Efficient Parallel Dynamic Programming[1]

(Revised)

Phillip G. Bradford
Indiana University
Department of Computer Science
215 Lindley Hall
Bloomington, IN 47405

(812) 855-3609

bradford@cs.indiana.edu

October 27, 1994

[1]An extended abstract of this paper is in the Proceedings of the 30th Annual Allerton Conference on Communication, Control and Computing, University of Illinois, 185-194, 1992. This is Technical Report # 352 (Revised), Indiana University. Hardcopy is available or a postscript file is obtainable via internet: cs.indiana.edu:/pub/techreports/TR352.ps.Z. **Tech-Report revision posted**: Oct. 1994.

Abstract

In 1983, Valiant, Skyum, Berkowitz and Rackoff showed that many problems with simple $O(n^3)$ sequential dynamic programming solutions are in the class $\mathcal{NC}$. They used straight line programs to show that these problems can be solved in $O(\lg^2 n)$ time with n^9 processors. In 1988, Rytter used pebbling games to show that these same problems can be solved on a CREW PRAM in $O(\lg^2 n)$ time with $n^6/\lg n$ processors. Recently, Huang, Liu and Viswanathan [23] and Galil and Park [15] give algorithms that improve this processor complexity by polylog factors.

Using a graph structure that is analogous to the classical dynamic programming table, this paper improves these results. First, this graph characterization leads to a polylog time and $n^6/\lg n$ processor algorithm that solves these problems. Second, there follows a subpolylog time and sublinear processor parallel approximation algorithm for the matrix chain ordering problem. Finally, this paper presents a $n^3/\lg n$ processor and $O(\lg^3 n)$ time algorithm that solves the optimal matrix chain ordering problem and the problem of finding an optimal triangulation of a convex polygon.

1.1 Introduction

Dynamic programming has long been useful in designing efficient sequential algorithms for a variety of combinatorial optimization problems. However, the standard sequential construction of dynamic programming tables has prevented the development of efficient parallel algorithms. With straightforward parallelization, many elementary dynamic programming algorithms require linear time at best. This paper gives a graph characterization that models dynamic programming tables. These graphs lead directly to polylog time algorithms for optimal matrix chain ordering, the optimal construction of binary search trees and the optimal triangulation of convex polygons. Further, tree decomposition of these graphs gives efficient polylog time parallel algorithms for the matrix chain ordering problem. The matrix chain ordering problem is the primary focus of this paper.

The dynamic programming paradigm is based on the *principle of optimality.* This principle is that for a structure to be optimal all of its well-formed substructures must also be optimal. Hence, the dynamic programming paradigm is essentially a top down algorithm design method. Conversely, the *greedy principle* basically is that if a substructure is optimal then it is in some optimal superstructure. In some sense this is a bottom up design method. Intuitively, the main results of the paper rely on finding applications of the greedy principle that enhance the efficiency of applying the principle of optimality.

This paper first shows how to solve several problems by finding a shortest path in an associated digraph. Next, it focuses on solving the matrix chain ordering problem building on work by Hu and Shing, and Berkman et al. Given a graph characterizing a dynamic programming table and by isolating monotonic and non-monotonic lists of adjacent matrix dimensions, we can ignore certain subgraphs while finding a shortest path.

1.1.1 Main Results of this Paper

This paper lays a foundation for new parallel algorithms that efficiently solve many combinatorial optimization problems. From this foundation, several algorithms that solve a variety of problems follow. The following problems are representative of those to which these methods can be applied (for formal definitions of these problems see [**3**, 11]):

- *optimal matrix chain ordering*: Find an optimal ordering to multiply a matrix chain together, where the matrices are pairwise compatible but of varying dimensions.
- *optimal convex polygon triangulation*: Find an optimal triangularization of a convex polygon given a triangle cost metric.
- *optimal binary search tree construction*: Build a binary search tree with minimal average lookup time given a totally ordered set of elements and their access probabilities.

The final sections of this paper focus on the matrix chain ordering problem and the convex polygon triangulation problem.

These three problems are representative of many that can be solved sequentially in $O(n^3)$ time with elementary dynamic programming algorithms, although there are faster algorithms that are more complex.

This paper shows how to transform all three problems to a minimum cost parenthesization problem on a weighted semigroupoid. This problem is then transformed to a shortest path problem on a special weighted digraph. Applying a shortest path algorithm based on matrix multiplication then gives a $O(\lg^2 n)$ time and $n^6/\lg n$ processor solution.

The rest of the paper is about the matrix chain ordering problem. Next follows a subpolylogarithmic time and sublinear processor algorithm that provides an approximate solution that is guaranteed to be within 15.5% of optimality. This algorithm follows from the work of Chin [10], and Hu and Shing [20] combined with that of Berkman et al. [4]. Finally, exploiting special properties of the digraphs that model the semigroupoids, a $O(\log^3 n)$ time algorithm using $n^3/\log n$ processors is given. In some sense this work extends the results of [1, 2, 26]. The work of Hu and Shing [19, 21, 22] supplies a basis for these algorithms, although they are built in a different framework.

This paper is an update of [6] and the full version of [7]. Much subsequent work has appeared, take for example [13, 29, 8, 30]. In addition, [12] reports very similar results to those in section 6 of this paper.

This paper takes all logarithms as base 2 and assumes the common-CRCW PRAM parallel model.

1.1.2 Previous Results

Using straight line arithmetic programs Valiant et al. [32] showed that many classical optimization problems with efficient sequential dynamic programming solutions are $\mathcal{NC}$. However, the algorithms they suggest appear to require $\Theta(\log^2 n)$ time and n^9 processors. Using pebbling games, Rytter [31] describes a general method to generate more efficient parallel algorithms for a class of optimization problems with dynamic programming solutions. He solves the three problems mentioned in the previous subsection in $O(\log^2 n)$ time with $n^6/\log n$ processors on a CRCW PRAM. Huang, Liu and Viswanathan [23] and Galil and Park [15] give algorithms that solve this problem in $O(\lg^2 n)$ time using $n^6/! \lg^5 n$ and $n^6/! \lg^6 n$ processors, respectively.

The three problems in Subsection 1.1.1 can be solved sequentially in $O(n^3)$ time with elementary dynamic programming algorithms, and in $O(n^2)$ time by more complex algorithms [33]. But the best serial solution of the matrix chain ordering problem known is Hu and Shing's $O(n \lg n)$ algorithm [19, 21, 22]. It has been conjectured that Hu and Shing's algorithm is optimal [28].

Finally, variations of the string editing problem are often solved with dynamic programming algorithms [11], and parallel polylog time solutions of these problems use $O(n^2)$ processors [1, 2, 26]. However, these string edit problems differ from the three in Subsection 1.1.1 and have elementary $O(n^2)$ sequential dynamic programming solutions. In particular, Apostolico et al. [1] use divide and conquer techniques to find shortest paths in special planar digraphs in $O(\lg n)$ time using $n^2/\lg n$ processors. Ibarra, Pong and Sohn [26] also use a graph characterization to solve such problems achieving similar results. Aggarwal and Park [2] also employ graph characterizations of these problems, but in addition they use properties of certain monotone arrays to find shortest paths.

1.1.3 Structure of this Paper

Section 2 gives the computational assumptions of this paper.

Section 3 defines the balanced minimum cost parenthesization problem (BPP). This problem models variations of the string edit problem. Finding shortest paths in planar weighted digraphs solves the BPP.

Section 4 generalizes the BPP to the minimum cost parenthesization problem (MPP). The MPP models problems such as the three in Subsection 1.1.1 and it can be solved by finding a shortest path in special *non*-planar weighted digraphs.

Section 5 focuses on the matrix chain ordering problem (MCOP), a special case of the MPP. Here, a list of weights represents the dimensions of the matrices and combinations of these weights make up the edge weights of the corresponding graph. The convex polygon triangulation problem is also a special case of the MPP. Alternatively, for the convex polygon triangulation problem a list of weights represents vertices of a convex polygon. Throughout this paper we focus on the MCOP since solutions to the MCOP are often given as standard examples of the dynamic programming paradigm. We model these weights and related graph nodes by the nesting levels of parentheses, using the all nearest smaller value problem of Berkman et al. [4, 5].

Section 6 contains a parallel approximation algorithm for the MCOP. This algorithm does a linear amount of work. Intuitively, this algorithm works by removing relatively heavy weights so that the remaining weight list closely approximates an optimal solution.

Section 7 gives the polylog time parallel algorithm for the MCOP and the convex polygon triangulation problem. It works in $O(\lg^3 n)$ time using $n^3/\lg n$ processors. This algorithm works by isolating atomic subgraphs and then connecting them to form a tree. Finally, a tree contraction algorithm computes a shortest path.

2.1 Definitions

This section contains definitions of structures that model associative products. With these definitions, problems including the three in Subsection 1.1.1 can be suitably addressed.

2.1.1 Computational Assumptions

A *weighted* semigroupoid $(S, R, \bullet, pc)$ is a semigroupoid $(S, R, \bullet)$ with a non-negative product cost function pc such that if $(a_i, a_k) \in R$ then $pc(a_i, a_k)$ is the cost of evaluating $a_i \bullet a_k$. The minimal cost of evaluating an associative product $a_i \bullet a_{i+1} \bullet \cdots \bullet a_k$ is denoted by $mp(i, k)$. That is,

$$mp(i,k) = \min_{i \le j < k} \{ \ mp(i,j) + mp(j+1,k) + f(i,j,k) \ \}$$

where $f(i,j,k) = pc(a_i \bullet a_{i+1} \bullet \cdots \bullet a_j,\ a_{j+1} \bullet a_{j+2} \bullet \cdots \bullet a_k)$.

Given a weighted semigroupoid and an associative product $a_1 \bullet a_2 \bullet \cdots \bullet a_n$ the problem of finding $mp(1,n)$ is the *minimum parenthesization problem* on a weighted semigroupoid (MPP).

Assume a globally accessible 4-tuple $(S, R, \bullet, pc)$ represents a weighted semigroupoid. Represent pc, R, and $\bullet$ by the array, $f(1..n, 1..n, 1..n)$, where

$$f(i,j,k) \quad = \quad \begin{cases} pc(a_i \bullet \cdots \bullet a_j,\ a_{j+1} \bullet \cdots \bullet a_k) & \text{if } (a_i \bullet \cdots \bullet a_j) \bullet (a_{j+1} \bullet \cdots \bullet a_k) \in S \\ \infty & \text{otherwise} \end{cases}$$

Given a_i and a_j, assume that both $a_i \bullet a_j$ and $pc(a_i, a_j)$ can be computed in constant time.

The paper assumes the standard elementary matrix multiplication algorithm. In addition, it assumes the cost of multiplying a matrix of dimension $w_i \times w_j$ by one of dimension $w_j \times w_k$ is $w_i w_j w_k$ operations. Also, the convex polygon triangulation problem assumes that a triangle with the vertices w_i, w_j and w_k has cost $w_i w_j w_k$.

This paper is about finding the minimum number of operations necessary to solve optimization problems. It does not address solving the optimization problems themselves. For instance, in solving the MCOP we derive the minimal number of operations needed to multiply n matrices. (In addition, we compute the actual optimal ordering of the matrices.) But we do not consider whether these matrices are to be multiplied sequentially or in parallel.

3.1 The Balanced Minimum Cost Parenthesization Problem

This section defines a subproblem of the MPP called the balanced parenthesization problem on a weighted semigroupoid (BPP). We use this problem to develop the MPP and to discuss some previous results.

$$\begin{array}{lllll}
S_{1,n} & & \text{is the start symbol} & & \\
S_{i,i+1} & \rightarrow & i \bullet j & \text{iff} & i+1=j \text{ and } i+1 \leq n \\
S_{i,j} & \rightarrow & i \bullet (S_{i+1,j}) \mid (S_{i,j-1}) \bullet j & \text{iff} & i < j-1
\end{array}$$

Figure 3.1: The Grammar L_1

Any string derived from L_1 is called a *balanced parenthesization* of n elements.

For every weighted semigroupoid with a balanced associative product of n elements we can construct a corresponding weighted digraph. As we shall see, finding a shortest path in such a graph solves the BPP for the given associative product.

Denote vertices by (i,j), where $1 \leq i \leq j \leq n$, and edges by $\rightarrow$, $\uparrow$ or $\nearrow$. Edge $(i,j) \uparrow (i-1,j)$ represents the product $a_{i-1} \bullet (a_i \bullet \cdots \bullet a_j)$, therefore it weighs $f(i-1,i-1,j)$. Similarly, $(i,j) \rightarrow (i,j+1)$ represents the product $(a_i \bullet \cdots \bullet a_j) \bullet a_{j+1}$ and it weighs $f(i,j,j+1)$. Also, $\nearrow$ represents an edge from $(0,0)$ to (i,i) for all $i, 1 \leq i \leq n$.

Definition 1 *Given an n-element weighted semigroupoid, the graph $G_n = (V,E)$ has vertices,*

$$V = \{(i,j) : 1 \leq i \leq j \leq n\} \cup \{(0,0)\}$$

and unit *edges,*

$$\begin{aligned}
E = \ & \{(i,j) \rightarrow (i,j+1) : 1 \leq i \leq j < n\} \cup \\
& \{(i,j) \uparrow (i-1,j) : 1 < i \leq j \leq n\} \cup \\
& \{(0,0) \nearrow (i,i) : 1 \leq i \leq n\}
\end{aligned}$$

and a weight function W where

$$\begin{array}{lcll}
W((i,j) \rightarrow (i,j+1)) & = & f(i,j,j+1) & 1 \leq i \leq j < n \\
W((i,j) \uparrow (i-1,j)) & = & f(i-1,i-1,j) & 1 < i \leq j \leq n \\
W((0,0) \nearrow (i,i)) & = & 0 & 1 \leq i \leq n
\end{array}$$

For example, see the graph G_4 in Figure 3.2.

Given a weighted semigroupoid, n^2 processors can construct a corresponding G_n graph in constant time, since each vertex has indegree and outdegree of at most two.

The following restricted instance of the matrix chain ordering problem is a special case of the BPP: Let "$\bullet$" denote matrix multiplication in the four-matrix instance, $M_1 \bullet M_2 \bullet M_3 \bullet M_4$ of the BPP. The problem is to minimize the cost of multiplying this chain while *excluding* the product $(M_1 \bullet M_2) \bullet (M_3 \bullet M_4)$, since it is not a balanced parenthesization.

Given a vertex (i,j) of G_n finding a shortest path from $(0,0)$ to (i,j) solves the minimum cost parenthesization problem for the balanced associative product $a_i \bullet a_{i+1} \bullet \cdots \bullet a_j$.

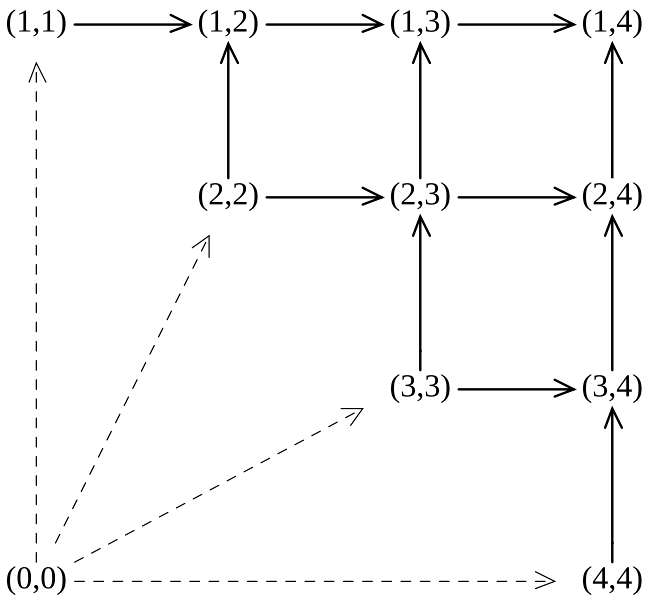
Figure 3.2: The Balanced Weighted Digraph G_4

Theorem 1 *Finding a shortest path from the vertex* $(0,0)$ *to vertex* $(1,n)$ *in* G_n *solves the minimal cost balanced parenthesization problem for an associative product of* n *elements.*

A proof is by induction on n.

Since G_n has $O(n^2)$ vertices, computing a minimum path in $O(\lg^2 n)$ time by a parallel matrix multiplication based minimum path algorithm takes $n^6/\lg n$ processors. With specialized minimum path algorithms the processor complexity improves dramatically [1, 2, 26]. These methods can compute a shortest path in $O(\lg^2 n)$ time with $n^2/\lg n$ processors on a CREW PRAM.

In [1, 2, 26] dynamic programming problems are also transformed into graph search problems; these variations of the BPP are used to solve string edit problems.

4.1 The Minimum Cost Parenthesization Problem

To represent these split parenthesizations we add edges to G_n called *jumpers*. Denote a horizontal jumper by $\Longrightarrow$, and a vertical jumper by $\Uparrow$. The vertical jumper $(i,j) \Uparrow (s,j)$ is $i-s$ units long and the horizontal jumper $(i,j) \Longrightarrow (i,t)$ is $t-j$ units long, where all non-jumper edges are of length 1. See Figure 4.3.

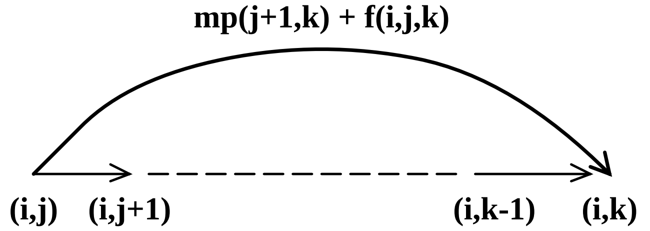

Figure 4.3: A Horizontal Jumper with its Associated Weight

The jumper $(i,j) \Longrightarrow (i,k)$ represents the product $(a_i \bullet \cdots \bullet a_j) \bullet (a_{j+1} \bullet \cdots \bullet a_k)$ and this jumper weighs $sp(j+1,k)$ plus $f(i,j,k)$. When finding a shortest path to (i,k) we take it on faith that a shortest path to $(j+1,k)$ will be computed in time for $sp(j+1,k)$ to be added to $f(i,j,k)$. Similar observations hold for vertical jumpers.

Definition 2 *The weighted digraph $D_n = (V, E \cup E')$, is a weighted digraph $G_n = (V,E)$ together with the* jumpers,

$$\begin{aligned} E' &= \{(i,j) \Longrightarrow (i,t) : 1 \le i < j < t \le n\} \cup \\ &\quad \{(s,t) \Uparrow (i,t) : 1 \le i < s < t \le n\} \end{aligned}$$

and each jumper has weight

$$\begin{aligned} W((i,j) \Longrightarrow (i,t)) &= sp(j+1,t) + f(i,j,t) & 1 < i < j < t \le n \\ W((s,t) \Uparrow (i,t)) &= sp(i,s-1) + f(i,s-1,t) & 1 \le i < s < t \le n \end{aligned}$$

For example, see the graph D_4 in Figure 4.4.

The following instance of the matrix chain ordering problem is a special case of the MPP: Let "$\bullet$" denote matrix multiplication in the four-matrix instance, $M_1 \bullet M_2 \bullet M_3 \bullet M_4$, of the MCOP. Say these matrices are of dimensions 5×10, 10×3, 3×20, and 20×6, respectively. In this case, the optimal product of matrices M_1, M_2, and M_3 is $(M_1 \bullet M_2) \bullet M_3$. But this is *not* a well-formed subproduct of the optimal matrix product of all four matrices, that is $(M_1 \bullet M_2) \bullet (M_3 \bullet M_4)$. This apparent lack of greediness seems to make techniques such as those of [1, 2, 26], fail to work for the MCOP.

Note the similarity of a D_n graph and a classical dynamic programming table, T, for the matrix chain ordering problem. The value of $sp(i,k)$ in a D_n graph is the same as $T[i,k]$ in the equivalent dynamic programming table.

Calculating a shortest path to (i,k) gives the minimum cost parenthesization of $a_i \bullet \cdots \bullet a_k$, for $1 \le i < k \le n$. So finding a shortest path from (i,k) to $(1,n)$ gives a minimal parenthesization of $a_1 \bullet \cdots \bullet a_{i-1} \bullet P \bullet a_{k+1} \bullet \cdots \bullet a_n$ where $P = (a_i \bullet \cdots \bullet a_k)$.

Theorem 2 *Finding a shortest path from $(0,0)$ to $(1,n)$ in D_n solves the minimal cost parenthesization problem for an associative product of n elements.*

There is a straightforward proof by induction on jumper length assuming Theorem 1.

The converse of this theorem also holds. That is, for any optimal parenthesization of a weighted semigroupoid there is a shortest path in a corresponding D_n graph.

It is not clear how to generalize the shortest path algorithms in [1, 2, 26] to D_n graphs. This is at least partially due to the jumper's weights; computing these weights seems difficult.

4.1.1 Constructing a D_n Graph

Constructing a D_n graph can be done by starting with a weighted digraph G_n and then performing incremental *path relaxation* [11] while adding new jumpers. A shortest path in D_n is found simultaneously. We accomplish these two goals by using a variation of a matrix multiplication based all pairs shortest path algorithm. We refer to the matrix multiplication based all pairs shortest path algorithm as a $(\min, +)$ matrix multiplication algorithm.

Jumper lengths are the basis of the arguments in this subsection.

Although G_n can be constructed in constant time with n^2 processors, it does not seem possible to construct D_n in constant time with as few processors. This is because the weight of a jumper $(i,j) \Longrightarrow (i,k)$ cannot be computed until $sp(j+1,k)$ becomes available.

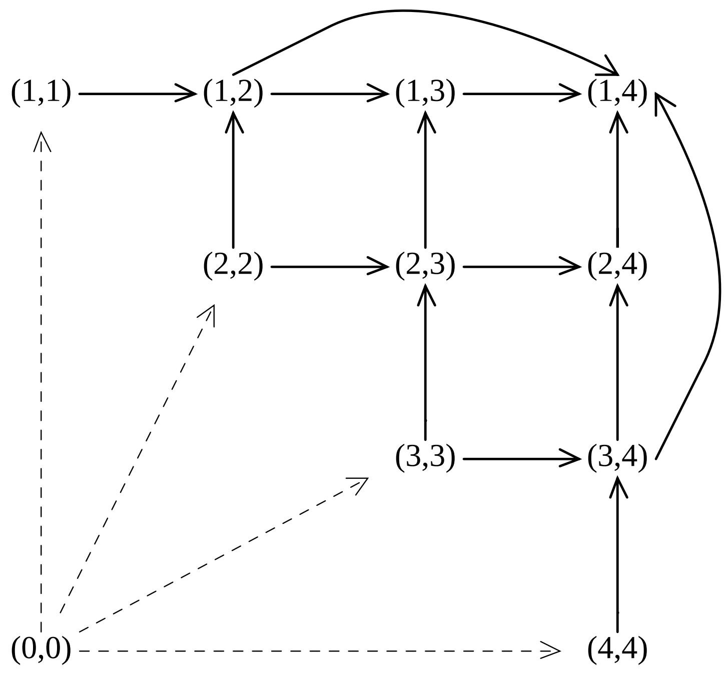

Figure 4.4: The Weighted Graph D_4

Lemma 1 *For all vertices (i,k) in a D_n graph, $sp(i,k)$ can be computed by a path having edges of length no larger than $\lceil (k-i)/2 \rceil$.*

Proof: Suppose that $(i,j) \Longrightarrow (i,k)$ is in a shortest path to (i,k) and $k-j > \lceil (k-i)/2 \rceil$. Hence,

$$\begin{aligned} sp(i,k) &= sp(i,j) + W((i,j) \Longrightarrow (i,k)) \\ &= sp(i,j) + sp(j+1,k) + f(i,j,k) \end{aligned}$$

But

$$W((j+1,k) \Uparrow (i,k)) = sp(i,j) + f(i,j,k)$$

so,

$$sp(i,k) = sp(j+1,k) + W((j+1,k) \Uparrow (i,k))$$

The jumper $(j+1,k) \Uparrow (i,k)$ is of length $j+1-i$. Therefore, since $j+1-i+k-j = k-i+1$ and $k-j > \lceil (k-i)/2 \rceil$, we know that $j+1-i \leq \lceil (k-i)/2 \rceil$.

On the other hand, a shortest path to $(j+1,k)$ cannot contain a jumper longer than $k-(j+1)$. Since $k-(j+1) < k-j$ this lemma follows inductively.
□

The proof of this last lemma leads directly to the following theorem.

Theorem 3 (Duality Theorem) *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 $(j+1,k) \Uparrow (i,k)$.*

Now we will show how to construct a D_n graph starting with a G_n graph. For any vertex (i,j) the value $j-i$ is the distance from $(0,0)$ to (i,j). So starting with a G_n graph, one $(\min,+)$ matrix multiplication computes all $sp(i,j)$ for any (i,j) where $j-i \leq 2^1-1$. Also, the minimum distances between all pairs of nodes in G_n up to 2 nodes apart are now available. With this, construct all jumpers of length 2. For length 2 horizontal jumpers, this is done by relaxing [11] them with the two horizontal edges they are directly above. In particular, having more than one path from a source to a destination relaxing is the process of finding the minimum of these paths. In Figure 4.5, the min operations are path relaxations.

Another, $(\min, +)$ matrix multiplication gives $sp(i, j)$ for all (i, j) such that $j - i \leq 2^2 - 1$. Continue this process inductively. Suppose that for all (i, j), where $j - i \leq 2^r - 1$, the values $sp(i, j)$ have been computed. At the same time the minimum distances between all pairs of nodes in G_n up to 2^r nodes apart have been computed. Now we can compute all jumpers of lengths ranging from $2^{r-1} + 1$ through 2^r and then relax them with the appropriate straight paths of lengths from $2^{r-1} + 1$ through 2^r. Performing another matrix multiplication makes the values for $sp(i, j)$, for all (i, j) where $j - i \leq 2^{r+1} - 1$, become available.

Lemma 2 *Assume all shortest paths have been calculated between each pair of vertices 2^{r-1} units apart. Suppose, for all (j, k) where $k - j \leq 2^{r-1} - 1$, the value $sp(j, k)$ is available. Then we can calculate $sp(i, t)$ with two $(\min, +)$ matrix multiplication for all (i, t) such that $t - i \leq 2^r - 1$.*

Proof: Suppose $sp(j, k)$ is available for all (j, k) where $k - j < 2^{r-1}$. Also, assume that the all pairs shortest paths have been calculated for all pairs of vertices of distance up to 2^{r-1}.

Now construct every jumper in D_n of length 2^{r-1} or smaller. Placing these jumpers, at the same time relaxing these paths, in D_n and performing one $(\min, +)$ matrix multiplication supplies $sp(i, t)$ for all $t - i < 2^r$ by Lemma 1.
□

The algorithm in Figure 4.5 is a modified $(\min, +)$ matrix multiplication all pairs shortest path algorithm. It is modified in that straight minimum paths are dynamically relaxed with new jumpers. This algorithm is basically the same as Rytter's algorithm [31].

```
for all $1 \leq i, j, u, v \leq n$ in parallel do
    $W[(i, j), (u, v)] \leftarrow \infty$
    $W[(0, 0), (i, i)] \leftarrow 0$
for all $1 \leq i, j \leq n$ in parallel do
    if $j + 1 \leq n$ then $W[(i, j), (i, j + 1)] \leftarrow f(i, j, j + 1)$
    if $i - 1 \geq 1$ then $W[(i, j), (i - 1, j)] \leftarrow f(i, i - 1, j)$
loop $2\lceil \lg n \rceil$ times
    for all $(0, 0) \leq (i, j) \leq (s, t) \leq (u, v) \leq (n, n)$ in parallel do
        $W[(i, j), (u, v)] \leftarrow \min\{\ W[(i, j), (s, t)] + W[(s, t), (u, v)],\ W[(i, j), (u, v)]\ \}$
        $W[(i, j), (u, j)] \leftarrow \min\{\ W[(i, j), (u, j)],\ W[(0, 0), (u, i - 1)] + f(u, i - 1, j)\ \}$
        $W[(i, j), (i, v)] \leftarrow \min\{\ W[(i, j), (i, v)],\ W[(0, 0), (j + 1, v)] + f(i, j, v)\ \}$
```

Figure 4.5: Modified $(\min, +)$ All Pairs Shortest Path Algorithm

Theorem 4 *Immediately after iteration r, for $1 \leq r \leq 2\lceil \lg n \rceil$, the algorithm in Figure 4.5, has computed $sp(i, t)$ for all $t - i \leq 2^r - 1$.*

Proof: The correctness of the first two loops is straightforward, so consider the third loop.

The outer loop iterates $2\lceil \lg n \rceil$ times because of Lemma 2.

The first line of the inner loop is the standard $(\min, +)$ matrix multiplication all pairs shortest path algorithm; provided we do not violate the adjacency matrix, its correctness follows from the correctness of such shortest path algorithms. An adjacency matrix has been violated when it can no longer be used to correctly determine shortest paths using the same $(\min, +)$ shortest path algorithm.

Justification of the next two lines follows by induction on jumper length. We only consider horizontal jumpers here, vertical jumpers follow immediately.

After the first iteration of the algorithm $sp(j,k)$ has been correctly calculated for all (j,k) where $k - j < 2$. After iteration r, for some $r \geq 1$, $sp(j,k)$ has been calculated for all (j,k) such that $k-j < 2^r$. At the beginning of iteration $r+1$, $sp(j,k)$ is calculated for all (j,k) such that $k-j < 2^r$ by the inductive hypothesis. During iteration $r+1$ the algorithm computes shortest paths of length between $2^r + 1$ and 2^{r+1}. By Lemma 2, this gives $sp(i,t)$, for all (i,t) such that $t - i < 2^{r+1}$.

Next in line 2, during iteration $r + 1$ the relaxation of straight vertical unit paths of length ranging from $2^r + 1$ to 2^{r+1} with vertical jumpers occurs. This is done by replacing $W[(i,j),(i,t)]$ with

$$\min\{\ W[(i,j),(i,t)],\ W[(0,0),(j+1,t)] + f(i,j,t)\ \}$$

for all $(j+1,t)$ such that $t - (j+1) < 2^{r+1}$. This does not violate the adjacency matrix since any new distance cost may be replaced by smaller, but positive, values because they have not been used yet in the calculation of other shortest paths. So relax the paths of length from $2^{r-1}+1$ to 2^r with the appropriate jumpers of the same lengths.

□

The algorithm in Figure 4.5 solves the three problems of Section 1.1.1 in $O(\lg^2 n)$ time with $n^6/\lg n$ processors.

Any cost function $f(i,j,k)$ can be used to solve the MPP as long as the shortest paths in the appropriate D_n graph remain the same.

5.1 The Matrix Chain Ordering Problem

This section discusses a subproblem of the MPP. In this subproblem the associative product costs are computed as $f(i,j,k) = w_i w_{j+1} w_{k+1}$ where $w_s = w_t$ iff $s = t$. All weights are positive integers and they are distinct only for convenience. With these associative product costs the MPP models the matrix chain order problem and the minimum cost triangulation of convex polygons. We derive the intuition of this section from the MCOP.

From here on D_n graphs are the central focus since the balanced version of this problem has been solved efficiently in [1, 2, 26].

Letting γ be a cyclic rotation function,

Theorem 5 (Deimel and Lampe [14]; Hu and Shing [21]) *Given an instance of the MCOP with the weight list* $l_1 = w_1, w_2, \ldots, w_{n+1}$ *and cyclically rotating it getting* $l_2 = w_{\gamma(1)}, w_{\gamma(2)}, \ldots, w_{\gamma(n+1)}$, *then finding an optimal parenthesization with* l_2 *provides an optimal solution to the original instance of the MCOP with* l_1.

Directly from this last theorem and Theorem 2 we have the next corollary.

Corollary 1 *Finding a shortest path in a* D_n *graph whose edge weights are constructed from either a weight list* l *or a cyclic rotation of* l *gives an optimal solution to the MCOP.*

In the rest of this paper let w_1 denote the smallest weight in any weight list.

5.1.1 Matrix Dimensions as Nesting Levels of Matching Parentheses

This subsection discusses an algorithmic technique that is central to the rest of the paper. Intuitively, this technique allows matrix dimensions to approximate the nesting level of matched parentheses.

Given an associative product where the level of each parenthesis in an optimal product is known, we can compute the parenthesization of this associative product by solving the following all nearest smaller value (ANSV) problem [4, 5]: Given $w_1, w_2, \ldots, w_n$ drawn from a totally ordered set, for each w_i find the largest j, where $1 \le j < i$, and smallest k where $i < k \le n$, so that $w_j < w_i$ and $w_k < w_i$ if such values exist.

Note that a weight list may contain weights without any matches. For example, in a monotone weight list there are no ANSV matches for any weight.

Now we investigate the effect monotonicity has on weight lists. This is interesting because we can solve any instance of the MCOP very efficiently if its weight list is monotonic.

As in [19] let $\|w_i : w_k\| = \sum_{j=i}^{k-1} w_j w_{j+1}$, where all such pair-wise sums can be computed by performing one parallel partial prefix.

Theorem 6 (Hu and Shing [19]) *The vector $F[i] = \|w_1 : w_i\|$ can be computed by a parallel partial prefix.*

As suggested by this last theorem, we can compute $\|w_i : w_k\|$ by performing one subtraction $\|w_i : w_k\| = F[k] - F[i]$. Therefore we can calculate the cost of any horizontal or vertical unit path in D_n by multiplying such a sum by the appropriate weight.

Let the i^{th} row of D_n be all vertices of the form (i, k) for $1 \le k \le n$ and the k^{th} column be vertices of the form (i, k) for $1 \le i \le k$. So, the unit path along the i^{th} row costs $w_i \|w_{i+1} : w_{n+1}\|$ where the unit path in the k^{th} column costs $w_{k+1}\|w_1 : w_k\|$.

Lemma 3 *Given a D_n graph with a weight list containing a sublist of increasing weights $w_i < w_{i+1} < \cdots < w_{k+1}$, in D_n the unit path $(0,0) \nearrow (i,i) \rightarrow \cdots \rightarrow (i,k)$ is cheaper than the unit path $(0,0) \nearrow (k,k) \uparrow (k-1,k) \uparrow \cdots \uparrow (i,k)$.*

Proof: Column k costs $w_{k+1}\|w_i : w_k\| = w_{k+1}\sum_{j=i}^{k-1} w_j w_{j+1}$ and row i costs $w_i \sum_{j=i+1}^{k} w_j w_{j+1}$ which is $w_i \sum_{j=i}^{k-1} w_{j+1} w_{j+2}$. Clearly, $w_{k+1} > w_i$ and $\sum_{j=i}^{k-1} w_{j+1} w_{j+2} > \sum_{j=i}^{k-1} w_j w_{j+1}$, hence the lemma follows.
□

Given a sublist of decreasing product weights $w_i > w_{i+1} > \cdots > w_{k+1}$ then the horizontal unit path $(0,0) \nearrow (i,i) \rightarrow \cdots \rightarrow (i,k)$ is more expensive than $(0,0) \nearrow (k,k) \uparrow (k-1,k) \uparrow \cdots \uparrow (i,k)$.

The two unit paths, $\mathcal{A} = (i,i) \rightarrow \cdots \rightarrow (i,k)$ and $\mathcal{B} = (i+1,i+1) \rightarrow \cdots \rightarrow (i+1,k) \uparrow (i,k)$ are *adjacent* horizontal unit paths.

Lemma 4 (Row Trade Off Lemma) *Given two adjacent horizontal unit paths $\mathcal{A}$ and $\mathcal{B}$ going to (i,k), which cost $w_i\|w_{i+1} : w_{k+1}\|$ and $w_{i+1}\|w_{i+2} : w_{k+1}\| + w_i w_{i+1} w_{k+1}$ respectively, one of the following conditions may hold:*

- *if $w_i < w_{i+1}$ and $w_{i+2} < w_{k+1}$ then $\mathcal{A}$ is cheaper*
- *if $w_i > w_{i+1}$ and $w_{i+2} > w_{k+1}$ then $\mathcal{B}$ is cheaper*

Proof: Since $\mathcal{A}$ costs $w_i||w_{i+1} : w_{k+1}||$ and $\mathcal{B}$ costs $w_{i+1}||w_{i+2} : w_{k+1}|| + w_i w_{i+1} w_{k+1}$, the difference in their costs is

$$d = w_i||w_{i+1} : w_{k+1}|| - w_{i+1}||w_{i+2} : w_{k+1}|| - w_i w_{i+1} w_{k+1}.$$

That is, $d = (w_i - w_{i+1})||w_{i+2} : w_{k+1}|| + (w_{i+2} - w_{k+1}) w_i w_{i+1}$ and when $w_i > w_{i+1}$ and $w_{i+2} > w_{k+1}$ then d is positive, hence $\mathcal{B}$ is cheaper.

The other case follows similarly.

□

There is a similar Column Trade Off Lemma, and its proof is almost identical.

5.1.2 Critical Nodes in D_n

This subsection relates the ANSV problem to the Row and Column Trade Off Lemmas. By solving the ANSV problem it is possible to isolate certain nodes that are central to finding shortest paths in D_n graphs.

The next definition is originally due to Hu and Shing [19], although they present it in a different framework: In D_n, a *critical node* is a node (i, k) such that $w_j > \max\{w_i, w_{k+1}\}$ for $i < j \leq k$. Note that there are no critical nodes of the form (j, j) for $1 \leq j \leq n$.

The row and column equations are,

$$\begin{array}{rcll} \mathcal{R} & = & (w_i - w_{i+1})||w_{i+2} : w_{k+1}|| + (w_{i+2} - w_{k+1}) w_i w_{i+1} & k+1 \neq i+2 \\ \mathcal{C} & = & (w_{k+1} - w_k)||w_{k-1} : w_i|| + (w_{k-1} - w_i) w_{k+1} w_k & k-1 \neq i \end{array}$$

Lemma 4 elucidates a key observation about the row and column equations. Particularly, only the order relationships of four weights is enough to determine whether equation $\mathcal{R}$ will be *necessarily* positive or *necessarily* negative. A similar observation holds for $\mathcal{C}$. This cannot be done for either $\mathcal{C}$ or $\mathcal{R}$ when both conditions $w_i < w_j$ and $w_j > w_{k+1}$, for $i < j \leq k$, hold. Generalizing for the possibility of an edge-connected path of critical nodes and associated row and column equations we say that $\mathcal{C}$ and $\mathcal{R}$ are order *indeterminate* when $w_j > \max\{w_i, w_{k+1}\}$ for all $j, i < j \leq k$. Critical nodes determine where both the row and column equations fail to provide any information. Further, critical nodes indicate where magnitude can overtake order in the row and column equations.

By solving the ANSV problem we can compute all critical nodes of a D_n graph. Although, Berkman et al. proved the next theorem for ANSV matches, it follows as a direct result of the relationship of matches in the weight list and critical nodes in D_n graphs.

Theorem 7 (Berkman et al. [4]) *All critical nodes can be computed in $O(\lg\lg n)$ and $O(\lg n)$ time using $n/\lg\lg n$ and $n/\lg n$ processors, respectively.*

Critical nodes (i, s) and (j, t) are not *compatible* when $s - i = t - j$ and there exists some vertex other than $(0, 0)$ that can reach both (i, s) and (j, t) by a unit path. The next theorem was originally proved by Hu and Shing in a different framework and follows from the properties of ANSV matches.

Theorem 8 (Hu and Shing [21]) *In any instance of the matrix chain ordering problem all critical nodes are compatible.*

Proof: Suppose D_n represents an instance of the matrix chain ordering problem with two non-compatible critical nodes (i, s) and (j, t). Since these are critical nodes, it must be that either $j < s < t$ or $i < j < s$. In either case we get a contradiction since both $w_k > \max\{w_i, w_{s+1}\}$, for

$i < k < s+1$, and $w_r > \max\{w_j, w_{t+1}\}$, for $j < r < t+1$, cannot hold.
□

A D_n graph can have as many as $n-1$ and as few as zero critical nodes. For example, a monotone weight list does not have any critical nodes.

The next lemma follows from the ANSV characterization of critical nodes.

Lemma 5 (Hu and Shing [21]) *Any D_n graph has at most $n-1$ critical nodes.*

Proof: Take a list of $n+1$ weights $w_1, w_2, \ldots, w_{n+1}$ making up the edge weights of some D_n graph.

A *representative weight* for the critical node (j,k) is the weight w_j with match $[w_i, w_{k+1}]$. Each unique critical node has a unique representative weight. Further, a list of $n+1$ weights can have at most $n-1$ representative weights, since neither w_1 nor w_{n+1} can be representative weights. Hence, there can be at most $n-1$ critical nodes.
□

A jumper $(i,j) \Longrightarrow (i,v)$ is said to *include* all critical nodes that are in some path r from $(0,0)$ to $(j+1,v)$, where r may contain jumpers too. Suppose there is a path r from $(0,0)$ to $(j+1,v)$ that includes all critical nodes in the set $\{\ (k,u)\ \}$, for all $j+1 \le k < u \le v$. Then we say any path q, containing only this one jumper $(i,j) \Longrightarrow (i,v)$, *includes* all critical nodes that are vertices of q and all critical nodes in r. Now generalize this notion recursively to paths with more than one jumper. Going even further, we can prove the next theorem.

Theorem 9 *In a D_n graph there is at least one path from $(0,0)$ to $(1,n)$ that includes all critical nodes.*

A proof of this theorem follows from the fact that all ANSV matches are compatible and each match represents a pair of parentheses. If a solution of the ANSV problem gives a parenthesization of all associative elements, then we are done. So consider the case where a solution of the ANSV problem only provides a partial parenthesization of the associative product. In this case, any arbitrary but legal completion of the parenthesization describes a path in the associated D_n graph.

5.1.3 Canonical Subgraphs of D_n

This subsection investigates the interaction between monotonicity and critical nodes. Weight lists can be broken into monotone sublists and sublists that have ANSV matches. Such sublists naturally lead to subgraphs that are useful for finding shortest paths.

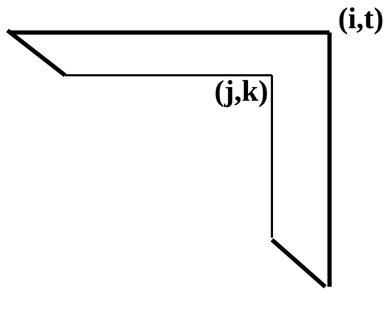

Figure 5.6: A $D_{(j,k)}^{(i,t)}$ Graph without $(0,0)$ and with No Jumpers Shown

A subgraph $D(i,t)$ of D_n is all vertices and edges of D_n that can reach (i,t) by a unit path; this includes $(0,0)$. Formally, $D(i,t)$ is all vertices $(j,k) \in V[D_n]$ such that $i \le j \le k \le t$ in addition to the associated edges. We also include $(0,0)$ in each subgraph. Also, when a graph $D(i,j)$ is such that $w_i, \ldots, w_{j+1}$ is monotonic, we say the *graph* $D(i,j)$ is monotonic.

Critical nodes may form a unit edge-connected maximal path p that isolates a subgraph. Suppose that p forms a contiguous maximal path starting at some critical node (j,k), where $k-j \geq 1$ and terminating at the critical node (i,t), then p isolates $D^{(i,t)}_{(j,k)}$. Saying the unit path p is maximal means that if there is any unit edge-connected path q of critical nodes that p is a subpath of, then $p = q$.

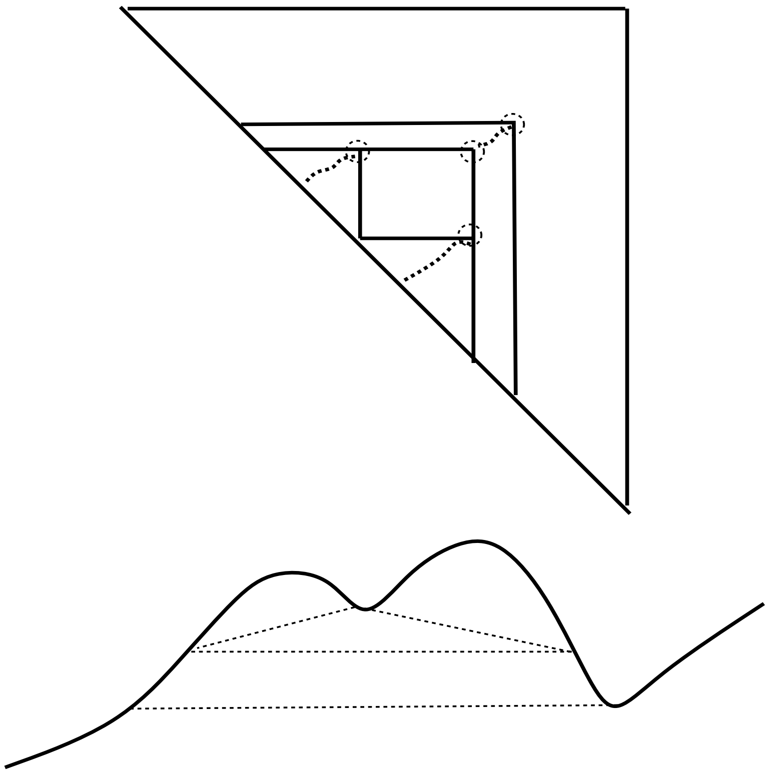

Figure 5.7: Several Canonical Subgraphs and Their Weight List

A *canonical subgraph* $D^{(i,t)}_{(j,k)}$ is the subgraph containing the maximal contiguous edge-connected path of critical nodes that begins at critical node (j,k) and terminates at critical node (i,t). The canonical subgraph $D^{(i,t)}_{(j,k)}$ has vertex set $V[D(i,t)] - V[D(j+1,k-1)] \cup \{(0,0)\}$ and the associated edges, see Figure 5.6. We write $D^{(i,t)}$ for a canonical subgraph of the form $D^{(i,t)}_{(j,j+1)}$, and p denotes the path of critical nodes in a canonical subgraph.

For instance, a canonical graph $D^{(i,t)}$ is a graph that in some sense isolates the weight list $w_i, w_{i+1}, \ldots, w_{t+1}$ where $w_k > \max\{w_j, w_s\}$ for all k, such that $i \leq j < k < s \leq t+1$.

In Figure 5.7 the weight list is represented as the contour below the D_n graph. In this contour, four key ANSV matches are represented by dotted lines. The four corresponding critical nodes are circled in the related D_n graph.

Canonical subgraphs are easily distinguishable because of properties of their critical nodes. So by Theorem 7, we can find all canonical subgraphs in $O(\lg\lg n)$ time using $n/\lg\lg n$ processors. Notice that there are only two basic kinds of canonical subgraphs.

Theorem 10 (Monotonicity Theorem) *Given a $D(i,u)$ graph with a monotone list of weights $w_i < w_{i+1} < \cdots < w_{u+1}$, the shortest path from $(0,0)$ to (i,u) is along the straight unit path $(0,0) \nearrow (i,i) \rightarrow (i,i+1) \rightarrow \cdots \rightarrow (i,u)$.*

Proof:
The proof is in two cases. In the first case, the theorem is shown to be true for G_n graphs while the second case shows that the theorem also holds for D_n graphs.

Case i: A unit path in $G(i,u)$

Let $(0,0) \nearrow (i,i) \rightarrow (i,i+1) \rightarrow \cdots \rightarrow (i,u)$ be a horizontal path with the associated vertical path $(0,0) \nearrow (u,u) \uparrow (u-1,u) \uparrow \cdots \uparrow (i,u)$.

By Lemma 3, the horizontal path is cheaper. Inductive application of the Row and Column Trade Off Lemmas (see Lemma 4) completes this case.

Case ii: A path with jumpers in $D(i,u)$

This case only addresses horizontal jumpers, since vertical jumpers follow by symmetry and Lemma 3.

Given an arbitrary path r from $(0,0)$ to (i,u) in $D(i,u)$ that includes one jumper, the unit path from (i,i) to (i,u) is shown to be cheaper than r. Suppose the jumper in this path is $(k,s) \Longrightarrow (k,t)$, for $i < k < s < t < u$. Since this is the only jumper in r, the path to (k,s) is $(k,k) \rightarrow \cdots \rightarrow (k,s)$.

Therefore, this problem has been reduced to finding a shortest path to (k,t). For, if the shortest path to (k,t) is a unit path then r cannot be a shortest path by case i, because $(k,s) \Longrightarrow (k,t)$ is the *only* jumper in r.

Still, since $W((k,s) \Longrightarrow (k,t)) = sp(s+1,t) + f(k,s,t)$ we must consider jumpers in the shortest path to $(s+1,t)$. Say a shortest path to $(s+1,t)$ is a unit path, then by case i it is the straight unit path $(s+1,s+1) \rightarrow \cdots \rightarrow (s+1,t)$. Since $f(k,s,t) = w_k w_{s+1} w_{t+1}$ and $W((k,s) \rightarrow (k,s+1)) = w_k w_{s+1} w_{s+2}$, it must be that $t > s+1$ so $f(k,s,t) > W((k,s) \rightarrow (k,s+1))$. Intuitively, we are trading the associative product cost $f(k,s,t)$ for the first edge of $(k,s) \rightarrow \cdots \rightarrow (k,t)$. This first unit edge is cheaper than the associative product cost. With this, because $(k,s+1) \rightarrow \cdots \rightarrow (k,t)$ costs $w_k \| w_{s+2} : w_{t+1} \|$ and the path $(s+1,s+1) \rightarrow \cdots \rightarrow (s+1,t)$ costs $w_{s+1} \| w_{s+2} : w_{t+1} \|$ and since $w_k < w_{s+1}$ the theorem holds.

Otherwise, say there is a horizontal jumper in r to $(s+1,t)$. Apply this case again inductively, until there is a jumper that derives its weight from a straight unit path. We are trading each jumper's associative product cost for the cost of the first unit edge in the associated unit path. These first unit edges are always cheaper than the related associative product cost. Eventually, the shortest path to (k,t) is shown to be $(k,k) \rightarrow \cdots \rightarrow (k,t)$. Hence, r is a unit path, but this means that it must be the straight unit path in row i by case i.

Handling a path with more than one jumper is straightforward. Inductively applying these two cases to jumpers successively farther away from $(0,0)$ completes the proof.
□

This theorem also holds for a monotone list of weights having the relation $w_i > w_{i+1} > \cdots > w_{j+1}$ where the shortest path is $(0,0) \nearrow (j,j) \uparrow (j-1,j) \uparrow \cdots \uparrow (i,j)$. Therefore, if the list of weights $w_i, w_{i+1}, \ldots, w_{j+1}$ is monotone then we do not have to construct any jumpers in $D(i,j)$.

We say that $D(i,t)$ *intersects* $D(j,v)$ iff $V[D(i,t)] \cap V[D(j,v)] - \{(0,0)\} \neq \emptyset$. From here on when we refer to monotone subgraphs we assume that they do not intersect any canonical subgraphs.

Theorem 11 *If $D(i,t)$ does not intersect any canonical graphs and has no critical nodes, then the weight list $w_i, w_{i+1}, \ldots, w_{t+1}$ is monotonic.*

Proof: Say there are no critical nodes in $D(i,t)$. Then there is at most *one* weight w_{j+1} where $j+1 \neq 1$ such that $w_i > \cdots w_j > w_{j+1} < w_{j+2} < \cdots < w_{t+1}$, otherwise $D(i,t)$ would contain critical nodes. But in this case, since $w_1 = \min_{1 \leq i \leq n+1}\{w_i\}$ it must be that $D(1,j+1)$ must contain the critical node $(1,j)$, which means that $D(i,t)$ would intersect with a canonical subgraph.

On the other hand, this means both the row and column equations begin and remain indeterminate so the following fact holds.

Fact 1: For all $(j,j+2) \in V[D(i,t)]$, either $w_j < w_{j+1} < w_{j+2} < w_{j+3}$ or $w_j > w_{j+1} > w_{j+2} > w_{j+3}$.

Applying the row or column equations to the nodes $(j,j+2)$, for all $j, i < j \leq t$, establishes this fact. For instance, let $\mathcal{R} = (w_j - w_{j+1})\|w_{j+2} : w_{j+3}\| + (w_{j+2} - w_{j+3})w_j w_{j+1}$, then $\mathcal{R}$ is order determinant so it must be either $w_j < w_{j+1}$ and $w_{j+2} < w_{j+3}$, or $w_j > w_{j+1}$ and $w_{j+2} > w_{j+3}$, but not both.

Suppose $w_j < w_{j+1}$ and $w_{j+2} < w_{j+3}$, and assume that $w_{j+1} > w_{j+2}$, otherwise if $w_{j+1} < w_{j+2}$ then $w_j < w_{j+1} < w_{j+2} < w_{j+3}$ so we are done. This means $w_j < w_{j+1}$ and $w_{j+1} > w_{j+2}$, therefore $w_{j+1} > \max\{w_j, w_{j+2}\}$ and this indicates $(j,j+1)$ is a critical node. This is a contradiction, hence it must be that $w_j < w_{j+1} < w_{j+2} < w_{j+3}$.

Using fact 1, the theorem follows inductively.

□

A lack of critical nodes implies the existence of a monotone subgraph. Just the same, a lack of ANSV matches in a section of a weight list indicates a monotone sublist. Assuming that $D(i,t)$ does not intersect with any canonical graphs we have the next corollary.

Corollary 2 *If $D(i,t)$ contains no critical nodes, then there are no jumpers in a shortest path from $(0,0)$ to (i,t). Moreover, if $D(i,t)$ contains no critical nodes then the shortest path from $(0,0)$ to (i,t) is a straight unit path.*

A proof of this follows immediately from Theorems 10 and 11.

Take a canonical subgraph $D^{(1,m)}$, where all critical nodes have been found and form a unit edge-connected path p. Removing the nodes and adjacent edges of p splits $D^{(1,m)}$ in two. These two pieces of $D^{(1,m)}$ are $\mathcal{U}$ for *upper* and $\mathcal{L}$ for *lower*. See Figure 5.8.

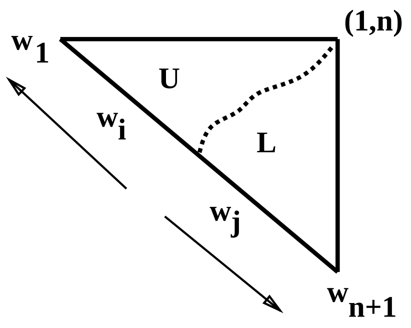

Figure 5.8: A D_n Graph Split by a Path of Critical Nodes, Arrows Point Toward Smaller Weights

Let $D(1,s)$ be the maximal well-formed subgraph of $\mathcal{U}$ and let $D(s+1,m)$ be the maximal well-formed subgraph of $\mathcal{L}$. By the maximality of $D(i,t)$ in $\mathcal{U}$ we mean that for any other well-formed subgraph $D(j,k)$, if $D(j,k) \subseteq \mathcal{U}$ then $D(j,k) \subseteq D(i,t)$.

Theorem 12 (Modality Theorem) *If $D(1,s) \subseteq \mathcal{U}$ and $D(s+1,n) \subseteq \mathcal{L}$, where both $D(1,s)$ and $D(s+1,m)$ are maximal, then $w_1 < w_2 < \cdots < w_{s+1}$ and $w_{s+2} > w_{s+3} > \cdots > w_m > w_{m+1}$.*

Proof: The path p splits $D^{(1,m)}$ into $\mathcal{U}$ and $\mathcal{L}$ where $D(1,s)$ and $D(s+1,n)$ are maximal, so $(s,s+1)$ is a critical node. Therefore, we know that $w_{s+1} > \max\{w_s, w_{s+2}\}$, so it must be that $w_s < w_{s+1}$ and $w_{s+1} > w_{s+2}$.

By Theorem 11, the weight lists $w_1, w_2, \ldots, w_s, w_{s+1}$ and $w_{s+1}, w_{s+2}, \ldots, w_m, w_{m+1}$ are both monotonic. Thus $w_s < w_{s+1}$, so it must be that $w_1 < w_2 < \cdots < w_s < w_{s+1}$. In addition, because $w_{s+1} > w_{s+2}$, we know that $w_{s+1} > w_{s+2} > \cdots > w_m > w_{m+1}$.
□

Take a $D^{(i,t)}_{(j,k)}$ canonical graph, then both $w_i < w_{i+1} < \cdots < w_j$ and $w_k > w_{k+1} > \cdots > w_{t+1}$ follow from Theorem 12.

6.1 A Parallel Approximation Algorithm for the MCOP

Combining results of Chin [10], and Hu and Shing [20] with those of Berkman et al. [4] this section develops an $O(\lg\lg n)$ time and $n/\lg\lg n$ processor approximation algorithm for the MCOP. This algorithm approximates the MCOP to within 15.5% of optimality. In addition, the processor time product of this algorithm is linear.

The approximation algorithm in this section consists of two stages. The first stage isolates relatively heavy weights by finding matrix products that must be in an optimal parenthesization. The isolation of such heavy weights provides optimal substructures that are in optimal superstructures—essentially giving a converse to the principle of optimality. The second stage is simply a greedy approach for finding a parenthesization once we have applied the first stage of the algorithm.

In this section, as before, by Corollary 1 rotate any given weight list so that w_1 is the smallest weight. For the next theorem let w_i, w_{i+1}, and w_{i+2} be three adjacent weights in a weight list of an instance of the MCOP where $w_i < w_{i+1}$ and $w_{i+1} > w_{i+2}$ which together means that $w_{i+1} > \max\{w_i, w_{i+2}\}$.

Theorem 13 (Hu and Shing [20]) *If*

$$w_1 w_i w_{i+2} + w_i w_{i+1} w_{i+2} < w_1 w_i w_{i+1} + w_1 w_{i+1} w_{i+2} \tag{6.1}$$

then the product $(a_i \bullet a_{i+1})$ is in an optimal parenthesization.

Proof of this Theorem is left to the literature, see [10] and [20] for different proofs. When $w_{i+1} > \max\{w_i, w_{i+2}\}$ fails to hold Equation 6.1 cannot hold, so there is no gain in assuming $w_{i+1} > \max\{w_i, w_{i+2}\}$.

Corollary 3 *If Equation 6.1 holds, then $w_{i+1} > \max\{w_i, w_{i+2}\}$.*

A proof follows from Equation 6.1 with $w_1 = \min_{1 \le i \le n+1}\{w_i\}$, and $w_1 w_i (w_{i+2} - w_{i+1}) + w_{i+1} w_{i+2} (w_i - w_1) < 0$ so it must be that $w_{i+2} - w_{i+1} < 0$. Also, starting with Equation 6.1 again and reassociating, we get $w_1 w_{i+2}(w_i - w_{i+1}) + w_i w_{i+2}(w_{i+2} - w_1) < 0$ so $w_i - w_{i+1} < 0$.

Unfortunately, the converse of this last corollary is not true. An ANSV match may not represent a minimal parenthesization in the MCOP. But any product that is in a minimal parenthesization by way of Equation 6.1 has been isolated by some match. Therefore, using the ANSV problem, the values in the weight list *approximate* the optimal level of parentheses.

A list of weights is *reduced* iff for all weights, say w_{i+1}, with ANSV match $[w_i, w_{i+2}]$ Equation 6.1 fails to hold, [10]. A reduced weight list may be non-monotonic.

Now we generalize Equation 6.1. Suppose by Theorem 13 that $(a_i \bullet a_{i+1})$ is in an optimal parenthesization. Now we can apply Theorem 13 to the list $l = w_1, \ldots w_{i-1}, w_i, w_{i+2}, w_{i+3}, \ldots, w_{n+1}$ in the same way. That is, if $w_i > \max\{w_{i-1}, w_{i+2}\}$ and $w_1 w_{i-1} w_{i+2} + w_{i-1} w_i w_{i+2} < w_1 w_{i-1} w_i + w_1 w_i w_{i+2}$, then the parenthesization given by the solution of the ANSV problem on l indicates that $(a_{i-1} \bullet \cdots \bullet a_{i+1})$ is optimal.

Given the weight list $l_1 = w_1, w_2, \ldots, w_{n+1}$ the approximation algorithm is [10, 18, 20]:

1. Reduce the weight list l_1 giving the weight list l_2, renumbering l_2 to be $l_2 = w_1, w_2, \ldots, w_{r+1}$ where $w_1 = \min_{1 \le i \le r+1}\{w_i\}$.

2. If l_2 has more than two weights, then compute the depths of the parentheses for the linear product $((\cdots(a_1 \bullet a_2) \bullet \cdots) \bullet a_r)$ of cost $w_1 \| w_2 : w_{r+1} \|$. With this, make the parenthesis discovered in Step 1 the appropriate amount deeper.

The depth of the parentheses determines the order to multiply the matrices.

Next we develop techniques to run this algorithm efficiently in parallel. Intuitively, by Theorem 13, if the match $[w_{i-1}, w_{i+1}]$ represents the nesting level of two parentheses in an optimal product, then we have characterized w_i's influence. Remove w_i from the weight list and recursively apply Theorem 13.

Suppose in solving the ANSV problem the weight w_j has the match $[w_i, w_k]$. Then, if

$$w_1 w_i w_k + w_i w_j w_k < w_1 w_j w_k + w_1 w_i w_j \tag{6.2}$$

and products $(a_i \bullet \cdots \bullet a_{j-1})$ and $(a_j \bullet \cdots \bullet a_{k-1})$ are both in an optimal parenthesization, then the product $(a_i \bullet \cdots \bullet a_{j-1}) \bullet (a_j \bullet \cdots \bullet a_{k-1})$ is also in an optimal parenthesization. Certainly, by Theorem 13 this is true when $i = j - 1$ and $j = k - 1$. In addition, Corollary 3 generalizes to suit Equation 6.2.

A weight list can be reduced in $O(\lg \lg n)$ time with $n/\lg \lg n$ processors on the common-CRCW PRAM model. We can reduce a weight list with two applications of the ANSV problem as follows.

Given the weight list l_1 the next algorithm outputs a reduced weight list. Let $A[1 \ldots n+1]$ be an array of $n+1$ integers all initialized to zero.

1. Solve the ANSV problem on the weight list l_1. Next check to see if there are any weights satisfying the condition described by Equation 6.1. If there are none, then output l_1 since it is reduced, then stop.

2. For all weights w_j in l_1 that have matches, say $[w_i, w_k]$, if w_j and w_i, w_k satisfy Equation 6.2, then assign a 1 to $A[j]$.

3. Now solve the ANSV problem on $A[1..n+1]$. If the nearest smaller values of $A[j]$ are in the match $[A[i], A[k]]$, then $(a_i \bullet \cdots \bullet a_{k-1})$ is in an optimal parenthesization. Removing all of the weights isolated by optimal parenthesizations gives a reduced weight list, which is output.

This algorithm produces a reduced weight list and optimal parenthesizations that have been isolated by the conditions of Equation 6.1.

The first step of this algorithm is correct by Theorem 13 and Corollary 3. The next theorem establishes the correctness of the last two steps of the algorithm.

Theorem 14 *If the ANSV match of $A[j]$ is $[A[i], A[k]]$ where $i < k$, then the product $(a_i \bullet \cdots \bullet a_{k-1})$ is in an optimal parenthesization.*

Proof: The array A contains values from the set $\{0, 1\}$, so if $A[j] = 0$ then $A[j]$ does not have an ANSV match. On the other hand, if $A[j] = 1$ then $A[j]$ must have an ANSV match since $A[1] = 0$ and $A[n+1] = 0$.

Now consider the case where $A[j]$ has match $[A[i], A[k]]$. This means that for all t such that $i < t < k$, $A[t]$ also has match $[A[i], A[k]]$. All of these matches are compatible, consequently all $A[t] = 1$ for $i < t < k$ are nested ANSV matches. This means there must be at least one list of three consecutive weights, say w_t, w_{t+1}, and w_{t+2}, that satisfy Equation 6.1. Now remove the middle such weight, w_{t+1}, and recursively continue this argument knowing that Equation 6.2 has marked the other such weights.
□

By Theorem 7, the three steps of this algorithm cost $O(\lg\lg n)$ time using $n/\lg\lg n$ processors on the common-CRCW PRAM or in $O(\lg n)$ time using $n/\lg n$ processors on the EREW PRAM and the common-CRCW PRAM.

Assume that $r + 1$ weights remain after reduction. Then renumbering and rotating the list of remaining weights gives $w_1, w_2, \cdots, w_{r+1}$ where $w_1 = \min_{1 \le i \le r+1}\{w_i\}$. The second step of the approximation algorithm requires that we now form the appropriate linear product with the remaining matrices.

The depth of the parentheses provides an approximation to within 15.5% of optimal for the MCOP. This is due to Chandra [9], Chin [10], and Hu and Shing [20].

Theorem 15 (Hu and Shing [20]) *If a weight list $w_1, w_2, \cdots, w_{r+1}$ is reduced, then the MCOP can be solved to within a multiplicative factor of* 1.155 *from optimal in constant time using n processors. This is done by choosing the linear parenthesization $((\cdots(a_1 \bullet a_2) \bullet \cdots) \bullet a_{r-1}) \bullet a_r$.*

From this theorem we see that, after a weight list is reduced, choosing a linear parenthesization with cost $w_1 \| w_2 : w_{r+1} \|$ gives the matrix product within a multiplicative factor of 1.155 from optimal.

The matrix chain order problem can be solved to within 15.5% of optimal in $O(\lg\lg n)$ time with $n/\lg\lg n$ processors.

The approximation algorithm given here is another problem whose solution built on the ANSV problem. This algorithm also shows that only a linear number of entries of a dynamic programming table give a nice approximate solution to the MCOP. That is, the path of critical nodes in the canonical subgraphs supply a good approximate solution for the matrix chain ordering problem.

This algorithm is built on the greedy principle more than the dynamic programming paradigm. In terms of the processor time product, this algorithm is optimal.

7.1 Solving the Matrix Chain Ordering Problem in Parallel

This section gives a polylog time algorithm for solving the matrix chain ordering problem that uses $n^3/\lg n$ processors. Throughout this section canonical subgraphs of the form $D^{(i,j)}$ are written as $D^{(1,m)}$ where $1 \le m \le n$. In addition, assume that D_n contains critical nodes, otherwise by Corollary 2 there is an immediate exact solution.

Canonical subgraphs can be treated atomically while finding a shortest path in a D_n graph. Further, using $(\min, +)$ matrix multiplication we can join these subgraphs together to form a shortest path in an entire D_n graph.

7.1.1 Shortest Paths Containing No Critical Nodes in Canonical Subgraphs

This subsection culminates in Theorem 18 which states that in a $D^{(1,m)}$ graph shortest paths have a very rigid structure; this result supplies the first step for finding shortest paths in canonical subgraphs. All results in this subsection apply to shortest paths from $(0,0)$ to $(1,m)$ in $D^{(1,m)}$ graphs and to shortest paths from (j,k) to (i,v) in $D^{(i,v)}_{(j,k)}$ graphs.

A path q with one jumper contains no critical nodes iff there are no critical nodes in q and there are no critical nodes in q's dual path. In particular, a jumper $(i,j) \Longrightarrow (i,k)$ contains no critical nodes if both (i,j) and (i,k) are not critical nodes and there are no critical nodes in a shortest path contributing to this jumper's weight. This generalizes to paths with more than one jumper.

In our terminology we have the following result of Hu and Shing [21, Corollary 3],

Theorem 16 (Hu and Shing [21]) *In any canonical graph, the sum of the two products* $w_i w_{j+1} w_{k+1} + w_{j+1} w_{j+2} w_{k+1}$ *where* $i < j < k$, *cannot contribute to the weight of any shortest path iff* $w_{k+1} > w_{j+1} > w_i$.

Next is a useful technical lemma.

Lemma 6 *In a* $D^{(1,m)}$ *graph if* $(i,t) \in V[\mathcal{U}]$ *then* $w_i < w_{t+1}$.

Proof: Since $(i,t) \in V[\mathcal{U}]$ and $i < t$, there must be some critical node $(i,u) \in V[p]$ where $t < u$. This means that $w_j > \max\{w_i, w_{u+1}\}$, for all $j, i < j \le u$. Since $i < t < u$ it must be that $w_i < w_{t+1}$.
□
A symmetric argument to that of Lemma 6 shows that $w_i > w_{j+1}$ for all nodes $(i,j) \in V[\mathcal{L}]$.

Theorem 17 *Any shortest path q to some vertex* (i,j) *in* $D^{(1,m)}$ *where q contains no critical nodes, except possibly* (i,j), *is a straight unit path.*

A proof of this theorem follows from an inductive application of Lemma 4 and Theorem 12.

The last theorem and all previous results of this section also apply to shortest paths from (j,k) to (i,v) in $D^{(i,v)}_{(j,k)}$ canonical subgraphs.

Jumpers of the form $(i,j) \Longrightarrow (i,k)$ such that $(j+1,k) \in V[p]$ are very important. Such a jumper contains at least one critical node, namely $(j+1,k)$.

Lemma 7 *If a horizontal shortest path q to* $(i,u) \in V[\mathcal{U}] \cup V[p]$, *is such that q has one jumper* $(i,j) \Longrightarrow (i,k)$ *where* $(j+1,k) \in V[p]$, *then q is equivalent to a shortest path to* $(j+1,k)$ *followed by* $(j+1,k) \Uparrow (i,k) \rightarrow \cdots \rightarrow (i,u)$.

The same holds for any such vertical path. A proof of this lemma follows from Lemma 1 and Theorem 3.

Suppose (j,k) and (i,t) are two critical nodes in a canonical graph $D^{(1,m)}$, where $i \le j \le k \le t$. Then there is a unit path of critical nodes from (j,k) to (i,t). With this, there are two important symmetric paths between (j,k) and (i,t):
The *upper angular* path of (j,k) and (i,t) is

$$(j,k) \Uparrow (i,k) \rightarrow \cdots \rightarrow (i,t)$$

and the *lower angular* path of (j,k) and (i,t) is

$$(j,k) \Longrightarrow (j,t) \uparrow \cdots \uparrow (i,t)$$

A *trivial* angular path has unit edges replacing the jumpers. That is, if the three unit edge connected critical nodes $(j,k) \rightarrow (j,k+1) \uparrow (j-1,k+1)$, then there is a trivial angular path $(j,k) \uparrow (j-1,k) \rightarrow (j-1,k-1)$ where $(j-1,k)$ is not a critical node. For examples of angular paths see Figure 7.9.

In a canonical subgraph the shortest path between any two critical nodes that contains no other critical nodes is an angular path. Angular paths are central to the rest of this section.

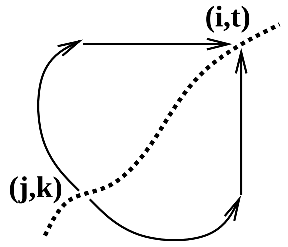

Figure 7.9: Two Angular Paths

A central result of this section is that a shortest path to a critical node (i,j) in a canonical graph may be along a path of critical nodes, then over an angular path then back to a subpath of critical nodes, then over an angular path and back to a subpath of critical nodes, etc.

Theorem 18 (Hu and Shing [21]) *A shortest path to a critical node (i,j) in a $D^{(1,m)}$ graph is either along a straight unit path to (i,j), along the path of critical nodes to (i,j), or along subpaths of critical nodes connected together by angular paths and finally to (i,j).*

A proof of this theorem follows inductively from Theorem 17 and the following. Say that there is some jumper $(i,j) \Longrightarrow (i,k)$ such that $(j+1,k) \notin V[p]$. Then, since $W((i,j) \Longrightarrow (i,k)) = f(i,j,k)+sp(j+1,k)$ and assume without loss that $(j+1,k) \in \mathcal{L}$, see Figure 7.10a. Now, by Theorem 17 we know the shortest path to $(j+1,k)$ includes the unit edge $(j+2,k) \uparrow (j+1,k)$ which has cost $w_j w_{j+1} w_{k+1}$. Additionally, assume that $(j+2,k) \notin V[p]$. But, we know $f(i,j,k) = w_i w_{j+1} w_{k+1}$ and by Lemma 6 since (i,j) and (i,k) are in $\mathcal{U}$ we know $w_i < w_{j+1} < w_{k+1}$. Therefore we can apply Theorem 16 showing that we cannot have jumper $(i,j) \Longrightarrow (i,k)$ in a minimal path in D_n. A similar case holds for Figure 7.10b.

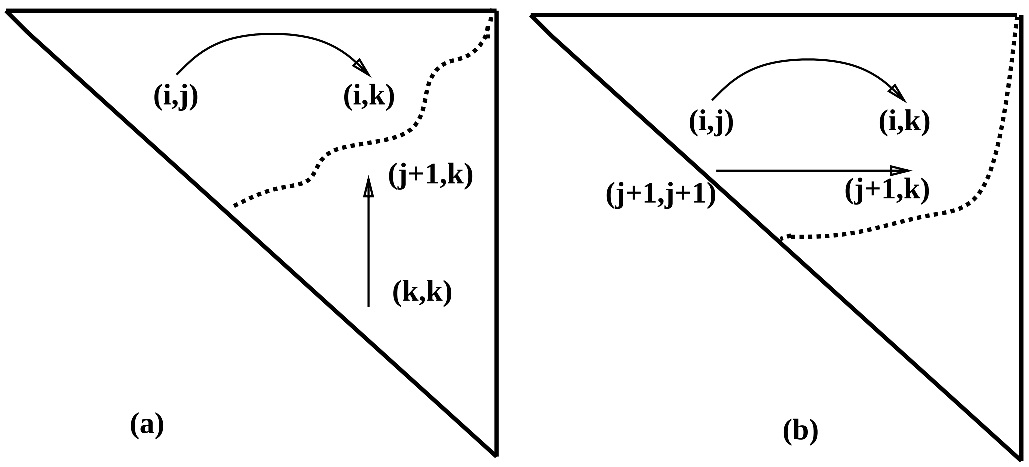

Figure 7.10: Two Jumpers and their Complimentary Paths

By the Duality Theorem and the compatibility of critical nodes, any jumpers over p have dual paths containing no jumpers over p giving the last case we discussed.

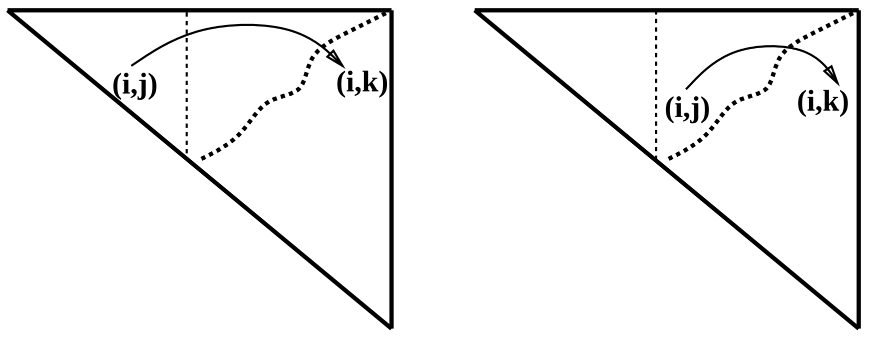

Figure 7.11: Two Jumpers Over the Path p

This last theorem and the next corollary also hold for shortest paths from (j,k) to (i,v) in $D^{(i,v)}_{(j,k)}$ graphs.

Corollary 4 *A shortest path from* $(0,0)$ *to a* non-*critical node* (s,t) *in a* $D^{(1,m)}$ *graph is either a straight unit path, or it is a minimal path to some critical node* (i,j) *and from* (i,j) *to* (s,t) *by an angular path.*

7.1.2 Combining the Canonical Graphs for an Efficient Parallel Algorithm

This subsection gives a parallel divide and conquer tool for finding minimal paths in D_n graphs based on work of Hu and Shing. This is done by isolating an underlying tree structure connecting the canonical subgraphs, so we can find shortest paths in these subgraphs individually while essentially ignoring the effect of the monotone subgraphs. This divide and conquer tool allows the computation of a shortest path in a D_n graph by applying variations of tree contraction techniques. These techniques incorporate special "leaf pruning" operations.

By Corollary 1, from here on assume

$$w_1 \;=\; \min_{1 \le i \le n+1} \{w_i\}$$

Next we present, without proof, a central result of Hu and Shing. In essence, this result gives some of the power of the greedy principle together with the principle of optimality. That is, with this result we can isolate some substructures that are necessarily in an optimal superstructure.

Theorem 19 (Hu and Shing [21]) *Given a weight list* $w_1, \ldots, w_{n+1}$ *with the three smallest weights* $w_1 < w_{i+1} < w_{v+1}$*, the products* $w_1 w_{i+1}$ *and* $w_1 w_{v+1}$ *are in some associative product(s) in an optimal parenthesization.*

There may be one or two fs that contain the products $w_1 w_{i+1}$ and $w_1 w_{v+1}$.

The next corollary is central to the results of this section. It essentially guarantees that certain easily computable nodes are a part of a shortest path in a given D_n graph. Where Hu and Shing use the results of the last theorem as a sequential divide and conquer tool, here it is made into a parallel tool.

Corollary 5 (Atomicity Corollary) *Given a weight list $w_1, \ldots, w_{n+1}$ with the three smallest weights $w_1 < w_{i+1} < w_{v+1}$ and $i+1 < v$, the critical nodes $(1,i)$ and $(1,v)$ are in a shortest path from $(0,0)$ to $(1,n)$ in D_n.*

A proof follows directly from Theorem 19.

Suppose $w_1 < w_{i+1} < w_{v+1}$ are the three smallest weights in D_n and both $D^{(1,i)}$ and $D^{(i+1,v)}$ are canonical subgraphs. Then clearly there is a shortest path from $(0,0)$ to $(1,i)$ in $D^{(1,i)}$. Also, applying the Atomicity Corollary (Corollary 5) to the subgraph $D(1,v)$, which also has the three smallest weights $w_1 < w_{i+1} < w_{v+1}$, shows that $(1,i)$ is in a minimal path to $(1,v)$. Therefore by the structure of D_n graphs the only contribution that $D^{(i+1,v)}$ can make to a shortest path to $(1,v)$ is by providing *sp* values for jumpers along the unit path $(1,i) \to \cdots \to (1,v)$. That is, there *may* be some jumper $(1,j) \Longrightarrow (1,k)$ such that $(j+1,k) \in V[D^{(i+1,v)}]$ and $(1,i) \to \cdots \to (1,j) \Longrightarrow (1,k) \to \cdots \to (i,v)$ is cheaper than $(1,i) \to \cdots \to (1,v)$. Shortly, in Lemma 9, we will see that we only have to consider jumpers $(1,j) \Longrightarrow (1,k)$ such that $(j+1,k)$ is a critical node in $D^{(i+1,v)}$.

Canonical Trees

Dividing a D_n graph into a tree of canonical subgraphs using the Atomicity Corollary (Corollary 5) is easily done in $O(\lg n)$ time with $n/\lg n$ processors by solving the ANSV problem.

In a D_n graph the structure joining all of the critical nodes is a *canonical tree*, see also Hu and Shing [19, 21, 22]. Define the leaves, edges, and internal nodes of a canonical tree as follows. Initially, in every canonical subgraph $D^{(1,m)}$ the critical node $(1,m)$ is a leaf and is denoted by $\overline{(1,m)}$ to distinguish it from other critical nodes. A $D^{(1,m)}$ canonical subgraph only contains one tree node, namely $\overline{(1,m)}$. On the other hand, an *isolated* critical node is a critical node with no critical nodes that are one unit edge away. The internal tree nodes are isolated critical nodes or $\overline{(i,v)}$ and $\overline{(j,k)}$ for $D^{(i,v)}_{(j,k)}$ canonical graphs, where $k \neq j+1$. A $D^{(i,v)}_{(j,k)}$ graph only contains the two tree nodes $\overline{(i,v)}$ and $\overline{(j,k)}$. Notice that all tree nodes are also critical nodes, thus they are compatible.

The edges of a canonical tree are the straight unit paths that connect tree nodes. Jumpers may reduce the cost of tree edges. An edge from $\overline{(i,j)}$ to $\overline{(i,k)}$ is denoted by $\overline{(i,j)} \to \cdots \to \overline{(i,k)}$ and all edges are directed towards $(1,n)$.

Since tree nodes are critical nodes with easily discernible properties, we can efficiently distinguish them in parallel. In addition, we can discard all monotone subgraphs since they have no influence on a shortest path to $(1,n)$, except if $D(1,i)$ is monotone and for some i, $w_1 = \min_{1\le i\le n+1}\{w_i\}$. That is, since $w_1 = \min_{1\le i\le n+1}\{w_i\}$, if $D(1,i)$ is monotone for some i, then a shortest path to $(1,n)$ will travel along the path $(1,1) \to \cdots \to (1,i)$. But this is the only case when a monotone graph contains a piece of a minimal path from $(0,0)$ to $(1,n)$.

For the next lemma, assume $w_1 = \min\limits_{1\le i\le n+1}\{w_i\}$

Lemma 8 *In a monotone subgraph $D(i,k)$ of D_n, the cost of a shortest path to (i,k) for $i > 1$ plus the associative weight $f(1,i-1,k)$ is more than the unit path $(1,i-1) \to \cdots \to (1,k)$.*

Proof: Since $D(i,k)$ is monotone, we either have $w_i < \cdots < w_{k+1}$ or $w_i > \cdots > w_{k+1}$. So by Theorem 10, the shortest path to (i,k) in $D(i,k)$ is either $(i,i) \to \cdots \to (i,k)$ or $(k,k) \uparrow \cdots \uparrow (i,k)$. Therefore, taking the jumper $(1,i-1) \Longrightarrow (1,k)$ with weight $sp(i,k) + f(1,i-1,k)$ we have two cases.

CASE i: The ordering of the weight list is $w_i < \cdots < w_{k+1}$.

In this case, the shortest path to (i,k) is $(i,i) \rightarrow \cdots \rightarrow (i,k)$. Clearly, $f(1,i-1,k) = w_1 w_i w_{k+1}$ and $W((1,i-1) \rightarrow (1,i)) = w_1 w_i w_{i+1}$. But since $w_{i+1} \leq w_{k+1}$, it must be $f(1,i-1,k) \geq W((1,i-1) \rightarrow (1,i))$. Along the same lines, $W((1,j) \rightarrow (1,j+1)) < W((i,j) \rightarrow (i,j+1))$ holds for all $j, i \leq j < k$.

CASE ii: The ordering of the weight list is $w_i > \cdots > w_{k+1}$.

In this case, the shortest path to (i,k) is $(k,k) \uparrow \cdots \uparrow (i,k)$. Since $f(1,i-1,k) = w_1 w_i w_{k+1}$, $W((1,k-1) \rightarrow (1,k)) = w_1 w_k w_{k+1}$, and $w_i \geq w_k$, we know that $f(1,i-1,k) \geq W((1,k-1) \rightarrow (1,k))$. Along the same lines, $W((1,i+j-1) \rightarrow (1,i+j)) < W((i+j+1,k) \uparrow (i+j,k))$ for all $j, 0 \leq j < k-i$.

□

Suppose D_n has fewer than $n-1$ critical nodes. Then D_n may have several disconnected canonical trees and one or more monotone subgraphs by Corollary 2. But, by Theorem 9 there is at least one path from $(0,0)$ to $(1,n)$ joining these canonical subtrees. From Lemma 8, after discarding irrelevant monotone subgraphs and for the moment ignoring $D^{(i,v)}_{(j,k)}$ graphs, there are several structures that $D^{(1,m)}$ graphs may form together. The relationships of tree nodes is the basis of all of these structures.

Let $\overline{(i,j)}, \overline{(j+1,k)}, \ldots, \overline{(u+1,v)}$ be neighboring leaves in $D(i,v)$ such that they are all in a canonical tree rooted at (i,v) or in no canonical tree at all. All of these leaves together can have the next relationships, or combinations of them.

CASE 1: The leaves are not joined together by internal tree nodes. For instance, this case occurs if $w_i < w_{j+1} < \cdots < w_{u+1} < w_{v+1}$.

CASE 2: The leaves form a binary canonical tree, see Figure 7.12. In this case, there are internal tree nodes in $D(i,v)$ that connect the leaves together.

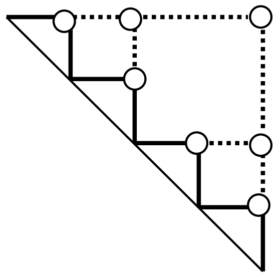

Figure 7.12: A Canonical Tree of $D^{(1,m)}$ Graphs, the Circles Denote Tree Nodes

We can solve Case 1 by creating a surrogate canonical tree and treating it as in Case 2. The viability of treating these situations as Case 2 is now shown. Take a list of r monotone leaves, none of which are in a canonical tree. Label the leaves $\overline{(i,j)}, \overline{(j+1,k)}, \overline{(k+1,t)}, \ldots, \overline{(u+1,v)}$ and without loss assume that $w_i < w_{j+1} < \cdots < w_{u+1} < w_{v+1}$ so these leaves are all in $D(i,v)$. Now, by the Atomicity Corollary we know that tree node $\overline{(i,j)}$ must be in a shortest path from $(0,0)$ to $\overline{(i,v)}$. Therefore, applying the Duality Theorem (Theorem 3) we know that a shortest path from $(0,0)$ to $\overline{(i,v)}$ goes from $(0,0)$ to $\overline{(j+1,v)}$ and then over the tree edge $\overline{(j+1,v)} \uparrow \cdots \uparrow \overline{(i,v)}$. Now, we can complete this argument by induction.

7.1.3 Finding Shortest Paths to All Critical Nodes in Canonical Subgraphs

This section discusses an algorithm for finding shortest paths in $D^{(1,m)}$ and $D^{(i,v)}_{(j,k)}$ canonical graphs. As before, p denotes the continuous path of critical nodes in a $D^{(1,m)}$ or $D^{(i,v)}_{(j,k)}$ graph.

First an $m^3/\lg m$ processor and polylog time algorithm for finding a shortest path to all critical nodes in a $D^{(1,m)}$ canonical subgraph is given. This is done by treating angular paths as edges so p is now an $(m+1)$-node graph with $\Theta(m^2)$ edges. That is, any angular path $(j,k) \Uparrow (i,k) \rightarrow \cdots \rightarrow (i,t)$ connecting the critical nodes (j,k) and (i,t) becomes *one edge* from (j,k) to (i,t) costing $W((j,k) \Uparrow (i,k)) + w_i \| w_{k+1} : w_{t+1} \|$. Of course, since (j,k) is a critical node but (i,k) is not a critical node and both are in the same canonical graph, it must be that $W((j,k) \Uparrow (i,k)) = w_i \| w_{i+1} : w_j \| + f(i,j-1,k)$. Now by Theorem 17, this also holds for $D^{(i,v)}_{(j,k)}$ canonical graphs. Further, using a $(\min,+)$ matrix multiplication shortest path algorithm, finding shortest paths in such an $(m+1)$-node graph can be done in $O(\lg^2 m)$ time with $m^3/\lg m$ processors. This results in a polylog time and $n^3/\lg n$ processor MCOP algorithm in the last subsections of the paper.

Next is the $O(\lg^2 m)$ time and $m^3/\lg m$ processor algorithm for finding a shortest path to each critical node in a $D^{(1,m)}$ graph. First compute all of the unit paths to nodes in p in constant time using m processors. Perhaps, we can best view these paths as edges from $(0,0)$ to the nodes in p. Further, the cost of each of the $\Theta(m^2)$ angular paths can be computed in constant time using m^2 processors, with preprocessing costing $O(\lg m)$ time and $m/\lg m$ processors. Now compute the shortest path to each node in p by treating every angular path as a weighted edge and applying a parallel $(\min,+)$ matrix multiplication all pairs shortest path algorithm to the nodes in p. This algorithm costs $O(\lg^2 m)$ time and $m^3/\lg m$ processors, and provides a shortest path from $(0,0)$ to every critical node in a $D^{(1,m)}$ graph. Compute the shortest paths from $(0,0)$ to all critical nodes in a $D^{(i,v)}_{(j,k)}$ canonical graph in the same way.

Theorem 20 *Given a $D^{(1,m)}$ graph we can compute a shortest path from $(0,0)$ to all nodes in p in $O(\lg^2 m)$ time using $m^3/\lg m$ processors.*

The results of this theorem also hold for finding shortest paths from all critical nodes in a $D^{(i,v)}_{(j,k)}$ canonical graph to (i,v).

Leaf Pruning and Band Merging

Here a basic technique for joining canonical subgraphs quickly in parallel is given. This technique is based on edge minimization and builds shortest paths in D_n graphs.

Take the two jumpers $(i,j) \Longrightarrow (i,t)$ and $(i,k) \Longrightarrow (i,u)$ in row i and without loss say $j < k$. Then they are not *compatible* iff $k < t < u$. If $(j+1,t) \in V[p]$ and $(k+1,u) \in V[p]$, then $(j+1,t)$ and $(k+1,u)$ are compatible. Consequently, any two jumpers in row i such as $(i,j) \Longrightarrow (i,t)$ and $(i,k) \Longrightarrow (i,u)$, where $(j+1,t) \in V[p]$ and $(k+1,u) \in V[p']$, must be compatible, where p and p' are possibly distinct paths of critical nodes. Notice that if p and p' are distinct, then they are still compatible. Given a D_n graph with the jumpers $(i,j) \Longrightarrow (i,t)$ and $(i,k) \Longrightarrow (i,u)$, let $\overline{p}$ be a minimal path from $(0,0)$ to $(1,n)$ in D_n. Then all jumpers $(i,j) \Longrightarrow (i,t)$ and $(i,k) \Longrightarrow (i,u)$ such that $(j+1,t) \in V[\overline{p}]$ and $(k+1,u) \in V[\overline{p}]$ are compatible.

Minimizing the cost of a straight unit edge path in a canonical tree by using jumpers is *edge minimizing*, and we will show that the jumpers only have to get their *sp* values from critical nodes. We will *only* edge minimize tree edges or straight unit paths in $D^{(i,v)}_{(j,k)}$ graphs. For example, let p be the path of critical nodes in $D(k,t)$ and consider the straight unit path $(i,j) \rightarrow \cdots \rightarrow (i,v)$. If the jumper $(i,k) \Longrightarrow (i,t)$ is such that $(k+1,t) \in V[p]$ and $(i,j) \rightarrow \cdots \rightarrow (i,k) \Longrightarrow (i,t) \rightarrow \cdots \rightarrow (i,v)$ is cheaper than $(i,j) \rightarrow \cdots \rightarrow (i,v)$, then we edge minimize $(i,j) \rightarrow \cdots \rightarrow (i,v)$ with $(i,k) \Longrightarrow (i,t)$.

The next procedure *edge minimizes* the unit path along the i^{th} row to the critical node (i,v) with all jumpers that get their *sp* values from the critical nodes $V[p]$,

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

and the same can be done for straight vertical unit paths. The minimal cost along the tree edge i to (i,v) is in $A[i,v]$, assuming only one connected path of critical nodes p. Notice by Theorem 6 that we can compute the cost of the straight unit (sub)paths in constant time with one processor.

Lemma 9 *When edge minimizing a tree edge $(i,j) \rightarrow \cdots \rightarrow (i,v)$ in a canonical subgraph we only have to consider jumpers $(i,k) \Longrightarrow (i,t)$ such that $(k+1,t) \in V[p]$.*

Proof: Take row i, and $(i,i) \rightarrow \cdots \rightarrow (i,v)$ assuming that some jumper $(i,s) \Longrightarrow (i,t)$ minimizes row i where $(s+1,t) \notin V[p]$ and $i < s < t \leq u$. Without loss, say $(s+1,t) \in V[\mathcal{U}]$, so a shortest path to $(s+1,t)$ is either the straight unit path $(s+1,s+1) \rightarrow \cdots \rightarrow (s+1,t)$ or this unit path with jumpers by Corollary 4.

All jumpers getting their *sp* value from a critical node in p must be compatible. Therefore, there can be only one jumper getting its *sp* value from a critical node in a shortest path from $(s+1,s+1)$ to $(s+1,t)$. Now there are three cases to consider,

Case i: A minimal path to $(s+1,t)$ is the straight unit path $(s+1,s+1) \rightarrow \cdots \rightarrow (s+1,t)$

By the Duality Theorem the minimal path along $(i,i) \rightarrow \cdots \rightarrow (i,s) \Longrightarrow (i,t)$ is equivalent to the path $(s+1,s+1) \rightarrow \cdots \rightarrow (s+1,t) \Uparrow (i,t)$. But $(s+1,s+1) \rightarrow \cdots \rightarrow (s+1,t) \Uparrow (i,t)$ is not an angular path, a straight unit path, or a path of critical nodes intermixed with angular paths getting their *sp* value from p. Therefore, we arrive at a contradiction of Corollary 4.

Case ii: A shortest path to $(s+1,t)$ is along the unit path from $(s+1,s+1)$ to $(s+1,t)$ and contains one or more jumpers whose *sp* values are *not* from $V[p]$.

This case would imply that a shortest path from $(0,0)$ to nodes in $\mathcal{U}$ are not angular paths or straight line unit paths contradicting Corollary 4.

Case iii: A shortest path to $(s+1,t)$ is along the unit path from $(s+1,s+1)$ to $(s+1,t)$ and contains one jumper that gets its *sp* value from a critical node in p.

Let $(s+1,j) \Longrightarrow (s+1,k)$ be a jumper such that $(j+1,k) \in V[p]$. Thus by the Duality Theorem (Theorem 3) we know that the path $(s+1,s+1) \rightarrow \cdots \rightarrow (s+1,j) \Longrightarrow (s+1,k) \rightarrow \cdots \rightarrow (s+1,t)$ followed by the jumper $(s+1,t) \Uparrow (i,t)$ costs the same as the path $(i,i) \rightarrow \cdots \rightarrow (i,s) \Longrightarrow (i,t)$. At the same time, the path $(s+1,s+1) \rightarrow \cdots \rightarrow (s+1,j) \Longrightarrow (s+1,k) \rightarrow \cdots \rightarrow (s+1,t)$ is equivalent to a shortest path to $(j+1,k)$ followed by the angular path $(j+1,k) \Uparrow (s+1,k) \rightarrow \cdots \rightarrow (s+1,t)$, by the Duality Theorem. This means a shortest path from $(0,0)$ to (i,t) is from $(0,0)$ to $(j+1,k)$ then $(j+1,k) \Uparrow (s+1,k) \rightarrow \cdots \rightarrow (s+1,t) \Uparrow (s+1,t)$, but this is a contradiction by Corollary 4, since this path is not an angular path.

Cases i and ii can occur at the same time, though the above arguments still hold.
□
This lemma easily generalizes to the case where p is a path of critical nodes in a conglomerate of canonical subgraphs. Lemma 9 also highlights the role well-formed subsolutions play in the dynamic programming solution of the MCOP.

Let $\overline{p}$ be a minimal path from $(0,0)$ to (k,t) in $D(k,t)$. Then we can extend the result of Lemma 9 to the case where we only have to consider jumpers $(i,k) \Longrightarrow (i,t)$ along tree edges $(i,j) \to \cdots \to (i,v)$ such that $(k+1,t) \in V[\overline{p}]$.

Theorem 21 *When edge minimizing a tree edge $(i,j) \to \cdots \to (i,v)$ in a canonical subgraph we only have to consider jumpers $(i,k) \Longrightarrow (i,t)$ such that $(k+1,t) \in V[\overline{p}]$.*

Proof: Since $(i,j) \to \cdots \to (i,v)$ is a tree edge it must be that $\max\{w_i, w_{v+1}\} < w_s$ for all $s, i \le s \le v+1$.

By contradiction, say the jumper $(i,k) \Longrightarrow (i,t)$ such that $(k+1,t) \notin V[\overline{p}]$ but $(k+1,t) \in V[p]$ is the jumper that minimizes the tree edge $(i,j) \to \cdots \to (i,v)$ more than any other jumper.

But since $(k+1,t) \notin V[\overline{p}]$, we can assume that the angular path $(r,s) \Uparrow (q,s) \to \cdots \to (q,u)$ is the angular path *in* $\overline{p}$ that goes around $(k+1,t)$, so $(r,s) \in V[\overline{p}]$ and $(q,u) \in V[\overline{p}]$. In this case, by the Duality Theorem (Theorem 3) a minimal path to (q,u) is $(q,q) \to \cdots \to (q,r-1) \Longrightarrow (q,s) \to \cdots \to (q,u)$ such that $(r,s) \in V[\overline{p}]$. In particular, notice that the jumper $(q,k) \Longrightarrow (q,t)$ saves no more than any other jumper by edge minimizing the unit path $(q,q) \to \cdots \to (q,u)$ and, since $(i,j) \to \cdots \to (i,v)$ is a tree edge, we know that $w_i < w_q$ and $f(i,k,t) < f(q,k,t)$. Thus, if $(q,k) \Longrightarrow (q,t)$ saves no more than the jumper $(q,r-1) \Longrightarrow (q,s)$ in row q, then $(i,k) \Longrightarrow (i,t)$ saves no more than any other jumper in row i.
□

This last theorem is very important. It says once a minimal path $\overline{p}$ is discovered in a subgraph $D(j,k)$ then during edge minimization we only have to consider jumpers with *sp* values from $\overline{p}$. Of course $V[\overline{p}] \subseteq V[p]$, where p is the path of critical nodes.

Theorem 22 *Given a tree node $\overline{(i,u)}$ where the graph $D(i,u)$ contains the leaves $D(i,j)$ and $D(j+1,u)$ so $\overline{(i,j)}$ and $\overline{(j+1,u)}$ are tree nodes, if $D(j+1,u)$ is a canonical subgraph, then a shortest path from $\overline{(i,j)}$ to $\overline{(i,u)}$ can be found in $O(u-j)$ operations.*

Proof: Since $(i,j), (i,u)$ and $(j+1,u)$ are all critical nodes, the three smallest weights in $w_i, \ldots, w_{u+1}$ are w_i, w_{j+1}, and w_{u+1}. Now assume without loss that $w_i < w_{u+1} < w_{j+1}$. Hence, by Corollary 5, tree node $\overline{(i,j)}$ must be in a shortest path from $(0,0)$ to (i,u). Therefore we will edge minimize the unit path $(i,j) \to \cdots \to (i,u)$. Otherwise, if $w_{u+1} < w_i < w_{j+1}$ then $(j+1,u)$ is in a shortest path from $(0,0)$ to (i,u). Therefore by the Theorem 3 we can edge minimize the path $(i,j) \to \cdots \to (i,u)$.

There could be a quadratic number of jumpers of the form $(i,k) \Longrightarrow (i,t)$ such that $j \le k < t \le u$. But by Theorem 21 we only have to consider jumpers along row i that get their *sp* values from $\overline{p}$. That is, only jumpers such as $(i,k) \Longrightarrow (i,t)$ where $(k+1,t) \in V[\overline{p}]$. The appropriate value of *sp*, which has been computed for each node in $\overline{p}$, can be retrieved and added to the appropriate value of f in constant time. Therefore we can find the minimal values for the paths between nodes (i,j) and (i,u) in $O(u-j)$ operations, since there are $O(u-j)$ such compatible jumpers.
□
The $O(u-j)$ operations are easily done in $O(\lg(u-j))$ time using $(u-j)/\lg(u-j)$ processors.

The last theorem holds for leaves in the canonical tree that have been combined and become conglomerates of other leaves and internal nodes. In this situation, jumpers derived from critical nodes in different subtrees are independent and compatible, so minimizing tree edges with them is done simultaneously. But canonical subgraphs of the form $D_{(j,k)}^{(i,v)}$ must be considered. In this case, take the leaf $D(j,k)$ that must be pruned in $D_{(j,k)}^{(i,v)}$, where $D(j,k)$ consists of the pruned subgraphs $D(j,t)$ and $D(t+1,k)$.

If there is a shortest path through (j,k) to (i,v), then by the Atomicity Corollary (Corollary 5) and depending on whether $w_j < w_{k+1}$ or $w_{k+1} < w_j$ either (j,t) or $(t+1,k)$, respectively, is in a shortest path to (j,k). Therefore, by Theorem 22 we would be done. But we must address the possibility that the cost of paths from $(0,0)$ to critical nodes throughout $D(j,k)$ can contribute to shortest paths to critical nodes from (j,k) to (i,v). In fact, we must combine the information about shortest paths in $D(j,k)$ with information about shortest paths in $D_{(j,k)}^{(i,v)}$. The combination of this shortest path information is done by edge minimizing unit paths in $D_{(j,k)}^{(i,v)}$ with *sp* values from critical nodes of $D(j,k)$.

The next theorem is another parallel divide and conquer tool, but it is for merging a $D(j,k)$ leaf into a $D_{(j,k)}^{(i,v)}$ graph.

Theorem 23 *In a $D_{(j,k)}^{(i,v)}$ graph with a shortest path $\overline{p}$ from (j,k) to (i,v) and suppose the four smallest weights are $w_i < w_{v+1} < w_{i+1} < w_v$ then $\overline{p}$ goes through one of*

1. *Either $(i+1,v)$ or $(i+1,v-1)$ or both*

2. *$(i,i+1)$ and $(i,v-1)$*

3. *$(v-1,v)$ and $(i+1,v)$*

Proof: Since $w_i < w_{v+1} < w_{i+1} < w_v$ and considering a $D_{(j,k)}^{(i,v)}$ graph, it must be that $(i,v),(i+1,v)$ and $(i+1,v-1)$ are all critical nodes, so $\overline{p}$ *may* go through them. Clearly, if $\overline{p}$ goes through $(i+1,v-1)$, then $\overline{p}$ cannot go through $(i,i+1)$ or $(v-1,v)$.

The shortest path $\overline{p}$ has a jumper up to row i, then $(i,i+1)$ will be in $\overline{p}$ in the sense of Theorem 9. Of course, if $\overline{p}$ goes through $(i,i+1)$, then $\overline{p}$ cannot go through either $(v-1,v)$ or $(i+1,v)$.

Finally, if $(v-1,v)$ is in a shortest path to (i,v), then $(i+1,v)$ is also and both $(i,i+1)$ and $(i+1,v-1)$ are not in this minimal path.
□
This last theorem also holds for $D^{(1,m)}$ graphs.

By Theorem 23 take a shortest path from $(0,0)$ to (i,v) that goes through $(i,i+1)$. Then there might be a straight unit path $(i,i+1) \rightarrow \cdots \rightarrow (i,v)$ connecting $(i,i+1)$ and (i,v). On the other hand, there *may* be some jumper $(i,j) \Longrightarrow (i,k)$ such that

$$(i,i+1) \rightarrow \cdots \rightarrow (i,j) \Longrightarrow (i,k) \rightarrow \cdots \rightarrow (i,v)$$

is cheaper than $(i,i+1) \rightarrow \cdots \rightarrow (i,v)$.

The only ways to merge canonical graphs together are given in Figure 7.13. Merging the two leaf canonical graphs as in Figure 7.13**a** is leaf pruning, and Figures 7.13**b,c** are band merging. Leaf pruning can be accomplished by edge minimizing alone after all shortest paths to critical nodes in the leaves have been found.

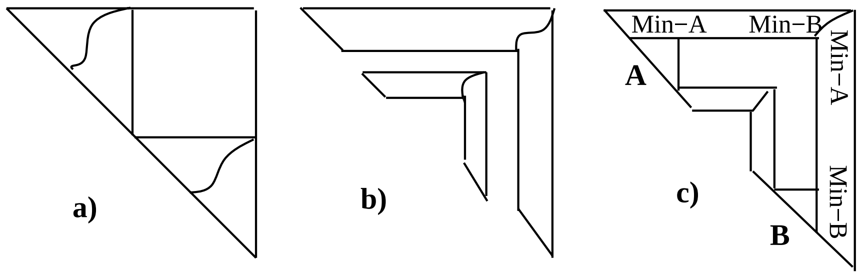

Figure 7.13: The Variations of Band Merging or Leaf Pruning

In Figure 7.13**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 Figure 7.13**b**.

To merge $D^{(i,v)}_{(j,t)}$ with $D^{(j,t)}_{(k,s)}$ can be done in $O(\lg^2 n)$ time using $n^3/\lg n$ processors. Let $\overline{p_1}$ be a minimal path from (i,v) back to $(0,0)$ *only* through $D^{(i,v)}_{(j,t)}$, and let $\overline{p_2}$ be a minimal path from (j,t) back to $(0,0)$ *only* through $D^{(j,t)}_{(k,s)}$. Now, merging these two graphs by finding all the angular paths from critical nodes in $\overline{p_2}$ forward to any critical nodes in $D^{(i,v)}_{(j,t)}$. Next, applying an all pairs shortest path algorithm combines these bands giving a shortest path from (i,v) back to $(0,0)$ through $D^{(i,v)}_{(k,s)}$.

7.1.4 Contracting a Canonical Tree

This subsection shows how to use edge minimization and band merging as the prune operations for a standard tree contraction algorithm. Together these algorithms complete an efficient solution to the MCOP.

In a tree that contains only $D^{(1,m)}$ canonical subgraphs, each prune operation is an edge minimization that joins leaves together, until the entire canonical tree is one leaf. Every critical node (i,j) has an associated variable $sp(i,j)$ where $sp(i,j)$ denotes the cost of a shortest path from $(0,0)$ to (i,j). Further, a canonical tree has at most $n-1$ critical nodes, so only $O(n)$ such variables are necessary.

Initially, all internal tree nodes have $sp(i,j) = \infty$ and all nodes in $\overline{p}$ in tree leaves have the minimum value from $(0,0)$ to (i,j) stored in $sp(i,j)$. That is, in the tree leaves the minimal paths $\overline{p}$ have been computed. In addition, for all $D^{(i,v)}_{(j,k)}$ graphs compute and store the value of the shortest path from all critical nodes in $D^{(i,v)}_{(j,k)}$ to (i,v). Compute these initial values using the methods of Subsection 7.1.3. Also, each tree edge $\overline{(i,j)} \rightarrow \cdots \rightarrow \overline{(i,k)}$ initially has weight $w_i \| w_{j+1} : w_{k+1} \|$. After the appropriate preprocessing, by Theorem 6, we can compute the initial cost of each of these $O(n)$ tree edges in constant time with one processor each.

Assume the standard tree contraction algorithm [24] except for the prune operation. Along with the new prune operation, there is an ordering of the leaves that prevents the simultaneous pruning of two adjacent leaves. Take two neighboring canonical graphs $D^{(i,j)}$ and $D^{(j+1,k)}$ with the two leaves $\overline{(i,j)}$ and $\overline{(j+1,k)}$ and the internal node $\overline{(i,k)}$, say $w_i < w_{k+1} < w_{j+1}$. Then prune leaf $\overline{(j+1,k)}$ since $\overline{(i,j)}$ is in a shortest path from $(0,0)$ to $\overline{(i,k)}$ by the Atomicity Corollary (Corollary 5). Otherwise, say $w_{k+1} < w_i < w_{j+1}$, then prune leaf $\overline{(i,j)}$. While, standard tree contraction algorithms use the Euler Tour Technique [24, 27] to number the leaves appropriately, as we have just seen the canonical nodes often provide a natural prune ordering. The viability of this natural leaf numbering follows by induction. Thus we apply the Euler Tour Technique in case the leaf ordering is arbitrary, otherwise take the natural pruning order. Take the tree made by connecting the two leaves $\overline{(i,j)}, \overline{(j+1,k)}$ to the internal node $\overline{(i,k)}$, then the pruning order is arbitrary if $w_i = w_{j+1} = w_{k+1}$.

In addition, the numbering of tree nodes by the Euler Tour Technique is used to prevent a band graph from being merged simultaneously with a band inside of it and a band around it. See standard tree collapsing techniques for details of this use of the Euler Tour Technique.

$$\begin{tikzcd}
{[i,j]} \arrow[r] & {[i,k]} \arrow[r] & {[i,u]} \\
 & {[j+1,k]} \arrow[u] &
\end{tikzcd}$$

Figure 7.14: A Small Canonical Tree

As Figure 7.14 depicts, take a canonical tree rooted at $\overline{(i,k)}$ which is the parent of tree nodes $\overline{(i,j)}$ and $\overline{(j+1,k)}$ which are siblings representing $D(i,j)$ and $D(j+1,k)$, respectively. Assume that $D(j+1,k)$ has been pruned into a leaf, and if $w_{k+1} < w_i < w_{j+1}$ then compute what contribution, if any, $\overline{(j+1,k)}$ makes to the shortest path to (i,k). This is determined by edge minimizing the tree edge $\overline{(i,j)} \to \cdots \to \overline{(i,k)}$ with all nodes in a minimal path from $(0,0)$ to $\overline{(j+1,k)}$. After this edge minimization, inactivate $\overline{(j+1,k)}$ and its parent $\overline{(i,k)}$, so $\overline{(j+1,k)}$ would not be pruned again.

Notice, the tree leaves depicted in Figure 7.15 can be pruned in any order. They are "pruned into" the edge joining them by first finding minimal paths to all critical nodes in each leaf and then, edge minimizing the edge joining them all with jumpers that get their *sp* values from the critical nodes in the leaves.

A linear list of leaves as those in Figure 7.15 can be pruned in any order. Therefore, choosing to do them simultaneously makes most sense. But, any number of nested bands must be merged in a way to avoid conflicts. This is easily done using the Euler Tour Technique for numbering appropriately for tree contraction, see [27].

The next lemma shows the correctness of pruning canonical subgraphs of the form $D^{(1,m)}$. This is necessary for canonical subtrees as in Figure 7.12.

Lemma 10 *Tree contraction of a tree of $D^{(1,m)}$ graphs with isolated internal tree nodes correctly computes a shortest path from $(0,0)$ to $(1,n)$ in D_n.*

Proof: The proof is by induction on the tree node depth, where a leaf is of depth 1.

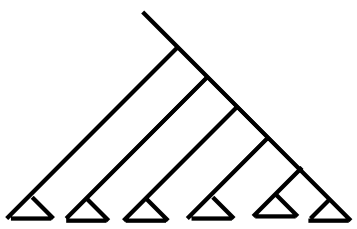

Figure 7.15: A Linear List of Tree Leaves

The case where the depth $d = 1$ is trivial, so consider when the depth is $d = 2$. Without loss say the canonical tree of depth 2 has nodes $\overline{(i,j)}, \overline{(j+1,k)}$ and $\overline{(i,k)}$, (see the subgraph $D(i,k)$ in Figure 7.14). Now by the Atomicity Corollary (Corollary 5) and since w_i, w_{j+1}, and w_{k+1} are the three smallest weights in the list $w_i, \ldots, w_{k+1}$ we know that $\overline{(i,j)}$ or $\overline{(j+1,k)}$ is in a shortest path from $(0,0)$ to $\overline{(i,k)}$. Thus, the prune operation above processes this properly.

Suppose the prune operation is correct for all trees T_d of depth d, and take the tree T_{d+1} of depth $d+1$. Without loss, say T_{d+1} contains two subtrees of depth d and the root of T_{d+1} is $\overline{(i,k)}$. In addition, if the two depth d subtrees of T_{d+1} have roots $\overline{(i,j)}$ and $\overline{(j+1,k)}$, then, by the properties of critical nodes, we know that the three smallest weights in $w_i, \ldots, w_{k+1}$ are w_i, w_{j+1}, and w_{k+1}. Without loss, say $w_{k+1} < w_i < w_{j+1}$, which by the Atomicity Corollary tree node $\overline{(i,j)}$ must be in a shortest path from $(0,0)$ to $\overline{(i,k)}$.

By the inductive hypothesis, we know that if all the subtrees rooted at $\overline{(i,j)}$ and $\overline{(j+1,k)}$ have been pruned, then we have a shortest path from $(0,0)$ to $\overline{(i,j)}$ and another to $\overline{(j+1,k)}$. Pruning leaf $\overline{(j+1,k)}$ finds all critical nodes that are in each of the two subtrees rooted at $\overline{(i,j)}$ and $\overline{(j+1,k)}$. The pruning operation is just an edge minimization to see which combinations of critical nodes may from a shortest path form $(0,0)$ to $\overline{(i,k)}$.
□

Lemma 10 shows that pruning allows us to build a shortest path from $(0,0)$ to $(1,n)$ in a tree made of leaf canonical graphs and isolated critical nodes. We can implement such tree pruning in $O(\lg n)$ time using $n/\lg n$ processors following standard tree contraction techniques. Specifically, either the Atomocity Corollary gives the leaf pruning ordering, or they can be pruned arbitrarily where we would choose an ordering like the one specified by the Euler Tour Technique.

Lemma 11 *Given two nested canonical graphs $D_{(j,u)}^{(i,v)}$ and $D_{(r,s)}^{(k,t)}$, where $j \le k < t \le u$, with no such canonical subgraph between them, we can join them with one merge operation.*

sA proof of this follows the proof of Lemma 10 in a straightforward manner.

Assume that the shortest paths in all canonical subgraphs are computed first at a cost of $O(\lg^2 n)$ time and $n^3/\lg n$ processors. Independently, compute the shortest paths of all of these canonical subgraphs. Then the pruning algorithm takes $O(\lg^2 n)$ time using $n^3/\lg n$ processors.

Now considering Lemmas 11 and 10 we can solve the MCOP by performing tree contraction with the prune operation.

Analyzing this algorithm gives the following theorem.

Theorem 24 *We can solve the MCOP in $O(\lg^3 n)$ time using $n^3/\lg n$ processors.*

This algorithm uses $O(n)$ nodes in a D_n graph to solve the MCOP. Thus we can solve the MCOP by using only $O(n)$ elements of a classical dynamic programming table.

Theorem 24 also applies to the optimal convex triangulation problem with the standard triangle cost metrics [11, 21].

7.2 Acknowledgments

Greg Shannon has been very supportive of this work; in particular he made me aware of reference [4]. Gregory J. E. Rawlins has been extremely helpful with the presentation, as was Sudhir Rao and Andrea Rafael. Alok Aggarwal has been an inspiration for this work. Artur Czumaj was also helpful. One of the anonymous referee's comments contributed greatly to this paper.

Bibliography

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

[3] S. Baase: *Computer Algorithms*, Second Edition, Addison-Wesley, 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] O. Berkman, B. Schieber and U. Vishkin: "Optimal Doubly Logarithmic Parallel Algorithms Based on Finding All Nearest Smaller Values," *J. of Algorithms*, Vol. 14, 344-370, 1993.

[6] P. G. Bradford: "Efficient Parallel Dynamic Programming," Technical Report # 352, Indiana University, April 1992.

[7] 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.

[8] 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, December 1992.

[9] A. K. Chandra: "Computing Matrix Chain Products in Near Optimal Time", IBM Research Report RC-5625, Oct. 1975.

[10] 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.

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

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

[13] A. Czumaj: "Parallel algorithm for the matrix chain product and the optimal triangulation problem (Extended Abstract)," *STACS 93*, Springer Verlag, LNCS # 665, 294-305, 1993.

[14] 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.

[15] Z. Galil and K. Park: "Parallel Dynamic Programming," 1991, Submitted.

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

[17] D. S. Hirschberg and L. L. Larmore, "The Least Weight Subsequence Problem", *SIAM J. on Computing*, Vol. 16, No. 4, 628-638, 1987.

[18] T. C. Hu: *Combinatorial Algorithms*, Addison-Wesley, 1982.

[19] T. C. Hu and M. T. Shing: "Some Theorems about Matrix Multiplication", *Proceedings of the 21st Annual IEEE Symposium on the Foundations of Computer Science*, 28-35, 1980.

[20] 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.

[21] 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.

[22] 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.

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

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

[25] L. L. Larmore and W. Rytter: "Efficient Sublinear Time Parallel Algorithms for the Recognition of Context-Free Languages", *Proceedings of 2nd Scandinavian Workshop on Algorithm Theory* 1992, Springer Verlag, LNCS #577, 1992.

[26] O. H. Ibarra, T.-C. Pong and S. M. Sohn: "Hypercube Algorithms for Some String Comparison Problems", *Proceedings of the IEEE International Conference on Parallel Processing*, 190-193, 1988.

[27] R. M. Karp and V. Ramachandran: "Parallel Algorithms for Shared Memory Machines", Chapter 17 in *Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity*, V. Van Leeuwen—editor, Elsevier, 1990.

[28] 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.

[29] 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.

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

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

[32] 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.

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