paths. Since $b \leq y$, by Lemma 11, the number of links in P_y must be larger than or equal to that of Q_b. Thus, this case cannot happen.

Case 2: $tl < x$. We execute path swap with respect to e and f to obtain a pair P' and Q' of paths. Both P' and Q' are candidate paths since the path swap affects only vertices not in $[1, tl]$. Both P' and Q' have h links. By Lemma 6 the sum of their weights is no more than $W_{h+1}(j) + W_{h-1}(j)$. Hence, one of them has weight bounded by the maximum of these two weights. Thus, the lemma holds in this case.

Case 3: $x \leq tl < a$. We execute path swap with respect to e and f to obtain a pair of paths P' and Q'. Since $tl < a$, the path P' given by connecting the prefix of Q ending at v_a with the suffix of P starting at v_y is a candidate path with h links. However, since $x \leq tl$, the path Q' given by connecting the prefix of P ending at v_x with the suffix of Q starting at v_b may not be a candidate path. As in Case 2, at least one of P' and Q' has weight bounded by $\max\{W_{h+1}(j), W_{h-1}(j)\}$. If P' is such a path, then we are done. Suppose that this is not the case, and the weight of P' is larger than this maximum. In this case the weight of Q' must be less than $\min\{W_{h+1}(j), W_{h-1}(j)\}$, thus less than $W_{h-1}(j)$.

Consider the prefix of Q' ending at v_b and denote it by R. By our construction R is the prefix of P ending at v_x followed by the edge (v_x, v_b). If R is a minimum-weight path from v_1 to v_b anchored by $[1, tl]$, then we are done since in this case Q' is a candidate path. Suppose that this is not the case. Let R' be a minimum-weight path from v_1 to v_b anchored in $[1, tl]$. We claim that the number of links of R' is the same as the number of links of R. To see this, observe that the number of links of the minimum-weight anchored paths from v_1 to v_y and from v_1 to v_a is the same as this of R. Since $a < b < y$, Lemma 11 implies that this is also the number of links of R'. Let Q'' be the path obtained from Q' by

replacing R by R'. Clearly, Q'' is a candidate path. Since the weight of Q'' is at most $W_{h-1}(j)$, and the number of links of Q'' is h, the lemma in this case follows. □

Fix some $tl < j \leq (t+1)l$. Lemma 14 implies that the weights of the candidate h-link paths from v_1 to v_j have no local maxima with respect to the link number. This implies that any local minimum must be also a global minimum. Consequently, given τ_{opt} we can find the minimum-weight candidate path to v_j sequentially using binary search in $O(\log m)$ time. Kruskal [13] showed that this search can be done in constant time using m processors. Thus, to find the minimum for all $tl < j \leq (t+1)l$ in constant time we need ml processors. It follows that without the knowledge of τ_{opt} this can be done using the parametric-search paradigm in $O(ml + n \log(ml))$ time. (Recall that the decision algorithm is the $O(n)$ minimum-weight path algorithm.)

We conclude with the description of step 2.

Step 2. We find an m in the range $[(K((i+1)l) - K(tl)), 2(K((i+1)l) - K(tl))]$ by a variant of binary search. Lemma 14 implies that the minimum-weight path from v_1 to $v_{(t+1)l}$ in $G(\tau_{\text{opt}})$ is given by the smallest h that locally minimizes the weight of the candidate h-link path to $v_{(t+1)l}$. Thus, it is easy to verify whether the path to $v_{(t+1)l}$ is indeed the right one. Based on this observation we "guess" m, and try to verify our guess. Initially, we set $m = 3$ and execute steps 3 and 4. If the minimum-weight candidate path found by the algorithm is a local minimum, then the path is the true minimum-weight path. Otherwise, we double m and repeat the process. The time complexity of the whole process is dominated by the time complexity of the last iteration.

Theorem 15. *The minimum k-link diameter path of a concave Monge DAG is found in $O(n\sqrt{k \log n} + n \log n)$ time and $O(n\sqrt{k \log n})$ space.*

Proof. We concentrate on the last iteration of step 2. The total amount of time required for step 1 in the n/l stages is $O(n^2/l \cdot \log l)$. Summing over all stages step 3 requires

$$O\left(\sum_{i=1}^{n/l} (K((t+1)l) - K(tl)) \cdot l\right) = O(kl).$$

Step 4 requires

$$O\left(\sum_{i=1}^{n/l} (K((t+1)l) - K(tl)) \cdot l + n \log n\right) = O(kl + n^2/l \cdot \log n).$$

We conclude that the algorithm requires $O(n^2/l \cdot \log n + kl)$. Setting

$$l = \min\{n\sqrt{\log n/k}, n\}$$

implies the claim on the time complexity. To get the claim on the space complexity note that $O(kl)$ space is needed to store the weights of the candidate paths. The rest of the computation requires linear space. □

4. Applications

We briefly describe how to apply our algorithm to get efficient algorithms for the five problems mention in Section 1.

Application I. Finding the maximum area inscribed k-gon of a convex n-gon. Boyce *et al.* [6] showed that if the maximum area k-gon containing a fixed vertex is given, then the maximum area k-gon can be found in $O(n \log n)$ time using the *interleaving property*. Aggarwal *et al.* [1] showed that the distance matrix involved in computing the maximum area inscribed polygon has the convex Monge property. Since finding the *maximum*-weight path in convex DAGs is equivalent to finding the *minimum*-weight path in concave DAGs, we can apply our algorithm to achieve an $O(n\sqrt{k \log n} + n \log n)$-time algorithm for the problem.

Application II. Finding the minimum area k-gon defined by k half-planes out of n half-planes defining a convex n-gon. Let $e_1, \ldots, e_n$ be the clockwise list of the edges of the convex n-gon, and let $e_{n+1} = e_1$. For $1 \leq i < j \leq n + 1$, define *area*(i, j) as the area of the region bounded by the right-extension of the edge e_i, left-extension of the edge e_j, and the convex chain from e_i to e_j of the polygon (Fig. 6). Note that *area*(i, j) can be infinity for some i and j. Define G to be the weighted DAG with vertex set $V = \{v_1, v_2, \ldots, v_{n+1}\}$, in which the weight of edge (i, j) is *area*(i, j), for $1 \leq i < j \leq n + 1$. It is not difficult to verify that G satisfies the concave Monge property. The minimum-weight k-link diameter path in G corresponds to the minimum area k-gon defined by the half-plane that corresponds to e_1 and $k - 1$ additional half-planes out of the remaining $n - 1$ half-planes defining the convex n-gon. As in the previous application, if the minimum area k-gon containing a fixed half-plane is given, then the minimum area k-gon can be found in $O(n \log n)$ time using the *interleaving property*. Thus, we can apply our algorithm to achieve the desired time bound.

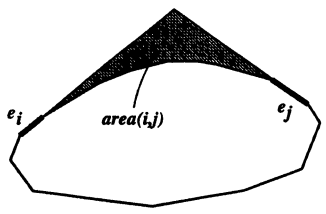

Fig. 6. *area*(i, j).

Application III. Computing k maximum-weight cliques in an interval graph $H(S)$, where S is a set of m intervals whose endpoints are integers in the closed interval $[1, n]$. This problem can be reduced to the problem of finding a minimum $(k+1)$-link path in a DAG with $n+2$ nodes as follows. Define a complete weighted DAG on the vertices $\{0, \ldots, n+1\}$, where the weight of edge (i, j), for $i < j$, is the total weight of the intervals in S contained in the (open) interval (i, j). Consider a $(k+1)$-link path $i_0 = 0, i_1, \ldots, i_k, i_{k+1} = n+1$. Note that the total weight of this path is the sum of the weights of all intervals that do not contain any of the points in $\{i_1, \ldots, i_k\}$. Thus, finding such a path of minimum-weight is equivalent to finding k points such that the total weight of the intervals containing these points is maximized. It is easy to see that the defined DAG satisfies the concave Monge property, and that each edge weight can be computed in $O(\log n)$ time after $O(m \log n)$-time preprocessing. Thus, we can obtain an

$$O(m \log n + n\sqrt{k \log n} \log n)\text{-time}$$

algorithm for this problem. A more efficient algorithm that achieves

$$O(m + n(\sqrt{k \log n} + \log n) \log \log n)$$

time by using a somewhat sophisticated data structure is given in [3].

Application IV. Finding a length-limited Huffman code. A length-limited Huffman code can be represented by a height-limited Huffman tree, defined as follows. Consider a binary tree storing n data $\{z_1, z_2, \ldots, z_n\}$ at leaves. The probability p_i that the data z_i is queried is known for each i. The Huffman tree is the tree with best average query time. (See Fig. 7.) The height-limited Huffman tree is the tree with best average query time under the condition that the height is no more than a given parameter k. Larmore and Przytycka [15] showed that the Huffman-tree

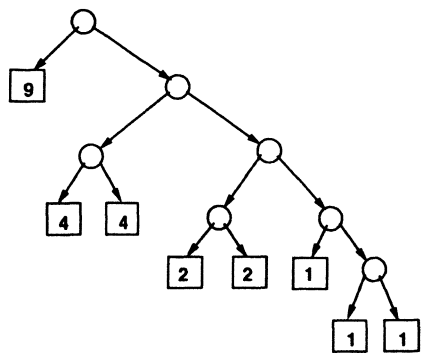

Fig. 7. Huffman tree. (The numbers in the leaves are proportional to the query probability.)

problem is reduced to the Least Weight Subsequence problem in a concave Monge array. It is not difficult to see that the length-limited problem is reduced to a minimum k-link path in a graph whose weights are given by the matrix defined in [15]. Thus, our algorithm can be applied to achieve the desired time bound.

Application V. Computing discrete quantization. First, we recall the definition of a discrete quantization. Let $f: \{x_1, x_2, \ldots, x_n\} \to \mathcal{R}$ be a real-valued function, where $\mathcal{R}$ is the set of the real numbers and $x_1 \le x_2 \le \cdots \le x_n$ are real numbers. Fix k, and consider a sorted set of real numbers $Z = \{z_1, z_2, \ldots, z_k\}$ and a mapping $\psi: \{1, 2, \ldots, n\} \to \{1, 2, \ldots, k\}$. The pair (Z, ψ) is called a quantization, and the sum $\sum_{i=1}^{n} f(x_i)(x_i - z_{\psi(i)})^2$ is called the error of the quantization. Optimal quantization is the one which minimizes the error.

We show how to reduce this problem to the problem of finding a minimum k-link path in a DAG with $n + 1$ nodes. For an interval I, define the weighted mean $\mu(I) = \sum_{s \in I} f(x_s)x_s / \sum_{s \in I} f(x_s)$. Wu [19] showed that the mapping ψ in the optimal quantization is a nondecreasing function, and that $z_j = \mu(\psi^{-1}(j))$, for $j = 1, \ldots, k$. Let $w(I) = \sum_{s \in I} f(x_s)(x_s - \mu(I))^2$. Then the error function coincides with $\sum_{j=1}^{k} w(\psi^{-1}(j))$. Hence, the function ψ represents the minimum k-link path in a DAG with nodes $\{0, 1, \ldots, n\}$, where the edge weight of (i, j) is $w((i, j])$. It is easy to see that this graph satisfies the concave Monge property, and that the values $\mu(I)$ and $w(I)$ can be computed in constant time after precomputing the prefix sums of x_i, $f(x_i)$, $f(x_i)x_i$, and $f(x_i)x_i^2$. Hence, we obtain an $O(n\sqrt{k \log n} + n \log n)$-time algorithm. This improves the result of Wu [19] by an $O(\sqrt{k/\log n})$ factor.

5. Concluding Remarks

It is still open whether it is possible to design an $O(n \log^c n)$-time algorithm (for some constant c) for computing a minimum k-link path in DAGs with the concave Monge property (in case $k = \Omega(\log^{2c-1} n)$). We are able to compute the minimum k-link path in only $O(n\sqrt{k \log n})$ time. One way to achieve such an algorithm is by designing a better parallel algorithm that can be used as a guide algorithm in the parametric search. Note that an n-processor polylogarithmic-time algorithm for computing a minimum-weight path in concave Monge DAGs would yield an $O(n \log^c n)$-time algorithm for computing a minimum k-link path in these DAGs. It is also worth noting that such a parallel algorithm would also yield an n-processor, polylogarithmic-time algorithm for computing a Huffman tree on n vertices without any height restriction using the techniques of Larmore and Przytycka [15].

Acknowledgments

The authors thank James K. Park for communicating the application to quantization. Also, they thank anonymous referees for many valuable comments; especially,

one of the referees who pointed out a serious gap in the initial version of the paper.

References

1. A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, Geometric Applications of a Matrix-Searching Algorithm, *Algorithmica* **2** (1987), 195–208.
2. A. Aggarwal and J. Park, Notes on Searching in Multidimensional Monotone Arrays, *Proc. 29th IEEE Symp. on Foundations on Computer Science*, 1988, pp. 497–512.
3. A. Aggarwal and T. Tokuyama, Consecutive Interval Query and Dynamic Programming on Intervals, *Proc. 4th Internat. Symp. on Algorithms and Computing*, 1993, pp. 466–475. Lecture Notes in Computer Science, Vol. 762. Springer-Verlag, Berlin.
4. T. Asano, Dynamic Programming on Intervals, *Proc. 2nd Internat. Symp. on Algorithms*, 1991, pp. 199–207. Lecture Notes in Computer Science, Vol. 557. Springer-Verlag, Berlin.
5. W. Bein, L. Larmore, and J. Park, The d-Edge Shortest-Path Problem for a Monge Graph, Preprint, 1992.
6. J. Boyce, D. Dobkin, R. Drysdale, and L. Guibas, Finding Extremal Polygons, *SIAM J. Comput.* **14** (1985), 134–147.
7. K. Chan and T. Lam, Finding Least-Weight Subsequences with Fewer Processors, *Proc. SIGAL Internat. Symp. on Algorithms*, 1990, pp. 318–327. Lecture Notes in Computer Science, Vol. 450. Springer-Verlag, Berlin.
8. B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, Diameter, Width, Closest Line Pair, and Parametric Searching, *Proc. 8th ACM Symp. on Computational Geometry*, 1992, pp. 120–129.
9. R. Cole, Slowing Down Sorting Networks to Obtain Faster Sorting Algorithms, *J. Assoc. Comput. Mach.* **34** (1987), 200–208.
10. G. Frederickson, Optimal Algorithms for Tree Partitioning, *Proc. 2nd ACM–SIAM Symp. on Discrete Algorithms*, 1991, pp. 168–177.
11. M. Klawe, A Simple Linear-Time Algorithm for Concave One-Dimensional Dynamic Programming, Technical Report 89-16, University of British Columbia, Vancouver, 1989.
12. M. Klawe and D. Kleitman, An Almost Linear-Time Algorithm for Generalized Matrix Searching, Technical Report RJ6275, IBM Almaden Research Center, 1988.
13. C. P. Kruskal, Searching, Merging and Sorting in Parallel Computation, *IEEE Trans. Comput.* **32** (1983), 942–946.
14. L. Larmore and D. Hirschberg, Length-Limited Coding, *Proc. 1st ACM–SIAM Symp. on Discrete Algorithms*, 1990, pp. 310–318.
15. L. Larmore and T. Przytycka, Parallel Construction of Trees with Optimal Weighted Path Length, *Proc. 3rd ACM Symp. on Parallel Algorithms and Architectures*, 1991, pp. 71–80.
16. L. Larmore and B. Schieber, On-Line Dynamic Programming with Applications to the Prediction of RNA Secondary Structure, *J. Algorithms* **12** (1991), 490–515.
17. N. Megiddo, Applying Parallel Computation Algorithms in the Design of Serial Algorithms, *J. Assoc. Comput. Mach.* **30** (1983), 852–865.
18. R. Wilber, The Concave Least Weight Subsequence Problem Revisited, *J. Algorithms* **9** (1988), 418–425.
19. X. Wu, Optimal Quantization by Matrix Searching, *J. Algorithms* **12** (1991), 663–673.

Received March 23, 1993, and in revised form November 30, 1993.