Matrix Chain Ordering in Polylog Time with $n/\lg n$ Processors

Phillip G. Bradford* Gregory J. E. Rawlins† Gregory E. Shannon‡

February 19, 1993

Abstract

This paper gives a $O(\lg^4 n)$ time and $n/\lg n$ processor algorithm for solving the matrix chain ordering problem and for finding optimal triangulations of a convex polygon on the Common CRCW PRAM model. This algorithm works by finding shortest paths in special digraphs modeling dynamic programming tables. These shortest paths are found cheaply using new and efficient techniques for exploiting monotonic problem constraints.

1 Introduction

Recently, much research has gone into designing efficient parallel algorithms for problems with elementary serial dynamic programming solutions. These problems include string editing [1, 3], context free grammar recognition [22, 21], and optimal tree building [2, 19]. Polylog time parallel algorithms for solving these problems use new approaches since straightforward parallelization of sequential dynamic programming algorithms produces 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 [7]. 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 [5, 6, 8, 9, 10, 13, 14, 15, 16, 17, 23, 26, 27, 28], 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 MCOP. In [24, 25] Ramanan very recently has independently given an $O(\lg^4 n)$ time and n processor algorithm for solving the MCOP.

*Department of Computer Science, Indiana University, 215 Lindley Hall, Bloomington, Indiana 47405. email: bradford@cs.indiana.edu.

†Department of Computer Science, Indiana University, 215 Lindley Hall, Bloomington, Indiana 47405. email: rawlins@cs.indiana.edu.

‡Department of Computer Science, Indiana University, 215 Lindley Hall, Bloomington, Indiana 47405. email: shannon@cs.indiana.edu.

1.1 Main Results of this Paper

Our approach follows [5], recasting the MCOP as a shortest path problem in a graph modeling a dynamic programming table. 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 ($O(\lg^4 n)$) and linear-processor ($n/\lg n$) parallel algorithm for the MCOP on the Common CRCW PRAM.

1.2 Previous Results

Elementary dynamic programming algorithms sequentially solve the matrix chain ordering problem in $O(n^3)$ time [7]. However, the best serial solution of the MCOP is Hu and Shing's $O(n \lg n)$ algorithm [16, 17]. Using straight line arithmetic programs, Valiant et al. [27] showed that many classical optimization problems with efficient sequential dynamic programming solutions were in $\mathcal{NC}$. Their algorithms require $\Theta(\lg^2 n)$ time and n^9 processors. Using pebbling games, Rytter [26] 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 [5], an algorithm was given that takes $O(\lg^3 n)$ time and $n^3/\lg n$ processors and [9] gave an algorithm that takes $O(\lg^3 n)$ time and $n^2/\lg^3 n$ processors. In addition, there are serial and parallel approximation algorithms for the MCOP [5, 6, 8, 15].

1.3 Structure of the Paper

Section 2, briefly reviews the interpretation of the MCOP as a shortest path graph problem from [5] and then summarizes the $n^3/\lg n$ processor algorithm. Section 3, isolates 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. Section 4, shows how to replace the all-pairs shortest path algorithm with parallel prefix and an all-pairs *comparison* algorithm. Finally, section 5 replaces the all-pairs comparison algorithm with applications of parallel prefix and binary search.

2 An $O(\lg^3 n)$ Time and $n^3/\lg n$ Processor MCOP Algorithm

This section briefly reviews the polylog time and $n^3/\lg n$ processor MCOP Algorithm from [5].

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 products $M_i \bullet \cdots \bullet M_k$. 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 [5].

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

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

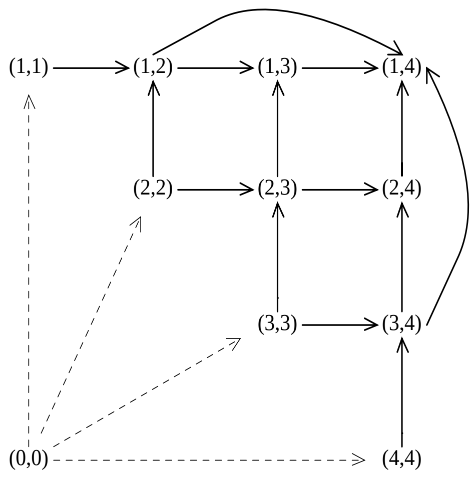

Figure 1: The Weighted Graph D_4

known as *unit* edges, together with the edges,

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

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. Just the same, 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 $(j+1,k) \Uparrow (i,k)$ both represent 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.

In [5] the MCOP was 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 [5]) *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* $(j+1,k) \Uparrow (i,k)$.

Further, 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 [5].

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 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, then for each parenthesis find its matching parenthesis by solving the all nearest smaller value (ANSV) problem [4]: Given $w_1, w_2, \ldots, w_n$, 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

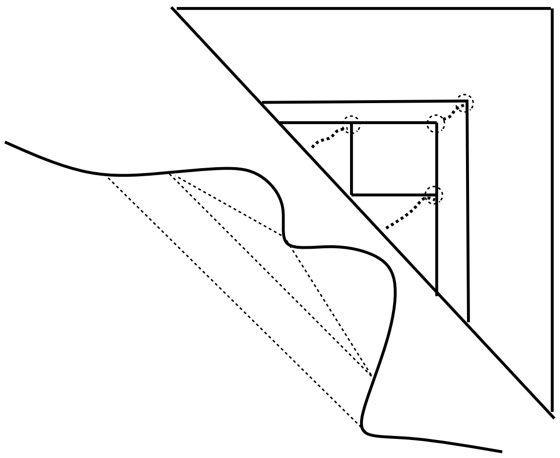

Figure 2: Two Leaf Subgraphs Inside an Band Subgraph with Critical Nodes Shown

each w find the largest j such that $1 \leq j < i$, and 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 a *critical node* is (i,k) such that $[w_i, w_{k+1}]$ is an ANSV match.

By solving the ANSV problem we can compute all critical nodes of D_n. Figure 2 shows a weight list at a 45^o angle below the x-axis and dashed lines representing four key ANSV matches. The four corresponding critical nodes are circled in D_n.

In our nomenclature, [4] shows that:

Theorem 2 *Computing all critical nodes costs $O(\lg n)$ time with $n/\lg n$ processors.*

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, then 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 [5].

2.2 Canonical Subgraphs of D_n

This subsection investigates the interaction between subgraphs containing critical nodes.

All vertices and edges that can reach (i,t) by a unit path form the subgraph $D(i,t)$. When $D(i,j)$ has a monotonic weight list $w_i, \ldots, w_{j+1}$, 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 $\{V[D(i,t)] - V[D(j+1,k-1)]\} \cup \{(0,0)\}$ and associated edges. A canonical subgraph of the form $D^{(i,t)}_{(j,j+1)}$, is a *leaf* canonical subgraph and is written $D^{(i,t)}$ and has the same nodes and edges as $D(i,t)$. 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 properties of their critical nodes by 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 $w_i \leq w_{i+1} \leq \cdots \leq w_{u+1}$, then a shortest path from $(0,0)$ to (i,u) is the straight unit path $(0,0) \nearrow (i,i) \rightarrow (i,i+1) \rightarrow \cdots \rightarrow (i,u)$, that costs

$w_i \sum_{j=i+1}^{u} w_j w_{j+1}$. On the other hand, if $D(i,t)$ has no critical nodes, then its associated weight list is monotonic. As in [16, 17, 5] 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 $2 \le i \le n+1$. 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 $i \le j \le k \le t$, then the *angular* paths of (j,k) and (i,t) are, (see Figure 3)

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

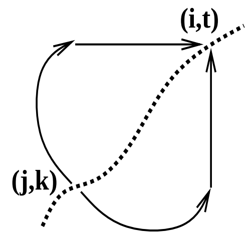

Figure 3: Two Angular Paths

Theorem 3 ([5]) *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 [5].

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 $2 \le i \le m+1$. Now find all critical nodes, then 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

This subsection discusses a tree contraction algorithm that contracts the tree structure joining the canonical subgraphs to form a shortest path in D_n, see also [16, 17, 5].

In D_n a *canonical tree* joins all of the canonical subgraphs. 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 $l_1 = w_1, w_2, \ldots w_{n+1}$, then 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, [16, 10]. So in the *rest of this paper* let w_1 denote the smallest weight in any weight list.

A result of Hu and Shing [16] and independently Yao [28] gives the next Corollary.

Corollary 1 (Atomicity Corollary [5]) *Given a weight list $w_1, \ldots, w_{n+1}$, with the three smallest weights w_1, w_{j+1}, and w_{k+1}, 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 means generally 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 $1 < i < m+1$. But, Corollary 1 can be used to break D_n into a tree of canonical graphs.

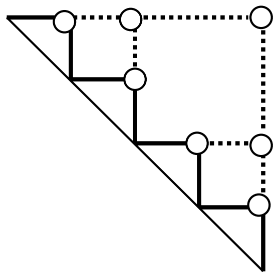

Figure 4: A Tree of Canonical Graphs, the Circles Denote Tree Nodes

Say 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 which are tree nodes.

The tree edge $(i, j) \to \cdots \to (i, v)$ 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 i^{th} row to the critical node (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

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

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 4 ([5]) *When edge minimizing a tree edge $(i, j) \to \cdots \to (i, v)$ in a canonical subgraph then 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 4, holds when $\overline{p}$ is a shortest path through a band from the front critical node back to $(0,0)$. Also, Theorem 4 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

This subsection shows how to contract a canonical tree efficiently in parallel.

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 $\overline{(i,k)}$, say $w_i < w_{k+1} < w_{j+1}$, 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 [20] when the raking order is arbitrary.

Given two nested bands, say $D^{(i,v)}_{(j,u)}$ is nested around $D^{(k,t)}_{(r,s)}$, that is $j \leq k < t \leq u$. Without loss, say 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. This is done by first edge minimizing 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

This section gives the $n^3/\lg n$ processor bottlenecks of the algorithm in section 2. In addition, this section gives 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

This subsection gives 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 5**a**. Merging two bands, see Figure 5**b**, and merging two bands that have contracted canonical trees between them, see Figure 5**c**.

In Figure 5**c**, 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 5**b**. 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.

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

Given the band $D^{(i,v)}_{(j,t)}$, let $\overline{p}^{(i,v)}_{(j,t)}$ denote a shortest path from (i,v) back to $(0,0)$ totally in $D^{(i,v)}_{(j,t)}$, see Figure 6. When there is no ambiguity, $\overline{p}^{(i,v)}_{(j,t)}$ will be written as $\overline{p}$. The nodes $V[\overline{p}] \cap V[p]$ are *super*

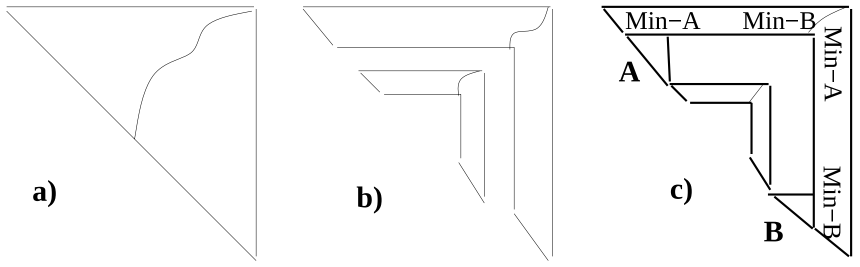

Figure 5: Bottlenecks 1, 2 and 3 for the $n^3/\lg n$ Processor Algorithm

critical nodes. Any two super critical nodes in $\overline{p}$ are connected by super critical nodes interspersed with angular paths by Theorem 3.

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.

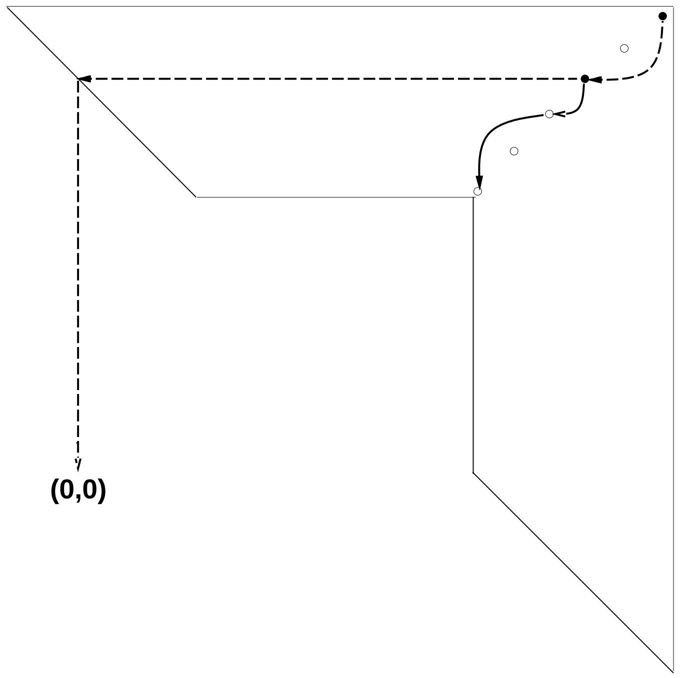

Figure 6: The Dashed Path is $\overline{p}$ and Black Nodes are Super Critical Nodes

3.2 A Metric for Finding Minimal Cost Angular Paths

This subsection gives a metric for finding a 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 the all of the relevant jumpers are nested. These jumpers get their *sp* values from super critical 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 5**c**, notice that any unit edge minimization using *sp* values from **A** or **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 $(i,s-1) \to \cdots \to (i,t)$.

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

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

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 $(i,s-1) \to \cdots \to (i,t)$. 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.

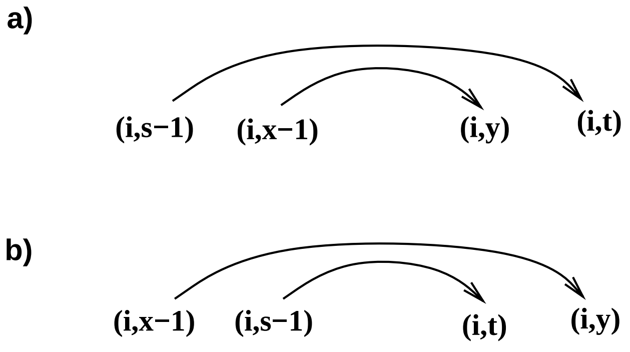

Figure 7: Two Different Nestings of two Jumpers

For nesting of Figure 7**a**, 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 $(i,s-1) \to \cdots \to (i,x-1)$ and $(i,y) \to \cdots \to (i,t)$ and $\Delta_i(s,t) > \Delta_i(x,y) > 0$. Similarly, for the nesting of Figure 7**b**, 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, in nesting of 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 $(i,x-1) \Longrightarrow (i,y)$. But on the other hand, for Figure 7**b**, 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 paths is only half the game for we also must consider the shortest paths forward.

Figure 8 is for the next Theorem, see also [16].

Theorem 5 *Given a leaf graph* $D^{(i,v)}_{(r,s)}$ *and any two critical nodes* (j,u) *and* (k,t) *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)*.*

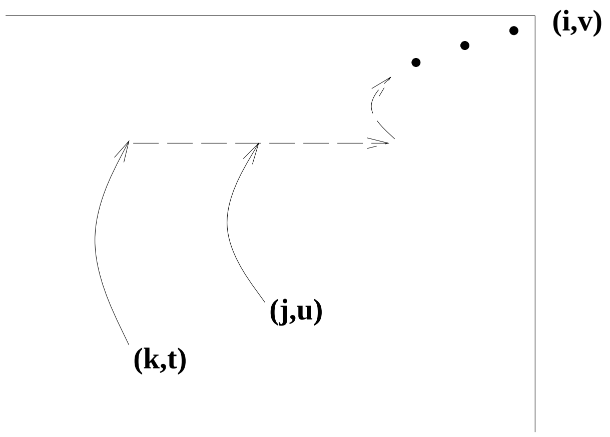

Figure 8: (j,u) Shadowing (k,t)'s Shortest Path Forward

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 "shadowed" by a shorter path from (j,u) forward to (i,v) then taking into account the f values. Naturally, Theorem 5 also holds for shortest paths forward in leaf graphs.

The next Theorem will also be useful.

Theorem 6 *Say $(i,s-1) \Longrightarrow (i,t)$ is in a shortest path forward, and in 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 effect the jumper $(s,t) \Uparrow (i,t)$ or the path from (i,t) to the front node of the present band.

4 An $O(\lg^3 n)$ Time and $n^2/\lg n$ Processor MCOP Algorithm

This section gives an $O(\lg^3 n)$ 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 super critical nodes have their shortest paths back to $(0,0)$ thorough their present bands. When merging two bands, by Theorem 4, we only have to consider shortest paths from super critical 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 super critical nodes must maintain a shortest path backwards to $(0,0)$ though 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 super critical 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 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 super critical node at hand back to $(0,0)$ through the present band. Initially, these pointers connect critical nodes and tree edges in the canonical tree.

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 super critical 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 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 super critical nodes as in Figure 10.

Figure 9 gives the induction invariant for merging two neighboring bands.

1. All critical nodes in both bands have their *front-ptr*s in trees of shortest paths that eventually go to super critical nodes which go to the front critical nodes of their respective bands.

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

Figure 9: Inductive Invariant for Band Merging

Figure 10 gives an example of the data structures for maintaining the inductive invariant. In this Figure only critical nodes are shown and the super critical 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 super critical 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. Notice, given a canonical graph and take all rows above p 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 super critical 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 a edge minimized unit path with the cost of its shortest path forward. Next, find the minimal 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. Assume there is a unit path of critical nodes from (x,y) to (s,t) to (r,u) as in Figure 11 for the next Lemma.

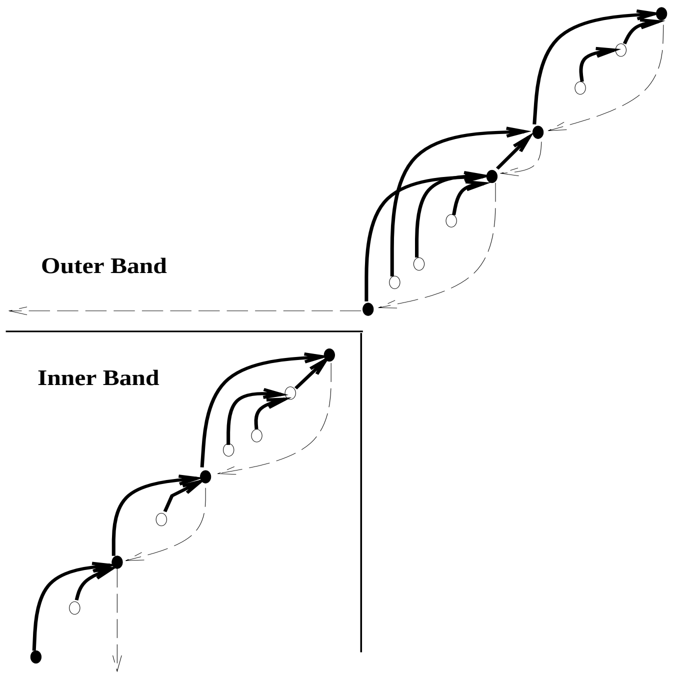

Figure 10: Solid Arrows: Forward Linked Lists of Trees; Dashed Arrows: Backward Linked Lists $\overline{p}$

Lemma 1 *Given a critical node (s,t) between the* super *critical node (x,y) and critical node (r,u) and say $i < r < s < x$ and row i is above row r, that is $w_i < w_r$, where rows i and r are above p then*

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

Proof: Start with $\Delta_r(x,y) \geq \Delta_r(s,t)$ which is,

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

and by some algebra we get the following (where $\|w_i : w_i\| = 0$),

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

and $sp(s,t) - sp(x,y)$ 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 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

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

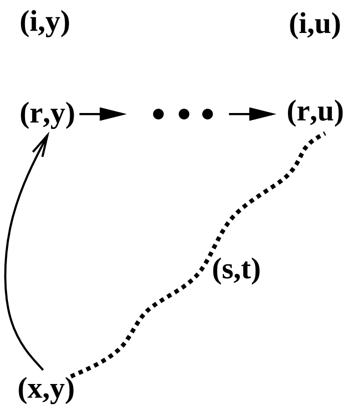

Figure 11: $(s,t) \notin V[\overline{p}]$ and the Angular Edge $(x,y) \Uparrow (r,y) \rightarrow \cdots \rightarrow (r,u)$

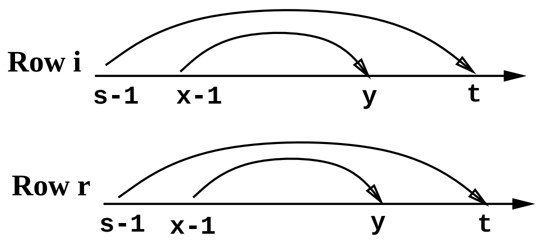

Figure 12: Two Jumpers in Different Rows

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

The next Theorem follows from Lemma 1.

Theorem 7 *Given a critical node (s,t) between the* super *critical node (x,y) and critical node (r,u) and say $i < r < s < x$ 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

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

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,

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

Given 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, say (s,t) is between (x,y) and (r,u) and $(s,t) \in V[\overline{p_i}]$ and if $(x,y) \in V[\overline{p_i}]$ and $(r,u) \in V[p_o]$ then Lemma 1 and Theorem 7 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.

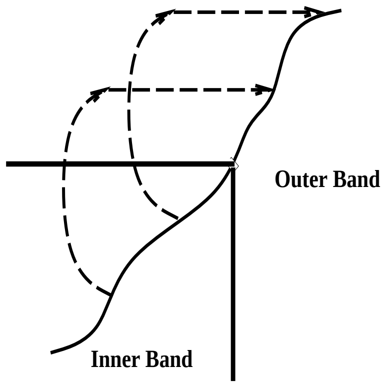

Figure 13: Conflicting Angular Paths *Between* Two Bands Being Merged

Two angular edges above p, say $(x,y) \Uparrow (r,y) \rightarrow \cdots \rightarrow (r,u)$ and $(i,j) \Uparrow (s,j) \rightarrow \cdots \rightarrow (s,t)$, are *compatible* if they don't cross each other. Compatibility is also holds for angular paths below p. Theorem 8 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. And 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 super critical 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 we can assume $(e,u) \Uparrow (d,u) \rightarrow \cdots \rightarrow (d,v)$ is $\overline{p}^{(d,v)}_{(e,u)}$. Now, say the angular edge $(e,u) \Uparrow (b,u) \rightarrow \cdots \rightarrow (b,y)$ is in $\overline{p}^{(a,z)}_{(e,u)}$, that is the minimal path from (a,z) back to $(0,0)$ through $D^{(a,z)}_{(e,u)}$.

Theorem 8 (Main Theorem) *In merging two nested bands computing shortest paths forward from super critical nodes of the inner band, we can consider only compatibly nested angular edges.*

Proof by contradiction: Suppose, $(e,u) \Uparrow (b,u) \rightarrow \cdots \rightarrow (b,y)$ is in $\overline{p}^{(a,z)}_{(e,u)}$, that is the angular edge $(e,u) \Uparrow (b,u) \rightarrow \cdots \rightarrow (b,y)$ is in a shortest path from (a,z) back to $(0,0)$ through $D^{(a,z)}_{(e,u)}$, see Figure 14. Take (d,v) in the band $D^{(d,v)}_{(e,u)}$, therefore (d,v) is between (e,u) and (a,z) in $D^{(a,z)}_{(e,u)}$.

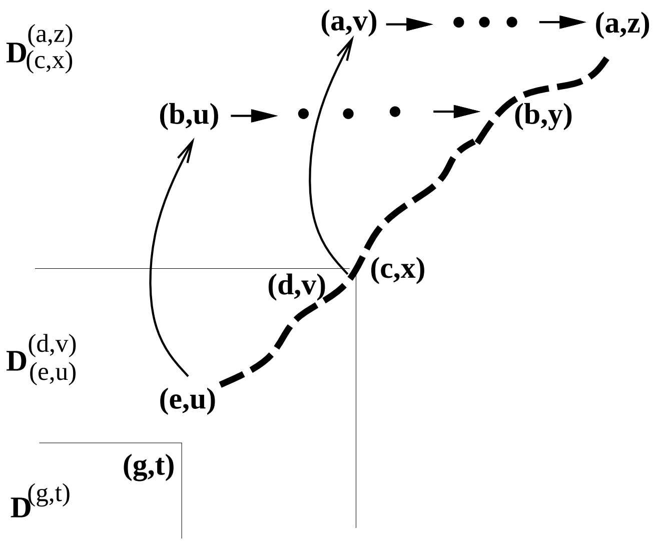

Figure 14: The Bands $D^{(a,z)}_{(c,x)}$, $D^{(d,v)}_{(e,u)}$ and the Leaf $D^{(g,t)}$

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 assume otherwise, say 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) traveling 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, 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* $D^{(a,z)}_{(e,u)}$ 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

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

And 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 $(b,e-1) \Longrightarrow (b,u)$ so we know,

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

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

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

is equivalent to the path

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

But, consider the path,

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

and it must be that $(a, e-1) \Longrightarrow (a, u)$ 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 both are above p, and 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,

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

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).

And because $(b, e-1) \Longrightarrow (b, u)$ edge minimizes row b, by Theorem 7 and since $\Delta_a(e, u) \geq \Delta_a(d, v)$ the $(a, e-1) \Longrightarrow (a, u)$ saves more in row a than $(a, d-1) \Longrightarrow (a, v)$.

Now, letting

$$\begin{aligned}
\mathcal{A} &= (a, a) \rightarrow \cdots \rightarrow (a, e-1) \Longrightarrow (a, u) \rightarrow \cdots \rightarrow (a, v) \\
\mathcal{D} &= (d, d) \rightarrow \cdots \rightarrow (d, e-1) \Longrightarrow (d, u) \rightarrow \cdots \rightarrow (d, v) \Uparrow (a, v)
\end{aligned}$$

see Figure 15.

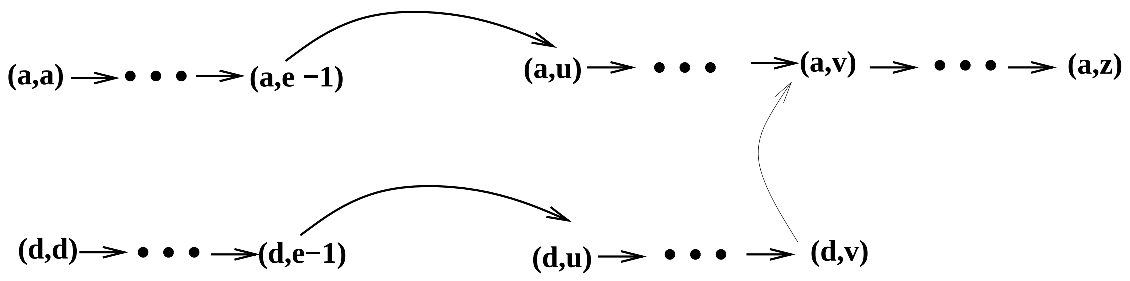

Figure 15: The Two Paths $\mathcal{A}$ and $\mathcal{D}$

$\mathcal{A}$ is a cheaper than $\mathcal{D}$ going from (a, z) back to $(0, 0)$ in $D^{(a,z)}_{(e,u)}$ by Theorem 7. 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 $(e, u) \Uparrow (a, u) \rightarrow \cdots \rightarrow (a, z)$ and not the angular path $(e, u) \Uparrow$

$(b,u) \to \cdots \to (b,y)$ which is a contradiction. We get this by applying Theorem 7 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 $D^{(a,z)}_{(e,u)}$ then none of the angular paths connecting super critical nodes in $D^{(g,t)}$ with paths forward $D^{(a,z)}_{(e,u)}$ change. This case is a straightforward application of the proof above and Theorem 6.
□

It is important to note that Theorem 8 only shows that angular paths that all start from super critical nodes in the same path back to $(0,0)$ are compatible. Theorem 8 doesn't say that all angular paths are always compatible.

Suppose, there is some angular path from a super critical node in the inner band, say (s,t), to the outer band that is in a shortest path from the front node of the outer band back to $(0,0)$. Then all super critical 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, all super critical nodes after (s,t) up to the front super critical node of the inner band have their shortest paths through nested angular paths connecting the inner and outer bands by Theorem 8. In fact, we can inductively apply this argument together with Theorem 5 giving:

Corollary 2 *Take the nested angular paths connecting two bands that are shortest paths forward from the different super critical 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 *Given a critical node (s,t) and take the i^{th} and r^{th} rows above p such that $i < r < s$ and $w_i < w_r$, then we have $\Delta_i(s,t) < \Delta_r(s,t)$.*

Proof: The function $\Delta_i(s,t)$ measures $(i,s-1) \Longrightarrow (i,t)$'s potential minimizing effect 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 $s = t-1$ gives

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

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,

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

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 super critical nodes since we are interested in merging two nested bands. Assume there is a unit path of critical nodes from (v, z) to (x, y) for the next Lemma.

Lemma 3 *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 we have*

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

Proof: Start with $\Delta_r(x, y) \geq \Delta_r(v, z)$ which is,

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

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

$\square$

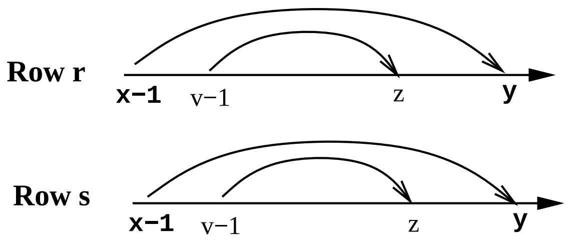

Figure 16: Two Jumpers in Different Rows

Theorem 9 *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 we have*

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 super critical 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 super critical nodes backwards eventually to $(0,0)$ and we can compute the *cost-to-back* weights using a parallel pointer jumping partial prefix.

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

Proof: Take 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, $(u,z) \notin V[\overline{p}^{(i,v)}_{(k,s)}]$. Notice that (u,z) may be a super critical node in $\overline{p}^{(i,v)}_{(j,t)}$.

Take the next cases,

Case i: Suppose $D^{(i,v)}_{(k,s)}$ is merged with another band nested around it, then since (u,z) is not in $V[\overline{p}^{(i,v)}_{(k,s)}]$ then by Theorem 4 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* super critical 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 super critical node and would have a minimal path back to $(0,0)$, but while the critical node (u,z) is not a *super* critical node it has no need of a shortest path back to $(0,0)$.

□

For every critical node we want to find a shortest path forward. This is because some angular path from some future inner band may terminate at any critical node. Therefore, after finding each super critical node's minimal cost to the front critical node of the outer band then do a tree partial prefix sum from the critical nodes to the super critical nodes so all critical nodes know their shortest paths to the front of the outer band.

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

Theorem 10 *After merging any two nested bands 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 8.

Theorem 10 shows the inductive invariant holds given the appropriate data structures and computations.

4.1 Merging Bands Using $n^2/\lg n$ Processors

This subsection shows 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 $w_i w_j w_k$, see [16].

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.

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 super critical 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-ptrs for these new edges
            let each super critical node $(x, y)$ have a pointer to a shortest path through
                $p^{(i,v)}_{(j,t)}$ to $(i, v)$
            for the super critical 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-ptrs giving any new super critical nodes
        and compute the values of cost-to-back for each new super critical node
3.  for all non-super critical nodes in $p^{(j,t)}_{(k,s)}$ in parallel do
        using pointer jumping expand the tree of front-ptrs through the new angular
            edges in $M$ and their minimal values to $(i, v)$. This gives trees joined by
            a linked list through the super critical nodes.
        with this find the shortest path to $(i, v)$ for all non-super critical
            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.
```

Figure 17: A $O(\lg n)$ Time and $n^2/\lg n$ Processor Algorithm for Merging Two Bands

The algorithm in Figure 17 costs $O(\lg n)$ time and $n^2/\lg n$ processors for merging two bands. Adding the cost of recursive doubling and tree contraction gives a factor of $O(\lg^2 n)$ time making the total cost $O(\lg^3 n)$ time and $n^2/\lg n$ 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 minimizing unit paths with contracted trees such as those depicted in the bottleneck of Figure 5**c**.

The **for** loops in step 2 compute the super critical nodes of the band that is being merged. 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 6, 8 and 10.

5 An $O(\lg^4 n)$ Time and $n/\lg n$ Processor MCOP Algorithm

This section reduces the processor complexity of the band merging algorithm of section 4 from $n^2/\lg n$ to $n/\lg n$. In this process the time complexity is raised by a multiplicative factor of $O(\lg n)$. The results of this section are based on a parallel divide and conquer form of binary search.

Theorem 7 and 9 supply the intuition for a parallel divide and conquer binary search algorithm that finds the jumpers that minimize each unit path in a canonical graph.

Theorem 11 *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 super critical node $(x-1, y)$ of the inner band to the front node of the outer band then to find shortest paths forward from other super critical nodes,*

- *It is sufficient to consider only larger nested jumpers in any row s below row r, that is $w_s > w_r$,*
- *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 Theorem 8 and Corollary 2.

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 super critical nodes originally in the inner band and the shortest path back to $(0, 0)$ through the two merged bands.

Say each band has m critical nodes, the next procedure finds shortest paths from all super critical nodes of the inner band to the front of the outer band. In addition, from 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 of the outer band.

Now assigning one processor to each unit path in the outer band and then summing the cost up to the critical node and the cost from the critical node to the front super critical node of the outer band gives a shortest path backwards from the front node of the outer band to $(0, 0)$. We can also compute these minimal paths in $O(\lg n)$ time using $n/\lg n$ processors. If a unit path has no edge minimizing jumpers, this algorithm just finds the shortest path forward for all super critical 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 5**c**. 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 $O(\lg^4 n)$ time and $n/\lg n$ processor matrix chain ordering algorithm.

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 a polylog factor of the best serial solution. This algorithm also solves a problem of finding an optimal triangulation of a convex polygon.

1. Find the middle super critical 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. Say this shortest path forward from the super critical 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

Figure 18: An $O(\lg^2 n)$ Time and $n/\lg n$ Processor Band Merging Algorithm

7 Acknowledgments

Conversations with Alok Aggarwal, Artur Czumaj, Ming Kao, Larry Larmore and Kunsoo Park were very helpful and enjoyable.

References

[1] A. Apostolico, M. J. Atallah, L. L. Larmore and S. H. McFaddin: "Efficient Parallel Algorithms for String Editing and Related Problems," *SIAM Journal on Computing*, Vol. 19, No. 5, 968-988, Oct. 1990.

[2] M. J. Atallah, S. R. Kosaraju, L. L. Larmore, G. L. Miller, and S.-H. Teng: "Constructing Trees in Parallel," *Proc. 1st Symp. on Parallel Algorithms and Architectures*, 499-533, 1989

[3] A. Aggarwal and J. Park: "Notes on Searching Multidimensional Monotone Arrays," *Proceedings of the* 29th *Annual IEEE Symposium on the Foundations of Computer Science*, 497-512, 1988.

[4] O. Berkman, D. Breslauer, Z. Galil, B. Schieber and U. Vishkin: "Highly Parallelizable Problems," *Symposium on the Theory on Computing*, 309-319, 1989.

[5] P. G. Bradford: "Efficient Parallel Dynamic Programming," Extended Abstract in the Proceedings of the 30^{th} *Allerton Conference on Communication, Control and Computation*, University of Illinois at Urbana-Champaign, 185-194, 1992.

[6] F. Y. Chin: "An $O(n)$ Algorithm for Determining Near-Optimal Computation Order of Matrix Chain Products," *Communications of the ACM*, Vol. 21, No. 7, 544-549, July 1978.

[7] T. H. Cormen, C. E. Leiserson and R. L. Rivest: *Introduction to Algorithms*, McGraw Hill, 1990.

[8] A. Czumaj: "An Optimal Parallel Algorithm for Computing a Near-Optimal Order of Matrix Multiplications," *SWAT*, Springer Verlag, LNCS # 621 , 62-72, 1992.

[9] A. Czumaj: "Parallel algorithm for the matrix chain product and the optimal triangulation problem," to appear in the proceedings of STACS '93.

[10] L. E. Deimel, Jr. and T. A. Lampe: "An Invariance Theorem Concerning Optimal Computation of Matrix Chain Products," North Carolina State Univ. Tech Report # TR79-14.

[11] Z. Galil and K. Park: *Parallel Dynamic Programming*, Manuscript, 1992.

[12] D. Hillis and G. L. Steele, Jr.: "Data Parallel Algorithms," *Communications of the ACM*, Vol. 29, No. 12, 1170-1183, Dec. 1986.

[13] S.-H. S. Huang, H. Liu, V. Viswanathan: "Parallel Dynamic Programming," *Proceedings of the* 2^{nd} *IEEE Symposium on Parallel and Distributed Processing*, 497-500, 1990.

[14] S.-H. S. Huang, H. Liu, V. Viswanathan: "A Sublinear Parallel Algorithm for Some Dynamic Programming Problems," *Theoretical Computer Science*, Vol. 106, 361-371, 1992.

[15] T. C. Hu and M. T. Shing: "An $O(n)$ Algorithm to Find a Near-Optimum Partition of a Convex Polygon," *J. of Algorithms*, Vol. 2, 122-138, 1981.

[16] T. C. Hu and M. T. Shing: "Computation of Matrix Product Chains. Part I," *SIAM J. on Computing*, Vol. 11, No. 3, 362-373, 1982.

[17] T. C. Hu and M. T. Shing: "Computation of Matrix Product Chains. Part II," *SIAM J. on Computing*, Vol. 13, No. 2, 228-251, 1984.

[18] J. JáJá: *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[19] D. G. Kirkpatrick and T. Przytycka: "Parallel Construction of Near Optimal Binary Search Trees," Proceedings of Symposium on Parallel Algorithms and Architectures, 234-243, 1990.

[20] R. M. Karp and V. Ramachandran: "Parallel Algorithms for Shared Memory Machines," Chapter in *Handbook of Theoretical Computer Science*, V. Van Leeuwen—editor, Elsevier, 1990.

[21] P. N. Klein and J. H. Reif: "Parallel Time $O(\lg n)$ Acceptance of Deterministic CFLs on an Exclusive-Write P-RAM," *SIAM J. on Computing*, Vol. 17, 463-485, 1988.

[22] L. L. Larmore and W. Rytter: "Efficient Sublinear Time Parallel Algorithms for the Recognition of Context-Free Languages," *SWAT*, Springer Verlag, LNCS #577, 121-132, 1991.

[23] P. Ramanan: "A New Lower Bound Technique and its Application: Tight Lower Bounds for a Polygon Triangularization Problem," *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, 281-290, 1991.

[24] 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, June, 1992.

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

[26] W. Rytter: "On Efficient Parallel Computation for Some Dynamic Programming Problems," *Theoretical Computer Science*, Vol. 59, 297-307, 1988.

[27] L. G. Valiant, S. Skyum, S. Berkowitz and C. Rackoff: "Fast Parallel Computation of Polynomials Using Few Processors," *SIAM J. on Computing*, Vol. 12, No. 4, 641-644, Nov. 1983.

[28] F. F. Yao: "Speed-Up in Dynamic Programming," *SIAM J. on Algebraic and Discrete Methods*, Vol. 3, No. 4, 532-540, 1982.