アルゴリズム 37－1
（1994．1．25）

Kリンク最短パス問題と行列探索

Alok Aggarwal, Baruch Schieber
IBM T. J. Watson Research Center
徳山 豪
日本ＩＢＭ東京基礎研究所

凹モンジュ性を持つ有向完全グラフのｋ個の辺を持つパスのうちで最短なものを求める効率の良いアルゴリズムを提案する。計算時間は $O(n\sqrt{k\log n}+n\log n)$ である。モンジュ性を持つ有向完全グラフの性質とともにパラメトリック探索が用いられる。応用として、（１）凸多角形に内接する最大のｋ角形（２）凸多角形に外接する最小のｋ角形（３）区間グラフの最大ｋクリーク（４）長さ限定最適コード（５）最適量子化の計算の高速化が得られる。

Finding a minimum-weight k-link path in graphs with Monge property and applications

Alok Aggarwal, Baruch Schieber
IBM T. J. Watson Research Center,
P.O. Box 218, Yorktown Heights, NY10598
Takeshi Tokuyama
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi, Kanagawa, 242 Japan. Email:ttoku@vnet.ibm.com

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 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 Figure 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.

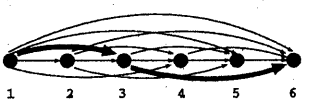

Figure 1: 2-link path of a complete DAG

A weighted DAG, G, satisfies the 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 $k = \Omega(\log n)$. From now on, we assume that this is the case.

We solve the problem using Megiddo's parametric search technique [15, 8] – a powerful technique for designing algorithms, especially in computational geometry [7]. 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, t_P the number of parallel steps, and p the number of processors in the guide algorithm. Applying parametric search, the resulting algorithm has 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 be sometimes sped up by refining the parallel guide algorithm [8, 9]; in fact, Frederickson [9] 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.

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 is Monge, then so is $G(\tau)$. 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 $1 \leq k \leq n - 1$, there exists a real number τ 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 not 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, $\ell(\tau - \xi) = W_\tau(Q) - W_\xi(Q) \geq W_\tau(P) - W_\xi(P) = k(\tau - \xi)$. Since $\tau - \xi > 0$, we have that $\ell \geq k$. □

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 there exists a link (i_1, j_1) of P_1 and a link (i_2, j_2) of P_2 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.e., $i_1 < i_2 < j_2 < j_1$, we have $W(Q_1) + W(Q_2) = W(P_1) + W(P_2) - w(i_1, j_1) - w(i_2, j_2) + w(i_1, j_2) + w(i_2, j_1) \leq W(P_1) + W(P_2)$. 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: Omitted in this version. □

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: Omitted in this version. □

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

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

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 $k < \ell \leq n$. Applying Lemma 3 it is easy to verify that for all $\tau \geq (P(k) - P(\ell))/(\ell - k)$, a minimum-weight k-link path in $G(\tau)$ is no larger than a minimum-weight ℓ-link path. By Corollary 7 $P(k) - P(\ell) \leq (\ell - k)(P(k) - P(k+1))$. 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 $1 \leq \ell < k$, 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 path 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 [17] 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, also the differences between weights are integral. Recall that any $P(k) - P(k+1) \leq \tau \leq P(k-1) - P(k)$ can be taken as τ_{opt}. Hence, there exists an integral τ_{opt} in the range $[-3U, 3U]$. 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]. □

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 [15]. 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 [15] 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 [13]. 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, 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 in which we perform 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 details 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 t-th stage, for $t = 0, \ldots, n/\ell - 1$, 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 $t\ell < j \leq (t+1)\ell$. 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 $v_{t\ell+1}, \ldots, v_{(t+1)\ell}$ 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 $1 \leq j \leq t\ell$ 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 $j > t\ell$, the minimum-weight path in $G(\tau_{opt})$ from v_1 to v_j anchored in $[1, t\ell]$ has either $K(t\ell)$ or $K(t\ell)+1$ links.*

Proof: Omitted in this version. □

For $t\ell < j \leq (t+1)\ell$, 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 $[1, t\ell]$ in $G(\tau_{opt})$, and their suffix consists of some (possibly zero) links in $[t\ell+1, (t+1)\ell]$. 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 $t\ell < j \leq (t+1)\ell$, compute the minimum-weight path in $G(\tau_{opt})$ from v_1 to v_j anchored in $[1, t\ell]$.

Step 2: Find an integer m satisfying $K(t+1)\ell) - K(t\ell) \leq m \leq 2(K((t+1)\ell) - K(t\ell))$.

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

Step 4: For each $t\ell < j \leq (t+1)\ell$, find the minimum-weight path in $G(\tau_{opt})$ among the candidate h-link paths from v_1 to v_j found in Step 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 $[1, t\ell]$ has either $K(t\ell)$ or $K(t\ell)+1$ links. For each $t\ell < j \leq (t+1)\ell$, we first find a minimum-weight path from v_1 to v_j anchored in $[1, t\ell]$ with $K(t\ell)$ links. Then, we find such a path with $K(t\ell)+1$ links. This computation is independent of τ since we compare between paths with the same number of links. Finally, we compare the two paths applying parametric search.

We show how to find the minimum-weight paths with $K(t\ell)$ links. Recall that we already computed the minimum-weight paths in $G(\tau)$ from v_1 to all vertices in $[1, t\ell]$. Since we are interested in minimum-weight paths anchored in $[1, t\ell]$ with $K(t\ell)$ links, we may consider only the vertices v_j in $[1, t\ell]$, such that the minimum-weight path from v_1 to v_j has $K(t\ell)-1$ links. Suppose that there are n' such vertices. Consider the $\ell \times n'$ matrix in which the (j,i)-th entry is the length of the minimum-weight path from v_1 to the i-th 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 Monge Property; and (ii) the minimum entry in row j corresponds to the minimum-weight path from v_1 to v_j with $K(t\ell)$ links. Hence, all the minimum-weight paths can be found in $O(n)$ time applying the matrix search algorithm of [1]. The minimum-weight paths with $K(t\ell)+1$ links are found similarly.

Now, for each $t\ell < j \leq (t+1)\ell$ we have the minimum-weight paths with $K(t\ell)$ links and $K(t\ell)+1$ links. To compare them we apply 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 $K(t\ell)$ links is of smaller weight in $G(\tau)$, and for all $\tau < \xi$ the path with $K(t\ell)+1$ 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 $\tau_{opt} > \xi$, 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 $\tau_{opt} < \xi$, 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 $O(\ell)$ processors in $O(1)$ time if the parameter τ is given. Hence, the associated parametric search

algorithm runs in $O(n \log \ell)$ 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 $v_{t\ell}, v_{t\ell+1}, \ldots, v_{(i+1)\ell}$. Let $\vec{x}$ be the vector of weights of the minimum-weight paths from v_1 to $v_{t\ell}, v_{t\ell+1}, \ldots, v_{(i+1)\ell}$ anchored in $[1, t\ell]$ in $G(\tau)$, for any $\tau \in (\tau_L, \tau_R)$. This vector is computed in previous stages. Let $\vec{x}_1$ be the vector whose j-th entry is the weight of the minimum-weight path from v_1 to v_j that contains $K(t\ell)$ links, if such a path exists, and infinity otherwise. Let $\vec{x}_2$ be the vector whose j-th entry is the weight of the minimum-weight path from v_1 to v_j that contains $K(t\ell)+1$ links, if such a path exists, and infinity otherwise. From Lemma 11 it follows that the noninfinity entries of $\vec{x}_1$ and $\vec{x}_2$ are contiguous. From Lemma 12 it follows that these paths have either $K(t\ell)$ or $K(t\ell)+1$ links. Thus, the vector $\vec{x}$ is the entrywise minimum of $\vec{x}_1$ and $\vec{x}_2$.

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

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

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 $h = K(t\ell), K(t\ell)+1, \ldots, K(t\ell)+m$ in $2m$ multiplications of an $\ell \times \ell$ 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 $O(\ell)$ 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 $K(t\ell) \le h \le K((i+1)\ell)$ and $t\ell < j \le (i+1)\ell$, and find the minimum-weight path among them at $\tau = \tau_{opt}$. 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 $t\ell < j \le (i+1)L$, let $W_h(j)$ be the weight of the candidate h-link path from v_1 to v_j. Then, $W_h(j) \le \max\{W_{h+1}(j), W_{h-1}(j)\}$.*

Proof: Omitted in this version. □

Fix some $t\ell < j \le (t+1)\ell$. 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 [10] showed that this search can be done in constant time using m processors. Thus, to find the minimum for all $t\ell < j \le (t+1)\ell$ in constant time we need $m\ell$ processors. It follows that without the knowledge of τ_{opt} this can be done using the parametric search paradigm in $O(m\ell + n\log(m\ell))$ 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 $[(K((i+1)\ell) - K(t\ell)), 2(K((i+1)\ell) - K(t\ell))]$ by a variant of binary search. Lemma 14 implies that the minimum-weight path from v_1 to $v_{(t+1)\ell}$ in $G(\tau_{opt})$ is given by the smallest h that locally minimizes the weight of the candidate h-link path to $v_{(t+1)\ell}$. Thus, it is easy to verify whether the path to $v_{(t+1)\ell}$ 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 the 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 roble 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.*

Proof: We concentrate at last iteration of Step 2. The total amount of time required for Step 1 in the n/ℓ stages is $O(n^2/\ell \cdot \log \ell)$. Summing over all stages Step 3 requires $O\left(\sum_{i=1}^{n/\ell}(K((t+1)\ell) - K(t\ell)) \cdot \ell\right) = O(k\ell)$. Step 4 requires $O\left(\sum_{i=1}^{n/\ell}(K((t+1)\ell) - K(t\ell)) \cdot \ell + n\log n\right) =$

$O(k\ell + n^2/\ell \cdot \log n)$. We conclude that the algorithm requires $O(n^2/\ell \cdot \log n + k\ell)$. Setting $\ell = \min(n\sqrt{\log n/k}, n)$ implies the theorem. □

4 Applications

The minimum k-link path in a concave Monge DAG has several applications. Given below are five such applications to geometric path finding (App. I and II), interval graphs (App. III), data optimization (App. IV), and data compression (App. 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 Figure 2.) For this problem Boyce, Dobkin, Drysdale and Guibas [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. Boyce *et al.* [6] showed that if the maximal area k-gon containing a fixed vertex is given, then the maximal area k-gon is 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 path in convex DAGs is equivalent to finding the minimum 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. 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 Figure 2.) This problem is the dual of the previous problem, and is thus solved in the same time complexity.

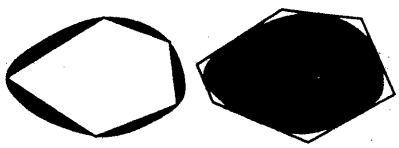

Figure 2: Max-area inscribing polygon and Min-area inscribed polygon

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 Figure 3.) Let S be 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 $O(m \log n + n\sqrt{k \log n} \log n)$ time algorithm for this problem. See [3] for a way to reduce the complexity to $O(m + n(\sqrt{k \log n} + \log n) \log \log n)$ by using a somewhat sophisticated data structure.

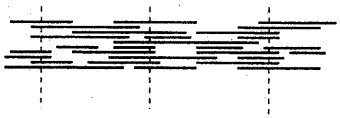

Figure 3: k maximum weight cliques of interval graph

Application IV. Given a weighted alphabet of size n, we want to find an optimal prefix-free binary code (length-limited Huffman Code) for the alphabet with the restriction that no code string be longer than length k. Larmore and Hirschberg [12] gave an $O(nk)$ time algorithm for this problem. 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. 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 [13] showed that the Huffman tree 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 [13]. Thus, our algorithm can be applied to solve the problem in

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

Application V. Let $f : \{x_1, x_2, \ldots, x_n\} \rightarrow \mathcal{R}$ be a real valued function, where $\mathcal{R}$ 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 $\psi : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, k\}$. 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 Figure 4. Wu [16] showed that computing optimal quantization can be reduced to finding a minimum-weight k-link path as follows: 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 [16] showed that the mapping ψ in the optimal quantization is a non-decreasing 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 $O(n\sqrt{k \log n} + n \log n)$ time algorithm. This improves the result of Wu [16] by an $O(\sqrt{k / \log n})$ factor.

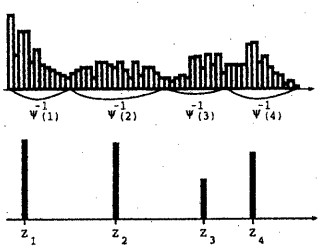

Figure 4: Quantization (k=4)

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), 497-512.

[3] A. Aggarwal and T. Tokuyama, Consecutive Interval Query and Dynamic Programming on Intervals, to appear in *Proc. 4th Int'l Symp. on Algorithms and Computing* (1993).

[4] T. Asano, Dynamic Programming on Intervals, *Proc. 2nd Int'l. Symp. on Algorithms, Lect. notes in Comput. Sci.* **557**, Springer–Verlag (1991), 199-207.

[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. on Computing* **14** (1985), 134-147.

[7] 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), 120-129.

[8] R. Cole, Slowing Down Sorting Networks to Obtain Faster Sorting Algorithms, *J. ACM* **34** (1987), 200-208.

[9] G. Frederickson, Optimal Algorithms for Tree Partitioning, *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms* (1991), 168-177.

[10] C.P. Kruskal, Searching, Merging and Sorting in Parallel Computation, *IEEE Trans. Computers* **C-32** (1983), 942-946.

[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] L. Larmore and D. Hirschberg, Length-Limited Coding, *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms* (1990), 310-318.

[13] L. Larmore and T. Przytycka, Parallel Construction of Trees with Optimal Weighted Path Length, *Proc. 3rd ACM Symp. on Parallel Algorithms and Architectures* (1991), 71-80.

[14] L. Larmore and B. Schieber, On-line Dynamic Programming with Applications to the Prediction of R-NA Secondary Structure, *J. of Algorithms* **12** (1991), 490-515.

[15] N. Megiddo, Applying Parallel Computation Algorithms in the Design of Serial Algorithms, *J. ACM* **30** (1983), 852-865.

[16] X. Wu, Optimal Quantization by Matrix Searching, *J. of Algorithms* **12** (1991), 663-673.

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