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

(Extended Summary)

ALOK AGGARWAL* BARUCH SCHIEBER* TAKESHI TOKUYAMA†*

Abstract

Let G be a weighted, complete, directed acyclic graph (DAG), whose edge weights obey the 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 $O(n\sqrt{K \log n})$ for the concave case and $O(n\alpha(n)\log^3 n)$ for the convex case. Our algorithm uses some properties of DAGs with Monge property together with a refined parametric search technique. We apply our algorithm (for the concave case) to get efficient solutions for the following problems, improving on previous results: (1) Finding the largest K-gon contained in a given polygon. (2) Finding the smallest K-gon that is the intersection of K halfplanes out of of given set of halfplanes defining an 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 $1 \leq i < j$, let $w(i, j)$ denote the weight associated with the arc (i, j). (See Figure 1.)

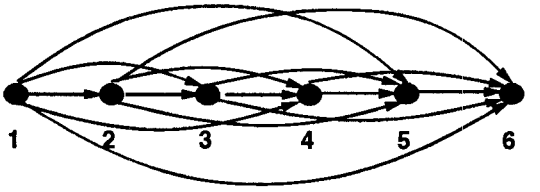

Figure 1: Complete DAG

We call a path in G a *K-link* path if the path contains exactly K arcs. For any two vertices, i and j, we call a path P a *minimum* K-link path if the path goes from i to j, contains exactly K links and among all such paths, it has the minimum weight. A weighted DAG, G, satisfies the concave Monge condition if $w(i, j) + w(i + 1, j + 1) \leq w(i, j+1)+w(i+1, j)$ holds for all $1 < i+1 < j < n$, and satisfies the convex Monge condition if the inequality is reversed.

In this paper, we are interested in computing the minimum K-link path from 1 to n in Monge DAGs, i.e., weighted DAGs whose weights satisfy either the concave Monge property or the convex 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, and in $O(nK\alpha(n))$ time for a convex Monge DAG, where $\alpha(.)$ is the inverse Ackermann's function.

*IBM – Research Division, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.

†IBM – Research Division, Tokyo Research Laboratory, 5-19 Sanban-cho, Chiyoda, Tokyo, 102, Japan.

9th Annual Comptutational Geometry,5/93/CA,USA

The main result of this paper is an $O(n\sqrt{K \log n})$ time algorithm for computing the minimum K-link path for the concave case and an $O(n\alpha(n)\log^3 n)$ time algorithm for the convex case.

We solve the convex case using Megiddo's parametric search [16] based on the efficient parallel algorithm of Chan and Lam [9]. For the concave case, the known parallel algorithms required for parametric search are not efficient enough; therefore, we use a *refined parametric search* technique instead. Parametric search is a powerful technique for designing algorithms, especially in computational geometry [7]; hence its refinement should also be a useful tool.

The original parametric search runs a generic parallel algorithm (on P processors) without knowing the key parameter τ, and calls a *decision algorithm* $\log P$ times at each stage in order to compute the comparisons. Frederickson [10] pointed out that it is sometimes inefficient to use a parallel algorithm for parametric search; in fact, the essential requirement for parametric search is to give a "nice" partial order of computation.

We use a similar philosophy, although the technique is different. In [10], "nice" partial orderings, as well as sorted matrix search techniques, speed up binary search, and consequently decrease the number of calls to the decision algorithm. In this paper, we construct a suitable partial order, so that only parameter-dependent comparisons are done in parallel. (This can be considered as an analogue of Valiant's comparison model [17] specialized for parametric search.) As a result, we can design a parametric search algorithm with a small number of calls to the decision algorithm as well as small total work.

The minimum K-link path in concave Monge graph has several applications. Given below are five such applications to geometric path finding (App. I and II), interval graph (App. III), data optimization (App. IV), and data compression (App. V):

Application I. Suppose that we are given a convex n-gon, and we want to compute the maximum area K-gon and the maximum perimeter K-gon that are contained in the given n-gon. (See Figure 2.) For this problem Boyce, Dobkin, Drysdale and Guibas [6] provided an $O(nK \log n)$ algorithm that was later improved by Aggarwal *et al.* [1] to $O(nK + n\log n)$. 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.

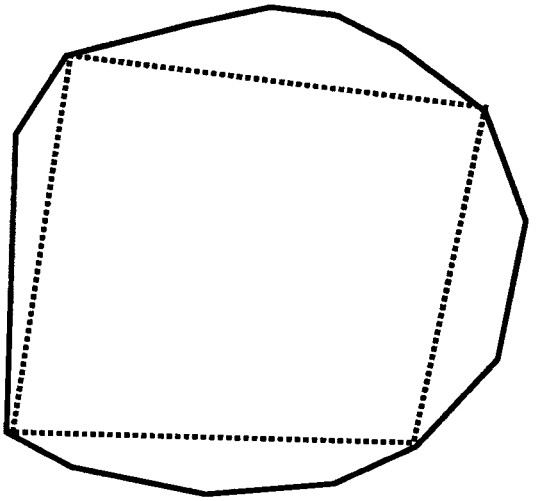

Figure 2: Max-area inscribing polygon

Application II. Suppose that we are given a convex n-gon and we want to compute the minimum area K-gon which is the intersection of K halfplanes out of a given set of halfplanes defining the n-gon. In other words, the minimum area circumscribing polygon touching edge-to-edge. (See Figure 3.) This problem is the dual of the previous problem, and thus is solved in the same time complexity.

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 4.) We give an $O(m+n\sqrt{K \log n}\log\log n)$ time algorithm, improving on a previous result of [4].

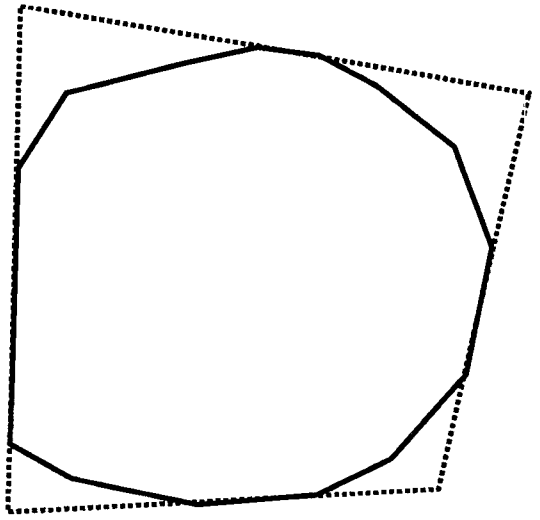

Figure 3: Min-area circumscribing polygon with edge-to-edge contact

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 [13] gave an $O(nK)$ time algorithm for this problem. Using the reduction of this problem to the min-weight K-link path problem [14], we solve it in $O(n\sqrt{K \log n})$ time.

Application V. Suppose that we are given a real valued function $f : \{x_1, x_2, \ldots, x_n\} \rightarrow \mathcal{R}$, where $\mathcal{R}$ are the reals and $x_1 \leq x_2 \leq \cdots \leq x_n$ are reals. Consider a sorted set of real numbers $Y = \{y_1, y_2, \ldots, y_K\}$ and a mapping $\psi : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, K\}$. We call the pair (Y, ψ) a quantization, and the sum $\sum_{i=1}^{n} f(x_i)(x_i - y_{\psi(i)})^2$ the error of the quantization. Optimal quantization is the one which minimizes the error. The continuous version of this problem is illustrated in Figure 5. Wu [18] showed that optimal quantization finding can be reduced to min-weight K-link path problem; hence it can be solved in $O(n\sqrt{K \log n})$ time applying our algorithm.

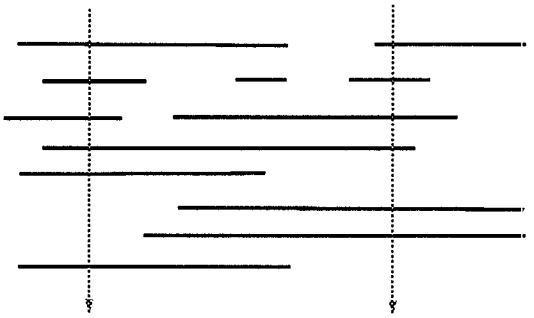

Figure 4: K-max clique of interval graph (K=2)

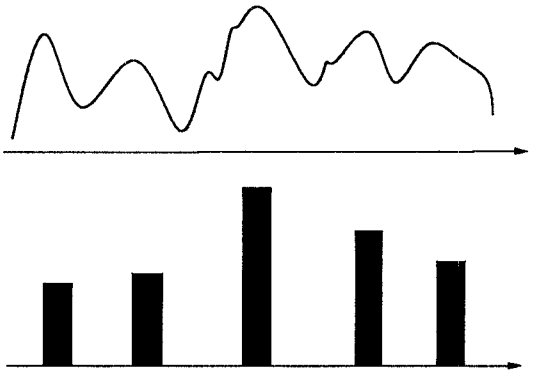

Figure 5: Quantization (K=5)

The remaining abstract is organized as follows. Section 2 describes the simple parametric search algorithms, and section 3 describes the refined parametric search algorithm and analyzes its time complexity. Section 4 briefly describes the applications.

2. The simple parametric search

Let G be our weighted DAG. For a real number τ, define the weighted DAG, $G(\tau)$, 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 $w(e) + \tau$ ($w(e)$ is the weight of e in G). Note that if G is Monge so is $G(\tau)$. Define a *diameter* path in G to be a path from 1 to n.

Lemma 2.1: *If for some τ the minimum weight diameter path in $G(\tau)$ has K links, then this path is the minimum weight K-link diameter path in G.*

Thus, in order to find the minimum weight K-link diameter path of G, it suffices to find a τ such that the minimum weight diameter path in $G(\tau)$ has K links. We denote this τ by τ_{opt}.

Lemma 2.2: *Suppose the minimum weight diameter path in $G(\tau)$ has K links. Then for every $\beta < \tau$, the minimum weight diameter path in $G(\beta)$ has at least K links.*

These two lemmas assure that we can solve the problem using binary search. Since the shortest path in a concave DAG (resp. convex DAG) is solved in $O(n)$ (resp. $O(n\alpha(n))$ time using [11, 12], we have a weakly polynomial $O(n \log(n+U))$ (resp. $O(n\alpha(n) \log(n+U))$ time algorithm for concave (resp. convex) minimum weight K-link path problem, where U is the maximum weight. (Bein *et al.* [5] discovered the above weakly polynomial algorithms independently.)

We make the above algorithms strongly polynomial by using the parametric search paradigm [16]. Assume that there is a parallel algorithm which computes the minimum weight path in $G(\tau)$ in $O(T_{par})$ time using P processors. Also assume that there is a sequential algorithm (*decision algorithm*) that computes it in T_{seq} time. Then, the parametric search scheme finds the minimum weight path among the paths with K links in G in $O(PT_{par} + T_{seq}T_{par} \log P)$ time.

For convex case, $T_{seq} = O(n\alpha(n))$ [12], and $T_{par} = O(\log^2 n)$ and $P = O(n)$ [9]. Thus, we obtain the following:

Theorem 2.3: *The minimum weight K-link diameter path can be computed in $O(n\alpha(n) \log^3 n)$ time for the convex case.*

For the concave case, it is known that $T_{seq} = O(n)$ [11, 15]. But, no polylogarithmic time parallel algorithm that uses $O(n)$ processors is known. 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 [14]. Thus, we have the following:

Theorem 2.4: *The minimum weight K-diameter path problem can be solved in $O(n\sqrt{n} \log^2 n)$ time for the concave case.*

The above time complexity for concave case is far from satisfactory, since for $n > K^2$, it is worse than the $O(nK)$ time algorithm given by using [1, 2]. In the next section we give a better algorithm by refining the parametric search technique.

3. The refined parametric search

We use a refined parametric search that uses a "partially" parallel algorithm instead of a "fully" parallel algorithm for the parametric search. Our aim is to find τ_{opt} for which the minimum weight path of $G(\tau_{opt})$ contains exactly K links. We design a generic algorithm for computing the minimum weight path of $G(\tau)$, where the comparisons depending on τ are performed in parallel, while the comparisons independent of τ are not. Like the usual parametric search, when the algorithm makes $O(P)$ pairwise independent comparisons depending on τ, we compute the critical values of such comparisons, and locate τ_{opt} in the sorted list of these values by calling the decision algorithm (the sequential shortest path algorithm) $O(\log P)$ times.

Without loss of generality we assume that the number of links in all minimum weight paths of $G(\tau)$ is the same. (If this is not the case we define the minimum weight paths to be the ones with the minimum number of links.) Let $k_\tau(i)$ be the number of links in the minimum weight path from v_1 to v_i in $G(\tau)$, and let $k(i)$ be the number of links

in the minimum weight path from v_1 to v_i in $G(\tau_{opt})$ (i.e., $k(i) = k_{\tau_{opt}}(i)$).

The following lemma is easily obtained from *interleaving property* [6] of paths in graphs with Monge property:

Lemma 3.1: *For all $\tau > 0$, and all $1 \leq i < j \leq n$, $k_\tau(i) \leq k_\tau(j)$.*

For a path P from v_1 to v_j (assuming that v_1 is to the left of v_j), the left endpoint of the last edge is called the *anchor* of P. If the anchor of a path is in an interval I, we say that the path has its anchor in I. We fix a number L, which will be set to be a suitable number in the analysis. For convenience we assume that both L and n/L are integers. For $1 \leq i \leq n/L$, define β_i and γ_i such that for any $\beta_i \leq \tau \leq \gamma_i$, the minimum weight path from v_1 to each vertex whose anchor is in $[1, iL]$ in $G(\tau)$ is the same as that in $G(\tau_{opt})$.

Our algorithm has n/L stages. Below, we describe stage $i+1$ in which we compute $\beta_{i+1}, \gamma_{i+1}$ and all minimum weight paths anchored in $[1, (i+1)L]$ in $G(\tau)$, for any $\beta_{i+1} \leq \tau \leq \gamma_{i+1}$, given β_i, γ_i and the weights (as linear functions of the parameter τ) of all minimum weight paths anchored in $[1, iL]$ in $G(\tau)$, for any $\beta_i \leq \tau \leq \gamma_i$. (Initially, $\beta_0 = 0$ and $\gamma_0 = \infty$.)

We first find all shortest paths from v_1 to $v_{iL}, v_{iL+1}, \ldots, v_{(i+1)L}$ in $G(\tau_{opt})$. The following lemma follows from Lemma 3.1 and the Monge property.

Lemma 3.2: *For any $j > iL$, the minimum weight path from v_1 to v_j anchored in $[1, iL]$ has either $k(iL)$ or $k(iL)+1$ links.*

For $iL < s \leq (i+1)L$, define the *candidate h-link path* from v_1 to v_s to be the minimum weight h-link path among the paths whose prefix is a minimum weight path anchored in $[1, iL]$ in $G(\tau)$, for any $\beta_i \leq \tau \leq \gamma_i$, and their suffix consists of some (possibly zero) links in $[iL+1, (i+1)L]$. Note that the minimum weight h-link path from v_1 to v_s in $G = G(0)$ need not be a candidate h-link path. However, it is easy to see the following lemma:

Lemma 3.3: *The minimum weight path from v_1 to v_s in $G(\tau_{opt})$ is one of the candidate h-link paths.*

The following is the key fact:

Proposition 3.1: *For a fixed h, the comparison between the weights of two h-paths is done independently of τ.*

Let A be the matrix of the edge weights between the vertices $v_{iL}, v_{iL+1}, \ldots, v_{(i+1)L}$. Let $\vec{x}$ be the vector of weights of the minimum weight paths from v_1 to $v_{iL}, v_{iL+1}, \ldots, v_{(i+1)L}$ anchored in $[1, iL]$ in $G(\tau)$, for any $\beta_i \leq \tau \leq \gamma_i$. This vector is computed in previous stages. From Lemma 3.2 it follows that these paths have either $k(iL)$ or $k(iL)+1$ links. Thus, the vector $\vec{x}$ is the entrywise minimum of $\vec{x}_1$ and $\vec{x}_2$, where the j-th entry of $\vec{x}_1$ (resp. $\vec{x}_2$) is the weight of the minimum weight path from v_1 to v_j that contains $k(iL)$ (resp. $k(iL)+1$) links if such exists, and ∞, otherwise. From Lemma 3.1 it follows that the noninfinity entries of $\vec{x}_1$ and $\vec{x}_2$ are contiguous.

Consider the semiring defined over the reals with the operation $\{\min, +\}$. For an $L \times L$ matrix P and an L vector $\vec{z}$, $\vec{w} = P\vec{z}$ is given by $w_s = \min_{i=1,2,\ldots,L}\{P_{s,i} + z_i\}$. The following lemma is obvious from the definition:

Lemma 3.4: *For a given h, all candidate h-link paths from v_1 to $v_{iL}, \ldots, v_{(i+1)L}$ are obtained by computing*

$$\min\{A^{h-k(iL)}\vec{x}_1, A^{h-k(iL)-1}\vec{x}_2\}.$$

The above operation is done independently of τ by Lemma 3.1. Actually, each term compared by using the min operation has a form $h\tau + w$ for an w independent of τ. Hence, for any given M, we can find all candidate h-link paths for $h = k(iL), k(iL)+1, \ldots, k(iL)+M$ 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.)

Suppose that we did this computation for some $M > k((i+1)L) - k(iL)$. Then, to find the shortest path from v_1 to v_j, for some $iL < j \leq (i+1)L$, in $G(\tau_{opt})$, we must compare the candidate h-link paths from v_1 to v_j, for $k(iL) \leq h \leq k((i+1)L)$, and

find the minimum path among them at $\tau = \tau_{opt}$. These comparisons depend on the parameter τ. Here, we apply the parametric search paradigm. When two paths are compared, we compute the critical value ξ of τ for the comparison. If this value is not in the interval $[\beta_i, \gamma_i]$, then the comparison is independent of τ, otherwise, we execute the sequential (linear time) shortest path algorithm for $G(\xi)$. If the shortest path has exactly K links, we can report ξ as our answer. If it contains less (resp. more) than K links, then $\tau_{opt} < \xi$ (resp. $\tau_{opt} > \xi$). Thus, we can replace one of β_i and γ_i by ξ, and continue the process.

The comparison of these M elements can be done in $O(\log M)$ time with $O(M)$ processors. Since we have to do it for all $iL < j \leq (i+1)L$, there are L such comparisons in parallel, and the number of processors becomes $O(LM)$. Hence, if we use the original Megiddo's parametric search, we call the sequential algorithm $O(\log M \log(ML))$ times. However, we can apply Cole's improved method [8], and the number of calls to the sequential algorithm is reduced to $O(\log(ML))$, and time complexity becomes $O(n\log(ML) + ML\log M)$.

Moreover, the problem can be considered as the matrix search in $M \times L$ matrix. The row indices of the location of minimum in the columns are non-decreasing sequence because of Lemma 3.1. Hence, we can reduce the time complexity to $O(n\log(ML) + \sqrt{L}M\log M)$ by computing the column minima of columns whose column indices are integer multiples of $\sqrt{L}$ in advance.

The only problem left is how to determine M such that $M > k((i+1)L) - k(iL)$. We would like to set M as small as possible in order to reduce the number of computations of the matrix multiplications. Below, we show how to find an M that is bounded from above by $2(k((i+1)L) - k(iL))$.

For $1 < h < n$, and for $iL < j \leq (i+1)L$, let $W_h(j)$ be the weight of the candidate h-link path from v_1 to v_j. We use the following lemma.

Lemma 3.5: *For $1 < h < n$, and for $iL < j \leq (i+1)L$,*

$$W_h(j) \leq \max\{W_{h+1}(j), W_{h-1}(j)\}.$$

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. Then, there is a link (v_x, v_y) in P and a link (v_s, v_t) in Q such that $x < s < t \leq y$, and the number of links in the subpath of P from v_1 to v_y is the same as in the subpath of Q from v_1 to v_s. We have four cases.

CASE 1: $t = y$. We cut P at v_y to obtain two pieces P_{left} and P_{right}. Similarly, we cut Q at $v_t = v_y$ to obtain two pieces Q_{left} and Q_{right}. We construct a path R_1 by "gluing" P_{left} and Q_{right}, and a path R_2 by "gluing" Q_{left} and P_{right}. Both R_1 and R_2 have h links. Since P and Q are candidate paths, so are R_1 and R_2. The sum of the weights of R_1 and R_2 is the same as the sum of the weights of P and Q. The lemma in this case follows.

CASE 2: $t < y$ and $x > iL$. From the Monge property, it is easy to see that $w(x,y) + w(s,t) \geq w(x,t) + w(s,y)$. We cut P at v_x and v_y to obtain three pieces P_{left}, (v_x, v_y), P_{right}. Similarly, we obtain Q_{left} and Q_{right} by cutting Q at v_s and v_t. We construct a path R_1 by "gluing" P_{left} and Q_{right} with the edge (v_x, v_t). Similarly, a path R_2 is obtained by "gluing" Q_{left} and P_{right} with (v_s, v_y). Both R_1 and R_2 have h links. Also, the prefixes of both R_1 and R_2 are minimum weight paths anchored in $[1, iL]$. Thus, both R_1 and R_2 can be considered as candidate h-link paths. The sum of the weights of R_1 and R_2 is not larger than the sum of the weights of P and Q. The lemma in this case follows.

CASE 3: $t < y$ and $s > iL \geq x$. Consider R_1 and R_2 as in the previous case. Since $s > iL$, the prefix of R_2 is a minimum weight path anchored in $[1, iL]$. If the weight of R_2 is less than or equal to the weight of P, then we are done. Suppose that the weight of R_2 is larger than that of P. Since the sum of the weights of R_1 and R_2 is not larger than the sum of the weights of P and Q, we have that the weight of R_1 is smaller than that of Q. However, R_1 may not be a candidate path since the prefix of R_1 from v_1 to v_t, denoted R'_1, is not necessarily the minimum weight path to v_t anchored in $[1, iL]$. Let R'_3 be a minimum weight path from v_1 to v_t anchored in $[1, iL]$. We claim that the number of links of R'_3 is the same as the number of links of R'_1. Observe that

the number of links of the minimum weight paths from v_1 to v_y and from v_1 to v_s is the same as this of R'_1. Since $s < t < y$, Lemma 3.1 implies that this is also the number of links of R'_3. Let R_3 be the path obtained by "gluing" R'_3 and Q_{right}. Clearly, the number of links of R_3 is h, and thus R_3 can be considered as a candidate h-link path. The lemma in this case follows.

CASE 4: $t < y$ and $s \leq iL$. Let P' be the prefix of P from v_1 to v_y and let Q' be the prefix of Q from v_1 to v_t. Since $s \leq iL$ both P' and Q' are anchored in $[1, iL]$. From the definition of a candidate path it follows that both P' and Q' are minimum weight paths. We construct a path R'_1 by "gluing" P_{left} and the edge (v_x, v_t). Similarly, a path R'_2 is obtained by "gluing" Q_{left} and (v_s, v_y). The sum of the weights of R'_1 and R'_2 is not larger than the sum of the weights of P' and Q'. Since P' and Q' are minimum weight paths this implies that R'_1 has the same weight as Q' and R'_2 has the same weight as P'. However, R'_1 and Q' (and also R'_2 and P') have different number of links. This contradicts our assumption that the number of links in the minimum weight paths to v_t (and v_y) is the same. □

We remark that since the above lemma is true for any i and L, we get the following as a corollary.

Corollary 3.6: *For $1 < h < n$, and for $1 \leq j \leq n$, let $\tilde{W}_h(j)$ be the minimum weight of an h-link path from v_1 to v_j. Then,*

$$\tilde{W}_h(j) \leq \max\{\tilde{W}_{h+1}(j).\tilde{W}_{h-1}(j)\}.$$

Lemma 3.5 implies that the weights of the candidate h-link paths from v_1 to $v(i+1)L$ have no local maxima with respect to the link number. This implies that any local minimum must be also a global minimum. Hence, in order to find the minimum candidate path from v_1 to $v_{(i+1)L}$, it suffices to find the h locally minimizing it. Thus, we can apply the doubling method for choosing a suitable M so that $M \leq 2(k((i+1)L) - k(iL))$. Initially, we set $M = 3$. If a locally minimum weight candidate path found by the algorithm, then it is the true minimum weight path. Otherwise, we set $M = 2M$ and repeat the process.

Now, we have the minimum weight paths from v_1 to v_j, for $j = 1, 2, \ldots, (i+1)L$ in $G(\tau_{opt})$ and an interval $J = (\beta_{i+1}, \gamma_{i+1})$ containing τ_{opt}. We further need all minimum weight paths anchored in $[1, (i+1)L]$. Since we have those anchored in $[1, iL]$, it suffices to find those anchored in $[iL+1, (i+1)L]$, and take the minimum of them for each terminal point. The latter operation is done in $O(n \log L)$ time applying parametric search based on parallel comparison as follows.

Since we have already computed the shortest paths from v_1 to each terminals in $[iL+1, (i+1)L]$ in $G(\tau_{opt})$, the anchored path finding can be considered as matrix search in an $n \times L$ Monge matrix. From Lemma 3.2, the minimum weight paths contain either $k((i+1)L)$ or $k((i+1)L)+1$ links. Hence, we only consider the paths such that the minimum weight path between its anchor and v_1 has $k((i+1)L) - 1$ or $k((i+1)L)$ links. For each of those two link numbers, the associated minimum weight paths are found in $O(n)$ time applying the matrix search, since the computation is independent of τ. Finally, we compare them applying parametric search. The comparison between the paths can be done with $O(L)$ processors and $O(1)$ parallel comparisons if the parameter τ is given. Hence, the associated parametric search algorithm needs, $O(n \log L)$ time. (Recall that the decision algorithm is the $O(n)$ shortest path algorithm.)

Theorem 3.7: *The minimum weight K-link diameter path of a DAG with concave Monge property is found in $O(n\sqrt{K \log n})$ time.*

Proof: Let $M_i = k(iL) - k((i+1)L)$. Recall that $k(n) = K$ from definition. The multiplication of an $L \times L$ Monge matrix by a vector is done in $O(L)$ time [1, 2]. Thus, the total time complexity consumed for the matrix multiplication is $O(\sum_{i=1}^{n/L} M_i L)$, which is $O(LK)$.

The parametric search part needs $O(n \log(M_i L) + n \log L + \sqrt{L} M_i \log M_i)$ time in each stage. Thus, $O(\frac{n^2}{L} \log(KL) + \sqrt{L} K \log K)$ time in total. Hence, the time complexity of the algorithm is $O(LK + \frac{n^2}{L} \log n)$. Setting $L = \frac{n\sqrt{\log n}}{\sqrt{K}}$, implies the theorem. □

4. Applications

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

Application I: Maximal area inscribed K-gon of a convex n-gon. 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 DAG is equivalent to finding the minimum path in concave DAG, we can apply our algorithm to achieve an $O(n\sqrt{K \log n} + n \log n)$ time algorithm for the problem.

Application II: Minimum area K-gon defined by K-halfplanes out of n-halfplanes defining an n-gon. This problem is the dual of the previous problem. Thus, we can apply our algorithm to achieve the desired time bound.

Application III: K maximum cliques in an interval graph. This problem was formulated in [4] as a minimum weight K-link path in a DAG on n nodes, where the weight $w(i,j)$ is the total weight of intervals contained in the (open) interval (i,j). It is easy to see that this 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 reducing the complexity to $O(m + n\sqrt{K \log n} \log \log n)$.

Application IV: Length limited Huffman Code. The length limited Huffman code is equivalent to the 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 Figure 6.) The height limited Huffman tree is the tree with best average query time under condition that the height is no more than a given parameter K. Larmore and Przytycka [14] 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 weight K-link path in a graph whose weights are given by the matrix defined in [14]. Thus, our algorithm can be applied to achieve the desired time bound.

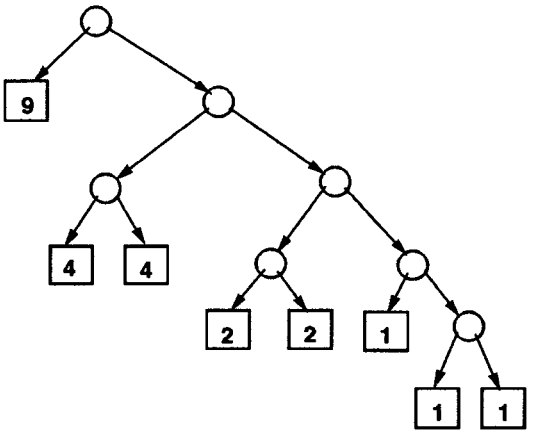

Figure 6: Huffman tree. (The numbers in leaves are proportional to the query probability.)

Application V: Discrete quantization. Let $Y = \{y_1, \ldots, y_K\}$ and $\psi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, K\}$ be the optimal quantization of $f : \{x_1, \ldots, x_n\} \to \mathcal{R}$. 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 [18] showed that ψ is a non-decreasing function, and that $y_j = \mu(\psi^{-1}(j))$. Let $c(I) = \sum_{s \in I} f(x_s)(x_s - \mu(I))^2$. Then, the error function coincides with $\sum_{j=1}^{K} c(\psi^{-1}(j))$. Hence, the function ψ represents the shortest K-link path in a DAG on the $n+1$ nodes $\{0, 1, \ldots, n\}$ where the edge weight of (i,j) is $c((i,j])$. It is easy to see that this graph satisfies the concave Monge property, and that the values $\mu(I)$ and $c(I)$ 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 $O(n\sqrt{K \log n})$ time algorithm.

5. Concluding Remarks

For DAGs whose weights obey the convex Monge property, we showed how to compute the minimum K-link path in $O(n\alpha(n)\log^3 n)$ time. However, for DAGs whose weights obey the concave Monge property, we are able to compute the minimum K-link path only in $O(n\sqrt{K\log n})$ time. This is because Megiddo's parametric search technique requires an efficient processor-time algorithm for computing the single pair shortest path for these DAGs. Indeed, if we can get an n processor polylogarithmic time algorithm for computing the shortest path in the concave case, then the minimum K-link path for DAGs with the concave Monge property can also be computed in almost linear time (i.e., within polylogarithmic factors). It is also worth noting that such an algorithm would also yield an n processor, polylogarithmic time algorithm for computing a Huffman tree on n vertices without any height restriction. Finally, note that while we have a fairly efficient algorithm for the convex case, we do not have any corresponding interesting applications.

Acknowledgement

The authors thank James K. Park for communicating the application to quantization.

References

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilbur, 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. of 29th FOCS* (1988), 497-512.

[3] A. Aggarwal and T. Tokuyama, Consecutive Interval Query and Dynamic Programming on Intervals, Preprint 1992.

[4] T. Asano, Dynamic Programming on Intervals, *Proc. ISA, 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 Computational Geometry* (1992), 120-129.

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

[9] K. Chan and T. Lam, Finding Least-Weight Subsequences with Fewer Processors, *Proc. SIGAL, Lect. notes in Comput. Sci.* **450**, Springer–Verlag (1990), 318-327.

[10] G. Frederickson, Optimal Algorithms for Tree Partitioning, *Proc. of 2nd SODA* (1991) 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, Technical Report RJ6275, IBM Almaden Research Center, 1988.

[13] L. Larmore and D. Hirschberg, Length-Limited Coding, *Proc. of 1st SODA* (1990) 310-318.

[14] L. Larmore and T. Przytycka, Parallel Construction of Trees with Optimal Weighted Path Length, *Proc. of 3rd SPAA* (1991) 71-80.

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

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

[17] L. Valiant, Parallelism in Comparison Problems, *SIAM J. Comput.* **4** (1975), 348-355.

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