SIAM J. COMPUT.
Vol. 27, No. 2, pp. 466–490, April 1998

EFFICIENT MATRIX CHAIN ORDERING IN POLYLOG TIME*

PHILLIP G. BRADFORD†, GREGORY J. E. RAWLINS‡, AND GREGORY E. SHANNON§

Abstract. The matrix chain ordering problem is to find the cheapest way to multiply a chain of n matrices, where the matrices are pairwise compatible but of varying dimensions. Here we give several new parallel algorithms including $O(\lg^3 n)$-time and $n/\lg n$-processor algorithms for solving the matrix chain ordering problem and for solving an optimal triangulation problem of convex polygons on the common CRCW PRAM model. Next, by using efficient algorithms for computing row minima of totally monotone matrices, this complexity is improved to $O(\lg^2 n)$ time with n processors on the EREW PRAM and to $O(\lg^2 n \lg\lg n)$ time with $n/\lg\lg n$ processors on a common CRCW PRAM. A new algorithm for computing the row minima of totally monotone matrices improves our parallel MCOP algorithm to $O(n \lg^{1.5} n)$ work and polylog time on a CREW PRAM. Optimal log-time algorithms for computing row minima of totally monotone matrices will improve our algorithm and enable it to have the same work as the sequential algorithm of Hu and Shing [*SIAM J. Comput.*, 11 (1982), pp. 362–373; *SIAM J. Comput.*, 13 (1984), pp. 228–251].

Key words. parallel, dynamic programming, matrix chain ordering, optimization

AMS subject classifications. 90C39, 68Q25, 03D15

PII. S0097539794270698

1. Introduction.

The design of efficient parallel algorithms for problems with elementary serial dynamic programming solutions has been the focus of much recent research. These problems include string editing [2, 5, 30], context-free grammar recognition [35, 37], and optimal tree building [7, 32]. Polylog time parallel algorithms for solving these problems use new approaches, since straightforward parallelization of sequential dynamic programming algorithms produce very slow (linear-time) parallel algorithms. Many efficient parallel algorithms designed to date rely on *monotonicity conditions* to give divide-and-conquer schemes. By "efficient" we mean that the processor-time product is within a polylog factor of the best sequential time.

The *matrix chain ordering problem* (MCOP) is to find the cheapest way to multiply a chain of n matrices, where the matrices are pairwise compatible but of varying dimensions. This problem can be found in many classic textbooks on parallel and sequential algorithms, such as [3, 18]. The MCOP is often the focus of dynamic programming research and pedagogy because of its amenability to an elementary dynamic programming solution. There has been significant sequential and parallel work on the MCOP [11, 17, 19, 20, 21, 25, 26, 27, 28, 29, 38, 40, 41, 39, 42, 43, 44] and a related convex polygon triangulating problem. However, until recently none of this work has given an efficient (linear-processor) polylogarithmic-time algorithm for the

*Received by the editors July 6, 1994; accepted for publication (in revised form) February 13, 1996. An extended abstract of this paper appeared in the *Proceedings of the 8th Annual IEEE International Parallel Processing Symposium*, Cancun, Mexico, H. J. Siegel, ed., IEEE Computer Society Press, Los Alamitos, CA, pp. 234–241. This paper also represents part of the first author's dissertation.

http://www.siam.org/journals/sicomp/27-2/27069.html

†Department of Computer Science, Indiana University, 215 Lindley Hall, Bloomington, IN 47405 (bradford@cs.indiana.edu). Current address: Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany (pbradford@bfm.com).

‡Department of Computer Science, Indiana University, 215 Lindley Hall, Bloomington, IN 47405 (rawlins@cs.indiana.edu).

§Department of Computer Science, Indiana University, 215 Lindley Hall, Bloomington, IN 47405 (shannon@cs.indiana.edu). Current address: 2501 N. Loop Drive, ISU Research Park, Ames, IA 50010 (shannon@neb.infostructure.com).

MCOP. (See Bradford [11].) Recently in [40] and [41] Ramanan independently gave an $O(\lg^4 n)$-time and n-processor algorithm for solving the MCOP on the CREW PRAM. Moreover, in [15] we gave a $O(\lg^4 n)$-time and $n/\lg n$ processor algorithm for solving the MCOP on the common-CRCW PRAM.

Algorithm-design paradigms often aid the design of efficient sequential algorithms. However, some algorithm-design paradigms may not lead to efficient parallel algorithms. In particular, some variations of the greedy paradigm appear to be inherently sequential [4]. This highlights the significance of research in parallel dynamic programming.

1.1. Main results of this paper. Our approach follows [11], recasting the MCOP as a shortest path problem in a graph modeling a dynamic programming table. (In fact, this paper is an update of a part of [10] and a revision of [15]; see also the first author's dissertation [12].) This graph has $O(n^2)$ nodes and with an all-pairs shortest paths algorithm finding a shortest path in this graph results in a $n^6/\lg n$ processor MCOP algorithm. Reducing the number of nodes to $O(n)$ using a tree decomposition and applying an all-pairs shortest path algorithm gives an $n^3/\lg n$ processor and polylog-time algorithm.

In this paper, we convert the successive applications of the brute force all-pairs shortest paths algorithm to successive applications of parallel partial prefix and binary search algorithms. As in the $n^3/\lg n$-processor algorithm, the applications of the prefix and binary search algorithms are controlled by a rake-compress paradigm operating on a tree-based decomposition of the original graph. All of this results in a polylog-time ($O(\lg^3 n)$) and linear-processor ($n/\lg n$) parallel algorithm for the MCOP on the common-CRCW PRAM. This improves our result of $O(\lg^4 n)$ time and $n/\lg n$ processors of [15]. In addition, using efficient algorithms for computing row minima on totally monotone matrices, our algorithm can run in $O(\lg^2 n \lg\lg n)$ time using $n/\lg\lg n$ processors on a common-CRCW PRAM or in $O(\lg^2 n)$ time using n processors on an EREW PRAM. Using the most efficient polylog-time parallel algorithms for computing row minima on totally monotone matrices, our algorithm can run in $O(\lg^{1.5} n \lg\lg n)$ time using $O(n\sqrt{\lg n})$ work on a common-CRCW PRAM or in $O(\lg^{2.5} n\sqrt{\lg\lg n})$ time using $O(n\sqrt{\lg n \lg\lg n})$ processors on an EREW PRAM. See Bradford, Fleischer, and Smid [14] for the most efficient polylog-time parallel algorithms to date for computing row minima in totally monotone matrices.

1.2. Previous results. Elementary dynamic programming algorithms sequentially solve the matrix chain ordering problem in $O(n^3)$ time; see [3, 18]. Several recent textbooks on the design and analysis of parallel algorithms discuss the MCOP; see [36, 23]. However, the best sequential solution of the MCOP is Hu and Shing's $O(n \lg n)$ algorithm [28, 29]. (Much work has been done on lower bounds on the MCOP and related problems; see [38, 39, 13].) Using straight-line arithmetic programs, Valiant et al. [43] showed that many classical optimization problems with efficient sequential dynamic programming solutions are in $\mathcal{NC}$. Their algorithms require $\Theta(\lg^2 n)$ time and n^9 processors. Using pebbling games, Rytter [42] gave more efficient parallel algorithms for a similar class of optimization problems costing $O(\lg^2 n)$ time with $n^6/\lg n$ processors. In [11], an algorithm was given that takes $O(\lg^3 n)$ time and $n^3/\lg n$ processors, and [20] gave an algorithm that takes $O(\lg^3 n)$ time and $n^2/\lg^3 n$ processors. In [40], Ramanan gives an extended abstract of an n-processor and $O(\lg^4 n)$-time CREW PRAM algorithm for solving the MCOP which came after our buggy version in [10]; his full version appears in [41]. A full version of our $n/\lg n$-processor and $O(\lg^4 n)$-time algorithm described and improved upon in this paper appears in [15]. In addition, there are serial and parallel approximation algorithms for the MCOP [11, 17, 19, 27].

1.3. Structure of the paper. In section 2 we briefly review the interpretation of the MCOP as a shortest path graph problem from [11] and then summarize the $n^3/\lg n$-processor algorithm. In section 3 we isolate this algorithm's $n^3/\lg n$-processor bottlenecks. The $n^3/\lg n$-processor cost of these bottlenecks is from an all-pairs shortest paths algorithm. In section 4 we show how to replace the all-pairs shortest path algorithm with parallel prefix and an all-pairs *comparison* algorithm. In section 5 we replace the all-pairs comparison algorithm with applications of parallel prefix and binary search. Finally, it is shown that the key problems solved in section 4, and more efficiently in section 5, can be solved by finding the row minima of a totally monotone matrix.

2. An $O(\lg^3 n)$ time and $n^3/\lg n$ processor MCOP algorithm.

This section contains a brief review of the polylog-time and $n^3/\lg n$-processor MCOP algorithm from [11].

Let T be an $n \times n$ dynamic programming table for the matrix chain ordering problem. It has entries $T[i,k]$ representing the cheapest cost of the matrix product $M_i \bullet \cdots \bullet M_k$. For any such T there is a graph D_n where the cost of a shortest path to node (i,k), denoted $sp(i,k)$, is the same as the final value of $T[i,k]$. Given a chain of n matrices, finding a shortest path from $(0,0)$ to $(1,n)$ in D_n solves the MCOP [11].

The weighted digraph D_n has vertices in the set, $\{(i,j) : 1 \le i \le j \le n\} \cup \{(0,0)\}$ and edges

$$\{(i,j) \rightarrow (i,j+1) : 1 \le i \le j < n\} \cup \{(i,j) \uparrow (i-1,j) : 1 < i \le j \le n\}$$
$$\cup \{(0,0) \nearrow (i,i) : 1 \le i \le n\},$$

known as *unit* edges, together with the edges

$$\{(i,j) \Longrightarrow (i,t) : 1 < i < j < t \le n\} \cup \{(s,t) \Uparrow (i,t) : 1 \le i < s < t \le n\},$$

known as *jumpers*; see the jumper from $(1,2)$ to $(1,4)$ in Figure 1. The unit edge $(i,j) \rightarrow (i,j+1)$ represents the product $(M_i \bullet \cdots \bullet M_j) \bullet M_{j+1}$ and weighs $f(i,j,j+1) = w_i w_{j+1} w_{j+2}$, which is taken as the cost of multiplying a $w_i \times w_{j+1}$ matrix and a $w_{j+1} \times w_{j+2}$ matrix. Similarly, the unit edge $(i,j) \uparrow (i-1,j)$ represents the product $M_{i-1} \bullet (M_i \bullet \cdots \bullet M_j)$ and costs $f(i-1,i-1,j) = w_{i-1} w_i w_{j+1}$. A shortest path to (i,k) through the jumpers $(i,j) \Longrightarrow (i,k)$ and $(j+1,k) \Uparrow (i,k)$ represents the product $(M_i \bullet \cdots \bullet M_j) \bullet (M_{j+1} \bullet \cdots \bullet M_k)$, and these jumpers weigh $sp(j+1,k) + f(i,j,k)$ and $sp(i,j) + f(i,j,k)$, respectively, where $sp(j+1,k)$ is the cost of a shortest path to node $(j+1,k)$ and $f(i,j,k) = w_i w_{j+1} w_{k+1}$. The jumper $(i,j) \Longrightarrow (i,t)$ is of length $t-j$. See Figure 1.

Using this model the MCOP can be solved in polylog time with $n^6/\lg n$ processors by using an all-pairs shortest path algorithm and exploiting the following theorem.

THEOREM 1 (Duality Theorem [11]). *If a shortest path from $(0,0)$ to (i,k) contains the jumper $(i,j) \Longrightarrow (i,k)$, then there is a* dual *shortest path containing the jumper $(j+1,k) \Uparrow (i,k)$.*

Furthermore, using a tree decomposition of D_n and an all-pairs shortest path algorithm, the MCOP was solved in polylog time using $n^3/\lg n$ processors [11].

2.1. Matrix dimensions as nesting levels of matching parentheses. The next four subsections show that using the list of matrix dimensions as nesting levels of

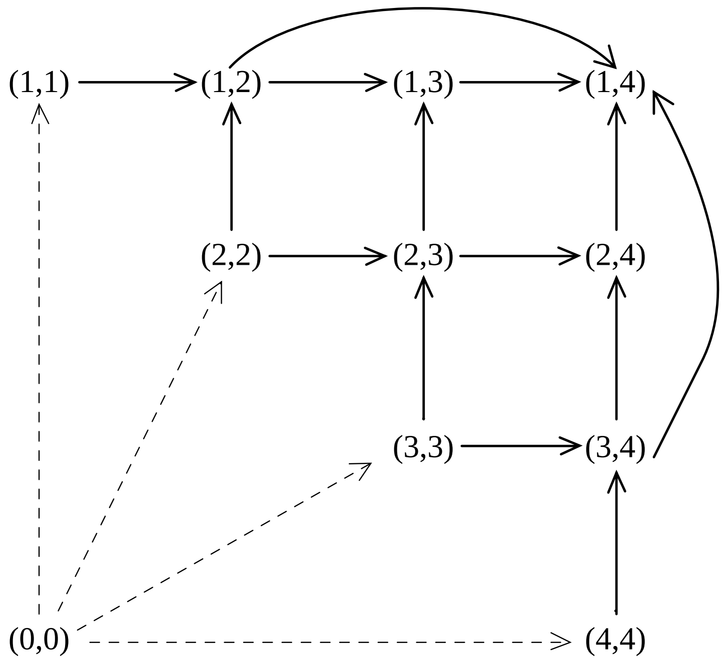

FIG. 1. *The weighted graph D_4.*

matching parentheses gives a tree decomposition of D_n that leads to efficient solutions of the MCOP.

Given an associative product with the level of each parenthesis known, for each parenthesis find its matching parenthesis by solving the all nearest smaller value (ANSV) problem [8, 9]: given weights $w_1, w_2, \ldots, w_n$, for each w find the indices, if they exist, of the nearest proceeding and succeeding weights both less than w. Let's call this pair of indices, if they exist, an ANSV *match.* That is, for each w the problem is to find the largest j where $1 \leq j < i$, and find the smallest k where $i < k \leq n$, so that $w_j < w_i$ and $w_k < w_i$, if such values exist. In D_n, (i, k) is a *critical node* if $[w_i, w_{k+1}]$ is an ANSV match.

By solving the ANSV problem we can compute all critical nodes of D_n. The bottom of Figure 2 depicts a list of matrix dimensions (called weights) and dashed lines representing four key ANSV matches. The four corresponding critical nodes are circled in D_n.

In our nomenclature, [8] shows that the following theorem holds.

THEOREM 2. *Computing all critical nodes costs $O(\lg n)$ time with $n/\lg n$ processors or in $O(\lg\lg n)$ time using $n/\lg\lg n$ processors on the common-CRCW PRAM.*

In addition, [16, 34] give the following theorem.

THEOREM 3. *Computing all critical nodes costs $O(\lg n)$ time with $n/\lg n$ processors on the EREW PRAM.*

Two critical nodes on the same diagonal are *compatible* if no vertices other than $(0, 0)$ can reach both of them by a unit path. Since a path of critical nodes represents a parenthesization, all critical nodes are compatible. Also, D_n has at most $n - 1$ critical nodes and there is at least one path from $(0, 0)$ to $(1, n)$ that includes all critical nodes [11].

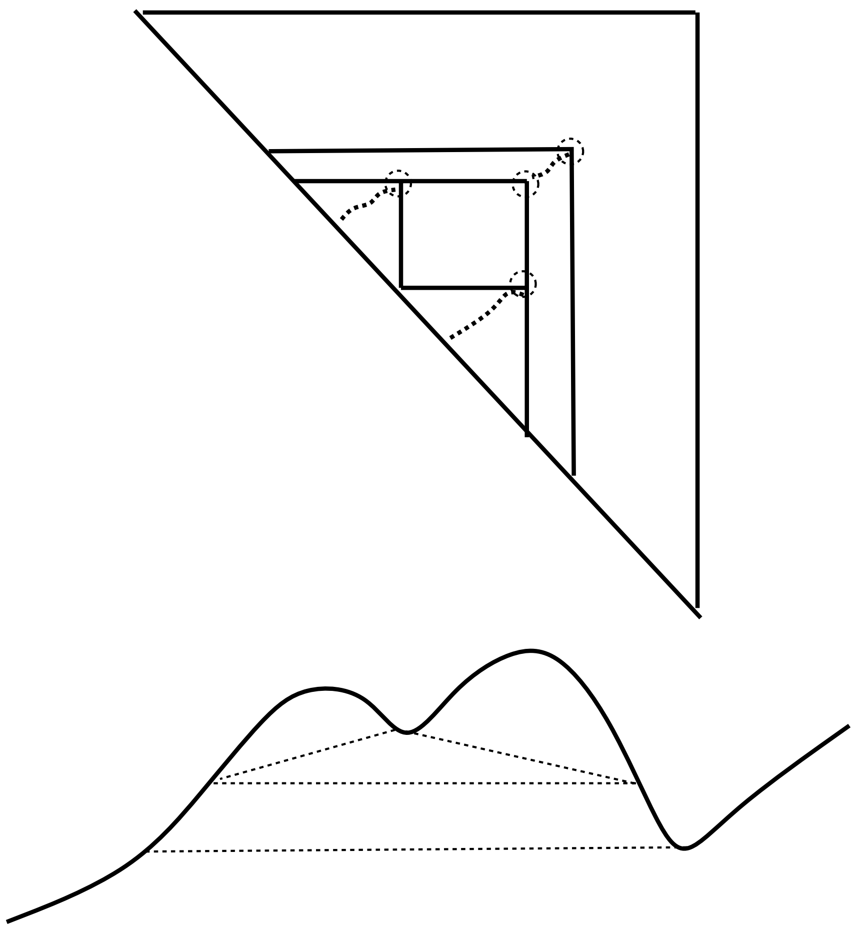
FIG. 2. *Two leaf subgraphs inside a band subgraph with critical nodes shown.*

2.2. Canonical subgraphs of D_n. In this subsection we investigate the interaction between subgraphs containing critical nodes.

All vertices and edges that can reach (i,t) by a unit path form the subgraph $D(i,t)$. Given $D(i,j)$ if the weight list $w_i, \ldots, w_{j+1}$ is monotonic, then $D(i,j)$ is monotonic. A *band canonical subgraph* $D^{(i,t)}_{(j,k)}$ is the subgraph containing the maximal unit edge-connected path of critical nodes beginning at critical node (j,k) and terminating at critical node (i,t) with the vertex set $\{(0,0)\} \cup (V[D(i,t)] - V[D(j+1,k-1)])$ and associated edges. A canonical subgraph of the form $D^{(i,t)}_{(j,j+1)}$ is a *leaf* canonical subgraph and is written $D^{(i,t)}$; it has the same nodes and edges as $D(i,t)$. The top of Figure 2 shows two leaf subgraphs nested inside of a band subgraph. Leaf and band subgraphs are the only two types of canonical subgraphs. Canonical subgraphs are easily distinguishable by the properties of their critical nodes shown in Theorem 2. From here on p denotes the path of critical nodes in band or leaf canonical subgraphs.

Given $D(i,u)$ with a monotone list of weights $w_i \le w_{i+1} \le \cdots \le w_{u+1}$, a shortest path from $(0,0)$ to (i,u) is the straight unit path $(0,0) \nearrow (i,i) \to (i,i+1) \to \cdots \to (i,u)$ that costs $w_i \sum_{j=i+1}^{u} w_j w_{j+1}$. On the other hand, if $D(i,t)$ has no

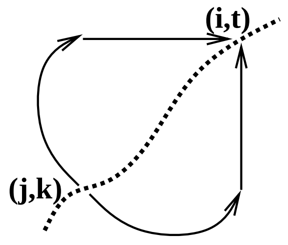

FIG. 3. *Two angular paths.*

critical nodes, then its associated weight list is monotonic. As in [28, 29, 11] let $\|w_i : w_k\| = \sum_{j=i}^{k-1} w_j w_{j+1}$, which is easily computable using differences of components of the parallel partial prefixes $\|w_1 : w_i\|$ for $2 \leq i \leq n+1$. This is useful since the unit path $(i,j) \to \cdots \to (i,k)$ costs $w_i\|w_{j+1} : w_{k+1}\| = w_i(\|w_1 : w_{k+1}\| - \|w_1 : w_{j+1}\|)$.

Suppose (j,k) and (i,t) are two critical nodes in a canonical graph such that from (j,k) we can reach (i,t) by a unit path, that is if $i \leq j \leq k \leq t$, then the *angular* paths of (j,k) and (i,t) are, (see Figure 3)

$$(j,k) \Uparrow (i,k) \to \cdots \to (i,t) \text{ and } (j,k) \Longrightarrow (j,t) \uparrow \cdots \uparrow (i,t).$$

THEOREM 4 (see [11]). *In a canonical subgraph the shortest path between any two critical nodes that contains no other critical nodes is an angular path or edge.*

In addition, any shortest path not including critical nodes is a straight path of unit edges. Thus, any shortest path to a critical node that contains no other critical nodes is a straight path of unit edges [11].

Now a polylog-time algorithm for finding shortest paths to all critical nodes in $D^{(1,m)}$ graphs is given. This algorithm takes $O(\lg^2 m)$ time and uses $m^3/\lg m$ processors.

First compute the parallel partial prefixes $\|w_1 : w_i\|$ for $2 \leq i \leq m+1$. Find all critical nodes. Now, in constant time using m processors compute the costs of all of the unit paths to nodes in p. Next compute the cost of the $O(m^2)$ angular paths in constant time with m^2 processors. Finally, compute the shortest path to each node in p by treating every angular path as an edge and applying a parallel all-pairs shortest path algorithm.

2.3. Combining the canonical graphs for an efficient parallel algorithm.

In this subsection we discuss a tree contraction algorithm that contracts the tree structure joining the canonical subgraphs to form a shortest path in D_n; see also [28, 29, 11].

In D_n a *canonical tree* joins all of the canonical subgraphs. A node in a canonical tree is critical node, say (i,j), and is written $\overline{(i,j)}$. Initially, for every leaf $D^{(i,j)}$ the critical node (i,j) is the tree leaf $\overline{(i,j)}$. Internal tree nodes are either isolated critical nodes or $\overline{(i,t)}$ and $\overline{(j,k)}$ in the band $D^{(i,t)}_{(j,k)}$. Tree edges are straight unit paths connecting tree nodes, and jumpers may reduce the cost of tree edges.

Given an instance of the MCOP with the weight list $l_1 = w_1, w_2, \ldots, w_{n+1}$, cyclically rotating it, getting l_2, and finding an optimal parenthesization for l_2 gives an optimal solution to the original instance of the MCOP with l_1, [28, 21]. So in the rest of this paper let w_1 denote the smallest weight in any weight list.

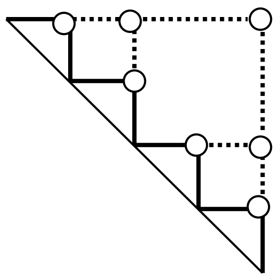

FIG. 4. *A tree of canonical graphs (the circles denote tree nodes).*

A result of Hu and Shing [28] leads directly to the next corollary.

COROLLARY 1 (Atomicity Corollary [11]). *Suppose a weight list* $w_1, \ldots, w_{n+1}$, *with the three smallest weights* w_1, w_{j+1}, *and* w_{k+1}, *is given such that* $1 < j < k-1$. *Then the critical nodes* $(1,j)$ *and* $(1,k)$ *are in a shortest path from* $(0,0)$ *to* $(1,n)$ *in* D_n.

For this corollary to work it is central that if w_1, w_{j+1}, and w_{k+1} are the three smallest weights; then $j+1 > 2$ and $k > j+1$. This generally means that Corollary 1 cannot be applied in a canonical subgraph. For instance, take the leaf $D^{(1,m)}$ where we can assume $w_1 < w_{m+1} < w_i$ for $1 < i < m+1$. However, Corollary 1 can be used to break D_n into a tree of canonical graphs; see Figure 4.

If D_n has fewer than $n-1$ critical nodes, then D_n may have disconnected canonical trees and monotone subgraphs. But there is at least one path joining these subtrees, and at the same time we can discount the monotone subgraphs. There are several relationships canonical graphs may have; these follow directly from the relationships of critical nodes that are tree nodes.

The tree edge $\overline{(i,j)} \to \cdots \to \overline{(i,v)}$ along row i initially costs $w_i \| w_{j+1} : w_{v+1} \|$ where $w_i < w_{v+1} < w_{j+1}$ are the three smallest weights in $D(i,v)$. Let $\overline{p}$ denote a shortest path of critical nodes in $D(j+1,v)$ from $(j+1,v)$ back to $(0,0)$. *Edge minimizing* the unit path along the ith row to the critical node $\overline{(i,v)}$ is done as follows. First let $L = w_i \| w_{i+1} : w_{v+1} \|$ and $W((i,k) \Longrightarrow (i,u)) = sp(k+1,u) + f(i,k,u)$, then compute

$$A[i,v] = \min_{\forall (k+1,u) \in V[\overline{p}]} \{L,\ w_i \| w_{i+1} : w_{v+1} \| - w_i \| w_{k+1} : w_{u+1} \| + W((i,k) \Longrightarrow (i,u))\}.$$

Since the three smallest weights in $D(i,v)$ are $w_i < w_{v+1} < w_{j+1}$, by Corollary 1 the cheapest cost to critical node (i,v) is now in $A[i,v]$.

THEOREM 5 (see [11]). *When edge minimizing a tree edge* $\overline{(i,j)} \to \cdots \to \overline{(i,v)}$ *in a canonical subgraph we only have to consider jumpers* $(i,k) \Longrightarrow (i,t)$ *such that* $(k+1,t) \in V[\overline{p}]$.

The critical node (i,u) in the band $D^{(i,u)}_{(j,k)}$ is the *front* critical node. In general, Theorem 5 holds when $\overline{p}$ is a shortest path through a band from the front critical node back to $(0,0)$. Also, Theorem 5 holds for leaves in the canonical tree that, after raking, have become conglomerates of other leaves, bands, and isolated critical nodes. Here, jumpers derived from critical nodes in different subtrees are independent so we can minimize tree edges with them simultaneously.

2.4. Contracting a canonical tree. In this subsection we show how to contract a canonical tree efficiently in parallel.

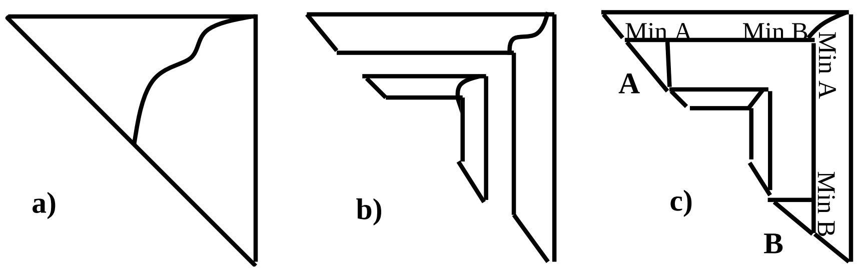

Fig. 5. *Bottlenecks* 1, 2, *and* 3 *for the* $n^3/\lg n$*-processor algorithm.*

Assume that all critical nodes (i, j) in tree leaves have the minimum cost back to $(0, 0)$ stored in $sp(i, j)$. Compute these values using an all-pairs shortest path parallel algorithm. There is an ordering of the leaves that prevents the simultaneous raking of two adjacent leaves. Given two neighboring leaves $D^{(i,j)}$ and $D^{(j+1,k)}$ with the three tree leaves $\overline{(i, j)}, \overline{(j + 1, k)}$, and $\overline{(i, k)}$, assume $w_i < w_{k+1} < w_{j+1}$. Then leaf $\overline{(j + 1, k)}$ must be raked, since $\overline{(i, j)}$ is in a shortest path from $(0, 0)$ to $\overline{(i, k)}$. Use the Euler tour technique [33] when the raking order is arbitrary.

Given two nested bands, assume $D^{(i,v)}_{(j,u)}$ is nested around $D^{(k,t)}_{(r,s)}$, that is, $j \leq k < t \leq u$. Without loss of generality, suppose any trees between $D^{(i,v)}_{(j,u)}$ and $D^{(k,t)}_{(r,s)}$ have been contracted. Then joining these bands costs $O(\lg^2 n)$ time with $n^3/\lg n$ processors. To achieve this, first, edge minimize all straight unit paths in $D^{(i,v)}_{(j,u)}$ with the shortest paths from critical nodes that are between $D^{(i,v)}_{(j,u)}$ and $D^{(k,t)}_{(r,s)}$ back to $(0, 0)$. Next, take all angular paths connecting these two bands and apply a parallel all-pairs shortest path algorithm merging the bands. Merging the bands $D^{(i,v)}_{(j,u)}$ and $D^{(j,u)}_{(r,s)}$ gives a shortest path from the front critical node (i, v) back to $(0, 0)$ through $D^{(i,v)}_{(r,s)}$. Incorporating this band merging with the tree contraction completes the polylog-time and $n^3/\lg n$-processor MCOP algorithm.

3. The structure of shortest paths in canonical subgraphs.

In this section we give the $n^3/\lg n$-processor bottlenecks of the algorithm in section 2. In addition, we give a metric for measuring the relative contributions of angular paths to shortest paths and some theorems about shortest paths forward from critical nodes in canonical graphs. From this section on, we only address rows in the canonical graphs; the arguments for columns follow immediately.

3.1. The $n^3/\lg n$ processor bottlenecks.

In this subsection we give the $n^3/\lg n$-processor bottlenecks of the algorithm sketched in section 2.

Three parts of the algorithm in section 2 use $n^3/\lg n$ processors. All other parts of this algorithm use a total of $n/\lg n$ processors and take $O(\lg n)$ time. The three bottlenecks are: finding shortest paths from all critical nodes in leaf graphs back to $(0, 0)$ (see Figure 5a); merging two bands (see Figure 5b); and merging two bands that have contracted canonical trees between them (see Figure 5c).

In Figure 5c, contracted trees **A** and **B** are used to edge minimize the unit paths marked by "**Min-A**" and "**Min-B**." Edge minimizing the unit paths in the outer band with the contracted trees gives an instance of the second bottleneck; see Figure 5b. Edge minimizing the unit paths in the outer band with the contracted trees costs $O(\lg n)$ time with $n^2/\lg n$ processors. In section 5 we will see how to perform such edge minimization in $O(\lg^2 n)$ time with $n/\lg n$ processors.

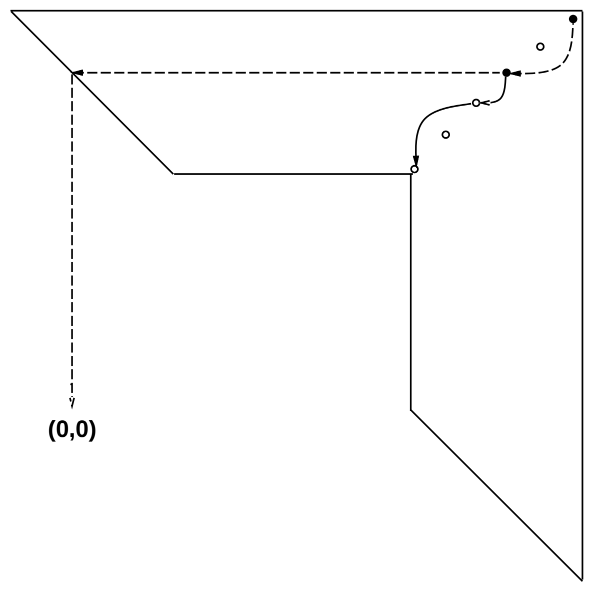

FIG. 6. *The dashed path is $\overline{p}$ and the two black nodes are supercritical nodes.*

Finding shortest paths back to $(0,0)$ from all critical nodes in a leaf graph, as in Figure 5a, will be done by breaking a leaf graph into nested bands. Therefore, finding efficient parallel methods of band merging and edge minimization will give an efficient parallel algorithm for the MCOP. So, the focus of the rest of the paper is finding efficient ways to get shortest paths from all critical nodes back to $(0,0)$ by edge minimization in leaf subgraphs partitioned as bands.

Given the band $D_{(j,t)}^{(i,v)}$, let $\overline{p}_{(j,t)}^{(i,v)}$ denote a shortest path from (i,v) back to $(0,0)$ totally contained in $D_{(j,t)}^{(i,v)}$; see Figure 6. When there is no ambiguity, $\overline{p}_{(j,t)}^{(i,v)}$ will be written as $\overline{p}$. Given a band $D_{(j,t)}^{(i,v)}$, whenever $\overline{p} = \overline{p}_{(j,t)}^{(i,v)}$ starts from the front critical node of the band it is in, the nodes $V[\overline{p}]$ are *super*critical nodes. Considering the minimal path back from the front critical node in Figure 6, we can see that only the two black critical nodes are supercritical nodes. Supercritical nodes of any band are all critical nodes in some minimal path back form the front critical node in the band back to $(0,0)$. Any two supercritical nodes in $\overline{p}$ are connected by supercritical nodes interspersed with the angular paths shown by Theorem 4.

When a canonical tree of D_n is totally contracted, then the final path $\overline{p}$ from $(1,n)$ back to $(0,0)$ gives the optimal order to multiply the set of n matrices. In addition, the cost of $\overline{p}$ is the minimal cost of multiplying the given chain of n matrices.

3.2. A metric for finding minimal cost angular paths. In this subsection we give a metric for finding minimal cost angular paths by using the equivalence of angular paths and jumpers along unit paths. This equivalence comes directly from Theorem 1.

When merging two bands, a unit path has at most one jumper minimizing it, since all the relevant jumpers are nested. These jumpers get their *sp* values from supercritical nodes of the inner band.

The influence of an angular edge can be taken as a jumper in a straight unit path by Theorem 1. In the case of Figure 5c, notice that any unit edge minimization using

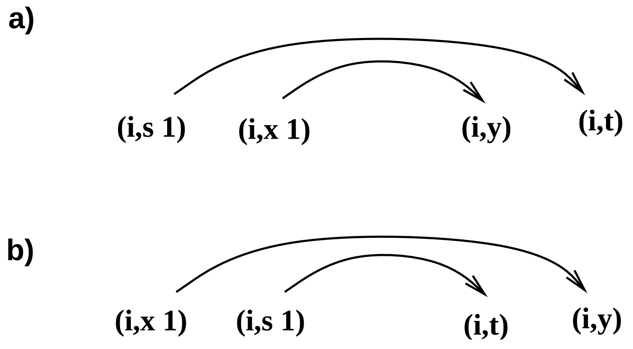

FIG. 7. *Two different nestings of two jumpers.*

sp values from $\mathbf{A}$ or $\mathbf{B}$ is independent of unit edge minimization using sp values from the inner band. Therefore, measuring the potential contribution of angular edges to shortest paths is done by measuring the potential contribution of jumpers to shortest paths along straight unit paths.

Take a node $(s,t) \in V[\overline{p}]$, where $sp(s,t)$ is the cost of a shortest path back to $(0,0)$ with respect to a band; then in row i we want to compare the cost of the jumper $(i,s-1) \Longrightarrow (i,t)$ with the cost of the associated unit path $(i,s-1) \rightarrow \cdots \rightarrow (i,t)$.

Given $(s,t) \in V[\overline{p}]$, take row i above p with the jumper $(i,s-1) \Longrightarrow (i,t)$; define

$$\Delta_i(s,t) = w_i \| w_s : w_{t+1} \| - [\ sp(s,t) + f(i,s-1,t)\ ].$$

If $\Delta_i(s,t) > 0$, then the jumper $(i,s-1) \Longrightarrow (i,t)$ provides a *cheaper* path along row i than the unit path $(i,s-1) \rightarrow \cdots \rightarrow (i,t)$. In particular, take both $(s,t) \in V[\overline{p}]$ and $(x,y) \in V[\overline{p}]$, and the two possible jumper nestings of Figure 7.

Considering the nesting of the jumpers in Figure 7a, if $\Delta_i(s,t) > \Delta_i(x,y) > 0$, then the jumper $(i,s-1) \Longrightarrow (i,t)$ "saves more" than the jumper $(i,x-1) \Longrightarrow (i,y)$ along row i because $(i,s-1) \Longrightarrow (i,t)$ doesn't have to deal with the paths $(i,s-1) \rightarrow \cdots \rightarrow (i,x-1)$ and $(i,y) \rightarrow \cdots \rightarrow (i,t)$ and $\Delta_i(s,t) > \Delta_i(x,y) > 0$. Similarly, for the nesting of the jumpers in Figure 7b, if $\Delta_i(x,y) > \Delta_i(s,t) > 0$, then the jumper $(i,x-1) \Longrightarrow (i,y)$ "saves more" than the jumper $(i,s-1) \Longrightarrow (i,t)$ along row i. Notice that considering the jumpers in Figure 7**b**, if $\Delta_i(s,t) > \Delta_i(x,y) > 0$, then the jumper $(i,s-1) \Longrightarrow (i,t)$ may or may not make row i cheaper than the jumper $(i,x-1) \Longrightarrow (i,y)$. On the other hand, in Figure 7b, if $(i,s-1) \Longrightarrow (i,t)$ makes row i cheaper than the jumper $(i,x-1) \Longrightarrow (i,y)$ does, then $\Delta_i(s,t) > \Delta_i(x,y) > 0$.

In $D^{(1,m)}$, if $(s,t) \in V[p]$, then above the path of critical nodes p the function $\Delta_i(s,t)$ is defined for all rows i such that $s > i \geq 1$.

Notice that edge minimizing a unit path is only half the game, for we also must consider the shortest paths forward.

Figure 8 is for the next theorem; also see [28].

THEOREM 6. *Let $D^{(i,v)}_{(r,s)}$ be a leaf graph and let (j,u) and (k,t) be any two critical nodes in $D^{(i,v)}_{(r,s)}$ such that there is a unit path from (k,t) to (j,u). Then a shortest path from (j,u) to (i,v) costs less than a shortest path from (k,t) to (i,v).*

A proof follows inductively by shadowing trivial angular paths without any jumpers, then showing that any shortest path from (k,t) forward to (i,v) can be

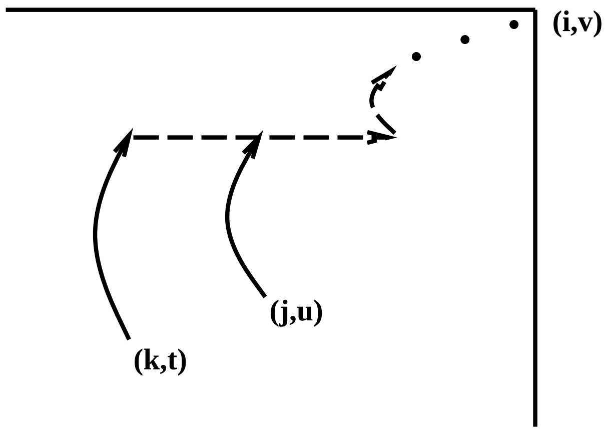

FIG. 8. (j, u) *shadowing* (k, t)*'s shortest path forward.*

"shadowed" by a shorter path from (j, u) forward to (i, v). While in the process we have taken into account the f values. Naturally, Theorem 6 also holds for shortest paths forward in leaf graphs.

The next theorem will also be useful.

THEOREM 7. *Let* $(i, s-1) \Longrightarrow (i, t)$ *be a shortest path forward. Suppose that the next band merging the value of* $sp(s, t)$ *decreases due to an edge minimization of row* s *or a lower row. Then* $(i, s-1) \Longrightarrow (i, t)$ *is still in a shortest path forward.*

A proof of this theorem follows directly from the basic notions of shortest paths. In particular, if the shortest path forward from the critical node (s, t) goes through $(s, t) \Uparrow (i, t)$, then making the path to (s, t) shorter will not affect the jumper $(s, t) \Uparrow (i, t)$ or the path from (i, t) to the front node of the present band.

4. A polylog-time and $n^2/\lg n$-processor MCOP algorithm.

In this section we give an $O(\lg^2 n)$-time and $n^2/\lg n$-processor algorithm for the MCOP. This algorithm works by using a key induction invariant that allows recursive doubling techniques to break through the bottlenecks given in the last section.

The basic idea of the algorithm is as follows. All critical nodes know their shortest paths to the front of the present bands they are in. Only supercritical nodes have their shortest paths back to $(0, 0)$ through their present bands. When merging two bands, by Theorem 5, we only have to consider shortest paths from supercritical nodes in the inner band to any critical node in the outer band. Therefore, all critical nodes must maintain a shortest path to the front of the band they are in. At the same time, all supercritical nodes must maintain a shortest path backwards to $(0, 0)$ through the band they are in. Much of this section supplies the details and correctness of this algorithm.

Each critical node in D_n has two pointers called *front-ptr* and *back-ptr* that represent angular edges. *Back-ptr*s are only used by supercritical nodes. With each *front-ptr* there are two values, *cost-of-front-ptr* and *cost-to-front*; and with each *back-ptr* there is one value, *cost-to-back*. *Cost-of-front-ptr* is the cost of the angular edge going forward to the front critical node in the present band, where the value of *cost-to-front* is the entire cost to the front critical node of the present band containing *front-ptr*. Similarly, the value of *cost-to-back* is the cost from the supercritical node at hand back to $(0, 0)$ through the present band. Initially, these pointers connect critical nodes and tree edges in the canonical tree.

1. All critical nodes in both bands have their *front-ptr*s in trees of shortest paths that eventually go to supercritical nodes. The supercritical nodes have their *front-ptr*s form a linked list that goes to the front (super)critical nodes of their respective bands.

2. In the two bands both shortest paths back to $(0, 0)$ of supercritical nodes are known. These shortest paths of supercritical nodes are made of linked lists of *back-ptrs* from the front (super)critical nodes of each band back through their respective bands to $(0, 0)$.

FIG. 9. *Inductive invariant for band merging.*

Let $D^{(i,v)}_{(j,t)}$ and $D^{(j,t)}_{(k,s)}$ be nested bands with paths of critical nodes labeled $p^{(i,v)}_{(j,t)}$ and $p^{(j,t)}_{(k,s)}$, respectively. Note (i, v) and (j, t) are the front critical nodes of these bands. As before, $\overline{p}^{(i,v)}_{(j,t)}$ and $\overline{p}^{(j,t)}_{(k,s)}$ are shortest paths from the front critical nodes back to $(0, 0)$ through the bands $D^{(i,v)}_{(j,t)}$ and $D^{(j,t)}_{(k,s)}$, respectively. Let $\overline{p}^{(i,v)}_{(j,t)}$ and $\overline{p}^{(j,t)}_{(k,s)}$ be made by two linked lists of *back-ptrs* along supercritical nodes back to $(0, 0)$ in their bands. It turns out that the shortest paths forward form all critical nodes in each of these bands and are made up of linked lists of trees of *front-ptr*s. We will see that this linked list of trees of *front-ptr*s is interconnected through the supercritical nodes as in Figure 10.

Figure 9 gives the induction invariant for merging two bands.

Figure 10 gives an example of the data structures for maintaining the inductive invariant. In this figure only critical nodes are shown and the supercritical nodes are black. The solid arrows are *front-ptr*s and the dashed arrows are *back-ptr*s.

Now, say (s, t) is a critical node but not a supercritical node, that is $(s, t) \in V[p]$ and $(s, t) \notin V[\overline{p}]$. There is a unique angular edge $(x, y) \Uparrow (r, y) \rightarrow \cdots \rightarrow (r, u)$ in $\overline{p}$ that "goes around" (s, t); see Figure 11. If we consider all rows above p in a given canonical graph, then $w_i < w_r$ implies that row i is "above" row r as in Figure 11. From here on we focus on finding shortest paths above the path p of critical nodes. The symmetric case of shortest paths below the path p of critical nodes follows.

Once we edge minimize all unit paths in $D^{(i,v)}_{(j,k)}$ with jumpers that get their *sp* values from supercritical nodes in $D^{(j,k)}_{(s,t)}$, then we can find the shortest path from (i, v) back to $(0, 0)$ through $D^{(i,v)}_{(s,t)}$. First, take one processor at each critical node in $D^{(i,v)}_{(j,k)}$ that sums the cost of the path back to $(0, 0)$, possibly through an edge-minimized unit path with the cost of its shortest path forward. Next, find the minimum of all of these sums, giving the shortest path from (i, v) back to $(0, 0)$ through $D^{(j,k)}_{(s,t)}$.

The basic intuition for the next lemma is that, if the shorter of two nested jumpers edge minimizes a unit path r, then any unit path above r with both of these jumpers is not minimized by the longer jumper; see Figure 12. For the next lemma, assume there is a unit path of critical nodes from (x, y) to (s, t) to (r, u) as in Figure 11.

LEMMA 1. *Let (s, t) be a critical node between the* super*critical node (x, y) and the critical node (r, u) and suppose that $i < r < s < x$ and row i is above row r. That is, $w_i < w_r$, where rows i and r are above p. Then*

$$\textbf{if } \Delta_r(x, y) \geq \Delta_r(s, t), \textbf{ then } \Delta_i(x, y) \geq \Delta_i(s, t).$$

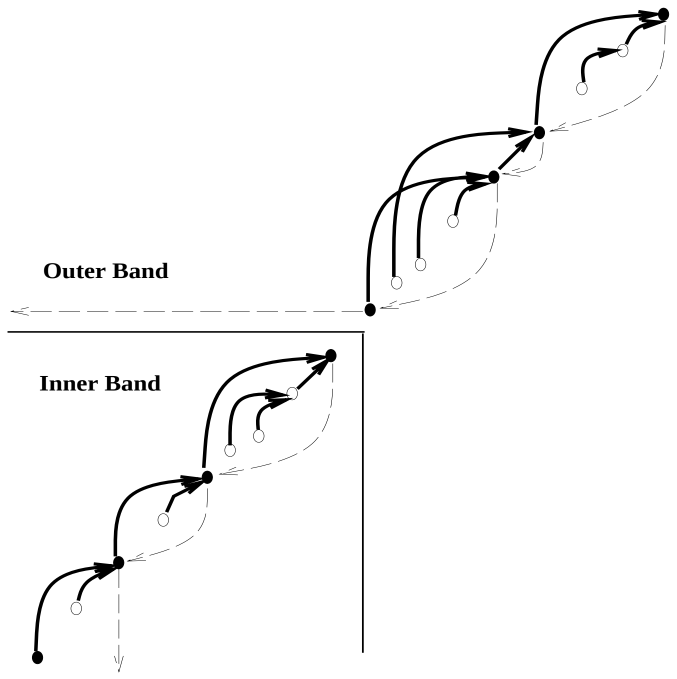

FIG. 10. *Solid arrows: forward linked lists of trees; dashed arrows: backward linked lists* $\overline{p}$*.*

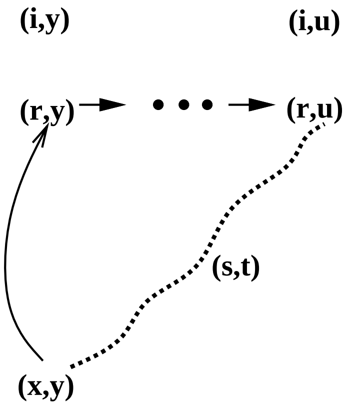

FIG. 11. $(s,t) \notin V[\overline{p}]$ *and the angular edge* $(x,y) \Uparrow (r,y) \to \cdots \to (r,u)$*.*

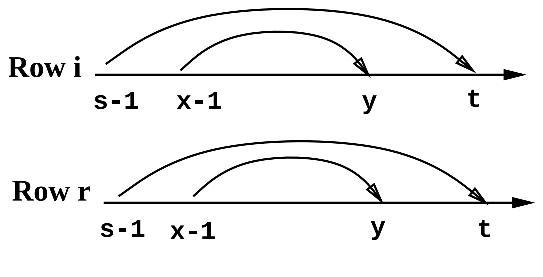

FIG. 12. *Two jumpers in different rows.*

Proof. Suppose $\Delta_r(x,y) \geq \Delta_r(s,t)$. This means

$$w_r\|w_x : w_{y+1}\| - [sp(x,y) + f(r,x-1,y)] \geq w_r\|w_s : w_{t+1}\| - [sp(s,t) + f(r,s-1,t)].$$

Using some algebra we obtain the following (where $\|w_i : w_i\| = 0$):

$$w_r[\ \|w_s : w_x\| + \|w_{y+1} : w_{t+1}\|\ ] < sp(s,t) - sp(x,y) + w_r(w_s w_{t+1} - w_x w_{y+1}).$$

Moreover, $sp(s,t) - sp(x,y)$ is always positive because $(r,x-1) \Longrightarrow (r,y)$ is nested inside of $(r,s-1) \Longrightarrow (r,t)$ and $f(r,s-1,t) < f(r,x-1,y)$. Therefore, if $sp(x,y) > sp(s,t)$, then a shortest path $\overline{p}$ would go through (s,t) to (r,u) and *not* over (s,t). In particular, if $sp(x,y) > sp(s,t)$, then since $f(r,x-1,t) > f(r,s-1,t)$, it must be the case that $sp(x,y) + f(r,x-1,t) > sp(s,t) + f(r,s-1,t)$. Therefore, row r would have been edge minimized by jumper $(r,s-1) \Longrightarrow (r,t)$ and *not* by $(r,x-1) \Longrightarrow (r,y)$; see Figure 12.

In addition, $w_s w_{t+1} - w_x w_{y+1} < 0$, since both (x,y) and (s,t) are critical nodes where $s \leq x < y \leq t$. So it must be that $w_x w_{y+1} - w_s w_{t+1} > 0$. Therefore, since

$$w_r[\ \|w_s : w_x\| + \|w_{y+1} : w_{t+1}\| + w_x w_{y+1} - w_s w_{t+1}\ ] < sp(s,t) - sp(x,y)$$

holds, and because $w_i < w_r$ and the term $sp(s,t) - sp(x,y)$ is independent of i and r, then $\Delta_i(x,y) \geq \Delta_i(s,t)$ follows. $\square$

The next theorem follows from Lemma 1.

THEOREM 8. *Let (s,t) be a critical node between the* super*critical node (x,y) and the critical node (r,u). Suppose $i < r < s < x$ and row i is above row r, that is, $w_i < w_r$, where rows i and r are above p. Then*

if $(r,x-1) \Longrightarrow (r,y)$ makes row r cheaper than $(r,s-1) \Longrightarrow (r,t)$ does,
then $(i,x-1) \Longrightarrow (i,y)$ makes row i cheaper than $(i,s-1) \Longrightarrow (i,t)$ does.

A proof follows from Lemma 1 and by the fact that the rows

$$(i,s-1) \rightarrow \cdots \rightarrow (i,x-1) \text{ and } (i,y) \rightarrow \cdots \rightarrow (i,t)$$

are cheaper than the rows

$$(r,s-1) \rightarrow \cdots \rightarrow (r,x-1) \text{ and } (r,y) \rightarrow \cdots \rightarrow (r,t),$$

and the change in f values between $(r,x-1) \Longrightarrow (r,y)$ and $(i,x-1) \Longrightarrow (i,y)$ is greater than the change of f values between $(r,s-1) \Longrightarrow (r,t)$ and $(i,s-1) \Longrightarrow (i,t)$. That is,

$$f(r,x-1,y) - f(i,x-1,y) > f(r,s-1,t) - f(i,s-1,t),$$

since w_x and w_{y+1} are both bigger than w_s and w_{t+1}. In addition, $w_r > w_i$; therefore

$$(w_r - w_i)[w_x w_{y+1} - w_s w_{t+1}] > 0.$$

Consider two nested bands with paths of critical nodes p_i for the inner band and p_o for the outer band, where $\overline{p_i}$ and $\overline{p_o}$ are shortest paths from the front critical nodes back to $(0,0)$ in each of these bands. Now, suppose (s,t) is between (x,y) and (r,u) and $(s,t) \in V[\overline{p_i}]$. If $(x,y) \in V[\overline{p_i}]$ and $(r,u) \in V[p_o]$, then Lemma 1 and Theorem 8

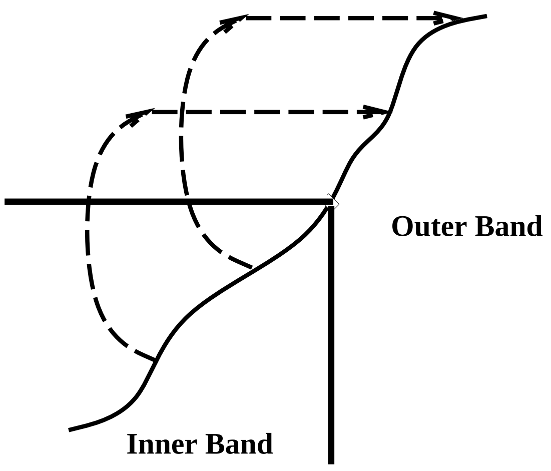

FIG. 13. *Conflicting angular paths* between *two bands being merged.*

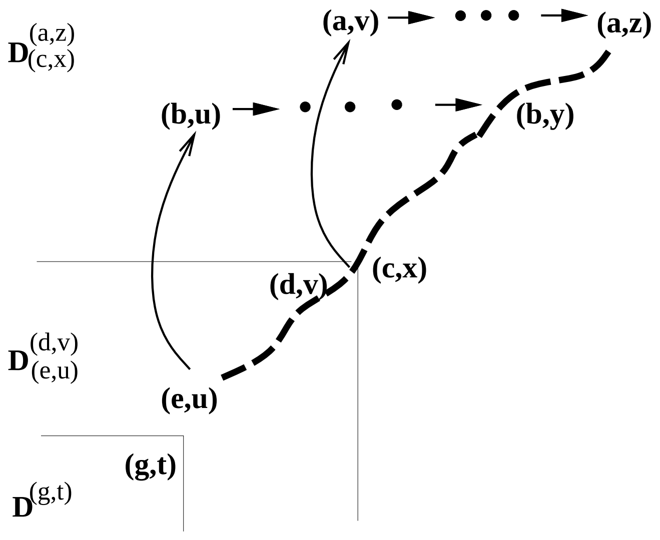

FIG. 14. *The bands* $D^{(a,z)}_{(c,x)}$, $D^{(d,v)}_{(e,u)}$ *and the leaf* $D^{(g,t)}$.

also hold. This is because $sp(s,t) - sp(x,y)$ is positive by an argument similar to that in the proof of Lemma 1.

Two angular edges above p, say $(x,y) \Uparrow (r,y) \to \cdots \to (r,u)$ and $(i,j) \Uparrow (s,j) \to \cdots \to (s,t)$, are *compatible* if they don't cross each other. Compatibility also holds for angular paths below p. Theorem 9 shows that *when merging* two bands and computing shortest paths forward, only compatible angular edges need to be considered. Figure 13 shows two conflicting angular paths.

Take the canonical graphs $D^{(a,z)}_{(c,x)}$, $D^{(d,v)}_{(e,u)}$ and $D^{(g,t)}$, where $D^{(g,t)}$ is nested inside of $D^{(d,v)}_{(e,u)}$ which is, in turn, inside of $D^{(a,z)}_{(c,x)}$; see Figure 14. Furthermore, assume that $D^{(a,z)}_{(c,x)}$ and $D^{(d,v)}_{(e,u)}$ are to be merged together. Then, in the next recursive doubling step, the new band $D^{(a,z)}_{(e,u)}$ will be merged with the leaf $D^{(g,t)}$. We can assume $D^{(g,t)}$ is a leaf or a band.

The next theorem assumes we have found a shortest path from supercritical nodes in $D^{(d,v)}_{(e,u)}$, through critical nodes in the outer band $D^{(a,z)}_{(c,x)}$; see Figure 14. We know $(d,v) \in V[\overline{p}^{(d,v)}_{(e,u)}]$ and, without loss of generality, we can assume $(e,u) \Uparrow (d,u) \to \cdots \to (d,v)$ is $\overline{p}^{(d,v)}_{(e,u)}$. Now, suppose the angular edge $(e,u) \Uparrow (b,u) \to \cdots \to (b,y)$ is in $\overline{p}^{(a,z)}_{(e,u)}$, where $\overline{p}^{(a,z)}_{(e,u)}$ is the minimal path from (a,z) back to $(0,0)$ through $D^{(a,z)}_{(e,u)}$.

THEOREM 9 (Main Theorem). *In merging two nested bands computing shortest paths forward from supercritical nodes of the inner band, we only need to consider compatibly nested angular edges.*

Proof. The proof is by contradiction. Suppose $(e,u) \Uparrow (b,u) \to \cdots \to (b,y)$ is in $\overline{p}^{(a,z)}_{(e,u)}$, that is, the angular edge $(e,u) \Uparrow (b,u) \to \cdots \to (b,y)$ is in a shortest path from (a,z) back to $(0,0)$ through $D^{(a,z)}_{(e,u)}$; see Figure 14. Suppose (d,v) is in the band $D^{(d,v)}_{(e,u)}$. Therefore (d,v) is between (e,u) and (a,z) in $D^{(a,z)}_{(e,u)}$. Now, when merging $D^{(g,t)}$ with $D^{(a,z)}_{(e,u)}$ we will show that a shortest path forward to (a,z) that goes through (d,v) must go through a critical node in row b or a critical node in a row below b.

Now, for the sake of a contradiction, assume otherwise. Suppose after merging $D^{(a,z)}_{(c,x)}$ with $D^{(d,v)}_{(e,u)}$ there is some shortest path from $(0,0)$ through (d,v) to (a,z). This shortest path travels through an angular path connecting the bands $D^{(a,z)}_{(c,x)}$ and $D^{(d,v)}_{(e,u)}$ and this angular path is conflicting with the angular path $(e,u) \Uparrow (b,u) \to \cdots \to (b,y)$. Say, without loss of generality, this conflicting angular path is $(d,v) \Uparrow (a,v) \to \cdots \to (a,z)$; see Figure 14. That is, we have conflicting angular paths since the shortest path from (d,v) forward goes through an angular path that terminates above row b, and the shortest path forward from (e,u) goes through an angular path that terminates in row b. But notice *in* $D^{(a,z)}_{(e,u)}$ that the shortest path from (a,z) back to $(0,0)$ still goes through the angular edge $(e,u) \Uparrow (b,u) \to \cdots \to (b,y)$.

In $D^{(a,z)}_{(e,u)}$ the angular edge $(d,v) \Uparrow (a,v) \to \cdots \to (a,z)$ can't be the shortest path forward from (d,v).

By Theorem 1, the shortest path to (b,y) through the angular edge

$$(e,u) \Uparrow (b,u) \to \cdots \to (b,y)$$

is equivalent to the path

$$(b,b) \to \cdots \to (b,e-1) \Longrightarrow (b,u) \to \cdots \to (b,y).$$

Moreover, since $\overline{p}^{(a,z)}_{(e,u)}$ goes through $(e,u) \Uparrow (b,u) \to \cdots \to (b,y)$, the jumper $(b,e-1) \Longrightarrow (b,u)$ edge minimizes row b. Thus, the jumper $(b,d-1) \Longrightarrow (b,v)$ saves at most as much as $(b,e-1) \Longrightarrow (b,u)$, and $(b,d-1) \Longrightarrow (b,v)$ is nested around $(b,e-1) \Longrightarrow (b,u)$. Thus,

$$\Delta_b(e,u) \geq \Delta_b(d,v).$$

Also, by Theorem 1, the shortest path from (d,v) to (a,z) that goes through the angular edge

$$(d,v) \Uparrow (a,v) \to \cdots \to (a,z)$$

is equivalent to the path

$$(a,a) \to \cdots \to (a,d-1) \Longrightarrow (a,v) \to \cdots \to (a,z).$$

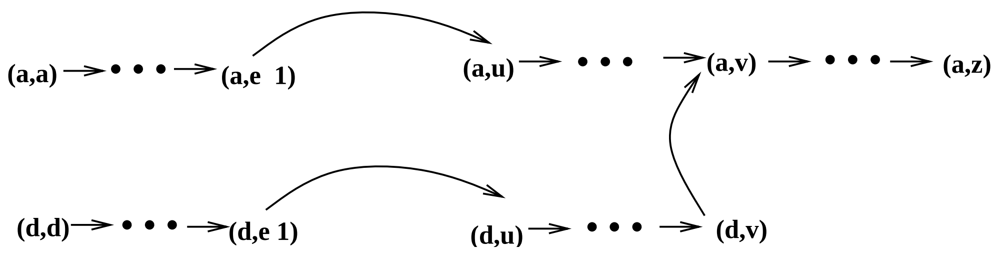

FIG. 15. *The two paths $\mathcal{A}$ and $\mathcal{D}$.*

But, consider the path

$$(a,a) \to \cdots \to (a,e-1) \Longrightarrow (a,u) \to \cdots \to (a,z),$$

and we know that the jumper $(a,e-1) \Longrightarrow (a,u)$ is nested inside of $(a,d-1) \Longrightarrow (a,v)$. In this case, it is possible that $d = e$ or $u = v$, but not both, since (d,v) is between (e,u) and (b,y) and $a < b$ and $w_a < w_b$, where row a is above row b and they are both above p. Furthermore, since the appropriate Δ values are defined, the following holds by Lemma 1:

$$\textbf{if } \Delta_b(e,u) \geq \Delta_b(d,v), \textbf{ then } \Delta_a(e,u) \geq \Delta_a(d,v).$$

Therefore,

$$\Delta_a(e,u) \geq \Delta_a(d,v),$$

which means the jumper $(a,e-1) \Longrightarrow (a,u)$ saves at least as much as the jumper $(a,d-1) \Longrightarrow (a,v)$ in a path to (a,z).

By Theorem 8, since $(b,e-1) \Longrightarrow (b,u)$ edge minimizes row b, and $\Delta_a(e,u) \geq \Delta_a(d,v)$, the jumper $(a,e-1) \Longrightarrow (a,u)$ saves more in row a than $(a,d-1) \Longrightarrow (a,v)$.

Now, take the two paths

$$\mathcal{A} = (a,a) \to \cdots \to (a,e-1) \Longrightarrow (a,u) \to \cdots \to (a,v),$$
$$\mathcal{D} = (d,d) \to \cdots \to (d,e-1) \Longrightarrow (d,u) \to \cdots \to (d,v) \Uparrow (a,v)$$

as in Figure 15.

$\mathcal{A}$ is cheaper than $\mathcal{D}$ going from (a,z) back to $(0,0)$ in $D_{(e,u)}^{(a,z)}$ by Theorem 8. Now, if $(d,v) \Uparrow (a,v)$ is in a shortest path forward from (d,v), then the shortest path forward from (e,u) must be through the angular path $(e,u) \Uparrow (a,u) \to \cdots \to (a,z)$ and not the angular path $(e,u) \Uparrow (b,u) \to \cdots \to (b,y)$, which is a contradiction. This follows by applying Theorem 8 to the jumpers $(b,d-1) \Longrightarrow (b,v)$ and $(b,e-1) \Longrightarrow (b,u)$ in row b and then up to row a, since (b,y) is between (d,v) and (a,z).

Now, suppose $D^{(g,t)}$ is merged with the outer band $D_{(e,u)}^{(a,z)}$. Then, none of the angular paths connecting supercritical nodes in $D^{(g,t)}$ with paths forward $D_{(e,u)}^{(a,z)}$ change. This case is a straightforward application of the proof above and Theorem 7. $\square$

It is important to note that Theorem 9 shows that only angular paths starting from supercritical nodes in the same path back to $(0,0)$ are compatible. Theorem 9 doesn't say that all angular paths are always compatible.

Suppose that there is some angular path from a supercritical node in the inner band, say (s,t), to the outer band that is in a shortest path from the front node

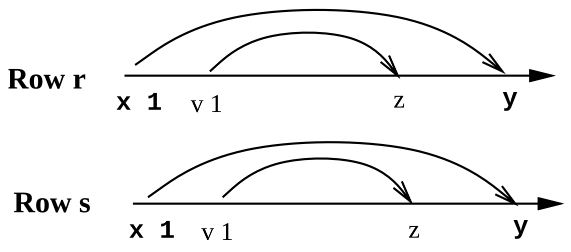

FIG. 16. *Two jumpers in different rows.*

of the outer band back to $(0,0)$. Then all supercritical nodes from (s,t) back to $(0,0)$ have their shortest paths forward through the angular path starting at (s,t). On the other hand, by Theorem 9 all supercritical nodes after (s,t) up to the front supercritical node of the inner band have their shortest paths through nested angular paths connecting the inner and outer bands. In fact, we can inductively apply this argument together with Theorem 6 giving the following corollary.

COROLLARY 2. *Consider the nested angular paths connecting two bands that are shortest paths forward from the different supercritical nodes of the inner band. Then, listing the path containing the outermost such angular path to the path containing the innermost such angular path gives more and more costly paths forward.*

The next lemma assumes we are merging two nested bands to find a shortest path from the front critical node of the outer band back to $(0,0)$.

LEMMA 2. *Let (s,t) be a critical node and let the ith and rth rows above p be such that $i < r < s$ and $w_i < w_r$. Then $\Delta_i(s,t) < \Delta_r(s,t)$.*

Proof. The function $\Delta_i(s,t)$ measures the potential minimizing effect of $(i,s-1) \Longrightarrow (i,t)$ on the path $(i,i) \to \cdots \to (i,u)$, where $(i,u) \in V[p]$ and $i < s < t \le u$. The cost of the jumper $(i,s-1) \Longrightarrow (i,t)$ is $sp(s,t) + f(i,s-1,t)$. Therefore, the difference $\Delta_{i+1}(s,t) - \Delta_i(s,t)$ is

$$(w_{i+1} - w_i)[\ \|w_s : w_{t+1}\| - w_s w_{t+1}\ ],$$

where $w_{i+1} > w_i$. Since the expression $\|w_s : w_{t+1}\| - w_s w_{t+1}$ is independent of the difference of weights w_i and w_{i+1} and $\|w_s : w_{t+1}\| - w_s w_{t+1} > 0$, because $(s,t) \in V[p]$. Also, when $s = t-1$ we have

$$\|w_s : w_{t+1}\| = w_s w_{s+1} + w_{s+1} w_{t+1}.$$

In addition, since $(s,t) \in V[p]$, it must be that $\max\{w_s, w_{t+1}\} < w_u$, for $s < u \le t$. Thus $\max\{w_s, w_{t+1}\} < w_{s+1}$. Therefore,

$$w_s w_{s+1} + w_{s+1} w_{t+1} > w_s w_{t+1}$$

and the proof follows inductively. □

The proof of the next lemma is similar to that of Lemma 1. The basic intuition here is that, if the longer of two nested jumpers edge minimizes a unit path r, then any unit path below r, with both of these jumpers, is not minimized by the shorter jumper; see Figure 16.

This next lemma only considers supercritical nodes since we are interested in merging two nested bands. For the next lemma assume there is a unit path of critical nodes from (v,z) to (x,y).

LEMMA 3. *Let (v, z) and (x, y) be two* super*critical nodes, where $r < s < x < v$, and assume $w_r < w_s$ such that rows s and r are above p. Then*

$$\textbf{if } \Delta_r(x, y) \geq \Delta_r(v, z), \textbf{ then } \Delta_s(x, y) \geq \Delta_s(v, z).$$

Proof. Assume $\Delta_r(x, y) \geq \Delta_r(v, z)$. Then

$$\begin{aligned} &w_r\|w_x : w_{y+1}\| - [\ sp(x, y) + f(r, x-1, y)\ ] \\ &\quad \geq w_r\|w_v : w_{z+1}\| - [\ sp(v, z) + f(r, v-1, z)\ ]. \end{aligned}$$

By Lemma 2 and, since each of these jumpers is of length at least 2, we know that $w_r w_v w_{z+1} < w_r\|w_v : w_{z+1}\|$ and $w_r w_x w_{y+1} < w_r\|w_x : w_{y+1}\|$. In addition, since $w_r < w_s$, we know that $f(r, x-1, y) < f(r, v-1, z)$ and $w_r\|w_x : w_{y+1}\| > w_r\|w_v : w_{z+1}\|$. Furthermore, the same holds in row s. Therefore, it must be that $\Delta_s(x, y) \geq \Delta_s(v, z)$. □

THEOREM 10. *Suppose we are given two* super*critical nodes (v, z) and (x, y), where $r < s < x < v$, and $w_r < w_s$ such that rows s and r are above p. Then*

if $(r, x-1) \Longrightarrow (r, y)$ *makes row r cheaper than* $(r, v-1) \Longrightarrow (r, z)$ *does,*

then $(s, x-1) \Longrightarrow (s, y)$ *makes row s cheaper than* $(s, v-1) \Longrightarrow (s, z)$ *does.*

A proof of this theorem follows from Lemma 3 and the fact that the change of the f values between $(r, v-1) \Longrightarrow (r, z)$ and $(s, v-1) \Longrightarrow (s, z)$ increases faster than the change in the f values between $(r, x-1) \Longrightarrow (r, y)$ and $(s, x-1) \Longrightarrow (s, y)$.

While merging $D^{(i,v)}_{(j,t)}$ and $D^{(j,t)}_{(k,s)}$ to form $\overline{p}^{(i,v)}_{(k,s)}$, the next lemma shows that we only need shortest path values backwards to $(0, 0)$ from supercritical nodes and we don't need shortest path values backwards to $(0, 0)$ from any other critical nodes. Hence, the *back-ptr*s will form a linked list between supercritical nodes backwards eventually to $(0, 0)$, and we can compute the *cost-to-back* weights using a parallel pointer jumping partial prefix computation.

LEMMA 4. *Suppose we are given $\overline{p}^{(i,v)}_{(j,t)}$ and $\overline{p}^{(j,t)}_{(k,s)}$ in $D^{(i,v)}_{(j,t)}$ and $D^{(j,t)}_{(k,s)}$, respectively. Consider critical nodes in the outer band, say $(u, z) \in V[p^{(i,v)}_{(j,t)}]$ and $(u, z) \notin V[\overline{p}^{(i,v)}_{(k,s)}]$; then we don't need shortest paths* back *to* $(0, 0)$.

Proof. Consider the angular path $(x, y) \Uparrow (q, y) \rightarrow \cdots \rightarrow (u, z)$ between $D^{(j,t)}_{(k,s)}$ and $D^{(i,v)}_{(j,t)}$. That is, $(x, y) \in V[\overline{p}^{(j,t)}_{(k,s)}]$ and $(u, z) \in V[p^{(i,v)}_{(j,t)}]$. But, suppose $(u, z) \notin V[\overline{p}^{(i,v)}_{(k,s)}]$ is the case. Notice that (u, z) may be a supercritical node in $\overline{p}^{(i,v)}_{(j,t)}$.

Consider the following cases.

Case i: Suppose $D^{(i,v)}_{(k,s)}$ is merged with another band nested around it.

Then, since (u, z) is not in $V[\overline{p}^{(i,v)}_{(k,s)}]$, by Theorem 5 we do not have to consider any angular paths starting from (u, z) going forward to critical nodes in the band nested around $D^{(i,v)}_{(k,s)}$.

Case ii: Suppose $D^{(i,v)}_{(k,s)}$ is merged with a smaller band inside $D(k, s)$.

Node (u, z) could be the terminal node of an incoming angular path contributing to a shortest path forward for *some* supercritical node in $D(k, s)$. In this case (u, z) needs to have a shortest path from (u, z) forward. Of course, in this case (u, z) could become a supercritical node and would have a minimal path back to $(0, 0)$. On the other hand, since the critical node (u, z) is not a *super*critical node it has no need of a shortest path back to $(0, 0)$. □

We want to find a shortest path forward for every critical node, since some angular path from some future inner band may terminate at any critical node. Therefore, after finding each supercritical node's minimal cost to the front critical node of the outer band, then compute a tree partial prefix sum from the critical nodes to the supercritical nodes. This lets all critical nodes know their shortest paths to the front of the outer band.

Suppose, through recursive doubling, we generate the band $D_{(j,t)}^{(i,v)}$ and the shortest path $\overline{p}_{(j,t)}^{(i,v)}$ from (i, v) back to $(0, 0)$ in this band. The next theorem shows that we can build the appropriate data structures to maintain the inductive invariant through recursive doubling.

THEOREM 11. *Suppose we have just merged any two nested bands into a new band. Then the front pointers of the new band form a tree and the back pointers of the new band form a linked list.*

There is a proof by induction based on Theorem 9.

Theorem 11 shows that the inductive invariant holds given the appropriate data structures and computations.

4.1. Merging bands using $n^2/\lg n$ processors. In this subsection we show how to merge two bands using $n^2/\lg n$ processors in $O(\lg n)$ time. This algorithm also merges two optimally triangulated convex polygons when all of the weights of one polygon are heavier than all of the weights of the other. Given a triangle with vertices w_i, w_j, and w_k its cost is $w_i w_j w_k$; also see [18, 28].

Recursively doubling the band merging algorithm while using the proper data structures and appropriate tree contracting gives the $n^2/\lg n$-processor and $O(\lg^3 n)$-time MCOP algorithm.

The algorithm in Figure 17 merges two bands in $O(\lg n)$ time using $n^2/\lg n$ processors. Adding the cost of recursive doubling and tree contraction gives a factor of $O(\lg^2 n)$ time to the entire algorithm, making the total cost for solving the MCOP $O(\lg^3 n)$ time using $n^2/\lg n$ processors.

The two **for** loops in step 1 of the algorithm in Figure 17 perform the edge minimizing. This is the only part of this algorithm that uses $n^2/\lg n$ processors. In $O(\lg n)$ time using $n^2/\lg n$ processors we can edge minimize unit paths with contracted trees such as those depicted in the bottleneck of Figure 5c.

The **for** loops in step 2 compute the supercritical nodes of the band that are being created by merging. Step 3 computes the shortest paths forward for all critical nodes in the inner band.

The base case for the recursive doubling can be established by breaking the canonical subgraphs into bands of constant width. Then for each band sequentially, let the $n/\lg n$ processors set up the inductive invariant in $O(\lg n)$ time. Number the nested bands consecutively according to their nestings by the Euler tour technique so the algorithm can track adjacent bands for merging.

The correctness of the algorithm in Figure 18 comes from Theorems 7, 9, and 11.

The time complexity of solving the MCOP can be reduced to $O(\lg^2 n)$ time with $n^2/\lg n$ processors by performing band merging and then tree contraction.

THEOREM 12. *Recursive doubling with band merging can be done simultaneously with tree contraction, thereby solving the MCOP in $O(\lg^2 n)$ time with $n^2/\lg n$ processors.*

Proof. Take any canonical tree T with nontrivial bands and leaves. Then T has at most $n - 1$ critical nodes. In general, for any arithmetic expression tree with $n - 1$ nodes, it takes $O(\lg n)$ time to contract it. In a canonical tree we have just seen

```
Take two adjacent nested bands, say $D^{(i,v)}_{(j,t)}$ nested around $D^{(j,t)}_{(k,s)}$, such that for
each band individually the inductive invariant holds.
1.  **for all** supercritical nodes $(x,y) \in V[\overline{p}^{(j,t)}_{(k,s)}]$ **in parallel do**
        **for all** angular edges from $(x,y)$ to all $(u,z) \in V[p^{(i,v)}_{(j,t)}]$ **in parallel do**
            find the angular edge between the bands that gives a shortest
                path from $(x,y)$ all the way to $(i,v)$, compute the
                *cost-of-front-ptr*s for these new edges
            let each supercritical node $(x,y)$ have a pointer to a shortest path
                through $p^{(i,v)}_{(j,t)}$ to $(i,v)$
            for the supercritical nodes in $\overline{p}^{(j,t)}_{(k,s)}$ put the angular edge that gives
                them a shortest path forward to $(i,v)$ in $M$
2.  **for all** angular edges in $M$ **in parallel do**
        find the shortest path $N$ from $(i,v)$ back to $(0,0)$ through $D^{(i,v)}_{(k,s)}$
    **for all** critical nodes in the path $N$ **in parallel do**
        using pointer jumping build the *back-ptr*s giving any new supercritical
        nodes and compute the values of *cost-to-back* for each new supercritical
        node
3.  **for all** *non*-supercritical nodes in $p^{(j,t)}_{(k,s)}$ **in parallel do**
        using pointer jumping expand the tree of *front-ptr*s through the new
            angular edges in $M$ and their minimal values to $(i,v)$. This gives
            trees joined by a linked list through the supercritical nodes.
        With this find the shortest path to $(i,v)$ for all non-supercritical
            nodes in $p^{(j,t)}_{(k,s)}$ by computing a partial prefix in a rooted tree.
            Also compute all of the new *cost-to-front* values using a parallel
            partial prefix.
```

FIG. 17. *An $O(\lg n)$-time and $n^2/\lg n$-processor algorithm for merging two bands.*

that each contraction operation (raking) can be done in $O(\lg n)$ time using $n^2/\lg n$ processors. This is because in the worst case a leaf raking operation in a canonical tree is the merging of two bands. Now, each band can be seen as no more than a linked list in the canonical tree that must be contracted where there is one leaf per list node. Now, we can simply take every band that has k critical nodes and is in any canonical tree, and we can assume that it has 2^c "linked list nodes" such that $2^{c-1} < k \leq 2^c$. With this, each raking operation will cost at most $O(\lg n)$ time using $n^2/\lg n$ processors. In addition, by assuming k is the nearest power of two greater than or equal to k, we are at most doubling the number of critical nodes in T. Hence the asymptotic bound we claim must hold. $\square$

5. An efficient polylog-time MCOP algorithm.

In this section we reduce the processor complexity of the band merging algorithm of section 4. The results of this section are based on a parallel divide-and-conquer form of binary search which is tied into some classical problems of finding row minima in totally monotone matrices.

Theorems 8 and 10 supply the basis for a parallel divide-and-conquer binary search algorithm that finds the jumpers that minimize each unit path in a canonical graph.

```
1.  Find the middle supercritical node in the inner band, say $(x-1, y)$.
2.  Using $m/\lg m$ processors and in $O(\lg m)$ time find a shortest path
    forward from $(x-1, y)$ to the front of the outer band. Suppose that this
    shortest path forward from the supercritical node $(x-1, y)$ has an angular
    edge between the two bands that terminates in row $r$.
3.  Split the jumpers into two sets:
    (a) Those smaller than or equal to $(r, x) \Longrightarrow (r, y)$; call them $S$. They are
        nested *inside* $(r, x) \Longrightarrow (r, y)$.
    (b) Those larger than or equal to $(r, x) \Longrightarrow (r, y)$; call them $L$. They are
        nested *around* $(r, x) \Longrightarrow (r, y)$.
4.  Do the following two steps in parallel:
    (a) Assign $|S|$ processors to rows $r$ up through 1 and recursively repeat this
        algorithm with the jumpers in $S$.
    (b) Assign $|L|$ processors to rows $r$ down through $m$ and recursively repeat
        this procedure with the jumpers in $L$.
```

FIG. 18. *An $O(\lg^2 n)$-time and $n/\lg n$-processor band merging algorithm.*

THEOREM 13. *Suppose that r is the row in the outer band such that the dual of $(r, x) \Longrightarrow (r, y)$ gives a shortest path forward from the supercritical node $(x-1, y)$ of the inner band to the front node of the outer band. Then to find shortest paths forward from other supercritical nodes,*

- *it is sufficient to consider only larger nested jumpers in any row s below row r, that is, $w_s > w_r$, and*
- *it is sufficient to consider only smaller nested jumpers in any row i above row r, that is, $w_i < w_r$.*

A proof of this theorem comes directly from Theorems 8 and 10.

The next algorithm replaces the two nested **for** loops in step 1 in the algorithm of Figure 17. This next algorithm gives shortest paths forward for all supercritical nodes originally in the inner band and a shortest path back to $(0, 0)$ through the two merged bands.

Assuming that each band has m critical nodes, the next procedure finds shortest paths from all supercritical nodes of the inner band to the front of the outer band. In addition, assume the shortest path information before the merging and all shortest paths to the front of the outer band. Then the shortest path back from the front node of the outer band is easily computed. As before, begin assuming the inductive invariant. Also, all jumpers in the next algorithm are jumpers that get their *sp* values from the inner band where the jumpers themselves are in unit rows or columns of the outer band.

The following algorithm is strikingly similar to those discussed in [1] and [2]. This key observation leads to some complexity improvements.

Now assign one processor to each unit path in the outer band. For each unit path, summing the cost to the critical node and the cost from the critical node to the front supercritical node of the outer band gives a shortest path backwards from the front node of the outer band to $(0, 0)$. These minimal paths can be computed in $O(\lg n)$ time using $n/\lg n$ processors. If a unit path has no edge minimizing jumpers, then this algorithm just finds the shortest path forward for all supercritical nodes in the inner band, since, in this case, the shortest path back to $(0, 0)$ from the front critical node of the outer band does not go through the inner band.

The algorithm in Figure 18 also breaks through the bottleneck of Figure 5c. It takes $O(\lg^2 n)$ time and uses $n/\lg n$ processors in the worst case. Considering the cost of the recursive doubling and the tree contraction gives the $O(\lg^3 n)$-time and $n/\lg n$-processor matrix chain ordering algorithm.

The next corollary shows that the algorithm given here can be improved by using efficient algorithms for finding row minima in totally monotone matrices. A $m \times n$ matrix M is *totally monotone* if every 2×2 submatrix is monotone. That is, for all $1 \leq i < k \leq m$ and $1 \leq j < l \leq n$, if $M[i, j] > M[i, l]$, then $M[k, j] > M[k, l]$.

This row minima problem is classical and has been shown to be at the root of many important problems; see for example [1, 2].

COROLLARY 3. *Solving the row minima problem on totally monotone matrices allows us to merge two bands.*

Proof. Given two nested bands to merge, for ease of exposition take only the horizontal straight unit paths of the outer band. Let each of these straight unit paths denote the row of a matrix M. Each column of M represents the jumpers that get their *sp* values from the supercritical nodes of the inner band. The first column represents the effect of the innermost jumper, the second column represents the effect of the immediate jumper containing it, etc. Similarly, several sets of independent jumpers give several totally monotone matrices.

By Theorem 13, M is a monotone matrix. But, any submatrix of M represents neighboring straight unit paths in the rows and neighboring jumpers along the columns. Similarly, every 2×2 submatrix is monotone. Since Theorems 8 and 10 still hold, we know that such a submatrix is also monotone since we can again apply Theorem 13. $\square$

Therefore, our algorithm is one of the many known to depend on the row minima problem on a totally monotone matrix. Hence, by the results of Aggarwal and Park [2] and Atallah and Kosaraju [6], our algorithm runs in $O(\lg^2 n \lg \lg n)$ time using $n/\lg \lg n$ processors on a common CRCW PRAM, or in $O(\lg^2 n)$ time using n processors on an EREW PRAM. For the EREW PRAM algorithm note that from pointer jumping to tree contraction the time complexity stays the same asymptotically.

An asymptotically optimal polylog-time row minima algorithm for totally monotone matrices would make the work of our MCOP algorithm the same as the work of Hu and Shing's $O(n \lg n)$ sequential algorithm. Very recently Bradford, Fleischer, and Smid [14] give an algorithm for computing the row minima of totally monotone matrices with $O(n\sqrt{\lg n})$ work and $O(\lg n \lg \lg n)$ time on a CREW PRAM (and several variations on other PRAM models). The results of [14] lead to an $O(n \lg^{1.5} n)$ work and polylog time CREW PRAM algorithm for the MCOP.

Hu and Shing's algorithm has the best known work for solving the MCOP to date [28, 29]. In this regard, in [38, 39] Ramanan shows that problems closely related to the MCOP have a $\Omega(n \lg n)$ lower bound. Furthermore, in [13] Bradford, Choppella, and Rawlins give several lower bounds for the MCOP on different models of computation, including a simple $\Omega(n \lg n)$ lower bound on the comparison based model for a constrained version of the MCOP.

6. Conclusions.

The study of efficient parallel algorithms for problems with elementary dynamic programming solutions is rich with interesting results. This paper gives an algorithm that solves the matrix chain ordering problem to within less than a log factor of the best serial solution. Furthermore, the best serial solution is in some sense optimal. This algorithm also solves a problem of finding an optimal triangulation of a convex polygon.

Acknowledgments. We acknowledge conversations with Alok Aggarwal, Venkatesh Choppella, Artur Czumaj, Ming Kao, Larry Larmore, and Kunsoo Park that were very helpful. In addition, conversations with Danny Chen were highlighted when, at Midwest Theory Day, he pointed out reference [16] which helps make our algorithm more efficient on the EREW PRAM.

REFERENCES

[1] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilbur, *Geometric applications of a matrix searching algorithm*, Algorithmica, 2 (1987), pp. 195–208.

[2] A. Aggarwal and J. Park, *Parallel searching multidimensional monotone arrays*, J. Algorithms, to appear; in Proc. 29th Annual IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 497–512.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[4] R. Anderson and E. W. Mayr, *Parallelism and the maximal path problem*, Inform. Process. Lett., 24 (1978), pp. 121–126.

[5] A. Apostolico, M. J. Atallah, L. L. Larmore, and S. H. McFaddin, *Efficient parallel algorithms for string editing and related problems*, SIAM J. Comput., 19 (1990), pp. 968–988.

[6] M. J. Atallah and S. R. Kosaraju, *An efficient parallel algorithm for the row minima of a totally monotone matrix*, J. Algorithms, 13 (1992), pp. 394–413.

[7] M. J. Atallah, S. R. Kosaraju, L. L. Larmore, G. L. Miller, and S.-H. Teng, *Constructing trees in parallel*, in Proc. 1st ACM Symp. on Parallel Algorithms and Architectures, ACM, New York, 1989, pp. 499–533.

[8] O. Berkman, D. Breslauer, Z. Galil, B. Schieber, and U. Vishkin, *Highly parallelizable problems*, in Proc. 21st Annual ACM Symposium on the Theory of Computing, ACM, New York, 1989, pp. 309–319.

[9] O. Berkman, B. Schieber, and U. Vishkin, *Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values*, J. Algorithms, 14 (1993), pp. 344–370.

[10] P. G. Bradford, *Efficient Parallel Dynamic Programming*, Technical Report 352, Indiana University, Bloomington, IN, April 1992 and October 1994 (revised).

[11] P. G. Bradford, *Efficient Parallel Dynamic Programming*, extended abstract in the Proceedings of the 30th Allerton Conference on Communication, Control and Computation, University of Illinois at Urbana-Champaign, 1992, pp. 185–194. Full version: Technical Report 352, Indiana University, Bloomington, IN, April 1992 and October 1994 (revised).

[12] P. G. Bradford, *Efficient Parallel Dynamic Programming*, Ph.D. dissertation, February 1995, Indiana University, Bloomington, IN; also Parallel Dynamic Programming Technical Report TR 424, February 1995, Indiana University.

[13] P. G. Bradford, V. Choppella, and G. J. E. Rawlins, *Lower bounds for the matrix chain ordering problem (extended abstract)*, in the Proceedings of LATIN '95: Theoretical Informatics, Lecture Notes in Comp. Sci. 911, R. Baeza-Yates, E. Goles, and P. V. Poblete, eds., Springer-Verlag, New York, 1995, pp. 112–130.

[14] P. G. Bradford, R. Fleischer, and M. Smid, *More efficient parallel totally monotone matrix searching*, J. Algorithms, 23 (1997), pp. 386–400.

[15] P. G. Bradford, G. J. E. Rawlins, and G. E. Shannon, *Matrix Chain Ordering in Polylog Time with $n/\lg n$ Processors*, Technical Report 360, Indiana University, Bloomington, IN, December 1992.

[16] D. Z. Chen, *Efficient geometric algorithms on the EREW PRAM*, in Proceedings of the 28th Allerton Conference on Communication, Control, and Computation, Monticello, IL, 1990, pp. 818–827. Full version in IEEE Trans. Parallel Distrib. Systems, 6 (1995), pp. 41–47.

[17] F. Y. Chin, *An $O(n)$ algorithm for determining near-optimal computation order of matrix chain products*, Comm. ACM, 21 (1978), pp. 544–549.

[18] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw–Hill, New York, 1990.

[19] A. Czumaj, *An optimal parallel algorithm for computing a near-optimal order of matrix multiplications*, in SWAT 92, Lecture Notes in Comput. Sci. 621, Springer-Verlag, New York, 1992, pp. 62–72.

[20] A. Czumaj, *Parallel algorithm for the matrix chain product and the optimal triangulation problem*, in STACS 93, Lecture Notes in Comput. Sci. 665, Springer-Verlag, New York, 1993, pp. 294–305.

[21] L. E. Deimel, Jr. and T. A. Lampe, *An Invariance Theorem Concerning Optimal Computation of Matrix Chain Products*, Technical Report TR79-14, North Carolina State Univ., Raleigh, NC.

[22] Z. Galil and K. Park, *Parallel algorithms for dynamic programming recurrences with more than $O(1)$ dependency*, J. Parallel Distrib. Comput., 21 (1994), pp. 213–222.

[23] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press, Cambridge, 1988.

[24] D. Hillis and G. L. Steele, Jr., *Data parallel algorithms*, Comm. ACM, 29 (1986), pp. 1170–1183.

[25] S.-H. S. Huang, H. Liu, and V. Viswanathan, *Parallel dynamic programming*, in Proc. 2nd Annual IEEE Symposium on Parallel and Distributed Processing, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 497–500.

[26] S.-H. S. Huang, H. Liu, and V. Viswanathan, *A sublinear parallel algorithm for some dynamic programming problems*, Theoret. Comput. Sci., 106 (1992), pp. 361–371.

[27] T. C. Hu and M. T. Shing, *An $O(n)$ algorithm to find a near-optimum partition of a convex polygon*, J. Algorithms, 2 (1981), pp. 122–138.

[28] T. C. Hu and M. T. Shing, *Computation of matrix product chains. Part* I, SIAM J. Comput., 11 (1982), pp. 362–373.

[29] T. C. Hu and M. T. Shing, *Computation of matrix product chains. Part* II, SIAM J. Comput., 13 (1984), pp. 228–251.

[30] O. H. Ibarra, T.-C. Pong, and S. M. Sohn, *Hypercube algorithms for some string comparison problems*, in Proc. IEEE International Conference on Parallel Processing, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 190–193.

[31] J. JáJá, *An Introduction to Parallel Algorithms*, Addison–Wesley, Reading, MA, 1992.

[32] D. G. Kirkpatrick and T. Przytycka, *Parallel construction of near optimal binary search trees*, in Proc. 2nd ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1990, pp. 234–243.

[33] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, Vol. A, V. Van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 869–941.

[34] S. K. Kim, *Optimal Parallel Algorithms on Sorted Intervals*, Technical Report TR 90-01-04, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1990.

[35] P. N. Klein and J. H. Reif, *Parallel time $O(\lg n)$ acceptance of deterministic CFLs on an exclusive-write P-RAM*, SIAM J. Comput., 17 (1988), pp. 463–485.

[36] V. Kumar, A. Grama, A. Gupta, and G. Kraypis, *Introduction to Parallel Computing*, Benjamin/Cummings, Redwood City, CA, 1994.

[37] L. L. Larmore and W. Rytter, *Efficient sublinear time parallel algorithms for the recognition of context-free languages*, in SWAT 91, Lecture Notes in Comput. Sci. 577, Springer Verlag, New York, 1991, pp. 121–132.

[38] P. Ramanan, *A new lower bound technique and its application: Tight lower bounds for a polygon triangularization problem*, in Proc. 2nd Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1991, pp. 281–290.

[39] P. Ramanan, *A new lower bound technique and its application: Tight lower bounds for a polygon triangularization problem*, SIAM J. Comput., 23 (1994), pp. 834–851.

[40] P. Ramanan, *An Efficient Parallel Algorithm for Finding an Optimal Order of Computing a Matrix Chain Product*, Technical Report WSUCS-92-2, Wichita State University, Wichita, KS, June 1992.

[41] P. Ramanan, *An Efficient Parallel Algorithm for the Matrix Chain Product Problem*, Technical Report WSUCS-93-1, Wichita State University, Wichita, KS, January 1993.

[42] W. Rytter, *On efficient parallel computation for some dynamic programming problems*, Theoret. Comput. Sci., 59 (1988), pp. 297–307.

[43] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, *Fast parallel computation of polynomials using few processors*, SIAM J. Comput., 12 (1983), pp. 641–644.

[44] F. F. Yao, *Speed-up in dynamic programming*, SIAM J. Algebraic Discrete Methods, 3 (1982), pp. 532–540.