ELSEVIER

Discrete Applied Mathematics 85 (1998) 1–24

DISCRETE APPLIED MATHEMATICS

Consecutive interval query and dynamic programming on intervals

Alok Aggarwal[a], Takeshi Tokuyama[b,*]

[a] *IBM Research Division, T. J. Watson Res. Ctr., P.O. Box 218, Yorktown Heights, NY 10598, USA*
[b] *IBM Research Division, Tokyo Res. Lab., 1623-14 Shimotsuruma, Yamato, Kanagawa, 242, Japan*

Received 2 April 1993; received in revised form 28 August 1996; accepted 22 December 1997

Abstract

Given a set of n points (nodes) on a line and a set of m weighted intervals defined on the nodes, we consider a particular dynamic programming (DP) problem on these intervals. If the weight function of the DP has convex or concave property, we can solve this DP problem efficiently by using matrix searching in Monge matrices, together with a new query data structure, which we call the *consecutive query* structure. We invoke our algorithm to obtain fast algorithms for sequential partition of a graph and for maximum K-clique of an interval graph.

Keywords: Dynamic programming; Matrix searching; Sequential partition; Clique covering; Interval query

1. Introduction

We consider special dynamic programming (DP) problems called *Monge interval DP* problems, motivated by its applications to *sequential partition of a graph* and *maximum K-clique of an interval graph.*

Let U be a set of n integers $\{1,2,...,n\}$. Given a non-decreasing sequence $f(i)$, $i = 1,2,\ldots,n-1$ such that $i < f(i) \leqslant n$, a set of ordered pairs $X = \{(i,j) : i < j \leqslant f(i)\}$, and a weight function $F : X \to \mathscr{R}$, let us define a function $D(j)$ by the initial condition $D(1) = 0$ and the recurrence formula

$$D(j) = \min_{i<j\leqslant f(i)} (D'(i) + F(i,j)), \tag{1}$$

where $D'(i)$ is computed from $D(i)$ in O(1) time. Consider the problem of computing $D(n)$, together with the sequence of indices $1 = \sigma(1) < \sigma(2) < \cdots < \sigma(l) = n$ satisfying $D(\sigma(i+1)) = D'(\sigma(i)) + F(\sigma(i), \sigma(i+1))$ for $i = 1,2,\ldots,l-1$.

* Corresponding author. E-mail: ttoku@trl.ibm.co.jp.

0166-218X/98/$19.00
PII S0166-218X(98)00021-3

This is a DP problem associated with the weight function F and the sequence f, and denoted by $DP(F,f)$. In order to simplify the notation, we write $DP(F)$ for $DP(F,f)$ unless f is needed explicitly.

An $O(n^2)$ time algorithm for solving $DP(F)$ was given by Kernighan [15], in which he used it as a subroutine for solving the sequential partition problem (defined later) of a graph with n vertices in $O(n^2)$ time.

Asano [1] defined the following special DP problems: Regard the set U as a set of points on a line, and let $\mathscr{I}$ be a set of m intervals, which are left-open and right-closed, and have their endpoints in U. Without loss of generality, we assume $m \geqslant n$. A weight $w(I)$ is associated with each interval. Define $W(i,j) = \sum_{I \in \mathscr{I} \text{ and } I \subset (i,j]} w(I)$, which is the sum of the weights of subintervals of $(i,j]$ in $\mathscr{I}$, and regard W as a weight function on $X = \{(i,j) : i < j \leqslant f(i)\}$. $DP(W)$ (to be precise, $DP(W,f)$) is called the *interval DP problem*. Asano [1] gave an $O(m \log n)$ time algorithm for the interval DP problem, and applied it to solved the sequential partition problem of a graph with n nodes and m edges in $O(m \log n)$ time.

1.1. Main results

We say a weight function F is concave (resp. convex) if the quadrangle inequality $F(i,j)+F(i+1,j+1) \leqslant F(i+1,j)+F(i,j+1)$ (resp. $F(i,j)+F(i+1,j+1) \geqslant F(i+1,j)+F(i,j+1)$) holds for such indices $1 \leqslant i,j \leqslant n$ that all four terms in the inequality are defined.

We consider the concave (resp. convex) interval DP problems, where the weight function W are concave (resp. convex). We also call them *Monge interval DP* problems, since the quadrangle inequality is often called the Monge inequality. Monge interval DP is an important class of interval DP problems; Indeed, if w is a nonnegative (resp. nonpositive) function, W becomes concave (resp. convex).

Specializing to Monge interval DP, we improve the time complexity: We give an $O(m+n \log\log n)$ time algorithm for a concave interval DP, and an $O(m+n \log n)$ time algorithm for a convex interval DP. Applying the above algorithms, we give efficient solutions of the sequential partition problem of graphs and the maximum K-clique problem of interval graphs.

1.2. Sequential partition of graph

Let $H = (V,E)$ be a graph with n vertices and m edges. An edge-weight function $w : E \to R^+$, a vertex-weight function $\psi : V \to R^+$, and a positive number K are given. Furthermore, we fix a linear ordering of the vertices of V labeled from 1 to n. For an integral sequence $0 = t(0) < t(1) < t(2) < \cdots < t(l) = n$ (for any l), we define $V_i = \{v_{t(i-1)+1}, v_{t(i-1)+2}, \ldots, v_{t(i)-1}, v_{t(i)}\}$. Then, V is a disjoint union of subsets (called *clusters*) $V_1, \ldots, V_l$, which is called a sequential partition of V. If the sum of weights of the vertices in each cluster is no more than K, the sequential partition is

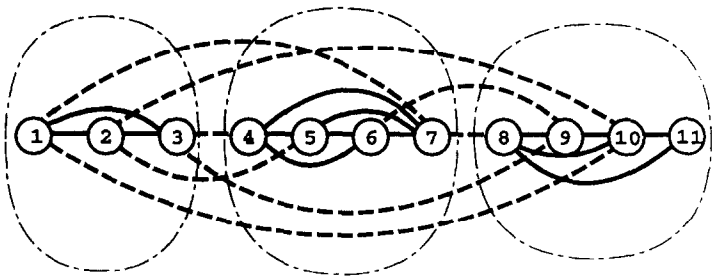

Fig. 1. Sequential partition of a graph, $n = 11$, $K = 4$. Each vertex and each edge have unit weights. Broken edges connect different clusters.

called a feasible partition (Fig. 1). We want to solve the following *sequential partition problem*:

> Find the feasible partition that minimizes the total weight of the edges connecting different clusters.

Sequential partition problem is a common heuristics to solve the (NP-hard) graph partition problem [11]; and hence a key subroutine in several practical applications (e.g. VLSI layout) [15]. By formulating the problem as a convex interval DP problem, we can solve the sequential partition problem in $O(m + n \log n)$ time, improving $O(n^2)$ time of Kernighan [15] and $O(m \log n)$ time of Asano [1].

1.3. Maximum K-clique of interval graph

Let $H = H(\mathscr{I}) = (V(\mathscr{I}), E(\mathscr{I}))$ be an interval graph associated with a set $\mathscr{I}$ of m intervals whose endpoints are among $U = \{1, 2, ..., n\}$. There is a one-to-one correspondence from $V(\mathscr{I})$ to $\mathscr{I}$, and two vertices of H are adjacent if and only if the corresponding intervals overlap with each other. Note that $H(\mathscr{I})$ has m vertices, and n becomes a hidden parameter of the graph. We fix a vertex-weight function $\psi : V(\mathscr{I}) \to R^+$. We want to solve the following *maximum K-clique problem:*

> Given a positive integer K, compute the set $\mathscr{C} = \{C_1, C_2, \ldots, C_K\}$ of cliques of H such that the total weight of vertices in the union of the cliques is maximized.

Fig. 2 shows an interval graph (right figure) associated with a set of nine segments (left figure, lifted up to two dimensional space), and its maximum 2-clique if each vertex weight is 1. The maximum K-clique problem on an interval graph has applications to classification rule generation in AI [13, 14] (especially to data discretization [14]).

Using our concave interval DP algorithm, we can compute the maximum K-clique of $H(\mathscr{I})$ in $O(m + n \min\{\log \Gamma \log\log n,\ 2^{O(\sqrt{\log K \log\log n})},\ K \log\log n\})$ time, where Γ is the maximum of the vertex weights (if we assume each weight is an integer).

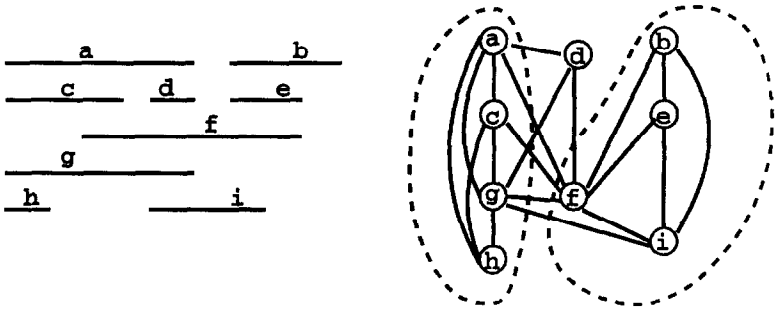

Fig. 2. Maximum 2-clique of an interval graph.

2. Consecutive interval query

2.1. Monge interval DP and interval query

Dynamic programming problems with concave or convex weight functions (Monge DP) are well-studied in the literature [3, 4, 8, 10, 18, 23]. A Monge DP problem $DP(F)$ for a concave (resp. convex) F can be solved in $O(T+nq(n))$ time [16, 18, 23] (resp. $O(T+n\alpha(n)q(n))$ time [17], where α is the inverse Ackerman function), if F can be implicitly stored using $O(T)$ preprocessing time so that its each value can be queried in $O(q(n))$ time. Hence, a Monge DP problem can be efficiently solved if its weight function can be queried efficiently.

For our Monge interval DP, the problem of querying a value $W(i,j)$ of W is a well-known problem called *interval query* (to be precise, interval query on a set of weighted intervals) in computational geometry. A naive solution method is to precompute the function W completely in $O(n^2)$ time, and to query a value of it in $O(1)$ time by using the table. Instead, a *range tree* [20] data structure answers the interval query in $O(\log n)$ time after spending $O(m \log n)$ time on preprocessing. However, because of the high preprocessing time, applying them as query data structures in known Monge DP algorithms will not improve Asano's $O(m \log n)$ time complexity.

Unfortunately, it appears to be difficult to obtain an efficient interval query data structure in $O(m)$ preprocessing time. We can construct an $O(n^{\varepsilon})$ time query data structure for any constant ε in $O(m)$ preprocessing time [6], but it only gives an $O(m+n^{1+\varepsilon})$ time algorithm for Monge interval DP.

In order to overcome this difficulty, we first construct data structures for what we call the consecutive query problem, and then show that our data structures give efficient amortized query time for solving our Monge interval DP problems.

We remark that a similar amortized query technique for a Monge DP problem was introduced by Hershberger and Suri [12] for solving the problem of computing the farthest distance within a simple polygon. This can be formulated as a concave DP problem, and $q(n)$ becomes the time needed to query the interior distance between two given vertices of a polygon. Although the current best data structure requires

an $O(\log n)$ query time, Hershberger and Suri [12] reduced $q(n)$ to a constant in an amortized sense, and obtained an optimal $O(n)$ algorithm for computing the farthest distance.

2.2. Consecutive query problem

Let us define the consecutive interval query problem (consecutive query, in short) given below.

Query $W(i+i_0, j+j_0)$ efficiently after we have queried $W(i,j)$.

The data structure for Monge interval DP consists of three different kind of consecutive query data structures: the first one answers in $O(|i_0|+|j_0|)$ time, the second in $O((|i_0|+|j_0|)/(\log n)^c + \log n)$ time for a constant c, the third is the h-skip data structure described in Section 2.5. The data structures are constructed by using *fractional cascading* [7] in $O(m)$ time.

It will be shown in later sections that these consecutive query data structures give an $O(\log\log n)$ (resp. $O(\log n)$) amortized query time for the query sequence in a solution of a concave (resp. convex) interval DP problem.

2.3. An $O(|i_0|+|j_0|)$ query time data structure

We first describe a data structure that answers the consecutive query in $O(|i_0|+|j_0|)$ query time. Without loss of generality, we assume that both i_0 and j_0 are nonnegative from now on. We remark that, in order to perform the consecutive interval query, it suffices to compute $W(i,j) - W(i+i_0, j)$ and $W(i+i_0, j) - W(i+i_0, j+j_0)$.

Let us denote the left and right endpoints of a interval I of $\mathscr{Z}$ by $left(I)$ and $right(I)$. In other words, $I = (left(I), right(I)]$.

Let $\mathscr{L}\mathscr{Z}_k$ be the set of intervals of $\mathscr{Z}$ whose left endpoint is k. Y_k is the sorted list of the right endpoints of $\mathscr{L}\mathscr{Z}_k$.

We define $\varphi(\mathscr{L}\mathscr{Z}_k, y) = \sum_{I \in \mathscr{L}\mathscr{Z}_k \text{ and } right(I) \leqslant y} w(I)$.

Lemma 2.1. $W(i,j) - W(i+i_0, j) = \sum_{k=i+1}^{i+i_0} \varphi(\mathscr{L}\mathscr{Z}_k, j)$

Proof. The value $W(i,j) - W(i+i_0, j)$ equals the total weight of intervals $(a,b]$ of $\mathscr{Z}$ satisfying that $i+1 \leqslant a \leqslant i+i_0$ and $1 \leqslant b \leqslant j$. This leads the lemma immediately. □

For all $k = 1, 2, \ldots, n$ and $\eta \in Y_k$, we precompute $\varphi(\mathscr{L}\mathscr{Z}_k, \eta)$, which are prefix sums of weights of $\mathscr{L}\mathscr{Z}_k$ sorted with respect to the right endpoints. Thus, for any nonnegative integer j, if we have the *location* of j (i.e. largest entry which is not larger than j) in the list Y_k, $\varphi(\mathscr{L}\mathscr{Z}_k, j)$ can be computed in $O(1)$ time.

Hence, what we need is a data structure which computes the locations of j in Y_k for $k = i+1, i+2, \ldots, i+i_0$ efficiently. For this purpose, we construct a data structure which is a slight modification of the *fractional cascading* data structure of Chazelle and Guibas [7].

In the data structure, an augmented list $\tilde{Y}_k$ is constructed for each list Y_k such that the location of j in Y_k can be computed in O(1) time from the location of j in $\tilde{Y}_k$ and vice versa. Further, the locations of j in $\tilde{Y}_{k+1}$ and $\tilde{Y}_{k-1}$ can be found in O(1) time from its location in $\tilde{Y}_k$. See Chazelle and Guibas [7] how to create such augmented lists for $k = 1,2,\ldots,n$ in O(m) time. We call the above structure the *left-end-based-structure*. It is clear from the properties of the data structure that we can find all the locations of j in Y_k for $k = i+1, i+2, \ldots, i+i_0$ in O(i_0) time, provided that we know the location of j in $\tilde{Y}_i$.

We construct a similar structure (called the *right-end-based-structure*) by exchanging the roles of the right endpoint and the left endpoint. The sorted list of the left endpoints of points of $\mathscr{R}\mathscr{L}_k$ (the intervals whose right endpoint is k) is denoted by X_k, and the associated augmented list is denoted by $\tilde{X}_k$. We can compute $W(i+i_0, j) - W(i+i_0, j+j_0)$ in O(j_0) time if we know the location of $i+i_0$ in $\tilde{X}_j$.

When we query $W(i,j)$, we store not only the value $W(i,j)$ but also both the location of j in Y_i and the location of i in X_j. Thus, we can find the location of j in $\tilde{Y}_i$ in O(1) time. Furthermore, we can find the location of $i+i_0$ in $\tilde{X}_j$ in O(i_0) time, since we can move at most i_0 steps up the list X_j to have the location of $i+i_0$, and we can have the location in $\tilde{X}_j$ in O(1) time from it.

Therefore, we can compute $W(i+i_0, j+j_0) - W(i,j)$ in O(i_0+j_0) time. Furthermore, we can simultaneously compute the location of $i+i_0$ in $\tilde{X}_{j+j_0}$ and that of $j+j_0$ in $\tilde{Y}_{i+i_0}$. Hence, we obtain the following:

Theorem 2.2. *We can query $W(i+i_0, j+j_0)$ after querying $W(i,j)$ in* O($|i_0| + |j_0|$) *time, by using the above data structure.*

Proposition 2.3. *The data structure can be computed in* O(m) *time.*

Proof. We prepare the sorted lists Y_k ($k = 1,2,\ldots,n$) using O(m) time for all columns. This can be done by first bucket-sorting all the elements of S with respect to the y-coordinate values, and then distributing the points into queues associated with columns. The computation of prefix sums $\varphi(\mathscr{L}\mathscr{L}_k, \eta)$ for $\eta \in Y_k$ for $k = 1,2,\ldots,n$ can be done in O(m) time in total by using a prefix-sum algorithm. Finally, the construction of the fractional cascading data structure can be done in O(m) time [7]. □

2.4. An O($\frac{|i_0|+|j_0|}{(\log n)^c} + \log n$) query time data structure

We next construct an O($\frac{i_0+j_0}{(\log n)^c} + \log n$) query time data structure for any given nonnegative integral constant c. Although we use only for $c = 1$ later to solve Monge interval DP, we here describe the data structure for a general c.

Let L be a natural number such that $L \approx \log n$. For simplicity, we assume that n/L^c is an integer (it is easy to remove this assumption). Let $\tau(s) = n/L^s$ for $s = 0,1,2,\ldots,c$.

Let us consider a set of vertices $\{v(s,t) |\ s = 0,1,\ldots,c;\ t = 1,2,\ldots,\tau(s)\}$. For each vertex $v(s,t)$, we draw an undirected edges to $v(s,t+1)$, $v(s-1,tL)$ and $v(s-1, (t-1)L+1)$ (if these vertices are defined). Now we have a graph G (Fig. 3).

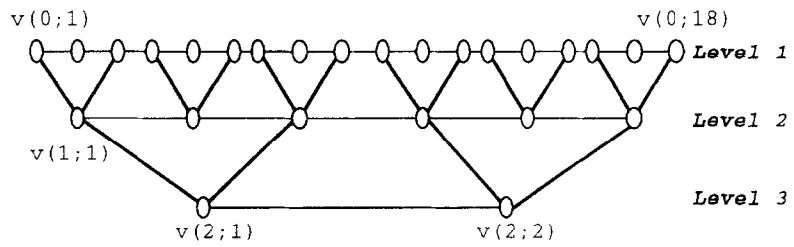

Fig. 3. Underlying graph of fractional cascading ($n = 18$, $L = 3$, $c = 2$).

For each $v = v(s,t) \in G$, we define closed intervals $J(v) = J(v(s,t)) = [(t-1)L^s + 1, tL^s]$. Let $\mathscr{L}\mathscr{Z}(v) = \{I \in \mathscr{Z} : left(I) \in J(v)\}$, and let $Y(v)$ be the sorted list of right endpoints of elements of $\mathscr{L}\mathscr{Z}(v)$. $\varphi(\mathscr{L}\mathscr{Z}(v), y)$ is similarly defined as $\varphi(\mathscr{L}\mathscr{Z}_k, y)$. We precompute prefix sums $\varphi(\mathscr{L}\mathscr{Z}(v), \eta)$ for all $v \in G$ and $\eta \in Y(v)$.

In $O(m)$ time, we can construct the fractional cascading data structure which has the underlying graph G, so that for any j, the data structure can answer the location of j in $Y(v)$ for all v on any path p of length l in G in $O(l + \log n)$ time (see [7] for details).

We construct an analogous structure exchanging the roles of right endpoint and left endpoint.

Theorem 2.4. *A data structure can be constructed in such a way that it answers the query $W(i+i_0, j+j_0)$ in $O(\frac{|i_0|+|j_0|}{(\log n)^c} + \log n)$ time as long as the query $W(i,j)$ has been answered before; this data structure can be constructed in $O(m+n)$ time, where c is a nonnegative integer and is considered as a constant.*

Proof. The interval $[i+1, i+i_0]$ can be represented as a disjoint union $\cup_{v \in Pi} J(v)$ of subintervals, such that the corresponding set Π of vertices lies on a path p, which has at most $2cL + (i_0/L^c)$ vertices, on the graph G (Fig. 4). $W(i+i_0, j) - W(i,j)$ equals the total sum of $\varphi(\mathscr{L}\mathscr{Z}(v), j)$ for $v \in \Pi$. Since $L \approx \log n$ and c is a constant, the query time is $O([i_0/(\log n)^c] + \log n)$. Similarly, we can compute $W(i+i_0, j+j_0) - W(i+i_0, j)$. □

2.5. *h-skip data structure*

In addition to the consecutive interval query data structures of Theorems 2.2 and 2.4, we use a variant of the consecutive interval query data structure called *h-skip structure* for a fixed integer h, which only deals with $W(i, jh)$ for nonnegative integers j, and can query $W(i+i_0, (j+j_0)h)$ in $O(i_0 + j_0)$ time if $W(i, jh)$ was queried before. This data structure is easy to construct in $O(n+m)$ time by rounding each interval $(s,t]$ in $\mathscr{Z}$ to $(s, h\lceil t/h \rceil]$ (if more than one interval is rounded to the same interval, we assign to the rounded interval the sum of the weights of the original intervals) and construct the data structure of Theorem 2.2 for those rounded intervals.

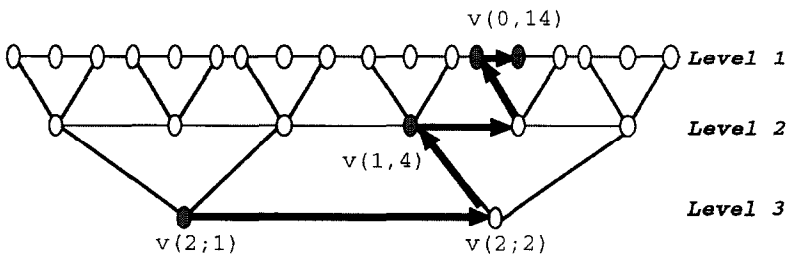

Fig. 4. (Black) vertices $v(2,1)$, $v(1,4)$, $v(0,13)$ and $v(0,14)$ associated with the decomposition $[1,9] \cup [10,12] \cup [13,13] \cup [14,14]$ of $[1,14]$, and the path p (bold arcs).

3. Algorithms for Monge interval DP problems

3.1. Matrix searching in a Monge matrix

First, we give a brief summary on *matrix searching*, which is a key subroutine in our algorithms.

A matrix $M = (M(i,j))_{i=1,2,\ldots,k;j=1,2,\ldots,l}$ is called a convex *Monge matrix* if

$$M(i,j) + M(i+1,j+1) \geqslant M(i,j+1) + M(i+1,j) \tag{2}$$

and a concave Monge matrix if

$$M(i,j) + M(i+1,j+1) \leqslant M(i,j+1) + M(i+1,j) \tag{3}$$

for $1 \leqslant i \leqslant l$ and $1 \leqslant j \leqslant l$. Entries of a matrix can be positive-infinity entries, and we permit $\infty \geqslant \infty$ to be correct in quadrangle relations.

For a matrix M, we define $m_j(M)$ and $r_M(j)$ (or $r(j)$ if we can fix M) to denote the value and the row index of the minimum entry of the jth column of M. The problem of computing $m_j(M)$ for all $j = 1,2,\ldots,l$ in an $n \times l$ matrix is called the *matrix searching problem.*

Although $\Omega(nl)$ time is necessary for matrix searching in a general matrix, it is well known [4] that matrix searching in a Monge matrix can be done in $O(n+l)$ time, if each entry can be queried in a constant time.

We consider the DP defined by Eq.(1) in the introduction, specializing for the weight function W (i.e. interval DP). We extend $W(i,j)$ so that it is ∞ unless $i < j \leqslant f(i)$, and let A be a matrix defined by $A(i,j) = D'(i) + W(i,j)$. By definition, $D(1) = 0$, $D(i) = m_i(A)$ for $i = 2,3,\ldots,n$, and we can compute $D'(i)$ from $D(i)$ in $O(1)$ time. Hence, it suffices to solve the matrix searching problem of A.

A matrix searching problem in a matrix M is called an "off-line" problem if any entry of $M(i,j)$ can be queried without any knowledge of column minima of M; otherwise, it is called an "on-line" problem. Matrix searching in the matrix A is an on-line problem, since we need $D(i)$, which is $m_i(A)$, in order to compute an entry $A(i,j)$. The following lemma shows the relation between matrix searching and concave interval DP (convex case will be discussed later):

Lemma 3.1. *If W is concave, A is a concave Monge matrix.*

Proof. If all four entries are defined (i.e. noninfinity), the Monge relation comes from the concavity of W. Otherwise, if either $A(i,j)$ or $A(i+1,j+1)$ is ∞, then $A(i+1,j)$ must be ∞ because of the monotonicity of f. Hence, the Monge relation holds. □

3.2. Off-line matrix searching using consecutive interval query

From now on, throughout this paper, we set n as the size of the original DP (i.e. size of A), and $h = \lfloor \log^2 n \rfloor$. Without loss of generality, we assume that n is an integral multiple of h. We use the h-skip data structure, as well as $O(|i_0| + |j_0|)$ time query data structure (Theorem 2.2) and $O((|i_0| + |j_0|)/\log n + \log n)$ time query data structure (Theorem 2.4 for $c = 1$).

A submatrix of A is called *rigid* if it has contiguous column indices and row indices. In this subsection, we consider a key subroutine of our solutions for Monge interval DP, namely, the off-line matrix searching in a rectangular rigid Monge submatrix M of A of size $k \times l$.

We investigate the order in which W is queried in an off-line matrix searching algorithm, and show that the consecutive query structures give an $O(\log \log n)$ amortized query time. Without loss of generality, we assume that M is concave Monge, since if M is a $k \times l$ convex Monge matrix, the matrix $\tilde{M}$ defined by $\tilde{M}(i,j) = M(i,k-j)$ is concave Monge.

Given an algorithm $\mathscr{A}$ that searches for minima of a matrix M, we define a directed path called the *search path* of $\mathscr{A}$ whose vertices are entries of M. (This path is not always a simple path, and may visit an entry several times). For two entries $M(i,j)$ and $M(i',j')$ in a matrix M, we define an *edge* between them with *length* $|i-i'| + |j-j'|$. Intuitively, the edge corresponds to a rectilinear shortest path between them in the matrix (regarded as a grid). The search path is defined incrementally according to the behavior of $\mathscr{A}$. Initially, it consists of a single vertex that corresponds to the first entry queried by $\mathscr{A}$. When $\mathscr{A}$ queries a new entry, we connect it to the path by the edge from the current destination of the search path to the new entry of M. The total sum $len(\mathscr{A})$ of the length of edges in the search path of $\mathscr{A}$ is called the *length* of the search path of the algorithm $\mathscr{A}$.

We assume that the starting entry of $\mathscr{A}$ has already visited before, and we need not worry about the query time for it. This assumption holds for our interval DP algorithms. The following two lemmas follow immediately from our definitions and Theorems 2.2 and 2.4.

Lemma 3.2. *If we use the consecutive interval query of Theorem 2.2 for querying entries of matrix A, the total cost of queries in an algorithm $\mathscr{A}$ is $O(len(\mathscr{A}))$.*

Lemma 3.3. *If we use the consecutive interval query of Theorem 2.4 for querying entries of matrix A, and an algorithm $\mathscr{A}$ queries P entries, the total cost of queries in $\mathscr{A}$ is $O(P \log n + len(\mathscr{A})/\log^c n)$.*

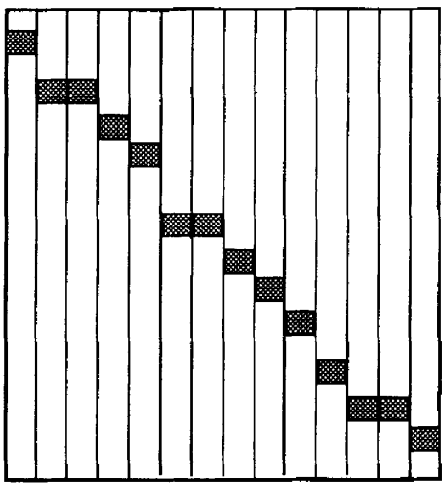

Fig. 5. Location of column minima in a concave Monge matrix.

A popular strategy for the matrix searching is the divide-and-conquer strategy. We start querying matrix entries from the north-west corner of the matrix to reach the top entry $M(1, \lceil l/2 \rceil)$ of the center-column, and then query all entries of the center column to find its minimum entry. The following lemma is fundamental, and is easily derived from the Monge property:

Lemma 3.4. *The row indices $r(i)$ $(i = 1, 2, \ldots, l)$ of the column minima of a concave (resp. convex) Monge matrix form a non-increasing (resp. non-decreasing) sequence (Fig. 5).*

Because of the above lemma, $r(i) \geqslant r(\lceil l/2 \rceil)$ if $i > \lceil l/2 \rceil$ and $r(j) \leqslant r(\lceil l/2 \rceil)$ if $j < \lceil l/2 \rceil$. Therefore, it suffices to compute the column minima in the north-west (upper-left) $r(\lceil l/2 \rceil) \times (\lceil l/2 \rceil - 1)$ submatrix and the south-east (lower right) $(k + 1 - r(\lceil l/2 \rceil)) \times (\lceil l/2 \rceil - 1)$ submatrix, which correspond to shaded regions in Fig. 6. We recursively process the northwest submatrix first, and the southeast submatrix next.

Lemma 3.5. *The length of the search path of the divide-and-conquer algorithm is* $O((k + l) \log l)$

Proof. The search path for the case where $l = 7$ is shown in Fig. 7. The length $len(\mathscr{A}(k, l))$ of the search path of the algorithm satisfies $len(\mathscr{A}(k, l)) \leqslant len(\mathscr{A}(r(\lceil l/2 \rceil), \lceil l/2 \rceil - 1) + len(\mathscr{A}(k + 1 - r(\lceil l/2 \rceil), \lceil l/2 \rceil - 1) + 2k + l$. Hence, $len(\mathscr{A}(k, l)) = O((k + l) \log l)$. □

Moreover, we use the following strategy (modified the one given in SMAWK algorithm in [3]): We first process the submatrix M' consisting of columns whose indices

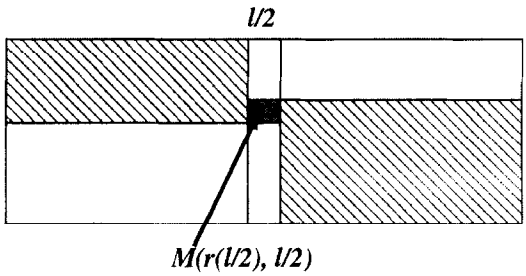

Fig. 6. Possible locations of minima.

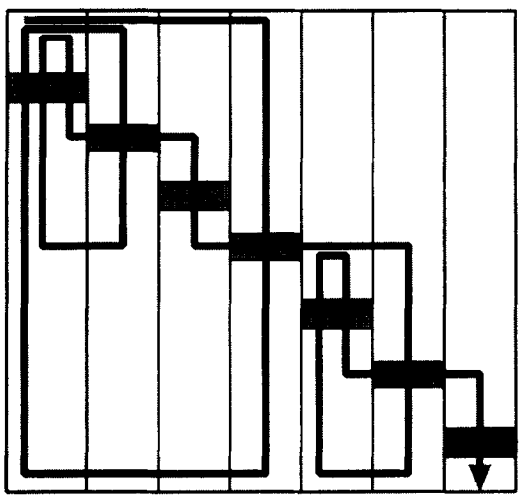

Fig. 7. Search path.

(in A) are integer multiples of h, and next process $M - M'$ using the information of column minima of M'.

Lemma 3.6. *All the minima of columns of M' can be computed in $O(k + l)$ time.*

Proof. M' has k contiguous rows and $s = \lfloor l/h \rfloor$ (noncontiguous) columns. We use the REDUCE subroutine of the SMAWK algorithm of Aggarwal et al. [3]. If REDUCE is applied to a $k' \times l'$ rigid submatrix of A where $k' > l'$, it removes $k' - l'$ rows and reduces the matrix searching problem to matrix searching in a submatrix of size $l' \times l'$. An example of the searching order of entries in REDUCE is given in Fig. 8. It is observed that length of any edge of the search path of REDUCE is at most two (see [3] for the behavior of REDUCE in details). Hence, $len(REDUCE) = O(k' + l')$.

In our matrix M', the column indices are not contiguous. However, since we deal only with columns whose indices are multiples of h, we can use the h-skip data structure for querying the entries of the matrix. Hence, if $k > s$, we can remove $k - s$ ($s = \lfloor l/h \rfloor$) rows from the searched matrix M' in $O(k + l)$ time.

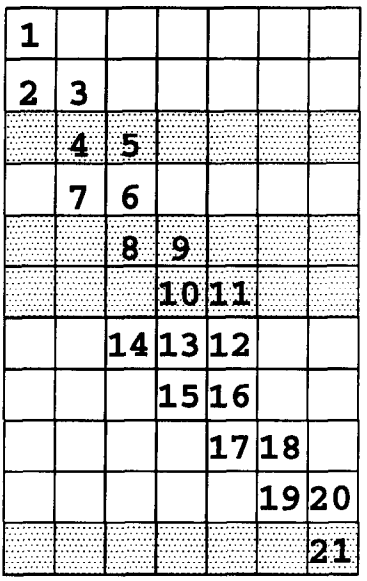

Fig. 8. Search order of REDUCE; Shaded rows are removed.

Thus, we have a matrix M'' that has at most s rows and s columns. However, neither the column indices nor the row indices are contiguous. We process this matrix by using the divide-and-conquer algorithm. The length of the search path is still $O((k+l)\log n)$, but the algorithm only queries $O(s\log n)$ entries. Thus, from Lemma 3.3, if we set $c = 1$, the algorithm spends $O(s\log^2 n + k + l) = O(k + l)$ time querying entries, and the time complexity of the algorithm is $O(k + l)$. □

Lemma 3.7. *All the column minima of M can be computed in $O((k + l)\log\log n)$ time.*

Proof. If $l \leqslant h$, we apply a divide-and-conquer strategy to solve in $O((l + h)\log l) = O((l + h)\log\log n)$. If $l > h$, we use Lemma 3.6 to compute all minima $r(ih)$ for $i = 1,2,\ldots, s$ ($s = \lceil l/h \rceil$) of M' (here, we identify indices of M and A without loss of generality). Given i, for every j such that $iL \leqslant j \leqslant (i+1)h$, $r(ih) \leqslant r(j) \leqslant r((i+1)h)$ because of the non-decreasing property of the column minimum locations. Therefore, it suffices to search in submatrices $M_1,\ldots, M_s$, such that M_i is a $k_i \times (h-1)$ matrix and $\sum_{i=1}^{s} k_i = k + s$ (Fig. 9). These submatrices can be computed in a total of $\sum_{i=1}^{s}(h + k_i)\log h = O((k + l)\log\log n)$ time. □

3.3. Klawe's Algorithm for Concave DP

We introduce Klawe's algorithm [16], which solves a concave DP problem in $O(nq(n))$ time if it takes $O(q(n))$ time to query each value of the weight function, and

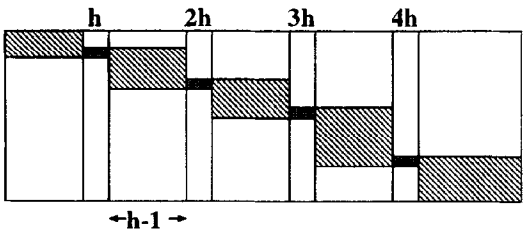

Fig. 9. Submatrices M_i (shaded regions).

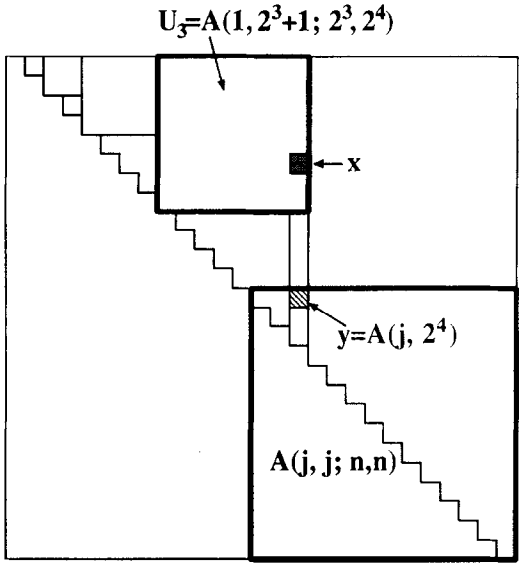

Fig. 10. Klawe's algorithm.

show that it can solve the concave interval DP problem in $O(m + n \log\log n)$ time by using the consecutive interval query data structures.

The algorithm of Klawe [16] is the following procedure DYN. Fig. 10 illustrate the state of the procedure when $i = 3$ (and $j = 13$). We use $A[a,s;b,t]$ to denote the submatrix of A associated with the (i,j) entries for $a \leqslant i \leqslant b$ and $s \leqslant j \leqslant t$. In other words, this is a rectangular submatrix whose upper-left corner entry is $A(a,s)$ and whose lower-right corner entry is $A(b,t)$.

To improve readability, in steps 5.2.2.1 and 5.2.2.2 of the procedure, we use a convention whereby $m_t(A[j,j;n,n])$ represents the minimum entry of the intersection of $A[j,j;n,n]$ and the tth column of A, which should be written as $m_{t-j+1}(A[j,j;n,n])$ mathematically, since it is the $(t-j+1)$th column of $A[j,j;n,n]$. We use the same convention for $m_t(U_i)$ in steps 5.2.2.1 and 5.3.1. Process 5.2.2 is designed to run concurrently with 5.2.1 so that 5.2.2.1 is invoked immediately after $m_t(A[j,j;n,n])$ has been computed in 5.2.1.

Procedure DYN(A) {Outputs $m_t(A)$ for $t = 1,2,\ldots,n$}
begin
1: $i = 0$;
2: $r = \min\{n, 2^{i+1}\}$;
3: Compute all the column minima of $U_i = A[1, 2^i + 1; 2^i, r]$;
4: $x =$ the minimum of the rightmost column of U_i; {located in the rth column of A}
  5: For $j = 2^i$ to r;
    5.1: Compute $y = A(j,r)$;
    5.2: If $y \leqslant x$; {Now all unknown minima are located in $A[j,j;n,n]$ or U_i }
      5.2.1: Call DYN($A[j,j;n,n]$);
      5.2.2: For $t = j$ to n; { This process runs concurrently with 5.2.1 }
        5.2.2.1: If $j \leqslant t \leqslant r$, $m_t(A) = min\{m_t(U_i), m_t(A[j,j;n,n])\}$;
        5.2.2.2: Else, $m_t(A) = m_t(A[j,j;n,n])$;
      5.2.3: Exit;
    5.3: Else (i.e. $y > x$)
{Now all column minima in $A[1, 2^i + 1; j, r]$ is located in U_i }
      5.3.1: $m_{j+1}(A) = m_{j+1}(U_i)$;
  6: End for ;
7: If $r = n$, exit; else, $i = i + 1$ and GOTO 2;
end;

Lemma 3.8 ((Klawe [16])). *The algorithm correctly computes all the column minima correctly.*

Proof. We give an outline, and refer to [16] for a detailed proof.

If we want to query an entry $A(s,j) = D'(s) + F(s,j)$ $(s < j \leqslant f(s))$ in the sth row of A, it is necessary that $m_s(A)$ should have been computed already. Note that $A(s,j) = \infty$ if $j \leqslant s$.

We first show that there is no deadlock. When we want to execute Step 3, $m_s(A)$ has already been computed for each $s = 1,2,3,\ldots,2^i$. Thus, Step 3 can be executed without a deadlock. Similarly, $m_j(A)$ has been already computed (in 5.3.1) when we want to execute Step 5.1. Since 5.2.2 is designed to run concurrently with 5.2.1 so that 5.2.2.1 is invoked immediately after $m_t(A[j,j;n,n])$ has been computed, there is no deadlock in computing the column minima of $A[j,j;n,n]$ in Step 5.2.1.

We next claim that the minima computed in the procedure are the true column minima of A. If $y > x$, Lemma 3.4 implies that there is no column minimum to the left of $(k, 2^{i+1})$ in the k-th row for $2^i + 1 \leqslant k \leqslant j$. Thus, the $(j+1)$-th column minimum must lie in the first 2^i rows, and hence, in U_i.

If $y \leqslant x$, we know that the minima of the columns to the right of the 2^{i+1}th column are located in or below the jth row, because of Lemma 3.4; Hence, we only need to search in $A[j,j;n,n]$ for these minima. For a column between the jth and the 2^{i+1}th, the column minimum is located either in $A[j,j;n,n]$ or in U_i. Thus, the procedure computes the correct column minima. □

Theorem 3.9. *The concave interval DP problem can be solved in* $O(m + n \log\log n)$ *time.*

Proof. We analyze Klawe's algorithm when it is applied to our concave interval DP problem. As a preprocess, we have constructed the three kinds of consecutive query data structure in $O(m)$ time.

When we start processing U_i, the column minima $m_j(A)$ have been already known for $j \leqslant 2^i$. Thus, matrix searching in U_i is an off-line problem.

Besides processing the submatrices U_i in Step 3, the algorithm computes entries on some specific columns (Step 5.1). We stop the algorithm when we call DYN($A[j,j,n,n]$) in Step 5.2.1. Then the length of the search path so far is $O(j)$ if we ignore the edges associated with processing of the submatrices U_i.

Thus, the length of edges except those corresponding to Step 3 in the search path of the whole algorithm is $O(n)$. Thus, the time complexity is dominated by the time taken to compute the column minima of rigid Monge submatrices U_i. Since the total of the heights and widths of these submatrices is $O(n)$, the theorem follows. □

3.4. Convex interval DP

This section uses the same notation as in the concave case, except that the weight function is convex. Unfortunately, different from the concave case, the matrix A defined by $A(i,j) = D'(i) + W(i,j)$ is not a convex Monge matrix; hence, we utilize a weaker property.

A matrix is called a partial Monge matrix if the Monge relation holds when all four entries in the quadrangle equality are non-infinity. Then, by definition, A is a partial convex Monge matrix such that $A(i,j)$ is non-infinity if and only if $i < j \leqslant f(i)$.

A partial convex Monge matrix M is called a *falling* staircase matrix (associated with g_M) if there exists a non-decreasing sequence g_M such that $M(i,j)$ is non-infinity if and only if $i \leqslant g_M(j)$ (Fig. 11). We often write g for g_M unless we explicitly need M.M is called a *reverse-falling* staircase if its transposition is a falling staircase matrix.

The matrix A can be decomposed into falling staircase submatrices and reverse-falling staircase submatrices (Fig. 12), so that each of these submatrix is rigid, and the total sum of the heights and widths of them is $O(n)$. This fact was shown by Aggarwal and Klawe [2] in a more general statement.

We process these submatrices of the matrix A from left to right. Each reverse-falling staircase submatrix in the decomposition has a form $A[a,b;s,t]$ such that $s < a$. Thus, when we compute this reverse-falling staircase submatrix, the values $D(i)$ have already been computed for all $i \leqslant s$, and therefore we can process it by using an off-line matrix searching algorithm. A falling submatrix in the decomposition has a form $A(a,a;s,s)$, for some $a < s$.

Therefore, it suffices to show that matrix searching in a falling staircase convex Monge submatrix of A can be done in $O(n \log n)$ time using consecutive query in an on-line fashion. Thus, from now on, we pretend that A itself is a falling staircase

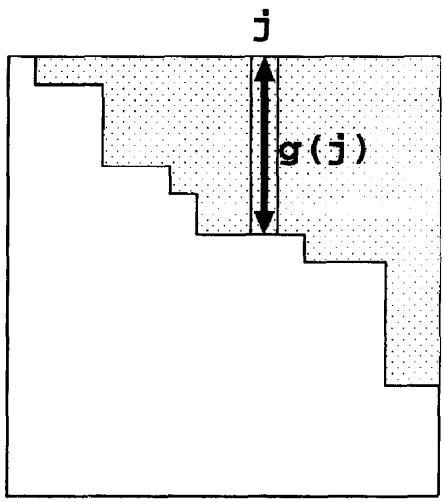

Fig. 11. Falling staircase matrix (shaded region has non-infinity entries) associated with a sequence g.

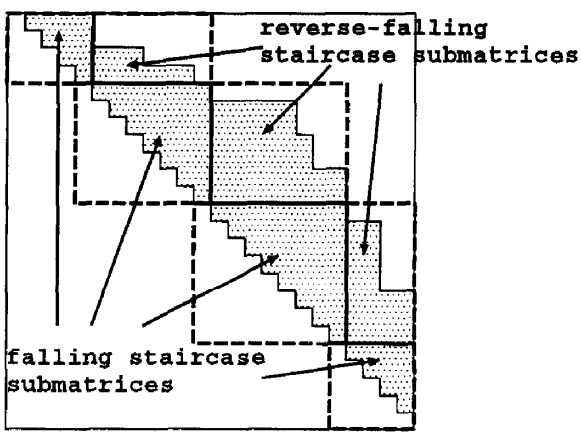

Fig. 12. Decomposition into falling and reverse falling matrices.

matrix associated with a sequence g, and values of W can be queried by using our consecutive query data structures. Note that $g(i) < i$ for $i = 1, 2, \ldots, n$ for an on-line problem.

Theorem 3.10. *Using consecutive query data structures, all the column minima of the matrix A can be computed in* $O(n \log n)$ *time in on-line fashion.*

We give a proof of the above theorem in the next subsection. Our following target result is an immediate corollary of Theorem 3.10.

Theorem 3.11. *Convex interval DP can be solved in* $O(m + n \log n)$ *time.*

3.5. Proof of Theorem 3.10

Our algorithm is obtained by modifying the following divide-and-conquer algorithm (which is different from the one in Section 3.2). Let $T(k, l)$ be the computing time for on-line matrix searching in an $k \times l$ falling staircase matrix M (with respect to g), which is a rigid submatrix of A. We, without loss of generality, assume l is even. We divide the matrix M into four submatrices $M_{NW} = M[1, 1; g(l/2), l/2)]$, $M_{NE} = M[1, l/2 + 1; g(l/2), l]$, $M_{SW} = M[g(l/2) + 1, 1; k, l/2]$, and $M_{SE} = M[g(l/2) + 1, l/2 + 1; k, l]$ in the upper-left (northwest), upper-right, lower-left, and lower-right to the $(l/2, g(l/2))$th entry, respectively. All the entries in the lower-left submatrix M_{SW} are infinities, and can therefore be ignored.

We first compute, in $T(g(l/2), l/2))$ time, the column minima of M_{NW}, which is a falling staircase matrix of size $g(l/2) \times l/2$. We next compute the column minima of M_{NE}; this is a rectangular Monge matrix, and we can process it in an off-line fashion. Thus, we can compute all the column minima of M_{NE} in $O((k + l) \log \log n)$ time using Lemma 3.7. Finally, we compute all the column minima of M_{SE}, which is a falling staircase matrix, in $T(k - g(l/2), l/2)$ time. Concurrently, we compare the column minima of M_{NE} with those of M_{SE} in $O(l)$ time. We have $T(k, l) = T(g(l/2), l/2) + T(k - g(l/2), l/2) + O((k + l) \log \log n)$, and hence the time complexity is $O((k + l) \log l \log \log n)$.

If we naively apply the above algorithm for $M = A$, the time complexity becomes $O(n \log n \log \log n)$ time, which is worse than our target. So, we modify the divide-and-conquer strategy using the idea that the processing time of A_{NE} (resp. A_{SE}) can be reduced making use of information obtained during the processing of A_{NW} (resp. A_{NW} and A_{NE}).

For $i = 1, \ldots, n/h$, let $B_i = A[g((i - 1)h + 1), (i - 1)h + 1; g(ih), ih]$ (Fig. 13). We define the union $B = \cup_{i=1,\ldots,n/h} B_i$ of those submatrices. We define $C = A - B$, which is the block falling staircase matrix obtained by removing the entries of B from A.

It is clear that the minimum $m_j(A)$ of the jth column of A is the minimum of $m_j(B)$ and $m_j(C)$. We assume temporarily that all the column minima of the matrix B have been computed in advance, and concentrate on how to process the matrix C. We define C_{NW}, C_{NE}, C_{SW}, and C_{SE} similarly to the submatrices of A. Since C is a block-upper-triangular matrix with block width h, we can stop the divide-and-conquer process when the width of a matrix becomes h. We use H (resp. H_{NW}, H_{NE}, H_{SE}) to denote the submatrix of C (resp. C_{NW}, C_{NE}, C_{SE}) consisting of all the columns whose column indices are integer multiples of h.

We can process H in the divide-and-conquer strategy in $O(n \log n)$ time, since we can compute the column minima of H_{NE} in $O(n)$ time (Lemma 3.6) when processing the rectangular matrix C_{NE}.

When we process $C_{NE} - H_{NE}$, we utilize the fact that the column minima of H_{NW} (as well as those of H_{NE}) are already known when we process C_{NE}, because of the

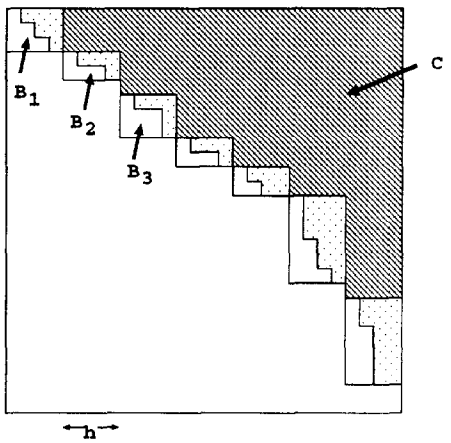

Fig. 13. Matrices B_i and matrix C.

properties of the divide-and-conquer strategy. Similarly, we know the column minima of H_{NW} and H_{NE} when we process C_{SE}.

Let $r(i) = r_C(i)$ be the row index of the minimum entry of the ith column of C. For any $j > i$, either $r(j) \leqslant r(i)$ or $r(j) > g_C(i)$, because of the Monge property. Notice the difference from the rectangular case in Lemma 3.7; the Monge relation holds only if the four terms in the relation are non-infinity.

Let us consider how we can reduce the computation time by using a knowledge of $r(ih)$. We have not yet computed the part of $C - H$ to the left of the $[(i-1)h]$th column when we compute $r(ih)$. If $ih < j$, either $r(j) \leqslant r(ih)$ or $r(j) > g_C(ih)$; also, if $i(h-1)+1 \leqslant j < ih$ then $r(j) \geqslant r(ih)$. Thus, the shaded region, which is the union of two rectangular matrices *left stripe* $A[1,(i-1)h+1; r(ih)-1, ih-1]$ and *right stripe* $A[r(ih)+1, ih+1; g_C(ih), n]$, in Fig. 14, cannot contain column minima. We prune away from $C - H$ the entries in the portion of the matrix that is located in the stripes for each $r(ih)$ $i = 1,2,\ldots, n/h$. If $r(ih) = g_C(ih)$, the right stripe is empty; in this case, we cut each rigid submatrix of the surviving part of $C - H$ containing both the $[r(ih)]$th row and $(r(ih)+1)$th row into two rigid submatrices by the horizontal line between these two rows. We denote the surviving part of the matrix by Q (Fig. 15) and prove the following claim:

Lemma 3.12. *Q consists of $O(n/\log^2 n)$ rigid submatrices $Q_1, \ldots, Q_s$. The matrix Q_i has size $q_i \times (h-1)$ and $\sum_{i=1}^{s} q_i = O(n)$.*

Proof. See Aggarwal and Klawe [2] for the proof that $\sum_{i=1}^{s} q_i = O(n)$, (Aggarwal and Klawe defined Q in a slightly different form). We, therefore, show that Q consists of $O(n/\log^2 n)$ rigid submatrices. See Fig. 16 to intuit the proof. We define $C[i]$ to be the ith block-column, which is the portion of C between the $((i-1)h+1)$th column and the ihth column of A. Note that $C[1]$ is empty, and $C[n/h]$ is the rightmost block-column.

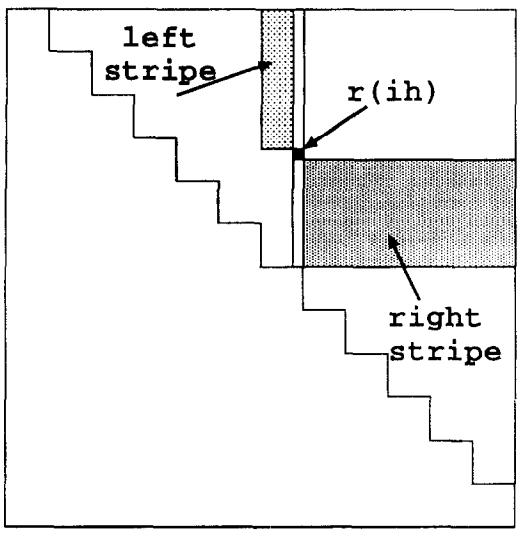

Fig. 14. Entries that can be pruned by using the minima $(r(ih), ih)$.

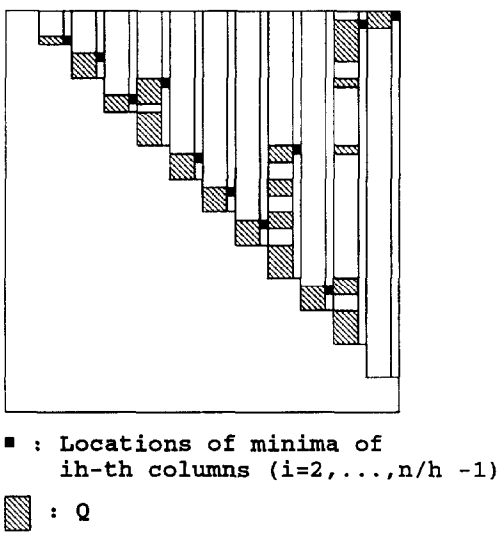

Fig. 15. Matrix Q.

We define $\gamma(i)$ as the number of connected components of $Q \cap C[i]$. We prune away the entries of C by using the row minima of H from left to right. Let $Q(j)$ be the matrix obtained by using $r(2h), \ldots, r(jh)$ to prune entries from $C - H$. By definition, $Q(n/h) = Q$ and $Q(1) = C - H$. It is easy to see that the number of connected components of $Q(j) \cap C[j]$ equals $\gamma(j)$.

We define $\mu(j)$ as the number of connected components of $Q(j) \cap C[n/h]$. The entries pruned because of $r(jh)$ form a horizontal stripe in $C[n/h]$. Thus, $\mu(j) \leqslant \mu(j-1) + 1$

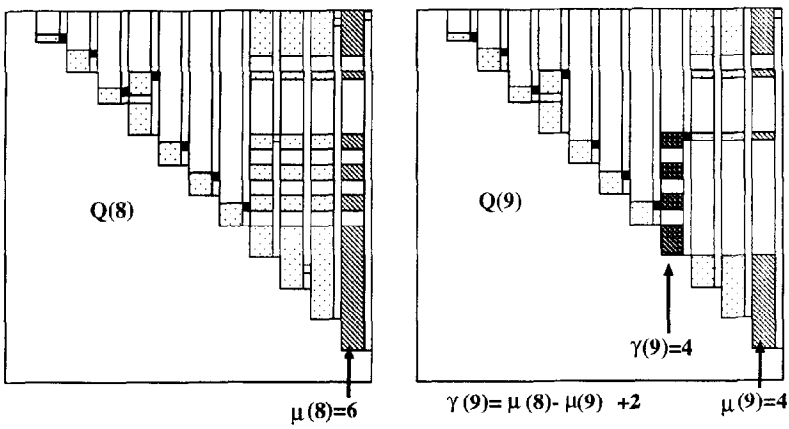

Fig. 16. Proof of Lemma 3.12.

and $\mu(1) = 1$. On the other hand, $\gamma(j) = \mu(j-1) - \mu(j) + 2$ if $j \leqslant n/h - 1$, since the right stripe pruned by $r(jh)$ contains $\gamma(j) - 2$ connected components of $Q(j) \cap C[n/h]$ if $r(jh) \neq g_C(jh)$, and it increments one connected components if $r(jh) = g_C(jh)$ (where $\gamma(j) = 1$). It is easy to see that $\gamma(n/h) \leqslant \mu(n/h - 1)$. Hence, $\sum_{j=1}^{n/h} \gamma(j) \leqslant 1 + 2(n/h - 1) - \mu(n/h - 1) + \gamma(n/h) \leqslant 2n/h$, and we have proved the claim. □

The right stripe $A[r(ih) + 1, ih + 1; g_C(ih), n]$ (or the horizontal separating line if $r(ih) = g_C(ih)$) has already separated the matrix into the northeast part and the southeast part when we need to divide the matrix at the ihth column and the $g_C(ih)$th row in the divide-and-conquer algorithm.

Thus, in the divide-and-conquer algorithm, each of the $O(n/\log^2 n)$ rigid submatrices of Q can be computed by using the off-line matrix searching algorithm of Lemma 3.7 in $(h + q_i)\log\log n)$ time, where q_i is the height of the submatrix. Hence, we can process Q in $O((hn/\log^2 n + \sum_{i=1}^{s} q_i)\log n \log n) = O(n \log\log n)$ time.

Finally, let us consider the processing of B. B_i is processed just after all column minima of C to the left of the ihth column have been computed. Since each B_i has $O(\log^2 n)$ consecutive columns, all column minima in B_i can be computed in $O((g_A(hi) - g_A(h(i-1)) + \log^2 n)(\log\log n)^2)$ time by applying the naive divide-and-conquer algorithm. Thus, we can compute all column minima of B in $O(n(\log\log n)^2)$ time.

In total, the time complexity of the algorithm is $O(n \log n)$.

4. Applications

The following lemma implies that the Monge properties are not very artificial conditions for interval DP problems:

Lemma 4.1. *If $w(e)$ is nonnegative for every $e \in \mathscr{I}$, $W(i,j)$ is a concave function. If $w(e)$ is nonpositive for every e, $W(i,j)$ is a convex function.*

Proof. We only prove the first statement, since the other statement can be proved similarly. Since $W(i,j) = \sum_{I \subset (i,j]} w(I)$,

$$(W(i,j+1) + W(i+1,j)) - (W(i,j) + W(i+1,j+1)) = w((i,j+1]) \geqslant 0.$$

This gives the concave Monge property. □

Before discussing applications, we consider the following modified version of the interval DP defined by the following recursion for fixed nonnegative integers a and b:

$$D(j) = \min_{i<j\leqslant f(i)} (D'(i) + W(i+a, j-b)).$$

Here, $W(i+a, j-b) = 0$ if $i + a \geqslant j - b$, $D(0) = 0$, and $D'(i)$ can be computed in $O(1)$ time from $D(i)$.

Lemma 4.2. *Given a set $\mathscr{I}$ of m weighted intervals on $U = \{1,2,\ldots,n\}$, $D(n)$ can be computed in $O(m + n \log\log n)$ (resp. $O(m + n \log n)$) time if the weight of each interval is nonnegative (resp. nonpositive).*

Proof. Given an interval $I = (i,j]$, we define $\pi_{a,b}(I) = (i-a, j+b]$. The problem is then equivalent to computing the interval DP on the set $\pi_{a,b}(\mathscr{I}) = \{\pi_{a,b}(I) | I \in \mathscr{I}\}$ of intervals. □

4.1. Sequential partition of a graph

We use notations given in section 1. We can consider a vertex of H as a node (integral point) of the interval $(0,n]$, the edge e between i and j as a sub-interval $I(e) = (i,j]$ of $(0,n]$, and a cluster V_i of a sequential partition as an interval $(t(i-1), t(i)]$. Then, the cost of edges in the cluster V_i is $\sum_{e\in E,\ I(e)\subset(t(i-1)+1,t(i)]} w(e) = W(t(i-1)+1, t(i))$, if we consider $\mathscr{I} = \{I(e) : e \in E\}$. We define a sequence $f(i)$ $i = 1,2,\ldots,n$ such that $f(i)$ is the largest integer satisfying the condition that the summation of the vertex weights of $\{v_{i+1},\ldots,v_{f(i)}\}$ does not exceed K.

We want to maximize $\sum_{i=1}^{l} W(t(i-1)+1, t(i))$ under the condition that $t(i) \leqslant f(t(i-1))$. Consequently, the sequential partitioning problem becomes one of computing the functions $D(i)$ on U, such that $D(0) = 0$ and

$$D(j) = \max_{i<j\leqslant f(i)} \{D(i) + W(i+1, j)\}.$$

Theorem 4.3. *The optimal sequential partition of H is obtained in $O(m + n \log n)$ time.*

Proof. It is easy to see that the sequence f can be computed in $O(n)$ time. By replacing each weight with its negative, we can formulate the rest of the computation

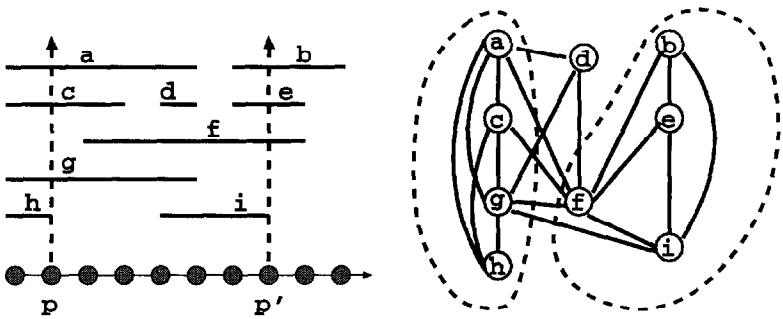

Fig. 17. Interval representation of cliques of an interval graph.

as a minimizing problem, where the weight of each interval is nonpositive. Thus, the problem is a convex interval DP, and can be solved in $O(m + n \log n)$ time. □

4.2. *Maximum K-clique of an interval graph*

For a set $\mathscr{C}$ of cliques of the interval graph $H(\mathscr{I})$, we denote $|\mathscr{C}|$ for the cardinality (number of cliques) of $\mathscr{C}$ and $\Psi(\mathscr{C})$ for the vertex-weight sum of the union of cliques of $\mathscr{C}$. Then, the maximum K-clique maximizes $\Psi(\mathscr{C})$ under the constraint that $|\mathscr{C}| = K$. In order to solve the maximum K-clique problem, we solve the following "parametric partial edge-covering with cliques (PPEC)" as a subproblem:

> Given a positive real number t, find a set $\mathscr{C}$ of cliques such that $\Psi(\mathscr{C}) - t|\mathscr{C}|$ is maximized.

Let $\mathscr{I}^p$ be the set of all intervals of $\mathscr{I}$ containing $p \in U$ in each of them. $\mathscr{I}^p$ corresponds to a clique C^p of $H(\mathscr{I})$. Also, any maximal clique has this form. We can assume that every clique in $\mathscr{C}$ of the solution of PPEC is a maximal clique, since otherwise we can replace a clique in $\mathscr{C}$ with a maximal clique containing it without decreasing $\Psi(\mathscr{C})$.

The dotted lines in the left panel of Fig. 17 represent p and p' such that the cliques associated with $\{a, c, g, h\}$ and $\{b, e, f, i\}$ are $\mathscr{I}^p$ and $\mathscr{I}^{p'}$, respectively.

Thus, PPEC is equivalent to the problem that finds a sequence $p_1 < p_2 < \cdots < p_l$ of nodes (l is not given in advance) such that $-lt + \sum_{I \in \cup_{i=1}^{l} \mathscr{I}^{p_i}} w(I)$ is maximized.

If we subtract $\sum_{I \in \cup_{i=1}^{l} \mathscr{I}^{p_i}} w(I)$ from the total sum of weights of intervals in $H(\mathscr{I})$, we have the total weight of intervals that contain no node of $\{p_1, \ldots, p_l\}$. Hence, the above problem is equivalent to the one that finds a set $p_1, p_2, \ldots, p_l$ of nodes such that $\sum_{i=1}^{l+1} (W(p_{i-1}, p_i - 1) + t)$ is minimized, where we set p_0 to be 0. Now, we have the following theorem:

Theorem 4.4. *PPEC can be solved in* $O(m + n \log \log n)$ *time.*

Proof. The problem can be solved by computing $D(n+1)$, using the recursion

$$D(j) = \min_{i<j}(D'(i) + W(i, j-1)) \quad \text{and} \quad D'(i) = D(i) + t.$$

Since $w(I)$ is nonnegative for each interval I, the problem is an instance of concave Monge DP. □

We search for a parameter value t_{opt} such that the solution of PPEC at $t = t_{\text{opt}}$ has exactly K cliques. Apparently, this solution gives the maximum K-clique of the original interval graph. Suppose we know that the solution of the PPEC has K_0 cliques at $t = t_0$. Then, $t_{\text{opt}} < t_0$ if $K > K_0$, and $t_{\text{opt}} > t_0$ if $K < K_0$. Thus, we can use a binary searching method to find t_{opt}. If Γ is the maximum of the weight, the time complexity is $O(m + n \log\log n \log \Gamma)$, since we can solve partial clique partition problems in $O(n \log \Gamma)$ time if the consecutive interval query data structure is given in advance. Note that preprocessing of the interval query is required only once.

We can apply the parametric searching technique [19] to make the above algorithm strongly polynomial. By applying an algorithm of Schieber [22] for solving a concave DP problem, we can solve the maximum K-clique problem in $O(n2^{O(\sqrt{\log k \log\log n})} \log\log n) = O(n2^{O(\sqrt{\log k \log\log n})})$ time.

A different approach to solving the maximum K-clique problem is the direct use of the matrix searching K times (see [3] for a similar solution for computing the extremal K-gon of a convex polygon). It follows from Lemma 3.7 that this approach requires $O(m + Kn \log\log n)$ time (we omit details).

Combining these three approaches, we have the following:

Theorem 4.5. *The maximum K-clique problem can be solved in $O(m + \min\{\log \Gamma \log\log n, 2^{O(\sqrt{\log K \log\log n})}, K \log\log n\} n)$ time.*

5. Concluding remarks

In this paper, we have dealt with the convex and concave cases of the dynamic programming problem on intervals. Our solution for the concave case runs in $O(m + n \log\log n)$ time, and is thus generally faster than the previous-best $O(m \log n)$ time algorithm. The solution for the convex case takes $O(m + n \log n)$ time, and faster than the previous $O(m \log n)$ time algorithm only if $n = o(m)$. Bridging this gap in the time complexity remains an open problem.

Whether a general interval DP problem can be solved in $o(m \log n)$ time remains another open question.

Acknowledgements

The authors thank the anonymous referees for many valuable comments.

References

[1] T. Asano, Dynamic programming on intervals, Internat. J. Comput. Geom. Appl. 3 (1991) 323–330.
[2] A. Aggarwal, M. Klawe, Applications of generalized matrix searching to geometric algorithms, Discrete Appl. Math. 27 (1990) 2–23.
[3] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, Geometric Applications of a matrix-searching algorithm, Algorithmica 2 (1987) 209–233.
[4] A. Aggarwal, J. Park, Notes on searching in multidimensional monotone arrays, Proc. 29th IEEE Symp. on Foundations of Computer Science, 1988, pp. 497–512.
[5] A. Aggarwal, B. Schieber, T. Tokuyama, Finding a minimum weight K-link path in graphs with Monge property and applications, Discrete Comput. Geom. 12 (1994) 263–280.
[6] A. Aggarwal, T. Tokuyama, Unpublished result.
[7] B. Chazelle, L. Guibas, Fractional cascading: I. A data structure technique, Algorithmica 1 (1986) 133–162.
[8] D. Eppstein, Sequence comparison with mixed convex and concave costs, J. Algorithm 11 (1990) 85–101.
[9] D. Eppstein, Z. Galil, R. Giancarlo, G. Italiano, Sparse dynamic programming, Proc. 1 ACM–SIAM Symp. on Discrete Algorithms, 1990, pp. 513–522.
[10] Z. Galil, K. Park, A linear-time algorithm for concave one-dimensional dynamic programming, Inform. Process. Lett. 33 (1990) 309–311.
[11] M. R. Garey, D. S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, W. H. Freeman, San Francisco, 1979.
[12] J. Hershberger, S. Suri, Matrix searching with the shortest path metric, Proc. 25th ACM Symp. on Theory of Computing, 1993, pp. 485–494.
[13] S. J. Hong, R-Mini: a heuristic algorithm for generating minimal rules from examples, in: Proc. PRICAI-94.
[14] S. J. Hong, Use of contextual information for feature ranking and discretization, IBM research report, RC-19664, 1994.
[15] B. Kernighan, Optimal sequential partitions of graphs, J. ACM 18 (1971) 34–40.
[16] M. Klawe, A simple linear time algorithm for concave one-dimensional dynamic programming, Technical Report, University of British Columbia, Vancouver, 1989.
[17] M. Klawe, D. Kleitman, An almost linear time algorithm for generalized matrix searching, Technical Report, IBM Almaden RC 1988.
[18] L. Larmore, B. Schieber, On-line dynamic programming with applications to the prediction of RNA secondary structure, J. Algorithms 12 (1991) 490–515.
[19] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, J. ACM 30 (1983) 852–865.
[20] F. Preparata, M. Shamos, Computational Geometry, An Introduction, 2nd ed., Springer, Berlin, 1988.
[21] R. Sedgewick, Algorithms, 2nd ed., Addison-Wesley, Reading, MA, 1988.
[22] B. Schieber, Computing a minimum-weight k-link path in graphs with the concave monge property, Proc. 6th ACM–SIAM SODA, 1995, pp. 405–411.
[23] R. Wilber, The concave least-weight subsequence problem revisited, J. Algorithms 9 (1988) 418–425.