On dynamic programming with underlying Monge path-decomposable array

Alfredo García, Pedro Jodrá, Javier Tejel[1]

Abstract

In this paper, we present an on-line algorithm to solve the dynamic programming recurrence $E[i_1] = \min_{i_2,i_3} \min_{i_4 \le i_1} \left\{ E[i_4 - 1] + w(i_1, i_2, i_3, i_4) \right\}$, for $i_1 = 1, \ldots, n_1$, where $E[0]$ is given and $W = \{w(i_1, i_2, i_3, i_4)\}$ is an $n_1 \times n_2 \times n_3 \times n_1$ weight Monge path-decomposable array. Our algorithm extends to multidimensional arrays the algorithm given by García et al. [5] for on-line searching of minima in Monge path-decomposable three-dimensional arrays.

Keywords: Monge arrays; Dynamic programming; Computational complexity

1 Introduction

1.1 Totally monotone arrays

A two-dimensional $n \times m$ array $W = \{w(i,k)\}$ is called monotone if the minimum entry in its i-th row lies below or to the right of the minimum entry in its $(i-1)$-st row. W is called totally monotone if $w(i,k) > w(i,k')$ implies $w(i',k) > w(i',k')$, for all $i < i'$ and $k < k'$. In other words, every 2×2 subarray is monotone. Let $k(i)$ be the smallest column index where the minimum entry in its i-th row is found. The main property of a totally monotone array is that $k(1) \le k(2) \le \ldots \le k(n)$, and this property is also satisfied by the row minima of any subarray.

Totally monotone arrays were introduced by Aggarwal et al. in the seminal paper [1], where they showed how a wide variety of problems in computational geometry could be reduced to the problem of finding the row minima in two-dimensional totally monotone arrays. Aggarwal et al. [1] described a $\Theta(n+m)$ algorithm to compute the row minima in $n \times m$ totally monotone arrays, provided that any entry can be computed in constant time. We shall refer to their algorithm as the SMAWK algorithm.

It is not hard to see that the SMAWK algorithm also computes linearly the row minima in a lower triangular totally monotone array. An $n \times m$ array W is said to be lower triangular if there are integer constants $1 \le c_1 \le \ldots \le c_n \le m$ such that $w(i,k)$ is defined for $k \le c_i$ and undefined otherwise. A lower triangular array is called totally monotone if the total monotonicity condition holds for any four defined entries of W which form a rectangular subarray.

Monge arrays are an important subclass of totally monotone arrays. An $n \times m$ array $W = \{w(i,k)\}$ is called Monge if $w(i,k) + w(i',k') \le w(i,k') + w(i',k)$, for $1 \le i < i' \le n$ and $1 \le k < k' \le m$. Although the total monotonicity condition is more restrictive than the Monge condition, in practical applications the most usual case of totally monotone arrays is that of Monge arrays. Especially in geometric settings, the Monge condition is closely related to the quadrangle inequality for convex quadrangles (the sum of the diagonals is greater than the sum of opposite sides, in any convex quadrilateral). A review on Monge properties and applications can be found in [4].

Aggarwal and Park [2] extended the notion of two-dimensional totally monotone arrays to multidimensional arrays and gave sequential algorithms for searching minima in such arrays. These authors defined two important subclasses of multidimensional totally monotone arrays called Monge path- and cycle-decomposable arrays. For $d \ge 3$, an $n_1 \times n_2 \times \ldots \times n_d$ d-dimensional array $W = \{w(i_1, \ldots, i_d)\}$ is called Monge path-decomposable if each of its entries satisfies

$$w(i_1, \ldots, i_d) = w^{(1,2)}(i_1, i_2) + w^{(2,3)}(i_2, i_3) + \ldots + w^{(d-1,d)}(i_{d-1}, i_d),$$

[1]Dpto. Métodos Estadísticos. Universidad de Zaragoza. E-mail: {olaverri,pjodra,jtejel}@posta.unizar.es

where for all $j < k$, the $n_j \times n_k$ array $W^{(j,k)} = \{w^{(j,k)}(i_j, i_k)\}$ is a two-dimensional Monge array. On the other hand, $W = \{w(i_1, \ldots, i_d)\}$ is called Monge cycle-decomposable if each of its entries satisfies

$$w(i_1, \ldots, i_d) = w^{(1,2)}(i_1, i_2) + \ldots + w^{(d-1,d)}(i_{d-1}, i_d) + w^{(d,1)}(i_d, i_1),$$

where for all $j < k$, the $n_k \times n_j$ array $W^{(j,k)} = \{w^{(j,k)}\}$ is Monge as above, and the $n_d \times n_1$ array $W^{(d,1)} = \{w^{(d,1)}(i_d, i_1)\}$ is Monge.

Let $W = \{w(i_1, \ldots, i_d)\}$ be an $n_1 \times \ldots \times n_d$ d-dimensional array. Henceforth each $(d-1)$-dimensional subarray of W corresponding to a fixed value of i_1 will be called a plane. Let $k_2(i_1), \ldots, k_d(i_1)$ be the second through d-th indices of the minimum entry in the plane i_1 of W, i.e.,

$$w(i_1, k_2(i_1) \ldots, k_d(i_1)) = \min_{\substack{1 \le i_j \le n_j \\ j=2,\ldots,d}} \{w(i_1, i_2, \ldots, i_d)\}.$$

If a plane contains multiple minima, we chose the first of the minima in the lexicographical order by the second through d-th indices. The $n_1 \times \ldots \times n_d$ d-dimensional Monge path- or cycle-decomposable arrays satisfy the property $k_j(1) \le \ldots \le k_j(n_1)$, for all $j = 2, \ldots, d$. Moreover, this property is fulfills by the minimum of any plane.

One problem that arises in connection with d-dimensional Monge path- and cycle-decomposable arrays is the *plane minima problem*, in which we wish to compute the minimum entry for each value of its first index i_1. Aggarwal and Park [2] showed that the plane minima problem for an $n_1 \times \ldots \times n_d$ d-dimensional array W can be solved in $\Theta(\sum_{j=1}^{d} n_j)$ time if W is Monge path-decomposable, and in $O((\sum_{j=2}^{d} n_j) \log n_1)$ time if W is Monge cycle-decomposable. We shall call AGPARK their algorithm. Aggarwal and Park applied these results to improve algorithms for some important problems described in [2, 3], such as the economic lot-sizing problem with backlogging, a well-known problem in operations research and management science.

Finally, it should be observed that the array-searching problems considered in [1, 2] are off-line problems. An off-line problem is a problem where each input is available at all times, while a problem where each input is available only after certain outputs have been computed is called an on-line problem.

1.2 Speeding up dynamic programming

Dynamic programming is an important technique that has been used widely in many disciplines, including operations research and computer science. In applying this technique, one tries to take advantage of certain properties of the problem under consideration to speed up the algorithm. It turns out that dynamic programming algorithms can often be speeded up if the underlying weight-array satisfy a total monotonicity property.

In [7], Larmore and Schieber considered the following one-dimensional dynamic programming recurrence

$$E[i] = \min_{1 \le k \le c_i} \{F[k] + w(i,k)\}, \quad i = 1, \ldots, n, \quad (1)$$

under the following assumptions: (i) $W = \{w(i,k)\}$ is an $n \times m$ weight array and each entry $w(i,k)$ can be computed in constant time, (ii) the integer constants $c_1, \ldots, c_n$ are given and satisfy $1 \le c_1 \le \ldots \le c_n \le m$, (iii) the values $F[k]$, for $k = 1, \ldots, c_1$, are given and (iv) the values $F[k]$, for $k = c_{i-1}, \ldots, c_i$, can be computed from $E[i-1]$ in constant time.

Recurrence (1) can be viewed as an on-line searching of the row minima in an $n \times m$ lower triangular array $A = \{a(i,k)\}$, with entries $\{a(i,k) = F[k] + w(i,k)\}$ defined for $1 \le k \le c_i$. Since the defined entries of columns $c_{i-1}+1, \ldots, c_i$ are available only after the minimum entry in row $i-1$ has been found, the problem of searching the row minima in A is an on-line problem, hence the SMAWK algorithm cannot be applied. Larmore and Schieber [7] developed a complicated $\Theta(n+m)$ algorithm to compute recurrence (1), provided that A is a totally monotone array, in particular when W is a Monge array. We shall refer to their on-line algorithm as the LARSH algorithm.

Some shortest paths problems on the plane were reformulated by García et al. [5] in terms of the following recurrence

$$E[i] = \min_{j=1,\ldots,n_2} \min_{1 \le k \le c_i} \{F[k] + b(i,j) + c(j,k)\}, \quad i = 1, \ldots, n_1, \quad (2)$$

under the following assumptions: (i) the integer constants $c_1, \ldots, c_{n_1}$ are given and satisfy $1 \leq c_1 \leq \ldots c_{n_1} \leq n_3$, (ii) the values $F[k]$ are given, for $k = 1, \ldots, c_1$, and (iii) the values $F[k]$, for $k = c_{i-1}, \ldots, c_i$, can be computed from $E[i-1]$ in constant time, for $i = 2, \ldots, n$, (iv) $B = \{b(i,j)\}$ and $C = \{c(j,k)\}$ are Monge weight-arrays of size $n_1 \times n_2$ and $n_2 \times n_3$, respectively, and their entries can be computed in constant time. In [5], García et al. provided a linear algorithm to solve dynamic programming equation (2), which we shall refer to as the MINIMA algorithm. This result allowed us to derive linear algorithms for problems such as the travelling salesman problem for points on a convex polygon and a segment line inside it [5], and the minimum latency problem for points on a line [6].

In this paper, we deal with the problem of speeding up a dynamic programming scheme extending recurrence (2) when the underlying weight-array is d-dimensional, with $d \geq 4$, and satisfy the Monge path-decomposable condition. In particular, we are concerned with a one-dimensional dynamic programming equation of the form

$$E[i_1] = \min_{i_2,\ldots,i_{d-1}} \min_{i_d \leq i_1} \{E[i_d - 1] + w(i_1, \ldots, i_d)\},$$

$$i_1 = 1, \ldots, n_1, \qquad (3)$$

where it is assumed that:

1. The value of $E[0]$ is given to begin the recurrence.
2. $W = \{w(i_1, \ldots, i_d)\}$ is an $n_1 \times \ldots \times n_{d-1} \times n_1$ d-dimensional Monge path-decomposable array and each entry $w(i_1, \ldots, i_d)$ can be computed in constant time.

For the sake of clarity, only the case $d = 4$ will be considered. The computational complexity of the on-line algorithm we describe is $O(n_1 + (n_2 + n_3) \log n_1)$ in time, speeding up the $O(n_1^2 \times n_2 \times n_3)$ time obvious algorithm. Moreover, the algorithm is linear in space.

2 The on-line dynamic programming algorithm

Let $W^j = \{w^j(i_j, i_{j+1})\}$ be an $n_j \times n_{j+1}$ two-dimensional Monge array, for $j = 1, 2, 3$. We consider the problem of computing the following one-dimensional dynamic programming recurrence

$$E[i_1] = \min_{\substack{1 \leq i_2 \leq n_2 \\ 1 \leq i_3 \leq n_3}} \min_{1 \leq i_4 \leq i_1} \Big\{ E[i_4 - 1] + w^1(i_1, i_2) + w^2(i_2, i_3) + w^3(i_3, i_4) \Big\}, \quad i_1 = 1, \ldots, n_1, \qquad (4)$$

where $E[0]$ is given and $w^j(i_j, i_{j+1})$ can be computed in constant time at all times, for all j. Since the constraint $i_4 \leq i_1$, we have $n_4 = n_1$.

In order to solve (4), we shall need the following definitions. Let $A = \{a(i_1, i_2, i_3, i_4)\}$ be the $n_1 \times n_2 \times n_3 \times n_1$ array given by $\{a(i_1, i_2, i_3, i_4) = w^1(i_1, i_2) + w^2(i_2, i_3) + w^3(i_3, i_4) + E[i_4 - 1]\}$ defined when $i_4 \leq i_1$. Observe that for fixed i_4, the on-line assumption implies the entries in the plane i_4 of A are available only after the minimum entry in the plane $i_1 = i_4 - 1$ of A has been found. Therefore, the problem of computing the dynamic programming recurrence considered here is equivalent to on-line searching of the minimum entry in each plane i_1 of the array A.

Furthermore, we consider the following arrays. Let $\overline{C} = \{\overline{c}(i_2, i_3, i_4)\}$ be the $n_2 \times n_3 \times n_1$ array with $\{\overline{c}(i_2, i_3, i_4) = w^2(i_2, i_3) + w^3(i_3, i_4) + E[i_4 - 1]\}$. For $i_1 = 1, \ldots, n_1$, let $A^{i_1} = \{a^{i_1}(i_2, i_3, i_4)\}$ be the $n_2 \times n_3 \times i_1$ array defined as follows

$$a^{i_1}(i_2, i_3, i_4) = w^1(i_1, i_2) + \overline{c}(i_2, i_3, i_4).$$

Observe that $\overline{C}$ and A^{i_1}, for $i_1 = 1, \ldots, n_1$, defined above are Monge path-decomposable arrays.

Clearly, $E[i_1]$ is the global minimum in A^{i_1}. We shall denote by $(I_2(i_1), I_3(i_1), I_4(i_1))$ the tuple (i_2, i_3, i_4), where $E[i_1]$ is achieved, i.e., $E[i_1] = a^{i_1}((I_2(i_1), I_3(i_1), I_4(i_1))$. We shall show that we need only $O(n_1 + n_2 + n_3)$ candidate tuples (i_2, i_3, i_4) to compute $(I_2(i_1), I_3(i_1), I_4(i_1))$, for all i_1.

Let $D = \{d(i_1, i_2, i_3)\}$ be the $n_1 \times n_2 \times n_3$ array defined as $\{d(i_1, i_2, i_3) = w^1(i_1, i_2) + w^2(i_2, i_3) + w^3(i_3, i_1)\}$. Let $(d_2(i_1), d_3(i_1))$ be the pair (i_2, i_3), where the minimum entry in each plane i_1 of D is found, for $i_1 = 1, \ldots, n_1$. D is a Monge cycle-decomposable array and its entries are available at all times. Thus, we can compute the pairs $(d_2(i_1), d_3(i_1))$, for $i_1 = 1, \ldots, n_1$, by applying the off-line AGPARK algorithm over D. Since

D is a Monge cycle-decomposable array, we have $d_2(i_1) \leq d_2(i_1+1)$, for $i_1 = 1, \ldots, n_1 - 1$.

The indices $d_2(i_1)$, for $i_1 = 1, \ldots, n_1$, play a key role in the reduction of the candidate tuples (i_2, i_3, i_4) where all the minima $E[i_1]$ can be found. The following lemma gives some properties of these tuples.

Lemma 1 *For* $i_1 = 1, \ldots, n_1$,
(1) $I_4(i_1) \leq i_1$,
(2) $d_2(I_4(i_1)) \leq I_2(i_1) \leq d_2(i_1)$,
(3) $d_3(I_4(i_1)) \leq I_3(i_1) \leq d_3(i_1)$. □

Given i_1, the previous lemma implies that $(I_2(i_1), I_3(i_1), I_4(i_1))$ is a tuple in the set R_{i_1} defined as follows

$$R_{i_1} = \{(i_2, i_3, i_4) : \ 1 \leq i_4 \leq i_1, \ 1 \leq i_3 \leq n_3\}.$$

In order to reduce the set of candidate tuples in R_{n_1}, we define the following arrays.

Let $\overline{C}' = \{\overline{c}'(i_2, i_3, i_4)\}$ be the subarray of $\overline{C}$ given by $\{\overline{c}'(i_2, i_3, i_4) = \overline{c}(i_2, i_3, i_4)\}$, for $d_2(1) \leq i_2 \leq d_2(n_1)$, $1 \leq i_4 \leq r_{i_2}$, and $1 \leq i_3 \leq n_3$.

Let $G = \{g(i_4, i_3)\}$ be the $n_1 \times n_3$ array defined by

$$g(i_4, i_3) = w^2(d_2(i_4), i_3) + w^3(i_3, i_4),$$

for $1 \leq i_4 \leq n_1$, and $1 \leq i_3 \leq n_3$.

Let $H = \{h(i_2, i_3, i_4)\}$ be the subarray of $\overline{C}'$ defined as follows

$$h(i_2, i_3, i_4) = w^2(i_2, i_3) + w^3(i_3, i_4) + E[i_4 - 1],$$

for $i_2 = d_2(1)+1, \ldots, d_2(n_1)$, $1 \leq i_3 \leq n_3$, and $1 \leq i_4 \leq r_{i_2-1}$, where $r_{i_2} = \max\{i_4 : \ d_2(i_4) \leq i_2\}$, for $i_2 = d_2(1), \ldots, d_2(n_1)$ (note that $r_{i_2} \leq r_{i_2+1}$, for all $i_2 = d_2(1), \ldots, d_2(n_1) - 1$).

Figure 1 represents the tuples (i_2, i_3, i_4) in R_{n_1} from which we define the arrays H and G, respectively. Grey points in Figure 1 correspond to the indices $d_2(i_1)$, for $i_1 = 1, \ldots, n_1$.

It is not hard to see that G is a two-dimensional Monge array. Moreover, G becomes available after the indices $d_2(i_1)$, for $i_1 = 1, \ldots, n_1$, have been computed. Then the minimum entry in each row of G can be computed in linear time by using the off-line SMAWK algorithm. Let $g(i_4)$ be the smallest index i_3 where the minimum entry in row i_4 of G is found, for $i_4 = 1, \ldots, n_1$. Since G is a two-dimensional Monge array, we have $g(i_4) \leq g(i_4+1)$, for $i_4 = 1, \ldots, n_1 - 1$.

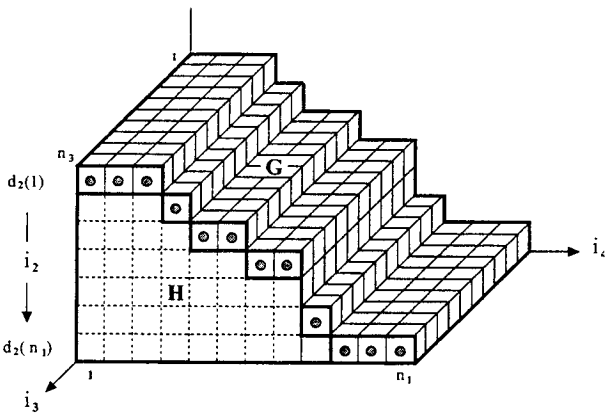

Figure 1: Arrays G and H from R_{n_1}.

The plane minima problem for H can also be solved linearly. This problem is equivalent to compute the following dynamic programming equation

$$H[i_2] = \min_{1 \leq i_3 \leq n_3} \min_{1 \leq i_4 \leq r_{i_2-1}} \Big\{ w^2(i_2, i_3) + w^3(i_3, i_4) + E[i_4 - 1] \Big\}, \quad i_2 = d_2(1)+1, \ldots, d_2(n_1).$$

Therefore, the minima $H[i_2]$, for $i_2 = d_2(1) + 1, \ldots, d_2(n_1)$, can be found linearly by using the on-line MINIMA algorithm. Let $(i_2, i_3^H(i_2), i_4^H(i_2))$ be the tuple (i_2, i_3, i_4) where the minimum $H[i_2]$ is achieved, for $i_2 = d_2(1)+1, \ldots, d_2(n_1)$, that is,

$$H[i_2] = w^2(i_2, i_3^H(i_2)) + w^3(i_3^H(i_2), i_4^H(i_2)) + E[i_4^H(i_2) - 1].$$

From the indices $g(i_4)$, for $i_4 = 1, \ldots, n_1$, and $d_2(i_1)$, for $i_1 = 1, \ldots, n_1$, the following lemma asserts the only candidate tuples in R_{n_1} where all the minima $E[i_1]$ can be found.

Lemma 2 *For* $i_1 = 1, \ldots, n_1$, *either*
(1) $I_3(i_1) = i_3^H(I_2(i_1))$ *and* $I_4(i_1) = i_4^H(I_2(i_1))$, *or*
(2) $I_2(i_1) = d_2(I_4(i_1))$ *and* $I_3(i_1) = g(I_4(i_1))$. □

Lemma 2 implies that $E[1], \ldots, E[n_1]$ are found in some of the following tuples (i_2, i_3, i_4) in R_{n_1}: either in tuples of the form $(i_2, i_3^H(i_2), i_4^H(i_2))$, for $i_2 = d_2(1)+1, \ldots, d_2(n_1)$, or in tuples of the form $(d_2(i_4), g(i_4), i_4)$, for $i_4 = 1, \ldots, n_1$. Let $\ell(i_1)$ be the total number of these tuples in R_{i_1},

for $i_1 = 1, \ldots, n_1$, and let m' be the total number in R_{n_1}, i.e., $m' = \ell(n_1)$. Since the planes i_2 of $\overline{C}$ are two-dimensional Monge arrays, we have $i_3^H(i_2) \leq g(r_{i_2})$, for $i_2 = d_2(1)+1, \ldots, d_2(n_1)$. We enumerate these m' tuples from 1 to m' beginning with $i_2 = d_2(1)$, and for fixed i_2, we enumerate them in the lexicographical order by their 3-th through 4-th indices. If (i_2, i_3, i_4) is a given tuple with number p, $1 \leq p \leq m'$, we denote its indices by $i_2(p) = i_2$, $i_3(p) = i_3$ and $i_4(p) = i_4$. Therefore

$$E[i_1] = \min_{1 \leq p \leq \ell(i_1)} \Big\{ w^1(i_1, i_2(p)) + w^2(i_2(p), i_3(p)) + w^3(i_3(p), i_4(p)) + E[i_4(p) - 1] \Big\}, \quad i_1 = 1, \ldots, n_1.$$

Let $A' = \{a'(i_1, p)\}$ be the $n_1 \times m'$ array with entries defined as

$$a'(i_1, p) = w^1(i_1, i_2(p)) + w^2(i_2(p), i_3(p)) + w^3(i_3(p), i_4(p)) + E[i_4(p) - 1],$$

for $1 \leq p \leq \ell(i_1)$. It is clear that

$$E[i_1] = \min_{1 \leq p \leq \ell(i_1)} \{a'(i_1, p)\}, \quad i_1 = 1, \ldots, n_1.$$

Also, we can verify the following property.

Lemma 3 *The $n_1 \times m'$ array A' is Monge.* □

Finally, we are in a position to solve recurrence (4). First, we solve the plane minima problem for the array D. Since the array G becomes available, we can compute the minimum entry in each row of G. Then, we compute on-line the minimum $E[i_1]$ in each row of A' by interleaving its computations with the computation of the minimum $H[i_2]$ in each plane of H. Our algorithm involves communication between two recursive processes of searching minima over the arrays H and A'. Both processes are active and intermediate values are communicated between them. A similar technique is used in [7, 5].

In step i_1, the values $E[1], \ldots, E[i_1 - 1]$, and $H(i_2)$, for $i_2 = d_2(1)+1, \ldots, d_2(i_1 - 1)$, have been computed. Hence, the planes of $\overline{C}'$ and H corresponding to i_4, for $i_4 \leq i_1$, become available. On the one hand, if $d_2(i_1 - 1) < d_2(i_1)$, we can obtain $i_3^H(i_2)$ and $i_4^H(i_2)$, for $i_2 = d_2(i_1 - 1)+1, \ldots, d_2(i_1)$, by applying the MINIMA algorithm over H. Then the first $\ell(i_1)$ columns of A' are defined. On the other hand, if $d_2(i_1 - 1) = d_2(i_1)$, the first $\ell(i_1)$ columns of A' are also available, because we have computed $g(i_1)$ previously. Therefore, the i_1-th row of A' is defined and we can activate the LARSCH algorithm over A' to obtain $E[i_1]$. The pseudocode in Figure 2 shows a way of making these computations. We shall call GMINIMA the algorithm described in this section.

Algorithm GMINIMA
begin
 comment: A, $\overline{C}$, D, H, G and A' defined above.
 comment: $E[0]$ is given.
 activate AGPARK over D to report $d_2(i_1)$, for $i_1 = 1, \ldots, n_1$.
 comment: r_{i_2}, for $i_2 = d_2(1), \ldots, d_2(n_1)$, are available.
 activate SMAWK over G to report $g(i_4)$, for $i_4 = 1, \ldots, n_1$.
 inicialize $p = 0$; $d_2(0) = d_2(1)$;
 for i_1 **from** 1 **to** n_1 **do**
  comment: Plane i_4 of $\overline{C}$, is available, for $i_4 = 1, \ldots, i_1$.
  if $d_2(i_1 - 1) < d_2(i_1)$ **then**
   comment: Plane i_2 of H is available, for $i_2 = d_2(i_1 - 1) + 1, \ldots, d_2(i_1)$.
   for i_2 **from** $d_2(i_1 - 1) + 1$ **to** $d_2(i_1)$ **do**
    activate MINIMA over H to report $(i_3^H(i_2), i_4^H(i_2))$.
    $p = p + 1$;
    $i_2(p) = i_2$; $i_3(p) = i_3^H(i_2)$; $i_4(p) = i_4^H(i_2)$;
   end for
  end if
 $p = p + 1$;
 $i_2(p) = d_2(i_1)$; $i_3(p) = g(i_1)$, $i_4(p) = i_1$; $\ell(i_1) = p$;
 comment: The row i_1 of A' is available.
 comment: The first $\ell(i_1)$ columns of A' are available.
 activate LARSCH over A' to compute $E[i_1]$.
 end for
end

Figure 2: The GMINIMA algorithm.

Theorem 1 *The GMINIMA algorithm runs in $O(n_1 + (n_2 + n_3) \log n_1)$ time and requires linear space.*

Proof. The AGPARK algorithm is applied over D to obtain $d_2(i_1)$, for $i_1 = 1, \ldots, n_1$. Then, the array G becomes available and the SMAWK algorithm can be applied over G to obtain $g(i_4)$, for all $i_4 = 1, \ldots, n_1$. D is an $n_1 \times n_2 \times n_3$ Monge cycle-decomposable array and the plane minima problem for D can be solved in $O((n_2 + n_3) \log n_1)$ time and $\Theta(n_1 + n_2 + n_3)$ space. G is an $n_1 \times n_3$ Monge array and the row minima can be found in $\Theta(n_1 + n_3)$. The MINIMA algorithm is activated over H and the LARSCH algorithm is activated over A' by interleaving the computations in both arrays. The size of H is at most $n_2 \times n_3 \times n_1$. Hence, the plane minima problem for H can be solved in $\Theta(n_1 + n_2 + n_3)$. A' is an $n_1 \times m'$ Monge array with $m' = n_1 + i_2(n_1) - i_2(1) \leq n_1 + n_2 - 1$. Thus, we can compute $E[1], \ldots, E[n_1]$ in $\Theta(n_1 + n_2)$. As each entry of D, G, H and A' can be computed in constant time, the computational complexity of the algorithm is proved. □

From the discussion above we therefore conclude the following.

Corollary 1 *Dynamic programming recurrence (4) can be computed in* $O(n_1 + (n_2 + n_3) \log n_1)$ *time and* $\Theta(n_1 + n_2 + n_3)$ *space.* □

3 Concluding remarks

We have presented an $O(n_1 + (n_2 + n_3) \log n_1)$ time algorithm for computing the dynamic programming recurrence (4). We point out that the complexity would become $\Theta(n_1 + n_2 + n_3)$ if a linear time algorithm was proved for off-line searching of minima in d-dimensional Monge cycle-decomposable arrays. However, it is not clear how to obtain an improvement of the algorithm described by Aggarwal and Park [2].

Generalizations of the on-line searching technique described in this paper allow us compute recurrence (3) when $d > 4$, but there are many other modifications of this recurrence that require further research.

References

[1] A. Aggarwal, M.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.K. Park. Notes on searching in multidimensional monotone arrays. *Proceedings of the 29th IEEE Symp. on Foundations of Computer Science*, October 1988, 497-512.

[3] A. Aggarwal and J.K. Park. Improved algorithms for economic lot size problems. *Operation Research Society of America* **41** (1993), 549-571.

[4] R.E. Burkard, B. Klinz and R. Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics* **70** (1996), 95-161.

[5] A. García, P. Jodrá and J. Tejel. An efficient algorithm for on-line searching of minima in Monge path-decomposable tridimensional arrays. *Information Processing Letters* **68** (1998), 3-9.

[6] A. García, P. Jodrá and J. Tejel. Un algoritmo óptimo para el cálculo del circuito de latencia mínima en un grafo camino con pesos. *XXV Congreso Nacional de Estadística e Investigación Operativa*, Abril 2000, 755-756.

[7] L.L. Larmore and B. Schieber. On-line dynamic programming with applications to the prediction of RNA secondary structure. *Journal of Algorithms* **12** (1991), 490-515.