Notes on Searching in Multidimensional Monotone Arrays

(preliminary version)

Alok Aggarwal* James Park†

Abstract

A two-dimensional array $A = \{a_{i,j}\}$ is called *monotone* if the maximum entry in its i-th row lies below or to the right of the maximum entry in its $(i-1)$-st row. (If a row has several maxima then we take the leftmost one.) An array A is called *totally monotone* if every 2×2 subarray (i.e., every 2×2 minor) is monotone. Totally monotone arrays were introduced by Aggarwal et al. [AKMSW87], who showed that several problems in computational geometry could be reduced to the problem of finding row maxima in totally monotone arrays. In this paper, we generalize the notion of two-dimensional totally monotone arrays to multidimensional arrays and exhibit a wide variety of problems involving computational geometry, dynamic programming, VLSI river routing, and finding certain kinds of shortest paths that can be solved efficiently by finding maxima in totally monotone arrays.

1 Introduction

1.1 Motivation

A two-dimensional array $A = \{a_{i,j}\}$ is called *monotone* if the maximum value in its i-th row lies below or to the right of the maximum value in its $(i-1)$-st row. (If a row has several maxima then we take the leftmost one.) An array A is called *totally monotone* if every 2×2 subarray (i.e., every 2×2 minor) is monotone. Now, if an oracle has an $n \times m$ totally monotone array A, then Aggarwal et al. [AKMSW87] showed that the row maxima of A can be found by asking for $O(n+m)$ entries from the oracle.

Although the question of finding the row maxima in a two-dimensional totally monotone array may seem rather odd at first glance, [AKMSW87] shows that several problems in computational geometry can be reduced to one or more instances of this problem. The following example illustrates one application (borrowed from [AKMSW87]) of searching in totally monotone arrays.

Suppose we are given a convex polygon and that we divide it into two convex chains P and Q containing n and m vertices, respectively, by removing two edges. Let $p_1, \ldots, p_n$ be the vertices of P in counterclockwise order and $q_1, \ldots, q_m$ be the vertices of Q in counterclockwise order. Then for $1 \leq i < k \leq n$ and $1 \leq j < l \leq m$, we observe that the sum of the diagonals of the convex quadrilateral formed by p_i, p_k, q_j, and q_l is greater than the sum of the opposite sides, i.e.,

$$\mathrm{d}(p_i, q_j) + \mathrm{d}(p_k, q_l) \geq \mathrm{d}(p_i, q_l) + \mathrm{d}(p_k, q_j). \quad (1)$$

Thus, if we imagine an $n \times m$ array A that contains the Euclidean distance from vertex $p_i \in P$ to vertex $q_j \in Q$ in $a_{i,j}$, then this array is totally monotone. Also, since any entry of this array can be computed in constant time (it being the Euclidean distance between two points), asking for an entry from the oracle simply corresponds to evaluating the distance between the corresponding points. Hence, by using [AKMSW87], we can find the farthest vertex in Q for every vertex in P by evaluating only $O(n+m)$ distances. *In fact, [AKMSW87] showed that the time required in addition to that for evaluating these $O(n+m)$ distances is also linear in $n+m$.* This implies that the farthest neighbor problem for convex chains can be solved in linear time and solves an open problem in [To83].

1.2 Previous Results

Since the monotone property of a 2×2 minor is closely related to the quadrangle inequality (namely, the sum of the diagonals is greater than the sum of opposite sides, in any convex quadrilateral), this combinatorial formulation turns out to be extremely useful. The

*IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598. This work was done while the author was visiting MIT during the spring of 1988.

†Laboratory for Computer Science, Massachusetts of Institute of Technology, Cambridge, MA 02139. Supported in part by the Defense Advanced Research Projects Agency under Contract N00014-87-K-825, the Office of Naval Research under Contract N00014-86-K-0593, and an NSF Graduate Fellowship.

0272-5428/88/0000/0497$01.00 © 1988 IEEE

utility of quadrangle inequalities was first observed by G. Monge in 1781 [Mo81]. Monge remarked that if unit quantities have to be transported from locations X and Y in the plane to locations Z and W (not necessarily respectively) in such a way as to minimize the total distance travelled, then the paths followed in transporting these quantities must not intersect. In 1961, A.J. Hoffman [Ho61] elaborated upon this idea by calling an $n \times m$ array $C = \{c_{i,j}\}$ a *Monge* array if $c_{i,j} + c_{i+1,j+1} \leq c_{i,j+1} + c_{i+1,j}$ for $1 \leq i < n$ and $1 \leq j < m$ and by providing a greedy algorithm for the transportation problem when the cost array $C = \{c_{i,j}\}$ is a Monge array. Thus, Monge and Hoffman's observations imply a greedy algorithm for the transportation problem when the sources lie on a line parallel to a line on which the sinks are located.

Observation 1 *If $C = \{c_{i,j}\}$ is a Monge array, then for $1 \leq i < k \leq n$ and $1 \leq j < l \leq m$, $c_{i,j} + c_{k,l} \leq c_{i,l} + c_{k,j}$.*

We follow Hoffman's terminology and call $c_{i,j} + c_{k,l} \leq c_{i,l} + c_{k,j}$ the *Monge condition* and $c_{i,j} + c_{k,l} \geq c_{i,l} + c_{k,j}$ the *inverse Monge condition*. Also, we will call an array Monge if it follows either the Monge condition or the inverse Monge condition.

Observe that every Monge array is totally monotone, but there are totally monotone arrays that are not Monge. Although the notion of totally monotone arrays is somewhat weaker than that of Monge arrays, it turns out that all the applications given in this paper, as well as those given in [AKMSW87,Wi88], actually obey the Monge or the inverse Monge condition. However, we usually use only total monotonicity and not the Monge condition to obtain our results.

Previous applications of the techniques introduced by [AKMSW87] for searching in totally monotone arrays include the following. [AKMSW87] showed that some channel routing problems can be solved in linear time using these techniques, thereby improving previous $O(n \lg n)$ time algorithms. Aggarwal and Suri [AS87] used array-searching to find the largest empty rectangle in $O(n \lg^2 n)$ time. Wilber [Wi88] applied array-searching techniques to an instance of dynamic programming and showed that the concave least-weight subsequence problem can be solved in linear time; this improved the result of Hirschberg and Larmore [HL85]. Finally, Klawe and Kleitman [KK88] used array-searching to improve a previous result of [AK87] in computational geometry.

Observation 2 *We defined an array $A = \{a_{ij}\}$ to be totally monotone if for all $i < k$ and $j < l$,*

$$a_{i,j} < a_{i,l} \quad \textit{implies} \quad a_{k,j} < a_{k,l} \qquad (2)$$

and we were interested in finding the maxima *in each row. This problem is equivalent to that of finding the* minima *in each row of a array whose 2×2 minors satisfy*

$$a_{i,j} > a_{i,l} \quad \textit{implies} \quad a_{k,j} > a_{k,l}. \qquad (3)$$

In fact, these problems are dual of each other and one can be transformed into the other by simply negating all entries of the array. Similarly, the problem of finding minima in each row when (2) holds and that of finding maxima in each row when (3) holds are dual of each other.

1.3 Main Results of this Paper

This paper presents a framework that allows us to solve efficiently a class of problems that obey either the Monge condition or the inverse Monge condition. This paper derives its inspiration from [AKMSW87] and [Ya80]; it generalizes and incorporates several results provided in these papers. (The relation of [AKMSW87] to this paper will be observed throughout, but that of [Ya80] is more implicit. We assume total familiarity of the reader with [AKMSW87] and [Ya80].)

Section 2 introduces the notion of multidimensional monotone arrays and provides sequential algorithms for searching in such arrays; Section 3 provides parallel array-searching algorithms. We use the problem of finding a maximum perimeter d-gon inscribed in a given convex n-gon as a prototype example. For this problem, we achieve the time bound of [AKMSW87] in the sequential case and improve bounds given in [ACGOY88] in the parallel case.

Section 4 presents the first set of applications of this paradigm to finding minimum area and minimum perimeter circumscribing polygons. For the minimum area circumscribing d-gon problem, we improve the best previous result of [AKMSW87] by a factor of $O((n \lg d)/(d + \lg n))$ and for the minimum perimeter circumscribing triangle problem, we improve the result of [De87] by an $O(n^2/\lg n)$ factor.

Section 5 presents efficient algorithms for dynamic programming problems that use the Monge condition or the inverse Monge condition. We give an $O(n^2)$ time algorithm for Frances Yao's [Ya80] dynamic programming problem using this paradigm. Although our algorithm is no better than Yao's in terms of asymptotic complexity (and probably worse in practice), it provides insight into other dynamic programming problems that use Monge conditions. For example, Section 5 also shows that the time complexity of a particular dynamic programming problem related to

biology can be improved from $O(n^2 \lg^2 n)$ [EGG88] to $O(n^2 \lg n)$.

Section 6 presents efficient sequential and parallel algorithms for river routing in VLSI. We point out that the results of [AKMSW87] can be used to solve the offset range problem in linear time under very weak assumptions regarding the routing rules for wires. This generalizes a recent result of Siegel [Si88], who was able to obtain a linear time algorithm for only a very restricted class of wiring rules. In the realm of parallel computation, Chang and JáJá [CJ88] have studied the separation problem and the optimal offset problem for rectilinear wires. Section 6 shows that the processor bounds for their algorithms can be improved by an $O(\lg n)$ factor. By incorporating the results of Subsection 3.1, we also generalize their results so that wires need not be rectilinear and the assumptions associated with the routing of wires are weaker.

Section 7 investigates the computation of shortest paths in certain directed, acyclic planar graphs in the realm of parallel computation. Such graphs arise in several contexts, including tomography and medical therapy [FKU77], optimal surface reconstruction from planar contours [FKU77], string editing and largest common subsequence problems [KF80,AAL87,Ma88], and in finding the maximal layers of n planar points [Ko86]. Subsection 7.1 obtains polylogarithmic time algorithms for the problem of surface reconstruction from planar contours; our processor-time product is $O(\lg n / \lg m)$ away from the best known sequential time for this problem [FKU77]. For string editing and the largest common subsequence problems, Subsection 7.1 improves either the time bound or the processor-time product of the algorithms given by [AAL87] and [Ma88]; in particular, we settle the open problem raised in [AAL87] by showing that factors of $O(\lg n)$ and $O(\lg n / \lg \lg n)$ can be saved in the time complexity without increasing the processor complexity[1]. Finally, for the maximal layers problem, Subsection 7.1 improves the time complexity of the known algorithm by a factor of $\lg n^2$.

[1] Just before sending this preliminary version to the publishers, we heard that Apostolico, Atallah, Larmore, and McFaddin have also settled the open problem raised in [AAL87].

[2] We are indebted to S. Rao Kosaraju [Ko86] for bringing the algorithm mentioned in Subsection 7.1 to our attention.

2 Multidimensional Arrays

2.1 Basic Definitions and Algorithms

For $d \geq 3$, let $A = \{a_{i_1,i_2,\ldots,i_d}\}$ be an $n_1 \times n_2 \times \cdots \times n_d$ array. Let $a_{i_1,i_2(i_1),\ldots,i_d(i_1)}$ be the maximum entry in the $(d-1)$-dimensional subarray corresponding to a particular value of i_1, i.e.,

$$a_{i_1,i_2(i_1),\ldots,i_d(i_1)} = \min_{i_2,\ldots,i_d} a_{i_1,i_2,\ldots,i_d}.$$

If the plane contains multiple maxima, we chose the first of the maxima ordered lexicographically by their second through d-th coordinates.

We will say that A is *monotone* if $i_1 > i_1'$ implies $i_k(i_1) \geq i_k(i_1')$ for all k between 2 and d. We will say that A is *totally monotone* if

1. every $2 \times 2 \times \cdots \times 2$ d-dimensional subarray of A is monotone, and
2. every $(d-1)$-dimensional plane of A, corresponding to a fixed value of its first coordinate, is totally monotone.

Observe that every totally monotone array is monotone and that every lower-dimensional subarray of a totally monotone array is totally monotone.

Two problems that arise in connection with multidimensional monotone arrays are the *plane maxima* problem, in which we wish to compute the array's maximum entry for each value of its first index i_1, and the *tube maxima* problem, in which we wish to compute the array's maximum entry for each pair (i_1, i_2) of first and second indices. Note that the tube maxima problem for a d-dimensional array can always be solved by solving the plane maxima problem for each of the array's $(d-1)$-dimensional planes (but this is not always the best approach — see Section 3).

Proposition 3 *For $d \geq 2$, the plane maxima of an $n_1 \times n_2 \times \cdots \times n_d$ d-dimensional totally monotone array A can be computed in $O((n_d + n_{d-1}) \prod_{k=1}^{d-2} \lg n_k)$ time.*

Note that the only place we take advantage of A's totally monotonicity is in solving certain two-dimensional subarrays — we can do almost as well (i.e., solve for the plane maxima in $O(n_d \prod_{k=2}^{d-1} \lg n_k)$ time) using only monotonicity.

2.2 An Example

To make the notions of monotone and totally monotone multidimensional arrays a little clearer, let us consider a concrete example. The maximum perimeter inscribed triangle problem is defined as follows:

given an n-vertex convex polygon P, find a triangle Q contained in P with maximum perimeter. It is easy to show that Q's vertices must be vertices of P. Thus, if $p_1, \ldots, p_n$ are the vertices of P and $\text{per}(i,j,k)$ is the perimeter of the triangle corresponding to p_i, p_j, and p_k, we want to find the i, j, and k maximizing $\text{per}(i,j,k)$.

To this end, we consider the three-dimensional array $A = \{a_{ijk}\}$ where

$$a_{ijk} = \begin{cases} \text{per}(i,j,k) & \text{if } i < j < k \\ -\infty & \text{otherwise.} \end{cases}$$

If we can find the maximum entry in A, we can solve the maximum perimeter inscribed triangle problem for P. *Note that we do not explicitly compute all of the entries in A. Rather, every time our array algorithm needs a particular entry, we calculate (in constant time) the perimeter of the corresponding triangle.* Also, it is easy to verify:

Observation 4 *The array A defined above is totally monotone.*

Since the array A is totally monotone and since we can compute any entry in constant time, we can find the array's largest entry, corresponding to the maximum perimeter inscribed triangle, in $O(n \lg n)$ time using Proposition 3. This result equals the result obtained by Boyce et al. in [BDDG85]; moreover, it represents a simpler solution to the problem, as Boyce et al. require a number of additional geometric properties of inscribed triangles that complicate their proof.

Similarly, it can be readily checked that the maximum perimeter inscribed d-gon problem reduces to finding the maximum entry in a d-dimensional totally monotone array. Since each entry in this array can be computed in $O(d)$ time, this yields an $O(dn \lg^{d-2} n)$ time algorithm. It remains open whether one can do better than $O(n \lg n)$ time for the inscribed triangle problem, but the inscribed d-gon problem can be solved more efficiently if we take advantage of some additional structure of the corresponding d-dimensional array; this additional structure is discussed in the next two subsections.

2.3 Monge-Composite Arrays

An important subclass of multidimensional totally monotone arrays is that of *Monge-composite* arrays. A d-dimensional array $A = \{a_{i_1,\ldots,i_d}\}$ is Monge-composite if it can be expressed as a sum of two-dimensional Monge arrays, all satisfying the Monge condition or all satisfying the inverse Monge condition. More precisely, each of its entries satisfies

$$a_{i_1,\ldots,i_d} = \sum_{k \neq l} w^{(k,l)}_{i_k,i_l},$$

where for all k and l, the $n_k \times n_l$ array $W^{(k,l)} = \{w^{(k,l)}_{i_k,i_l}\}$ is a Monge array.

Proposition 5 *Every Monge-composite array A is totally monotone.*

Two special cases of Monge-composite arrays are *path- and cycle-decomposable* arrays. An array $A = \{a_{i_1,\ldots,i_d}\}$ is path-decomposable if each of its entries satisfies

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

where the w's are entries from two-dimensional Monge arrays as before. A is cycle-decomposable if each of its entries satisfies

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

Theorem 6 *The plane maxima of an $n_1 \times \cdots \times n_d$ d-dimensional path-decomposable array A can be computed in $O(\sum_{k=1}^{d} n_k)$ time.*

Proof The proof is by induction on d. The base case of $d = 2$ is trivial. For $d \geq 2$, we assume by induction that the theorem holds for all lower-dimensional path-decomposable arrays. Since $A = \{a_{i_1,\ldots,i_d}\}$ is path-decomposable, we can write

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

where the w's are entries from two-dimensional Monge arrays. Now consider the $n_2 \times \cdots \times n_d$ $(d-1)$-dimensional array $B = \{b_{i_2,\ldots,i_d}\}$ where

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

By induction, we can find the plane maxima of B in $O(\sum_{k=2}^{d} n_k)$ time. Since the maximum entry in the plane of A corresponding to a particular value of the first index i_1 is just

$$\min_{i_2} \left\{ w^{(1,2)}_{i_1,i_2} + \min_{i_3,\ldots,i_d} \left\{ w^{(2,3)}_{i_2,i_3} + \cdots + w^{(d-1,d)}_{i_{d-1},i_d} \right\} \right\},$$

we need only find the row maxima in the array formed by adding the vector of B's plane maxima into $W^{(1,2)}$. Since this array is just a two-dimensional Monge array, we can find its row maxima in an additional $O(n_1 + n_2)$ time, which yields the desired result. ■

Theorem 7 *The plane maxima of an $n_1 \times \cdots n_d$ d-dimensional cycle-decomposable array A can be computed in $O((\sum_{k=2}^{d} n_k) \lg n_1)$ time.*

Proof It is easily verified that each plane of a cycle-decomposable array is path-decomposable. This means we can compute a particular plane's maximum entry in $O(\sum_{k=2}^{d} n_k)$ time. Thus, using our standard divide-and-conquer approach, we can compute all plane maxima in $O((\sum_{k=2}^{d} n_k) \lg n_1)$ time. ■

2.4 More Examples

Returning to the maximum perimeter inscribed d-gon problem, note that the d-dimensional array corresponding to this problem is cycle-decomposable. Thus, we can apply Theorem 7 and obtain an $O(dn \lg n)$ time algorithm for the problem. It is possible to reduce this time bound to $O(dn+n \lg n)$ using one additional trick, which we will present in the final version of this paper.

As Boyce et al. [BDDG85] note, the same approach can also be used to solve the maximum area inscribed d-gon problem in $O(dn + n \lg n)$ time. Although the corresponding d-dimensional array *is not quite totally monotone*, it can be easily shown that we only need to consider certain subarrays of this array, and these subarrays are totally monotone.

Another application of array-searching is to the following problem: given a convex n-gon P and a distinguished vertex v of P, find the longest d-link non-intersecting path inside P from v to every other vertex of P. Theorem 6 can be used to obtain such paths in $O(dn)$ time. (This result is implicit in [AKMSW87].)

Note that not all applications involve path- or cycle-decomposable arrays; consider, for example, the problem of finding a maximum weight d-clique of vertices from a convex n-gon, where the weight of a clique is defined as the sum over all pairs of vertices in the clique of the distances between the two vertices. An instance of this problem corresponds to a d-dimensional Monge-composite array, each of whose entries may be computed in $O(d^2)$ time, and thus may be solved in $O(d^2 n \lg^{d-2} n)$ time. And, the d-dimensional array we consider in Subsection 4.2 in connection with the minimum perimeter circumscribing triangle problem is not even Monge-composite.

3 Array-Searching in Parallel

In this section, we provide resource bounds for array-searching in three models of parallel computation: the CREW-PRAM model, the CRCW-PRAM model, and Valiant's comparison model [LLMPW87]. For the sake of brevity, we ignore the issue of processor allocation in our discussion of PRAM algorithms; details regarding processor allocation will be provided in the final version of this paper. Also, in this version we consider only square arrays; we will give results for $n \times m$ arrays, $n \neq m$, in the final version.

3.1 Two-Dimensional Arrays

Suppose we are given an $n \times n$ totally monotone array A.

Proposition 8 *Given the maximum entry in every r-th row of A, $1 \leq r \leq n$, we can compute the remaining row maxima in $O(r + \lg(n/r))$ time on a CREW-PRAM or in $O(r + \lg\lg(n/r))$ time on a CRCW-PRAM using n/r processors.*

Sketch of Proof The row maxima provided to us partition the array into r regions (containing a total of $O(rn)$ entries) in which the remaining row maxima must lie. These regions can be further partitioned into n/r $r \times O(r)$ subarrays. Since each of these subarrays is totally monotone, we can assign a single processor to each subarray and apply the sequential array-searching algorithm given in [AKMSW87] to obtain all their row maxima in $O(r)$ time. We can then do all the necessary combining of subarray maxima from the same region in $O(\lg(n/r))$ time on a CREW-PRAM and in $O(\lg\lg(n/r))$ time on a CRCW-PRAM. ■

By applying Proposition 8 repeatedly, we obtain the following:

Corollary 9 *For $1 \leq r \leq n$ we can compute the row maxima of A in $O((\lg n/\lg r)(r + \lg(n/r)))$ time on a CREW-PRAM or in $O((\lg n/\lg r)(r + \lg\lg(n/r)))$ time on a CRCW-PRAM using n/r processors.*

For $r = n^{\epsilon}$, where $\epsilon > 0$ is a constant, this yields an optimal processor-time product for both CREW- and CRCW-PRAMs. For $r = \lg n$, we obtain an $O(\lg^2 n/\lg\lg n)$ time, $n/\lg n$ processor CREW-PRAM algorithm for computing A's row maxima. For $r = \lg\lg n$, we obtain an $O(\lg n \lg\lg n/\lg\lg\lg n)$ time, $n/\lg n \lg n$ processor CRCW-PRAM algorithm.

We can obtain even better time bounds (but worse processor-time complexities) using a divide-and-conquer approach; for such an approach we need the following:

Observation 10 *Given an $n \times k$ totally monotone array B, $k \leq n$, suppose we know the maximum in every $\lfloor n/k \rfloor$-th row of B. Then $O(n + k)$ operations*

suffice for computing the remaining row maxima of B. Furthermore, these $O(n+k)$ operations can be performed in $O(\lg(n+k))$ time using $(n+k)/\lg(n+k)$ processors on a CREW-PRAM or in $O(\lg\lg(n+k))$ time using $(n+k)/\lg\lg(n+k)$ processors on a CRCW-PRAM.

Proposition 11 *The row maxima of A can be computed in $O(\lg n \lg\lg n)$ time using $n/\lg\lg n$ processors on a CREW-PRAM.*

Proof Consider a $\sqrt{n} \times n$ array B formed by taking every $\sqrt{n}$-th row of A. Partition this array into $\sqrt{n}$ subarrays such that the i-th subarray S_i contains columns numbered $i\sqrt{n}+1$ through $(i+1)\sqrt{n}$. In Step 1, recursively compute the row maxima in each S_i (in parallel). This yields $\sqrt{n}$ candidates for the maximum value of the i-th row of B. In Step 2, assign enough processors to each row of B to compute the maximum of the row's $\sqrt{n}$ candidates in $O(\lg n)$ time. This yields the maximum value in every $\sqrt{n}$-th row of A. For $0 \le i < \sqrt{n}$, let k_i denote the index of the column in which the maximum value of row $i\sqrt{n}$ lies. Then the maximum values in rows $i\sqrt{n}+1$ through $(i+1)\sqrt{n}-1$ lie in columns k_i through k_{i+1}. Let $k_{\sqrt{n}} = n$ and for $0 \le i < \sqrt{n}$, let a_i be the maximum integer such that $k_{i+1} - k_i = a_i\sqrt{n} + b_i$, $0 \le b_i < \sqrt{n}$. For $0 \le j < a_i$, let $S_{j,i}$ be the subarray of A that contains rows $i\sqrt{n}+1$ through $(i+1)\sqrt{n}-1$ and columns $k_i + j\sqrt{n}+1$ through $k_i + (j+1)\sqrt{n}$. Let R_i be the subarray formed by rows $i\sqrt{n}+1$ through $(i+1)\sqrt{n}-1$ and columns $k_i + a_i\sqrt{n}+1$ through $k_i + a_i\sqrt{n} + b_i$ and let R'_i be the $b_i \times b_i$ subarray formed by taking every b_i-th row of R_i. In Step 3, recursively solve (in parallel) the row maxima problem for $S_{j,i}$ and R'_i for $0 \le j < a_i$ and $0 \le i < \sqrt{n}$. In Step 4, use Observation 10 to solve the row maxima problem in T_i for $0 \le i < \sqrt{n}$ in $O(\lg(\sqrt{n}+b_i)) = O(\lg n)$ time. In Step 5, compute the maximum value in each of rows $i\sqrt{n}+1$ through $(i+1)\sqrt{n}-1$ by taking the maximum of the values obtained for the row by solving $S_{0,i}, S_{1,i}, \ldots, S_{a_i-1,i}$, and R_i and the row's k_i-th column entry. As the total number of entries that are candidates for the maximum value in any row of A is bounded from above by n, Step 5 takes $O(\lg n)$ time and solves the row maxima problem for A.

To analyze the processor and time bounds, note that steps 2, 4, and 5 take $O(\lg n)$ time, whereas steps 1 and 3 take at most $T(\sqrt{n})$ time, where $T(n)$ denotes the time complexity of solving A. Consequently, $T(n) \le 2T(\sqrt{n}) + O(\lg n)$, which, together with $T(1) = O(1)$, yields the required time bound. Also, it can be easily verified that if $OP(n)$ denotes the number of operations required by this algorithm, then

$$OP(n) = O(n) + \sqrt{n}\,OP(\sqrt{n}) + \sum_{i=0}^{\sqrt{n}-1} \{a_i\,OP(\sqrt{n}) + OP(b_i)\},$$

which, together with $OP(1) = O(1)$, yields $OP(n) = O(n \lg n)$. Consequently, by Brent's theorem [Br74], the number of processors required is $O(n/\lg\lg n)$. ■

Using a proof similar to that given for Proposition 11, we can show:

Proposition 12 *In the CRCW-PRAM model, n processors suffice to compute the row maxima of A in $O(\lg n)$ time.*

We can achieve the same result in Valiant's comparison model, using a relatively simple divide-and-conquer approach:

Observation 13 *$\lg n$ rounds of n comparisons apiece suffice to compute A's row maxima.*

A superlinear number of processors allows us to obtain optimal time algorithms.

Proposition 14 *In the CREW-PRAM model, we can compute the row maxima of A in $O(\lg n)$ time using $n \lg n$ processors.*

Sketch of Proof The proof is obtained by modifying Observation 13 so that it can be applied to CREW-PRAMs and by using *pipelining* suitably. The pipelining technique used here is reminiscent of Cole's merge sort [Co86]. ■

Observation 15 *In the CRCW-PRAM model, we can compute the row maxima of A in $O((1/\epsilon)\lg(1/\epsilon))$ time using $n^{1+\epsilon}$ processors for any $\epsilon \ge 1/\lg n$.*

Our results for computing row maxima in two-dimensional totally monotone arrays are summarized in Table 1.

Returning to our prototype example of the maximum perimeter inscribed triangle problem, it can be easily seen that Boyce et al.'s [BDDG85] technique can be modified to solve this problem in $O(\lg^3 n)$ time using n processors on a CREW-PRAM. However, by using Proposition 11, we can solve this problem in $O(\lg^2 n \lg\lg n)$ time on an $n/\lg\lg n$ processor CREW-PRAM and in $O(\lg^2 n)$ time on an n processor CRCW-PRAM. In a similar vein, Boyce et al.'s technique can be modified to solve the maximum

model	*time*	*processors*
Valiant's	$O(\lg n)$	n
CREW	$O(\lg n \lg\lg n)$	$n/\lg\lg n$
	$O(\lg^2 n/\lg\lg n)$	$n/\lg n$
	$O(\lg n)$	$n\lg n$
CRCW	$O(\lg n)$	n
	$O(\lg n \lg\lg n/\lg\lg\lg n)$	$n/\lg\lg n$
	$O((1/\epsilon)\lg(1/\epsilon))$	$n^{1+\epsilon}$

Table 1: Row Maxima Results

perimeter d-gon problem in $O(d\lg^2 n+\lg^3 n)$ time using n processors on a CREW-PRAM. Using Proposition 11, we can solve it in $O((d+\lg n)\lg n\lg\lg n)$ time on an $n/\lg\lg n$ processor CREW-PRAM and in $O((d+\lg n)\lg n)$ time on an n processor CRCW-PRAM. However, for larger values of d (for example, for $d = n^{\epsilon}$, $\epsilon > 0$), this algorithm no longer takes polylogarithmic time, i.e., it is not an NC^+ algorithm [ACGOY88]. Consequently, we next consider the plane and tube maxima problems in a parallel context in order to obtain an NC^+ algorithm.

3.2 Multidimensional Arrays

For multidimensional totally monotone arrays, we consider both the plane maxima and tube maxima problems. We will restrict our attention to three-dimensional arrays for the sake of brevity; our results generalize in a natural manner.

Let A be an $n \times n \times n$ totally monotone three-dimensional array. We can obtain a naive solution to the plane maxima problem by applying our basic divide-and-conquer technique (given in Proposition 3):

Observation 16 *If we can compute the row maxima of an $n \times n$ totally monotone two-dimensional array in t time using p processors in either the CREW-PRAM or the CRCW-PRAM model, we can compute the plane maxima of A in $O(t\lg n)$ time using p processors in the same model.*

It remains open whether the time complexity given by Observation 16 can be improved. For the tube maxima problem, we improve upon the naive algorithm and obtain optimal processor-time products in both the CREW- and CRCW-PRAM models. Proposition 11 can be readily used to obtain:

Observation 17 *In the CREW-PRAM model, we can compute the tube maxima of A in $O(\lg n\lg\lg n)$ time using $n^2/\lg n\lg\lg n$ processors.*

We can reduce the time bounds even further as follows:

Proposition 18 *In the CREW-PRAM model, we can compute the tube maxima of A in $\Theta(\lg n)$ time using $n^2/\lg n$ processors.*

Proof This proof can be obtained using either Proposition 14 or using a proof technique given in the proof of Proposition 11. Although the proof that uses ideas from Proposition 11 can be used to prove Propositions 19 and 20, in this version we provide the proof using Proposition 14 because it is easier. First we show that using m^2 processors, we can compute the tube maxima of an $m \times m \times m$ totally monotone array B in $O(\lg m)$ time. Then we improve the processor bound to obtain the required result.

Consider every $(\lg m)$-th plane corresponding to a fixed value of B's first index. Using Proposition 14, compute in $O(\lg m)$ time the tube maxima in these $m/\lg m$ planes using $m\lg m$ processors per plane (i.e., m^2 total processors). Then, in each plane corresponding to a fixed value of B's second index, we fill in the rest of the plane's column maxima, corresponding to the remaining tube maxima. By Proposition 8, this can be done in $O(\lg m)$ time using $m/\lg m$ processors per plane (i.e., a total of $m^2/\lg m$ processors).

Now consider an $n \times n \times n$ array $A = \{a_{ijk}\}$. For $0 \le \ell < \lg n$, let A_ℓ be the $(n/\lg n) \times (n/\lg n) \times (n/\lg n)$ subarray of A containing entries a_{pqr_ℓ}, where $p = i\lg n$, $q = j\lg n$, and $r_\ell = \ell(n/\lg n) + k$ for $1 \le i,j \le n/\lg n$ and $1 \le k \le \lg n$. By assigning $n^2/\lg^2 n$ processors to each A_ℓ, i.e., by assigning a total of $n^2/\lg n$ processors, we can compute the tube maxima for A_ℓ in $O(\lg n)$ time. Now the maximum corresponding to the $(i\lg n, j\lg n)$-th tube of A is simply the maximum of the corresponding tube maxima in $A_0,\ldots,A_{\lg n-1}$ and since $1 \le i,j \le n/\lg n$, we can compute these maxima in $O(\lg n)$ additional time. Then, in every $(\lg n)$-th plane corresponding to a fixed value of A's second index, we fill in the rest of the plane's column maxima. This can be done in $O(\lg n)$ time using $n/\lg n$ processors per plane (i.e., $n^2/\lg^2 n$ total processors) using Proposition 8. Finally, in each plane corresponding to a fixed value of A's first index, we fill in the rest of the plane's column maxima, corresponding to the remaining tube maxima. This can be done in $O(\lg n)$ time using $n/\lg n$ processors per plane (i.e., $n^2/\lg n$ total processors). ■

Proposition 19 *In the CRCW-PRAM model, we can compute the tube maxima of A in $O((\lg\lg n)^2)$ time using $n^2/(\lg\lg n)^2$ processors.*

model	*time*	*processors*
CREW	$O(\lg n)$	$n^2/\lg n$
CRCW	$O((\lg n \lg n)^2)$	$n^2/(\lg \lg n)^2$
restricted CRCW	$O(\lg \lg n)$	$n^2/\lg \lg n$

Table 2: Tube Maxima Results

Using the results of [FRW88], we can show:

Proposition 20 *If all of A's entries are integers from $\{1, \ldots, n^c\}$, where c is some constant, then a CRCW-PRAM can compute the tube maxima of A in $O(\lg \lg n)$ time using $n^2/\lg \lg n$ processors.*

Our tube maxima results for three-dimensional totally monotone arrays are summarized in Table 2.

The results regarding tube maxima improve upon the time bounds of [AKLMT88] and [AAL87] by factors of $O(\lg n)$ and $O(\lg n/\lg \lg n)$ without any deterioration in the processor time-product.

We now return to our prototype example and show how to obtain fast parallel algorithms for the maximum perimeter inscribed d-gon problem for any value of d. Frances Yao [Ya80] has given a procedure that, for $1 \leq i \leq j \leq n$, computes the longest $(d-1)$-link non-intersecting path in a given n-gon that starts from vertex i and ends at vertex j. This procedure can be easily parallelized and it essentially requires $O(\lg d)$ stages where each stage solves a tube maxima problem for an $n \times n \times n$ array. Furthermore, if we are given the largest $(d-1)$-link path from vertex i to vertex j, for all $i \leq j$, then we can compute the maximum perimeter d-gon by simply adding the edge (i, j) to the path from i to j, for all $i \leq j$, and then taking the largest perimeter d-gon among the $O(n^2)$ d-gons so obtained. Consequently, it is easy to verify:

Proposition 21 *The maximum perimeter inscribed d-gon problem can be solved in $O(\lg k \lg n)$ time using $n^2/\lg n$ processors on a CREW-PRAM and in $O(\lg k(\lg \lg n)^2)$ time using $n^2/(\lg \lg n)^2$ processors on a CRCW-PRAM.*

Note that the processor-time product required by our algorithms is $O(n^2 \lg k)$, which is a factor of $O((n \lg k)/(k + \lg n))$ worse than the best sequential bound given in Section 2.4 or [AKMSW87].

4 Circumscribing Polygons

The first set of applications of the array-searching paradigm is to finding minimum area and minimum perimeter circumscribing polygons. Given an n-vertex convex polygon P and an integer d between 3 and n, we want to find a minimum area or minimum perimeter d-gon Q containing P.

In [BDDG85], Boyce et al. were not able to solve the problem of circumscribing a convex n-gon (or a set of n planar points) with a minimum area d-gon. Aggarwal, Chang, and Yap [ACY85] showed that a minimum area circumscribing d-gon can be found in $O(n^2 \lg d \lg n)$ time, and this is further improved to $O(n^2 \lg d)$ in [AKMSW87]. We extend the techniques of Boyce et al. [BDDG85] in a non-trivial manner to obtain an $O(nd + n \lg n)$ time algorithm for the area minimization problem. We present this algorithm in Subsection 4.1.

As for the perimeter minimization problem, the only known results are for $d = 3$. DePano [De87] shows that the minimum perimeter circumscribing triangle can be computed in $O(n^3)$ time. In Subsection 4.2, we improve this to $O(n \lg n)$ time using a straightforward application of our techniques for searching in three-dimensional totally monotone arrays.

4.1 Area Minimization

In finding a minimum area circumscribing d-gon, we first use the techniques of [BDDG85] to obtain a minimum area *flush* circumscribing d-gon. (A circumscribing polygon is flush with P if all its edges are flush with P, i.e., all of its edges contain edges of P.) We then use this flush d-gon to obtain an arbitrary circumscribing d-gon with optimal area, with the help of a lemma due to DePano [De87].

The techniques given by Boyce et al. [BDDG85] can be extended in a straightforward manner to finding a flush minimum area circumscribing d-gon in $O(dn \lg n + n \lg^2 n)$ time. (This was pointed out in the concluding section of [BDDG85].) Furthermore, the techniques of [AKMSW87] reduce the complexity of this problem to $O(dn + n \lg n)$ time. In [De87], DePano provides the following geometric characterization of a minimum area circumscribing d-gon:

Lemma 22 ([De87]) *Let P be any convex n-gon. For $3 \leq d \leq n$, there exists a minimum area d-gon Q circumscribing P that has a least $d-1$ edges flush with P. Furthermore, if some edge e of Q is not flush with P, then:*

1. *e is* balanced, *i.e., its midpoint lies on P, and*
2. *e's two neighboring edges (those edges of Q sharing an endpoint with e) intersect on the same side of e as P lies.*

This allows us to relate minimum area flush circumscribing d-gons and minimum area arbitrary circumscribing d-gons:

Observation 23 *Some minimum area circumscribing d-gon* interleaves *every minimum area flush circumscribing d-gon, i.e., the contact points (edges and vertices of P) of the optimal arbitrary circumscribing d-gon must alternate with contact points (edges only) of the optimal flush circumscribing d-gon.*

Thus, once we have computed an optimal flush circumscribing d-gon, we have d intervals of edges, overlapping only at their endpoints, such that some optimal circumscribing d-gon has a contact point in each of these intervals. Furthermore, DePano's lemma tells us that there exists an optimal circumscribing d-gon Q with at most one non-flush edge, and if this non-flush edge exists, then its neighbors intersect on the same side of the non-flush edge as P. Because of this, it is easy to verify:

Observation 24 *There are at most three intervals in which the non-flush edge can lie.*

Let the d intervals of edges be denoted by $I_1, I_2, \ldots, I_d$ in counterclockwise order around P. Let $v_1, \ldots, v_n$ be the vertices of P in counterclockwise order and let e_i, $1 \leq i \leq n$, be the edge of P connecting v_i and $v_{(i+1) \bmod n}$. We define an *optimal flush* $(d-1)$*-chain* $V_{i,j}$ to be the convex polygonal chain satisfying the following conditions: (i) the chain lies entirely outside P, (ii) its first edge is flush with e_i, its $(d-1)$-st edge is flush with e_j, and successive edges are flush with edges of P in clockwise order, and (iii) among all possible chains satisfying (i) and (ii), the area enclosed between this chain and P is minimum. Finally, if $e_i \in I_{k-1}$ and $e_{j+1}, \ldots, e_{j+\ell}$ denote the edges in I_{k+1}, define an *optimal 3-chain* $V'_{i,j+s}$ to be the convex polygonal chain satisfying the following conditions: (i) the chain lies entirely outside P, (ii) its first and third edges are flush with e_i and e_{j+s}, respectively, and its second edge touches some vertex (or is flush with an edge) of I_k, and (iii) among all possible chains satisfying (i) and (ii), the area enclosed between this chain and P is minimum.

Now, we can check the possibility of the nonflush edge lying in interval I_k as follows. Let e_i be the middle edge in I_{k-1}. We need to (a) find the optimal flush $(d-1)$-chain $V_{i,j+s}$ for $1 \leq s \leq \ell$, and (b) find the optimal 3-chain $V'_{i,j+s}$ for $1 \leq s \leq \ell$. Again, by suitably modifying the technique given by Boyce et al. (and improved by [AKMSW87]), we have:

Lemma 25 *$V_{i,j+1}, \ldots, V_{i,j+\ell}$ can all be computed in $O(n)$ time.*

This allows us to accomplish (a); for (b), we need the following lemma:

Lemma 26 *$V'_{i,j+1}, \ldots, V'_{i,j+\ell}$ can all be computed in $O(n)$ time.*

Sketch of Proof For $1 \leq s \leq \ell$, if the second edge of $V'_{i,j+s}$ only touches some vertex in I_k, then because of Lemma 22, this edge must be balanced. This balancing condition provides some additional properties that help us in obtaining $V'_{i,j+1}, \ldots, V'_{i,j+\ell}$ in $O(n)$ time. Similar properties have been used in [OAMB86] and [ACY85]; we will provide the details in the final version of this paper. ■

At this point, Boyce et al.'s basic divide-and-conquer approach can be used to yield:

Theorem 27 *Given a convex n-gon, a minimum area circumscribing d-gon can be computed in $O(nd + n \lg n)$ time.*

4.2 Perimeter Minimization

In obtaining his $O(n^3)$ time algorithm for the minimum perimeter circumscribing triangle problem, DePano [De87] shows that every convex n-gon has a minimum perimeter circumscribing triangle with at least one edge flush. He also proved the following lemma:

Proposition 28 ([De87]) *For any triple (i, j, k), let $T_{i,j,k}$ be the minimum perimeter circumscribing triangle with its first edge $\hat{e}_1$ flush with edge e_i of P, its second edge $\hat{e}_2$ containing vertex v_j of P, and its third edge $\hat{e}_3$ containing vertex v_k of P. Then there exists a point p_2 on $\hat{e}_2$ and P and a point p_3 on $\hat{e}_3$ and P, such that the length of $\hat{e}_2$ between p_2 and the endpoint it shares with $\hat{e}_1$ equals the length of $\hat{e}_3$ between p_3 and the endpoint it shares with $\hat{e}_1$.*

Clearly, this proposition allow us to compute $T_{i,j,k}$ (and its perimeter) in constant time. Thus, by considering all n^3 triples (i, j, k), DePano finds a minimum perimeter circumscribing triangle in $O(n^3)$ time.

Another way of looking at this problem is in terms of an $n \times (2n-2) \times (2n-1)$ array $A = \{a_{ijk}\}$ defined as follows. For $1 \leq i \leq n$ and $i < j < k < i + n$, we let a_{ijk} be the perimeter of $T_{i, j \bmod n, k \bmod n}$, provided this triangle exists. If this triangle does not exist, we let a_{ijk} be ∞. For all other values of i, j, and k, we also define a_{ijk} to be ∞. Using Proposition 28, each entry of A can be computed in constant time. And, since the perimeter of the minimum perimeter circumscribing triangle is simply the minimum entry in A, the following proposition provides the basis of an $O(n \lg n)$ time algorithm:

Proposition 29 *A is totally monotone.*

It is unclear whether A is Monge-composite, since fixing any two indices of A does not fix the both the edges of the circumscribing triangle corresponding to these indices. Nevertheless, Proposition 29 allows us to apply the array-searching algorithm given in Proposition 3 and obtain:

Theorem 30 *Given a convex n-gon P, a minimum perimeter triangle Q circumscribing P can be computed in $O(n \lg n)$ time.*

It remains open whether a minimum perimeter circumscribing triangle can be found in $o(n \lg n)$ time and whether there exist efficient algorithms for computing minimum circumscribing d-gons.

5 Dynamic Programming

5.1 Frances Yao's Paper Revisited

In [Ya80], Yao considered the following dynamic programming problem. Let $W = \{w(i,j)\}$ be a Monge array satisfying the Monge condition and the following additional monotonicity constraint: $w(i',j') \leq w(i,j)$ for $i \leq i' \leq j' \leq j$. Let $C = \{c(i,j)\}$ be defined so that $c(i,i) = 0$ for $1 \leq i \leq n$ and

$$c(i,j) = w(i,j) + \min_{i<k\leq j} \{c(i,k-1) + c(k,j)\}$$

for $1 \leq i < j \leq n$. In [Ya80], Yao showed that all of the entries of C can be computed in $O(n^2)$ time. She used this result to provide a simple alternate solution to Knuth's problem regarding optimal binary trees [Kn73]; she also extended this to the computation of optimal t-ary trees. In obtaining her $O(n^2)$ time bound, Yao showed the following:

Lemma 31 ([Ya80]) *C is a Monge array satisfying the Monge condition, i.e., $c(i,j) + c(i',j') \leq c(i,j') + c(i',j)$ for $i \leq i' \leq j \leq j'$.*

Thus, C is clearly a totally monotone array. Unfortunately, the entries of C are not directly available, so the result of [AKMSW87] seems inapplicable, at least in a straightforward manner.

Now consider the three-dimensional array $D = \{d(i,k,j)\}$, where we define $d(i,k,j) = c(i,k-1) + c(k,j) + w(i,j)$ when $i < k \leq j$ and ∞ otherwise. Since we now have $c(i,j) = \min_{1\leq k\leq n} d(i,k,j)$, we have transformed the problem of computing C to the problem of computing the tube maxima of D. D is clearly Monge-composite (and thus totally monotone), but again the array's entries are not directly available. At this point, however, we can apply the results of Wilber [Wi88].

Each plane of D, corresponding to a particular value of the first index $\hat{i}$, is totally monotone, since D is totally monotone. Moreover, the problem of computing the column maxima in the plane corresponding to $\hat{i}$ is equivalent to the concave least-weight subsequence problem considered in [HL85] and [Wi88]:

$$\begin{aligned} \text{find} \quad f(q) &= \min_{0\leq p<q} g(p,q) \quad \text{for } 1 \leq q \leq m \\ &\text{where} \quad f(0) = 0 \\ \text{and} \quad g(p,q) &= f(p) + v(p,q) \quad \text{for } 0 \leq p < q \leq m. \end{aligned}$$

In our context, $m = n - \hat{i}$, $f(q) = c(\hat{i}, q + \hat{i})$, $g(p,q) = d(\hat{i}, p + \hat{i} + 1, q + \hat{i})$, and $v(p,q) = c(p + \hat{i} + 1, q + \hat{i}) + w(\hat{i}, q + \hat{i})$. Note that the array $V = \{v(p,q)\}$ is Monge since C is Monge. Provided $c(i,j)$ is known for $i > \hat{i}$ (so that $v(p,q)$ can always be computed in constant time), we can solve this problem in $O(m)$ time using the algorithm given by Wilber in [Wi88]. Thus, to compute the tube maxima of D, we need only apply Wilber's algorithm n times, first to the plane corresponding to $\hat{i} = n$, then to the plane corresponding to $\hat{i} = n-1$, and so on down to the plane corresponding to $\hat{i} = 1$. This gives us all of C's entries in $O(n^2)$ time.

Now suppose the array C is defined in terms of a weight array W that satisfies the inverse Monge condition and, in addition, $w(i',j') \geq w(i,j)$ for $i \leq i' \leq j' \leq j$. By adopting the procedure given above, we can show:

Proposition 32 *The array C can be computed in $O(n^2\alpha(n))$ time, where $\alpha(n)$ denotes the inverse Ackermann's function.*

Proof This proof is identical to that given above except that now each plane of the three-dimensional array D, corresponding to a particular value of the first index $\hat{i}$, is no longer totally monotone or Monge. Instead, the plane is a *staircase monotone* array as defined by Aggarwal and Klawe [AK87]. The problem of computing the column maxima in the plane corresponding to $\hat{i}$ is thus equivalent to the *convex* least-weight subsequence problem, and Klawe and Kleitman [KK88] have shown that this problem can be solved in $O(n\alpha(n))$ time. ■

It remains open whether the time complexity given in Proposition 32 can be improved from $O(n^2\alpha(n))$ to $O(n^2)$.

5.2 Waterman's Problem

The sequence of ribonucleotides of an RNA is called its *primary structure*. When the primary structure of a *single-stranded RNA* is known, the question of

which bases form pairs becomes important and this is referred to as the *secondary structure* of RNA. It turns out that primary structure of RNA is easy to evaluate and in [Wa78], Waterman argued that understanding the secondary structure of RNA is closely related to solving some dynamic programming problems. One of the dynamic programming problems involves computing the entries of an array $E = \{e(i,j)\}$, where

$$e(i,j) = \min_{\substack{1 \le i' < i \\ 1 \le j' < j}} \{c(i',j') + w(i'+j', i+j)\},$$

such that for $k \le p \le l \le q$,

$$w(k,l) + w(p,q) \le w(k,q) + w(p,l) \qquad (4)$$

and $c(i,j)$ can be computed from $e(i,j)$ in constant time. Waterman and Smith [WS86] provided an $O(n^3)$ time algorithm for computing the entries of E. Recently, Eppstein, Galil, and Giancarlo [EGG88] have obtained an $O(n^2 \lg^2 n)$ time algorithm for this problem. (They also obtained an $O(n^2 \lg n \lg \lg n)$ time algorithm for the special case of w being a logarithmic function or other simple function.) Using array-searching, we show:

Theorem 33 *All $e(i,j)$ can be computed in $(n^2 \lg n)$ time if the $w(i,j)$ obey the Monge condition (4) or the inverse Monge condition.*

Sketch of Proof We only prove the theorem for the case when $c(i,j) = e(i,j)$; this proof can be easily generalized to any $c(i,j)$ that can be computed from $e(i,j)$ in constant time.

We begin with some definitions and observations. The entry $e(i,j)$ *strictly dominates* the entry $e(i',j')$ if $i > i'$ and $j > j'$. An entry depends only on those entries it strictly dominates. The *diagonal* d_k of E contains those entries $e(i',j')$ such that $i' + j' = k$.

Observation 34 *The only entry on a particular diagonal d_k of E that we need to consider in computing $e(i,j)$ is the minimum entry $e(i',j')$ on d_k that is strictly dominated by $e(i,j)$. (We will refer to this entry as a* diagonal minimum.*)*

With this is mind, we define

$$d_k(i,j) = \min_{\substack{i' < i, j' < j \\ i'+j'=k}} e(i',j').$$

Thus

$$e(i,j) = \min_{k < i+j-1} \{d_k(i,j) + w(k, i+j)\}.$$

Now we can describe our algorithm for computing the entries of E. We use a divide-and-conquer approach, reminiscent of an algorithm given by Aggarwal and Suri [AS87] for finding the largest empty corner rectangle. We begin by partitioning E into four $n/2 \times n/2$ subarrays A, B, C, and D, as shown in Figure 1, and recursively computing the entries of A in $T(n/2)$ time. We then compute the *effect* of A on B, i.e., for each entry of B, we obtain a upper bound on its value based on the entries of A. (Each entry in B is the minimum of a number of terms; some of these terms depend only on the entries of B; the rest depend only on the entries of A; we compute the minimum of those terms dependent on the entries of A.) We will explain later how this may be accomplished in $O(n^2)$ time. Next, we recursively compute the entries of B in $T(n/2)$ time, taking into account the effect of A on B, i.e., in assigning an entry of B its value, we always take the minimum of the value we have recursively computed for this entry based on the entries of B, and the value we have computed for this entry based on the entries of A. We repeat this process for C, computing the effect of A on C in $O(n^2)$ time and then recursively computing C in $T(n/2)$ time. Finally, we compute the effect of A on D, the effect of B on D, and the effect of C on D, all in $O(n^2)$ time, and then recursively computed D in $T(n/2)$ time.

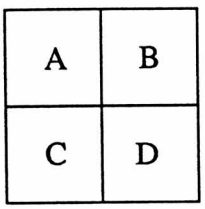

Figure 1:
Partitioning Scheme

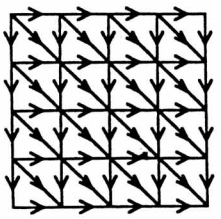

Figure 2:
Grid DAG

This yields the recurrence $T(n) \le 4T(n/2) + O(n^2)$ for the time to compute all entries of an $n \times n$ array. Since $T(1) = O(1)$, $T(n) = O(n^2 \lg n)$.

We can compute the effect of A on B as follows (computing the effect of A on C, A on D, B on D, and C on D may be done in an analogous manner). For $1 \le i \le n/2$, each entry $e(i, j+n/2)$ in row i of B strictly dominates the same set of entries in A and thus depends on the same diagonal minima. Consequently, consider the two-dimensional array $X_i = \{x_i(k,j)\}$ where $x_i(k,j) = d_k(i, 1+n/2) + w(k, i+j+n/2)$. $e(i, j+n/2)$ can then be expressed as the minimum of two quantities

$$\min_{k < i+j+n/2-1} x_i(k,j)$$

and

$$\min_{\substack{1<i'<i \\ 1<j'<j}} \{e(i', j' + n/2) + w(i' + j' + n/2, i + j + n/2)\}.$$

As the first of these quantities depends only on entries in A and the second depends only on entries in B, computing the effect of A on B reduces to computing the row minima in all n of the X_i.

Observation 35 *For $1 \leq i \leq n/2$, X_i is a Monge array (satisfying either the Monge or inverse Monge condition, depending on the condition that the w's satisfy).*

Lemma 36 *The effect of A on B, specifically the row minima of X_i for $1 \leq i \leq n/2$ can be computed in $O(n^2)$ time.*

Proof of Lemma The $d_k(i, n/2)$ for $1 \leq i \leq n/2$ and $2 \leq k \leq n$ can be computed in $O(n^2)$ time, since $d_k(1, n/2) = e(1, k-1)$ and $d_k(i, n/2)$ can be computed from $d_k(i-1, n/2)$ and $e(i, k-i)$ in constant time. Once this has been done, each entry in each of the X_i can be computed in constant time, so we can just apply the algorithm of [AKMSW87] n times and obtain all the desired row minima in an additional $O(n^2)$ time. ■

This completes our proof of Theorem 33. ■

Note that this divide-and-conquer approach can be used to obtain an $O(n \lg n)$ time solution to the one-dimensional version of Waterman's problem, namely, the concave or convex least-weight subsequence problem considered by [HL85,KK88,Wi88]. Wilber [Wi88] solves the concave least-weight subsequence problem in $O(n)$ time by an elegant modification of [AKMSW87]; this suggests that perhaps an $o(n^2 \lg n)$ solution to the concave version of Waterman's problem [EGG88,Wa78] might be obtainable.

6 VLSI River Routing

Let $p_1 < p_2 < \cdots < p_n$ be points on a line segment P that is horizontally imbedded in the plane. (We identify a point p_i with its offset relative to the leftmost point of P.) Let x_i be the x-coordinate of p_i in the plane. We call x_1 the *offset* of P and the y-coordinate of p_1 the *separation* of P. Let $q_1 < q_2 < \cdots < q_n$ be points on a line segment Q imbedded horizontally in the plane so that q_1 is at the origin. (We identify a point q_i with its x-coordinate.) A *routing* is a set of n nonintersecting continuous curves (wires), with the i-th curve going from p_i to q_i, that satisfy a set of *design rules*. The design rules are determined by circuit technology. At a minimum, there is a requirement that the distance between any two wires be at least some fixed constant, which we may take to be 1. We are concerned with three problems:

1. *Minimum Separation Problem.* Given a fixed offset for P, find the minimum separation that allows a valid routing.
2. *Offset Range Problem.* Given a wiring rule and a separation, find all offsets permitting a legal wiring for that separation.
3. *Optimal Offset Problem.* Find the offset for P for which the minimum separation for P that allows a valid routing is minimized.

We say that the points are monotone if $x_i \leq q_i$ for all i or if $x_i \geq q_i$ for all i. We may assume that the points are monotone, for if they are not they may be partitioned into maximal monotone blocks and the routing for each block may be done independently. Without loss of generality, assume $x_i \leq q_i$ for all i. Since the points are monotone, we can assume that the wires are monotone, i.e., the y-coordinate of a wire is nonincreasing as the x-coordinate increases. For $1 \leq i \leq n$ and $1 \leq j \leq n-i$, the j-th *barrier* about q_i is defined to be the set of points that delimit the closest possible approach to q_i of the monotone wire going from p_{i+j} to q_{i+j}. The barriers are determined by the design rules.

Let $H_\rho(x)$ denote a family of barrier curves (with ρ denoting the index) such that for $x < \rho$, $0 < H_\rho \leq \rho$ and $H_\rho(x)$ is concave, and for $x \geq \rho$, $H_\rho(x) = 0$. A family of similar curves that is concentric with respect to the origin can be denoted by $H_\rho = \rho h(x/\rho)$. (See [SD81] for details.) For $1 \leq i, j \leq n$, Siegel and Dolev defined the concept of a left-offset function $L(i, j)$ and a right-offset function $R(i, j)$ such that the relative offset interval for P is

$$\left[\max_{1 \leq i,j \leq n} L(i, j), \min_{1 \leq i,j \leq n} R(i, j)\right].$$

Siegel and Dolev also showed:

Theorem 37 ([SD81]) *Suppose $r \geq 0$, $s \geq 0$, and $j \geq i + r$. Then if $L(i, j) \leq L(i + r, j)$, $L(i, j + s) \leq L(i + r, j + s)$. Similarly, if $R(i, j) \geq R(i + r, j)$, then $R(i, j + s) \geq R(i + r, j + s)$.*

Using this theorem, it is readily seen that:

Observation 38 *The $n \times n$ arrays $L = \{L(i, j)\}$ and $R = \{R(i, j)\}$ are totally monotone. Furthermore, if $L(i, j)$ and $R(i, j)$ can be computed in constant time, then the offset range problem can be solved in linear time by finding the row maxima in L and R.*

This observation generalizes the results of Siegel [Si88], who shows an $O(n)$ time algorithm for the offset range problem when barriers are polygonal in nature.

In [SD81], Siegel and Dolev show that the optimal offset problem can be solved for a particular set of wiring rules in $O(\Psi(n)\lg n)$ time if the separation problem for these wiring rules can be solved in $O(\Psi(n))$ time. Consequently, using [AKMSW87], we can obtain an $O(n\lg n)$ time algorithm for the optimal offset problem for most wiring rules used in practice. It remains open whether an $o(n\lg n)$ time algorithm can be provided for this problem. Indeed, Mirazaian [Mi87] has a provided an elegant $\Theta(n)$ time algorithm for this problem for rectilinear wiring; however, his algorithm exploits the properties associated with such a wiring, and it is unlikely that his techniques can be extended to other wiring models.

In [CJ88], Chang and JáJá provide parallel algorithms for the minimum separation problem and the optimal offset problem for the river routing of rectilinear wires. They provide an n processor, $O(\lg n)$ time CREW-PRAM algorithm for the minimum separation problem and an n processor, $O(\lg^2 n)$ time CREW-PRAM algorithm for the optimal offset problem. In this subsection, we improve the processor bounds and then extend these results for routing rules with weaker assumptions.

Proposition 39. *The minimum separation problem for n rectilinear wires can be solved in $\Theta(\lg n)$ time using $n/\lg n$ processors on a CREW-PRAM and in $\Theta(\lg\lg n)$ time using $n/\lg\lg n$ processors on a CRCW-PRAM.*

Proposition 40 *The offset range problem for n rectilinear wires can be solved in $\Theta(\lg n)$ time using $n/\lg n$ processors on a CREW-PRAM and in $\Theta(\lg\lg n)$ time using $n/\lg\lg n$ processors on a CRCW-PRAM.*

At this point, we observe that the simple divide-and-conquer technique given by Mirazaian yields an $n/\lg n$ processor, $O(\lg^2 n)$ time CREW-PRAM algorithm and an $n/\lg\lg n$ processor, $O(\lg n\lg\lg n)$ time CRCW-PRAM algorithm solving the optimal offset problem. This improves the corresponding processor bounds for these problems given in [CJ88] by a factor of $O(\lg n)$.

If the routing rules for wires obey the assumptions given in [SD81], then we have the following:

Corollary 41 *The minimum separation and offset range problems for n wires that obey the wiring rules given by Siegel and Dolev can be solved in $O(\lg n\lg\lg n)$ time using $n/\lg\lg n$ processors on a CREW-PRAM and in $O(\lg n)$ time using n processors on a CRCW-PRAM. Furthermore, the optmial offset problem for these wires can be solved in $O(\lg^2 n\lg\lg n)$ time using $n/\lg\lg n$ processors on a CREW-PRAM and in $O(\lg^2 n)$ time using n processors on a CRCW-PRAM.*

It remains open whether the optimal offset problem can be solved in $\Theta(\lg n)$ time on an $n/\lg n$ processor CREW-PRAM, even for rectilinear wires. Also, it remains open whether any of our techniques can be used to improve the processor and/or time bounds given by Chang and JáJá [CJ88] for routability testing within a rectilinear polygon. Finally, obtaining better processor-time bounds than those given in Corollary 41 also remains open.

7 Finding Shortest Paths in Certain Planar Graphs

Given a directed, acyclic $(2m+1)\times(n+1)$ grid graph G (as shown in Figure 2), with a non-negative cost $c((i,j),(k,l))$ associated with each of its arcs $(v_{i,j}, v_{k,l})$, Fuchs, Kedem, and Uselton [FKU77] showed that the problem of optimal surface reconstruction from planar contours can be reduced to finding the minimum cost paths in G that start from $v_{i,0}$ and end at $v_{m+i,n}$ for $i = 0,1,\ldots,m-1,m$. [FKU77] also presents an $O(nm\lg m)$ time sequential algorithm for computing all these paths. (To be precise, the costs on all diagonal edges of G are infinite for the problem discussed in [FKU77]. However, by considering the diagonal edges also, our algorithms can be extended to incorporate [KF80].)

In a separate paper, Kedem and Fuchs [KF80] showed that finding the minimum cost path which begins at $v_{0,0}$ and ends at $v_{2m,n}$ can be used to solve the string editing problem considered in [WF74] (see Subsection 7.1 also) and that finding minimum cost paths which begin at $v_{i,0}$ and end at $v_{m+i,n}$ can be used to solve the circular string-to-string correction problem. Consequently, the sequential algorithm of [FKU77] can be employed to solve both these problems in $O(nm\lg m)$ time.

Recently, Apostolico et al. [AAL87] and Mathies [Ma88] have independently provided parallel algorithms that find the minimum cost path in G starting from $v_{0,0}$ and ending at $v_{2m,n}$. (These two papers discuss the problem in the context of string editing and largest common subsequences only; see Subsection 7.1.) Both algorithms can be easily extended to find minimum cost paths that begin at $v_{i,0}$ and

end at $v_{i+m,n}$ for $i = 0, \ldots, m$. Below, we briefly discuss a divide-and-conquer algorithm; a variant of this algorithm appears in [AAL87]. Our algorithm incorporates the results of Subsection 3.2 to improve the time bounds given in [AAL87] without affecting the processor-time product by more than a constant factor. Later, in Subsection 7.1, we apply these ideas to several different problems and compare our results with those given in the literature.

In this section, we will refer to nodes in G with in-degree zero as sources and those with out-degree zero as sinks. We will also refer to the problem of finding the shortest paths from all sources to all sinks as the all minimum cost paths problem. Note that the output of this problem is a $(2m+n) \times (2m+n)$ array whose (i,j)-th element contains the cost of minimum cost path from the i-th source to the j-th sink.

For the sake of simplicity, we assume $n = 2m$, i.e., G is an $m \times m$ grid DAG. Let DIST_G be an $n \times n$ array containing the lengths of all minimum cost paths that begin at the top or left boundary of G and end at its right or bottom boundary. Then divide the $n \times n$ grid into four $n/2 \times n/2$ grids A, B, C, and D as shown in Figure 1. In parallel, recursively solve the all shorest paths problem for A, B, C, and D, thereby obtaining the four distance arrays DIST_A, DIST_B, DIST_C, and DIST_D. Then (i) use DIST_A and DIST_B to obtain $\text{DIST}_{A \cup B}$, (ii) use DIST_C and DIST_D to obtain $\text{DIST}_{C \cup D}$, and (iii) use $\text{DIST}_{A \cup B}$ and $\text{DIST}_{C \cup D}$ to obtain DIST_G.

Until now, our algorithm is identical to that given in [AAL87]; however, it differs form this point onwards, since Apostolico et al. obtain their time and processor bounds by describing an efficient procedure for executing steps (i), (ii), and (iii), whereas we show below how each of these steps can be executed by finding the tube maxima in a $\Theta(n) \times \Theta(n) \times \Theta(n)$ three-dimensional totally monotone array. Since the tube maxima problem can be solved very efficiently using the results of Subsection 3.2, we will improve upon the results given in [AAL87].

Let $v_1, \ldots, v_m$ ($w_1, \ldots, w_m$, respectively) be the m points on the left and top boundaries of A (B, respectively) in clockwise order. Similarly, let $x_1, \ldots, x_m$ ($y_1, \ldots, y_m$, respectively) be the m points on the bottom and right boundaries of A (B, respectively) in counterclockwise order. Then we claim:

Observation 42 DIST_A *and* DIST_B *are both Monge arrays.*

Now let $A' = a'(i,j)$ and $B' = b'(i,j)$ denote the $n \times n/2$ and $n/2 \times n$ arrays obtained by deleting the first $n/2$ columns of A and the last $n/2$ rows of B, respectively. Consider the $n \times n/2 \times n$ three-dimensional array whose (i,j,k)-th entry contains $a'(i,j) + b'(j,k)$. Clearly, this array is Monge-composite and hence totally monotone. Furthermore, $\text{DIST}_{A \cup B}$ is a $3n/2 \times 3n/2$ array and its (i,k)-th entry is either an entry of DIST_A (this happens when $k \leq n/2$), or an entry of DIST_B (when $k > n/2$ and $n < i \leq 3n/2$), or the minimum entry in the (i,j)-th tube of this three-dimensional array.

Theorem 43 *The all minimum cost paths problem for an $n \times n$ grid DAG can be solved in $O(\lg^2 n)$ time using $n^2/\lg n$ processors on a CREW-PRAM.*

Sketch of Proof From Observation 42 and algorithm given above, it follows that $T(n) \leq T(n/2) + T'(n)$ and $OP(n) \leq 4\,OP(n) + OP'(n)$, where $T(1)$ and $OP(1)$ are constants and $T'(n)$ and $OP'(n)$ denote the time and number of operations, respectively, required to find the tube maxima of a $\Theta(n) \times \Theta(n) \times \Theta(n)$ totally monotone array. Now from Proposition 18, we have $T'(n) = O(\lg n)$ and $OP'(n) = O(n^2)$ for a CREW-PRAM. Consequently, $T(n) = O(\lg^2 n)$ and $OP(n) = O(n^2 \lg n)$. Using Brent's theorem [Br74], it can be readily checked that the number of processors used in $O(n^2/\lg n)$. ■

Similarly, by incorporating Propositions 19 and 20, we have:

Theorem 44 *The all minimum cost paths problem for an $n \times n$ grid DAG can be solved in $O(\lg n (\lg\lg n)^2)$ time using $n^2/(\lg\lg n)^2$ processors on a CRCW-PRAM.*

Theorem 45 *The all minimum cost paths problem for an $n \times n$ grid DAG can be solved in $O(\lg n \lg\lg n)$ time using $n^2/\lg\lg n$ processors on a CRCW-PRAM if all arc costs are integers from $\{1, 2, \ldots, n^c\}$ for some constant c.*

Above, we showed how to solve the all minimum cost paths problem for $n = 2m$; below we state the results for general n and m; these results can be readily derived along the lines of Theorems 43, 44, and 45.

Corollary 46 *Suppose $m \leq n$. Then the all shortest paths problem for a $2m \times n$ DAG can be solved (i) in $O(\lg m \lg n)$ time using $mn/\lg m$ processors on a CREW-PRAM, (ii) in $O(\lg n (\lg\lg m)^2)$ time using $mn/(\lg\lg m)^2$ processors on a CRCW-PRAM, and (iii) in $O(\lg n \lg\lg m)$ time using $mn/\lg\lg m$ processors on a CRCW-PRAM if the costs on edges are non-negative integers from $\{1, \ldots, n^c\}$ for some constant c.*

The following two questions remain unresolved regarding the all minimum cost paths problem. First, the best sequential algorithm for this problem for a $2m \times n$ grid DAG takes $O(mn \lg m)$ time, whereas the only known lower bound is $\Omega(mn)$. It seems that the array-searching framework should be useful in improving the upper bound. Second, the processor-time product of our algorithm is $O(nm \lg n)$, which is a factor of $O(\lg n / \lg m)$ away from the sequential bound.

7.1 Applications

In biological research, tomography and medical diagnosis, manufacturing design, and architecture, it is often useful to reconstruct a three-dimensional solid from a set of cross-sectional contours; this reconstruction often helps in comprehending the object's structure and it facilitates it automatic manipulation. Fuchs, Kedem, and Uselton [FKU77] propose a procedure that reduces this problem to a special case of the all minimum cost paths problem for a grid DAG. Consequently, the bounds given in Corollary 46(i) and (ii) also hold for the problems considered in [FKU77].

Wagner and Fischer [WF74] have given a sequential $O(nm)$ time algorithm for solving the string-editing problem. Kedem and Fuchs [KF80] have provided an $O(nm \lg m)$ time sequential algorithm for the circular string-editing problem. Since the all minimum cost paths problem for an $(n+1) \times (m+1)$ grid DAG solves both these problems, Corollary 46(i) and (ii) also apply to these problems. Also, because the largest common subsequence problem [AAL87,Ma88] is a special case of the string editing problem where all costs are small integers, the largest common subsequence problem can be solved in parallel in the time and processor bounds given by Corollary 46(iii). This improves the time bounds given in [AAL87,Ma88] by a factor of $O(\lg m)$ for the CREW-PRAM and by a factor of $O(\lg m / \lg \lg m)$ for the CRCW-PRAM. Also, our result has the same processor-time product as [AAL87] and it improves upon [Ma88] by a factor of $O(\lg m)$.

In the conclusion of [AAL87], Apostolico et al. remark that if they increase the number of processors by a factor of $O(m)$, then they could improve their time bounds on the string editing problem by a factor of $O(\lg m)$ using trivialized versions of their algorithms. They also left as an interesting open question whether any improvement in time complexity can be achieved at the expense of only a polylogarithmic factor in processor complexity. Our Corollary 46, when applied to string editing, shows that a factor of $O(\lg m)$ ($O(\lg m / \lg \lg m)$, respectively) improvement in time complexity for a CREW-PRAM (CRCW-PRAM, respectively) is achievable *without any deterioration in processor complexity*; this settles the open problem raised in [AAL87].

Given n distinct points $p_1, \ldots, p_n$ in the plane, point $p_i = (x_i, y_i)$ *dominates* $p_j = (x_j, y_j)$ if $x_i \geq x_j$ and $y_i \geq y_j$. A point is called maximal (and is said to lie in the first maximal layer) if it is not dominated by any other point. If we remove all the points in the maximal layers indexed $1, 2, \ldots, j-1$, then the j-th maximal layer is composed of all maximal points in the remaining set. The problem of computing the maximal layers arises frequently in computational geometry and statistics, and a $\Theta(n \lg n)$ time sequential algorithm for the problem is known.

In 1986, Rao Kosaraju [Ko86] communicated to us a simple but elegant procedure for finding maximal layers in parallel. This procedure reduces this problem to that of finding maximum cost paths from n specially marked nodes to node (n, n) in an $(n+1) \times (n+1)$ grid DAG. Consequently, a simple extension of the techniques given at the beginning of this section can be used to solve the problem of finding these maximum cost paths. Moreover, since only binary costs are assigned to arcs, Theorems 43 and 45 can be used to compute the maximal layers of n planar points.

The processor-time complexity of our solution to the maximal layers problem is $O(n^2 \lg n)$, whereas the sequential algorithm takes $\Theta(n \lg n)$ time. Consequently, it remains open whether the processor-time complexity can be improved.

Acknowledgements

The authors thank Avrim Blum, Rao Kosaraju, Tom Leighton, Gary Miller, Mark Newman, and Baruch Schieber for stimulating discussions. The authors specially thank Baruch Schieber for pointing out [AAL87] and Gad Landau for pointing out [FKU77].

References

[ACY85] A. Aggarwal, J.S. Chang, C.K. Yap, "Minimum Area Circumscribing Polygons," The Visual Computer, Vol. 1, 1985, pp. 112–117.

[ACGOY88] A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing, C.K. Yap, "Parallel Computational Geometry," Algorithmica, Vol. 3, No. 3, 1988, pp. 293–326.

[AK87] A. Aggarwal, M. Klawe, "Applications of Generalized Matrix Searching to Geometric Algorithms," Discrete Applied Math., 1988, to appear; also Tech. Report, IBM Almaden Research Center, 1987.

[AKMSW87] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, R. Wilber, "Geometric Applications of a Matrix-Searching Algorithm," Algorithmica, Vol. 2, 1987, pp. 209–233.

[AS87] A. Aggarwal, S. Suri, "Fast Algorithms for Computing the Largest Empty Rectangle," Proc. 3rd Annual Symp. on Computational Geometry, 1987, pp. 278–290.

[AAL87] A. Apostolico, M.J. Atallah, L.L. Larmore, "Efficient Parallel Algorithms for String Editing and Related Problems," Tech. Report, Purdue University, Oct. 1987.

[AKLMT88] M.J. Atallah, S.R. Kosaraju, L.L. Larmore, G. Miller, S. Teng, "Constructing Trees in Parallel," unpublished manuscript, May 1988.

[BDDG85] J.E. Boyce, D.P. Dobkin, R.L. Drysdale, L.J. Guibas, "Finding Extremal Polygons," SIAM J. of Computing, Vol. 14, 1985, pp. 134–147.

[Br74] R.P. Brent, "The Parallel Evaluation of General Arithmetic Expressions," J. of ACM, Vol. 2, No. 2, 1974, pp. 201–206.

[CJ88] S.-C. Chang, J. JáJá, "Parallel Algorithms for River Routing," Proc. 17th International Conference on Parallel Processing, Aug. 1988.

[Co86] R. Cole, "Parallel Merge Sort," Proc. 27th IEEE FOCS, 1986, pp. 511–516.

[De87] N.A.A. DePano, "Polygon Approximation with Optimized Polygonal Enclosures: Applications and Algorithms," Ph.D. thesis, Dept. of Computer Science, The Johns Hopkins Univ., 1987.

[EGG88] D. Eppstein, Z. Galil, R. Giancarlo, "Speeding Up Dynamic Programming with Applications to the Computation of RNA Structure," manuscript, Columbia University, April 1988.

[FRW88] F.E. Fich, P.L. Ragde, A. Wigderson, "Relations Between Concurrent-Write Models of Parallel Computation," SIAM J. of Computing, Vol. 17, No. 3, June 1988, pp. 606–627.

[FKU77] H. Fuchs, Z.M. Kedem, S.P. Uselton, "Optimal Surface Reconstruction from Planar Contours," CACM, Vol. 20, No. 10, Oct. 1977, pp. 693–701.

[HL85] D.S. Hirschberg, L.L. Larmore, "The Least Weight Subsequence Problem," SIAM J. of Computing, Vol. 16, No. 4, Aug. 1987, pp. 628–638.

[Ho61] A.J. Hoffman, "On Simple Linear Programming Problems," Convexity, Proc. Symposia in Pure Mathematics, Vol. 7, American Mathematical Society, Providence, RI, 1961, pp. 317–327.

[KF80] Z.M. Kedem, H. Fuchs, "On Finding Several Shortest Paths in Certain Graphs," Proc. 18th Allerton Conference on Communication, Control and Computing, Oct. 1980, pp. 677–683.

[KK88] M.M. Klawe, D.J. Kleitman, "An Almost Linear Time Algorithm for Generalized Matrix Searching," Tech. Report, IBM Almaden Research Center, 1988.

[Ko86] S.R. Kosaraju, personal communication, Oct. 1986.

[Kn73] D.E. Knuth, "Optimum Binary Search Trees," Acta Informatica, Vol. 1, 1973, pp. 14–25.

[LLMPW87] F.T. Leighton, C.E. Leiserson, B. Maggs, S. Plotkin, J. Wein, "Theory of Parallel and VLSI Computation: Lecture Notes of 18.435/6.848," MIT Research Seminar Series, MIT/LCS/RSS 1, March 1988.

[Ma88] T.R. Mathies, "A Fast Parallel Algorithm to Determine Edit Distance," Tech. Report CMU-CS-88-130, Carnegie Mellon University, April 1988.

[Mi87] A. Mirazaian, "River Routing in VLSI," J. of Comp. Systems and Sciences, Vol. 34, 1987, pp. 43–54.

[Mo81] G. Monge, "Déblai et remblai," Mémoires de l'Académie des Sciences, Paris, 1781.

[OAMB86] J. O'Rourke, A. Aggarwal, M. Baldwin, S. Maddila, "An Optimal Algorithm for Finding Minimal Enclosing Triangles," J. of Algorithms, Vol. 7, 1986, pp. 258–268.

[Si88] A. Siegel, "Fast Optimal Placement for River Routing," in preparation.

[SD81] A. Siegel, D. Dolev, "The Separation for General Single-Layer Wiring Barriers," Proc. CMU Conference on VLSI Systems and Computations, eds. H.T. Kung, B. Sproull, G. Steele, Computer Science Press, 1981, pp. 143–152; portions of this paper also appear in "Some Geometry for General River Routing," SIAM J. of Computing, Vol. 17, No. 3, June 1988, pp. 583–605.

[To83] G.T. Toussaint, "The Symmetric All-Farthest Neighbor Problem," Comp. and Math. Applications, Vol. 9, No. 6, 1983, pp. 747–753.

[WF74] R.A. Wagner, M.J. Fischer, "The String to String Correction Problem," J. of ACM, Vol. 21, No. 1, 1974, pp. 168–173.

[Wa78] M.S. Waterman, "Secondary Structure of Single-Stranded Nucleic Acids," in *Studies in Foundations and Combinatorics, Adv. in Math. Suppl. Studies, Vol. 1* (G.C. Rota, ed.), Academic Press, New York, 1978, pp. 167–212.

[WS86] M.S. Waterman, T.F. Smith, "Rapid Dynamic Programming Algorithms for RNA Secondary Structure," Adv. in Applied Math., Vol. 7, 1986, pp. 455–464.

[Wi88] R. Wilber, "The Concave Least Weight Subsequence Problem Revisited," J. Algorithms, 1988, to appear.

[Ya80] F.F. Yao, "Efficient Dynamic Programming Using Quadrangle Inequalities," Proc. 12th ACM STOC, 1980, pp. 429–435.