# An Approximation Algorithm for the Matrix Tree Multiplication Problem

**Mahmoud Abo-Khamis**
RelationalAI, Berkeley, CA, USA

**Ryan Curtin**
RelationalAI, Atlanta, GA, USA

**Sungjin Im**
University of California, Merced, CA, USA

**Benjamin Moseley**
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

**Hung Ngo**
RelationalAI, Berkeley, CA, USA

**Kirk Pruhs**
Department of Computer Science, University of Pittsburgh, PA, USA

**Alireza Samadian**
Department of Computer Science, University of Pittsburgh, PA, USA

**Abstract**

We consider the Matrix Tree Multiplication problem. This problem is a generalization of the classic Matrix Chain Multiplication problem covered in the dynamic programming chapter of many introductory algorithms textbooks. An instance of the Matrix Tree Multiplication problem consists of a rooted tree with a matrix associated with each edge. The output is, for each leaf in the tree, the product of the matrices on the chain/path from the root to that leaf. Matrix multiplications that are shared between various chains need only be computed once, potentially being shared between different root to leaf chains. Algorithms are evaluated by the number of scalar multiplications performed. Our main result is a linear time algorithm for which the number of scalar multiplications performed is at most 15 times the optimal number of scalar multiplications.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis

**Keywords and phrases** Matrix Multiplication, Approximation Algorithm

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2021.6

**Funding** *Sungjin Im*: Supported in part by NSF grants CCF-1617653 and CCF-1844939.
*Benjamin Moseley*: Supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.
*Kirk Pruhs*: Supported in part by NSF grants CCF-1907673, CCF-2036077 and an IBM Faculty Award.

**Acknowledgements** We want to thank David Fernandez-Baca for discussions and pointers related to Markovian phelogeny trees.

## 1 Introduction

An instance of the Matrix Tree Multiplication problem consists of an arborescence $T = (V, E)$. There is a positive integer dimension $d_v$ associated with each vertex $v$, and a $d_u$ by $d_v$ matrix $M_{u,v}$ associated with each directed edge $(u, v)$. Let $r$ be the root of $T$ and $L$ be the collection of leaves of $T$. The output is, for each leaf $\ell \in L$, the product of the matrices on the directed path from the root $r$ to that leaf $\ell$ (in that order). We restrict our attention to algorithms that use the standard matrix multiplication algorithm to multiply two matrices, which uses $ijk$ scalar multiplications to multiply an $i$ by $j$ matrix by a $j$ by $k$ matrix. We evaluate algorithms based on the aggregate number of scalar multiplications that they use. If the tree $T$ has a single leaf, then this is the classic Matrix Chain Multiplication problem that is commonly covered in the dynamic programming chapter of introductory algorithms textbooks (e.g. [4]). However, it is important to note that a Matrix Tree Multiplication instance is not equivalent to a disjoint collection of Matrix Chain Multiplication instances, one for each leaf. This is because multiplications that are shared between various chains need only be computed once, not once for each chain.

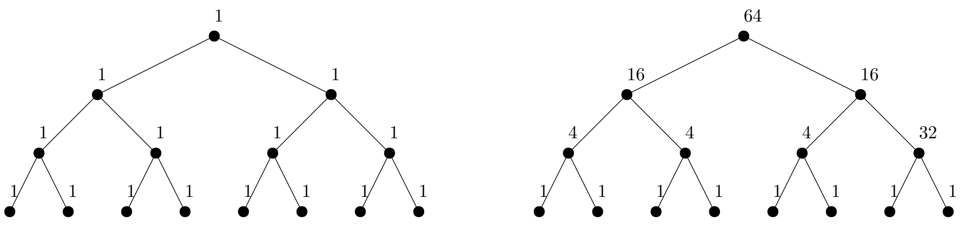
To help the reader appreciate this difference, let us consider two instances of Matrix Tree Multiplication where $T$ is a balanced binary tree of depth $\lg n$ with $n$ leaves. In the first instance, depicted in Figure 1a, all dimensions are 1. Then every feasible solution for every root-to-leaf path/chain uses $\lg n - 1$ scalar multiplications. However, the aggregate number of scalar multiplications can be quite different for different feasible solutions. To see this, if $u$ is an ancestor of $v$ in $T$, let $M_{u,v}$ denote the product of the matrices between $u$ and $v$ in $T$. We now consider two feasible solutions:

- **Top-Down:** For each root-to-leaf path $r = v_1, v_2, \dots v_k$ in $T$, the $i^{th}$ matrix multiplication is $M_{v_1,v_{i+2}} = M_{v_1,v_{i+1}} M_{v_{i+1},v_{i+2}}$ for $i \in [1, k-2]$.
- **Bottom-Up:** For each root-to-leaf path $r = v_1, v_2, \dots v_k$ in $T$, the $i^{th}$ matrix multiplication is $M_{v_{k-i-1},v_k} = M_{v_{k-i-1},v_{k-i}} M_{v_{k-i},v_k}$ for $i \in [1, k-2]$.

For Top-Down the computation of a matrix $M_{v_1,v_h}$ can be shared by all root-to-leaf paths with leaves in the subtree $T_{v_h}$ of $T$ rooted at $v_h$. If we charge the computation of $M_{v_1,v_h}$ to the vertex $v_h$, then each vertex that is neither the root nor a child of the root is charged exactly once and this charge is one. Thus, the objective value for Top-Down is $n - 3$. In contrast, for Bottom-Up none of its matrix multiplications can be shared between different paths. Thus, the objective value for the Bottom-Up algorithm is $\Theta(n \lg n)$. Thus, conceptually the advantage of Top-Down is that it minimizes the number of matrix multiplications, and maximizes the number of root-to-leaf paths that can utilize each particular matrix multiplication.

We remark that when the dimensions higher in the tree are significantly larger than the dimensions lower in the tree Bottom-Up can be significantly cheaper than Top-Down. This is because the individual matrix multiplications can be significantly cheaper. As an example consider the instance, depicted in Figure 1b. Here the dimension of a vertex at height $h$ is $2^{2h}$. Therefore, leaves have dimension 1, and the dimensions increase geometrically by a factor of 4 as one goes up the tree, with the root ultimately having dimension $n^2$. The cost of Top-Down is clearly $\Omega(n^6)$ as there are individual matrix multiplications high in the tree that have this cost. On the other hand, the cost Bottom-Up is $O(n^5)$ as its cost increases geometrically up the tree, and the last matrix multiplications cost $O(n^5)$.

One motivation for our consideration of the Matrix Tree Multiplication problem comes from Markovian models of phylogenetic trees (see for example [9, Chapter 7]). In this setting the leaves are the taxa (for example, the DNA strands for known strains of some virus such as COVID), and the internal nodes represent conjectured historic ancestors of the leaf taxa. The phylogenetic tree $T$ is thus a conjectured explanation of the evolutionary history of the leaf taxa. The matrices represent transition probabilities for mutation of a taxon in particular state to a taxon in some other state over some period of time. Multiplying the matrices on a root-to-leaf path results in an aggregate transition probability from an initial taxon state to a final leaf taxon state. This can then be used in a variety of ways, for example to find the



**(a)** A tree where all nodes have weight one.

**(b)** A tree where the node weight are geometrically decreasing.

**Figure 1** Two example trees.

initial taxon state that most likely resulted in the leaf taxa states, or for determining the likelihood that $T$ would result in the known leaf taxa given an initial distribution of states. Such applications motivate the consideration of the Matrix Tree Multiplication problem. Additionally, we are primarily interested because we believe that the problem is an interesting and natural generalization of a known classical algorithmic problem.

Another motivation for the Matrix Tree Multiplication problem comes from automatic differentiation (AD) [5, 1], which is widely used today in machine learning [1]. In AD, we are given a differentiable multivariate function $f : \mathbb{R}^p \to \mathbb{R}^q$ for some $p$ and $q$ and our target is to compute the derivative of each of the $q$ outputs with respect to each of the $p$ inputs. Those derivatives can be arranged together in a $(q \times p)$ Jacobian matrix of $f$. The function $f$ is typically represented as a computer program or a computation graph $G$ which is a DAG where vertices and directed edges represent variables and elementary functions being applied to those variables. Now consider the special case where $G$ has a tree structure $T = (V, E)$ where each vertex $v$ corresponds to a set of $d_v$ variables for some number $d_v$ and each directed edge $(u, v)$ corresponds to a multivariate function $f_{u,v} : \mathbb{R}^{d_u} \to \mathbb{R}^{d_v}$. Let $M_{u,v}$ be the transpose of the Jacobian matrix of $f_{u,v}$. Let $r$ be the root of the tree and $L$ the set of leaves. Thanks to the (multivariate) chain rule [5], the problem of computing the derivatives of the root's variables with respect to the variables of each leaf reduces to computing the multiplication of the matrices $M_{u,v}$ along each leaf-to-root path, hence to an instance of Matrix Tree Multiplication.

The standard textbook dynamic programming algorithm for the Matrix Chain Multiplication problem computes a parenthesization that results in the minimum number of scalar multiplications in time $O(n^3)$. This optimal parenthesization can be computed by a significantly more complicated algorithm that runs in time $O(n \log n)$ [7, 8]. It seems quite challenging to extend these approaches to the Matrix Tree Multiplication problem, even on very simple instances. For example, we do not know how to compute the optimal number of scalar multiplications in polynomial time even in the case that $T$ has only 2 leaves. This is because it's not clear if there are subproblems the optimum solutions to which lead to that to the original problem.

Thus, we consider approximation algorithms. First, let us review what is known in terms of approximation algorithms for Matrix Chain Multiplication. [3] and [7] cite [2] as giving a 2-approximation algorithm for the Matrix Chain Multiplication problem. [1] A 1.25 approximation algorithm was later given in [3], and finally a 1.15 approximation algorithm was given in [6]. In each case an optimal parenthesization can be computed in linear or nearly linear time.

[1] [2] is an IBM technical report that does not seem to be available on the web, and the IBM library is closed during the COVID outbreak.

Our main result is a linear-time 15-approximation algorithm, that we call the Cut Contraction algorithm, for the Matrix Tree Multiplication problem.

The rest of the paper is organized as follows. Section 2 gives a technical overview of the algorithm design and analysis. Section 3 introduces some additional notation and terminology that we will use. Section 4 describes the multiplications performed Cut Contraction algorithm (ignoring implementation details). Section 5 analyzes the approximation ratio for the Cut Contraction algorithm. Section 4.3 briefly discusses how to implement the Cut Contraction algorithm to get a linear-time algorithm that can output a parenthesization for each root to leaf path that has approximation ratio at most 15.

## 2 Technical Overview

To build some intuition, let us begin by giving a greedy 2-approximation algorithm for Matrix Chain Multiplication (which is presumably the algorithm given in [2]). Assume the vertices are $1, 2 \ldots, n$. Let $m = \arg\min_i d_i$ be the index of the minimum dimension. If you think of the chain as a path graph, then $d_m$ is the min vertex cut. Intuitively the algorithm multiplies the min-cut out to the end. So the algorithm first computes the matrix products $M_{m-i,m} = M_{m-i,m-i+1} M_{m-i+1,m}$ for $i = [2, m-1]$, $M_{m,m+i} = M_{m,m+i-1} M_{m+i-1,m+i}$ for $i = [2, n-m]$, and then finally computes $M_{1,n}$ by multiplying $M_{1,m}$ by $M_{m,n}$. This algorithm uses

$$d_1 d_n d_m + \sum_{i=1}^{m-2} d_i d_{i+1} d_m + \sum_{i=m+1}^{n-1} d_i d_{i+1} d_m$$

scalar multiplications. Observe that in any feasible solution it must be the case that for each $i \notin [m-1, m]$, the cost of the matrix multiplication that involves $M_{i,i+1}$ is at least $d_i d_{i+1} d_m$. Thus, a lower bound of the cost of optimal, that we call the edge cut lower bound, is:

$$\frac{d_1 d_n d_m + \sum_{i=1}^{m-2} d_i d_{i+1} d_m + \sum_{i=m+1}^{n-1} d_i d_{i+1} d_m}{2}$$

The factor of two comes from the fact that each matrix multiplication involves two matrices. Note that the upper bound on the cost of the algorithm is then twice this edge cut lower bound. Therefore, intuitively the edge cut lower bound assumes every edge/matrix gets multiplied by its preferred dimension, and this algorithm gives every edge its preference.

As a first step toward generalizing the algorithmic design to trees, we develop three cut based lower bounds for trees. The first lower bound is what we call the edge cut lower bound. Roughly speaking, the edge cut lower bound is

$$\sum_{(u,v)\in E} d_u d_v \alpha(u, v)/2$$

where $\alpha(u, v)$ is the minimum aggregate dimension of a cut that separates the edge $(u, v)$ from either the root or the leaves. This edge cut lower bound assumes every edge/matrix gets multiplied by its preferred dimensions. The second lower bound is what we call the root-leaf lower bound. Roughly speaking, the root-leaf lower bound is

$$\sum_{\ell\in L} d_r d_\ell \beta(\ell)$$

where $L$ is the set of leaves and $\beta(\ell)$ is the minimum dimension on the path from $r$ to $\ell$, excluding the endpoints. The final lower bound is what we call the vertex cut lower bound.

Roughly speaking, the vertex cut lower bound is

$$\sum_{v \in V} d_v \gamma(v)$$

where $\gamma(v)$ is the product of the min-cut separating $v$ from the root and the aggregate dimensions of the min-cut separating $v$ from the leaves.

With those cut lower bounds in hand, the natural path forward would be to design an algorithm in such a way that one could analyze its approximation ratio by comparing to these cut lower bounds. However, there are instances such that no matrix multiplication can be charged to the cut lower bounds (at least in a natural way). Further, there are instances where these cut lower bounds are too loose in aggregate and are more than a constant factor less than optimal. One such example is when $T$ is a a complete balanced binary tree where the dimension of the root is $n$, the dimension for vertices of height $h \in [0, \frac{\lg n}{2}]$ is $2^h$, and the dimension of the rest of the vertices are $\sqrt{n}$. For this instance, all the cut lower bounds are $O(n^2)$, but the optimal solution has cost $\Theta(n^{5/2})$.

Our algorithm for Matrix Tree Multiplication first "reduces" the tree by performing all the matrix multiplications that can naturally be charged to the cut lower bounds. Roughly speaking the multiplications that can not be charged to these cut lower bounds are those in which the middle dimension $d_v$ corresponds to a vertex $v$ that is itself the min-cut of $T_v$, the subtree of $T$ rooted at $v$. Thus, in the resulting reduced tree $R$, every node is the min-cut of its own subtree. Further it is relatively straight-forward to also ensure that the dimensions of the vertices on any root-to-leaf path in the reduced tree $R$ form a geometrically decreasing sequence. Our algorithm then performs the multiplications on the reduced tree $R$ in top-down order. We show that the above-mentioned properties of the reduced tree $R$ are sufficient to allow us to use a charging argument to directly bound the cost of these top-down multiplications by a constant factor times the cost of any arbitrary feasible solution for $T$. Here we directly charge to the optimal and not the lower bounds.

## 3 Notation and Terminology

We use $r$ to denote the root, and $T_v$ to denote the subtree rooted at vertex $v$. For any vertex $v$ and any sets of vertices $A$, let $\Pi_v(A)$ denote the set of vertices in $A$ that are descendant of $v$. Given a set $A$, we denote the collective dimensions of the vertices in $A$ by $W(A)$, that is $W(A) = \sum_{x \in A} d_x$.

We use $v \prec u$ to denote that $v$ is a strict ancestor of $u$ in $T$. We write $\preccurlyeq$ to denote that $v$ is an ancestor of $u$ and also $u$ could equal $v$. Given a vertex $v$ of $T$, we call a collection $C$ of vertices is a *cut* in $T_v$ if the removal of the vertices in $C$ leaves no remaining $v$ to leaf path in $T_v$, and there are no two vertices $u$ and $v$ in $C$ such that $v \prec u$. Given a vertex $v$ of $T$, we call a cut $C$ in $T_v$ the *min-cut* of $T_v$ if its vertices have the minimum cumulative dimensions among all cuts; that is $C = \arg\min_{C' \in \mathcal{C}} \sum_{v \in C'} d_v$.

## 4 The Cut Contraction Algorithm Description

Our Cut Contraction algorithm first partitions the tree $T$ into various components. This is described in Subsection 4.1. An algorithm, which we call Reduce, then performs matrix multiplications that can be charged to the cut lower bounds. This is described in Subsection 4.2. Finally the Top-Down algorithm, described in the introduction, is applied to the resulting reduced tree $R$.

## 4.1 Classifying Vertices in the Tree

We explain how the vertices of $T$ are classified by the algorithm before any matrix multiplications are performed. The first cut $C_1 = \mathcal{C}(r)$ is the min cut of $T$. To compute $C_{i+1}$, the algorithm iteratively considers the non-leaf vertices $u \in C_i$, and then considers all paths $P$ from $u$ to every leaf $\ell$ in $T_u$. Let $v$ be the vertex with the least depth (closest to $u$) on $P$ such that $d_v < d_u/2$. We call the vertex $v$ a *checkpoint* vertex. If $v$ does not exists then the leaf $l$ is included in $C_{i+1}$. If $v$ exists then min-cut of the subtree $T_v$ is added to $C_{i+1}$. Note that the min cut of $T_v$ can be $v$ itself.

Let $D_i^v$ be the vertices between a non-leaf vertex $v \in C_i$ and the descendants of $v$ in $C_{i+1}$ including $v$ and the descendants. That is, $D_i^v = \{u : v \preccurlyeq u \text{ and } \exists x \in C_{i+1}\ u \preccurlyeq x\}$. $U_i^v$ are the vertices in $D_i^v$ that prefer $v$ over the cut $C_{i+1}$ and $S_i^v$ are the vertices that prefer $C_{i+1}$. Formally, $U_i^v = \{u : u \in D_i^v \text{ and } d_v \leq \sum_{w \in \Pi_u(C_{i+1})} d_w\}$ and $S_i^v = D_i^v \setminus U_i^v$. Finally $D_i = \cup_{v \in C_i} D_i^v$ are the level $i$ intermediate vertices, $U_i = \cup_{v \in C_i} U_i^v$ are the level $i$ upper vertices and $S_i = \cup_{v \in C_i} S_i^v$ are the level $i$ lower vertices.

▶ **Observation 1.** *For all vertices $v \in C_i$, the vertices in $U_i^v$ is a connected component of $T$ that includes $v$.*

## 4.2 The Reduce Algorithm Description

Initially for every path $r = u_1, u_2, \ldots, u_k$ from $r$ to every vertex $u_k \in C_1$ of length at least two hops the algorithm computes the matrix products $M_{u_j,u_k} = M_{u_j,u_{j+1}} M_{u_{j+1},u_k}$, for $j \in [1, k-2]$.

Next the algorithm iteratively performs matrix multiplication on matrices between $C_i$ and $C_{i+1}$ for $i = 1, 2, ....$ To multiply matrices between $C_i$ and $C_{i+1}$ the algorithm iteratively considers vertices $v \in C_i$. The algorithm next iteratively considers vertices $u \in \Pi_v(C_{i+1})$. Let $v = u_1, u_2, \ldots, u = u_k$ be the path from $v$ to $u$. If $k \geq 3$ the algorithm then multiplies the matrices on this path in manner that we now describe (otherwise the algorithm does nothing on this path). Let $m$ be minimum such $u_{m+1}$ is not in $U_i^v$. Note that it could be that all of $u_2, \ldots, u_k$ are in $S_i^v$ and thus $m = 1$, or all of $u_1, \ldots, u_k$ are $U_i^v$ and thus $m = k$. If $m \geq 2$ the algorithm multiplies the matrices in $U_i^v$ in top-down order. That is, it computes the matrix products $M_{v,u_j} = M_{v,u_{j-1}} M_{u_{j-1},u_j}$ for $j \in [2, m]$. If $k - m \geq 2$ the algorithm multiplies the matrices in $S_i^v$ in bottom-up order. That is, the algorithm computes the matrix products $M_{u_{k-j},u_k} = M_{u_{k-j},u_{k-j+1}} M_{u_{k-j+1},u_k}$ for $j \in [2, k-m]$. Finally, if $2 \leq m \leq k-1$ the algorithm computes the matrix product $M_{v,u_k} = M_{v,u_m} M_{u_m,u_k}$. Let the resulting tree be $R$.

## 4.3 Linear Time Implementation

Here we sketch the key steps to make the algorithm run in linear time. To find the min-cuts of *every* subtree, the algorithm can start from the leaves and make them the min-cut of their subtree. Recursively in a bottom up fashion the algorithm can find the minimum cut of all the subtrees. For each vertex, the algorithm compares its dimension with the summation of the min-cuts of its children.

Once this is known, $C_1$ can be found in linear time. In order to find the checkpoints and the next cuts, we only need to perform a depth first search over the tree. Similarly, finding the sets $U_i^v$ and $S_i^v$ can be done by a depth first search over the vertices of the tree. After this step, the multiplications are well defined.

## 5 Cut Contraction Approximation Analysis

In subsection 5.1 we state and prove three cut based lower bounds on optimal. In subsection 5.2 we prove some structural properties of the classification of vertices. In subsection 5.3 we analyze the Reduce algorithm. Finally in subsection 5.4 we analyze the Top-Down algorithm on the reduced tree $R$.

### 5.1 The Cut Lower Bounds

Let $\mathcal{C}(v)$ be a min-cut of the subtree $T_v$, and let $\mathcal{C}^-(v)$ be the minimum cut of $T_v$ subject to the constraint that the cut does not contain $v$. Let $h(v)$ be the vertex $x$ with the minimum dimension subject to the constraint that $r \preccurlyeq x \prec v$. For an edge $(u,v) \in E$ define $\alpha(u,v)$ as follows:

$$\alpha(u,v) = \begin{cases} d_{h(u)} & \text{if } u \neq r \text{ and } v \in L \\ W(\mathcal{C}^-(v)) & \text{if } u = r \text{ and } v \notin L \\ \min\left(d_{h(u)}, W(\mathcal{C}^-(v)\right) & \text{otherwise} \end{cases}$$

For a leaf $\ell \in L$ define $\beta(\ell)$ as follows:

$$\beta(\ell) = \min_{u \text{ s.t.} r \prec u \prec \ell} d_u$$

For a vertex $v \in V$ that is neither the root nor a leaf, define $\gamma(v)$ as follows

$$\gamma(v) = d_{h(v)} \cdot W(\mathcal{C}^-(v))$$

▶ **Lemma 2** (Edge Cut Lower Bound).

$$\sum_{(u,v)\in E} d_u d_v \alpha(u,v) \leq 2 \cdot \mathrm{Opt}$$

**Proof.** Let $P_{u,v}$ be the set of all root-to-leaf paths passing an edge $(u,v)$. Let $A_{(u,v)}$ be the set of vertices $q$ for which, the optimum algorithm has made a multiplication of cost $d_u d_v d_q$. That is, the algorithm has performed either the multiplication $M_{u,v} M_{v,q}$ or $M_{q,u} M_{u,v}$ and let $O_{u,v}$ be the total cost of these multiplications. That is, $O_{u,v} = \sum_{q \in A_{(u,v)}} d_u d_v d_q$.

In any feasible solution, for every path $p$ in $P_{u,v}$, there should be one vertex in $p$ that is in $A_{u,v}$. That is because, in order for the algorithm to find the final product of the matrices in $p$, at some point, it must multiply $M_{u,v}$ to some other matrix in $p$. Therefore, if no ancestor of $u$ is in $A_{u,v}$, we know $A_{u,v}$ must be a cut (or its superset) in $T_v$ that is not equal to $\{v\}$. If there exists a vertex $x \in A_{u,v}$ such that $x \prec u$, then we know $O_{u,v} \geq d_x d_u d_v \geq d_{h(u)} d_u d_v$. Otherwise, as $A_{u,v}$ is a cut in $T_v$, we have $W(A_{u,v}) \geq W(\mathcal{C}^-(v))$; thus, $O_{u,v} \geq W(A_{u,v}) d_u d_v \geq W(\mathcal{C}^-(v)) d_u d_v$. Therefore, in either case, $O_{u,v} \geq d_u d_v \alpha(u,v)$. Summing over all edges $(u,v)$, we get the following value for the total cost of the multiplications involving the matrix of an edge in $T$:

$$2\mathrm{Opt} \geq \sum_{(u,v)\in E} O_{u,v} \geq \sum_{(u,v)\in E} d_u d_v \alpha(u,v)$$

We get the factor of 2 because each matrix multiplication only involves two matrices and therefore is counted at most twice. ◀

▶ **Lemma 3** (Root-Leaf Cut Lower Bound).

$$\sum_{\ell \in L} d_r d_\ell \beta(\ell) \le \mathrm{Opt}$$

**Proof.** Since the optimal solution is feasible, it must perform a multiplication of the form $M_{r,u}M_{u,\ell}$ for each leaf $\ell$ in order to computed $M_{r,\ell}$, which must cost at least $d_r d_\ell \beta(\ell)$, and cannot be shared among different leaves. ◀

▶ **Lemma 4** (Vertex Cut Lower Bound). *Let $V'$ be the set of vertices in $T$ that are neither a root nor a leaf. Then,*

$$\sum_{v \in V'} d_v \gamma(v) \le \mathrm{Opt}$$

**Proof.** Fix an edge $(u, v)$ and consider all the root to leaf paths that pass through $(u, v)$. Any feasible solution needs to compute the final product of the matrices lying on all of these paths. We first prove the following claim: for any root-to-leaf path $P$ that contains the edge $(u, v)$ there exists a multiplication of the form $M_{a,u}M_{u,b}$ that the feasible solution computes where $a$ and $b$ are two vertices in $P$ and $a \prec u \prec b$.

We can find this multiplication by the following procedure. We first consider the last multiplication that is performed on the path between $r$ and a leaf node $\ell$. Let $M_{r,w}M_{w,\ell}$, be that multiplication. If $w = u$, then we have found the multiplication. If $w \prec u$, then we recurse on the last multiplication that the algorithm has performed to calculate $M_{u,\ell}$ until we find a multiplication of the form $M_{a,u}M_{u,b}$. Lastly, if $u \prec w$, then we recurse on the last multiplication that the algorithm has performed to compute $M_{r,u}$.

Now, for an edge $(u, v)$, let $A_{(u,v)}$ be the set of all pairs of vertices $(a, b)$ such that the algorithm has computed $M_{a,u}M_{u,b}$. From the above claim we can conclude that the set $B_{(u,v)} = \{b \ : \ (a, b) \in A_{(u,v)}\}$ is a cut in the subtree $T_v$, because for every root-to-leaf path that has the edge $(u, v)$, there exists a pair of $(a, b)$ in $A_{(u,v)}$ that is on that path and $v \prec b$. Then, since these sets of multiplications are disjoint with respect to different edges $(u, v)$, we can get the following lower bound:

$$\mathrm{Opt} \ge \sum_{(u,v)\in E} \sum_{(a,b)\in A_{(u,v)}} d_a d_b d_u \ge \sum_{(u,v)\in E} d_{h(u)} d_u \sum_{b \in B_{(u,v)}} d_b \ge \sum_{(u,v)\in E} d_{h(u)} d_u W(\mathcal{C}(v)).$$

Rewriting the last summation, by summing over all vertices $u$ and then all the edges $(u, v)$ connected to $u$, and the lemma follows:

$$\mathrm{Opt} \ge \sum_{(u,v)\in E} d_{h(u)} d_u W(\mathcal{C}(v)) \ge \sum_{u \in V} d_u \gamma(u). \qquad ◀$$

## 5.2 Structural Properties

Lemma 5 states that vertices between $v$ and the cut $C(v)$ inherit their min-cut from $C(v)$. Lemma 6 lower bounds the size of min-cuts $C(u)$ for vertices $u \in D_i$. Lemma 7 observes that the dimension of every vertex in a set $C_i$ must be smaller than the dimension of any ancestor. Lemma 8 lower bounds the cut size for an edge $(u, w) \in D_i$. Lemma 9 observes that nodes in $R$ are min-cuts of their subtree. Lemma 10 observes that the dimensions are geometrically decreasing on root to leaf paths in the reduced tree $R$.

▶ **Lemma 5.** *Let $u$ be a descendant of $v$ in $T$ such that $u$ also has a descendant in $\mathcal{C}(v)$. Then $\Pi_u(\mathcal{C}(v))$ is a min-cut in $T_u$.*

**Proof.** We prove the claim by contradiction. Let us assume $\Pi_u(\mathcal{C}(v))$ is not a min-cut of $T_u$. Note that there is no vertex $x \in \mathcal{C}(v)$ such that $x \preccurlyeq u$; because, if that was the case, we could remove any vertex in $\mathcal{C}(v)$ that is descendant of $u$ and obtain a smaller cut. Therefore, since every $v$ to leaf path including the ones passing through $u$ have a vertex in $\mathcal{C}(v)$, we can conclude that $\Pi_u(\mathcal{C}(v))$ is a cut in $T_u$, and since we have assumed that it is not a min-cut, we can conclude $W(\mathcal{C}(u)) < W(\Pi_u(\mathcal{C}(v)))$.

Now we create a new cut in $T_v$ by removing the vertices in $\Pi_u(\mathcal{C}(v))$ from $\mathcal{C}(v)$ and adding $\mathcal{C}(u)$. The weight of the new cut is $W(\mathcal{C}(v)) - W(\Pi_u(\mathcal{C}(v))) + W(\mathcal{C}(u))$ which is smaller than $W(\mathcal{C}(v))$ and that is a contradiction with the fact that $W(\mathcal{C}(v))$ was the min-cut of $T_v$. ◀

▶ **Lemma 6.** *For all nonleaf vertices $v \in C_i$, and for all vertices $u \in D_i^v$ it must be the case that $W(\mathcal{C}(u)) \geq \min(d_v/2, W(\Pi_u(C_{i+1})))$.*

**Proof.** If there is a vertex $x$ in $\mathcal{C}(u)$ such that $d_x \geq d_v/2$ then the proof is trivial. Now we assume that for all vertices $x \in \mathcal{C}(u)$, we have $d_x < d_v/2$.

We divide the proof into two cases. In the first case assume that there is a checkpoint vertex $t$ such that $v \prec t \preccurlyeq u$. Then by definition, $\Pi_t(C_{i+1})$ is a min-cut of $T_t$. Furthermore, since $t$ is an ancestor of $u$, we have $\Pi_u(\Pi_t(C_{i+1}) = \Pi_u(C_{i+1})$. Then using Lemma 5, we can conclude $\Pi_u(C_{i+1})$ is a min-cut of $T_u$; therefore, $W(\mathcal{C}(u)) = W(\Pi_u(C_{i+1}))$.

In the second case, assume that there is no checkpoint between $v$ and $u$. Then for all the vertices $x \in \mathcal{C}(u)$, there exists a checkpoint vertex $t$ such that $u \prec t \preccurlyeq x$; that is because $d_x \leq d_v/2$ and there is no checkpoint above $v$. Let $T$ denote all such checkpoints, then $T$ is a cut in $T_u$ and a cut between $u$ and $\mathcal{C}(u)$. Therefore, $\bigcup_{t\in T} \Pi_t(\mathcal{C}(u)) = \mathcal{C}(u)$, and based on the definition of $C_{i+1}$ and the fact that $T$ is a cut in $T_u$, we have $\Pi_u(C_{i+1}) = \bigcup_{t\in T} \Pi_t(C_{i+1}) = \bigcup_{t\in T} \mathcal{C}(t)$.

For any checkpoint in $t$, using Lemma 5, we know $\Pi_t(\mathcal{C}(u))$ is a min-cut of $T_t$, and as a result $W(\Pi_t(\mathcal{C}(u))) = W(\mathcal{C}(t)) = W(\Pi_t(C_{i+1}))$. Summing over all vertices in $T$ we get $W(\mathcal{C}(u)) = W(\Pi_u(C_{i+1}))$. ◀

▶ **Lemma 7.** *For any nonleaf vertex $v \in C_i$ and for all ancestors $u$ of $v$, it must be the case that $d_v \leq d_u$.*

**Proof.** We use induction on $i$. For the base case of $i = 1$, we know $C_1$ is the min-cut of $T$, and if there was a vertex $u$ such that $u \prec v$ and $d_u < d_v$, we could create a smaller cut by replacing $v$ with $u$ in $C_1$.

For $i > 1$, let $q$ be the ancestor of $v$ in $C_{i-1}$. We show that $d_v$ is smaller than $d_u$ for all vertices $u$ where $q \preccurlyeq u \prec v$. Then by induction, it will be smaller than all of its ancestors because $q$ is a non leaf vertex in $C_{i-1}$. Since $v$ is not a leaf, there exists a checkpoint vertex $t$ between $v$ and $q$. Then as $v$ is in $\mathcal{C}(t)$, for all $u$ where $t \preccurlyeq u \preccurlyeq v$, we have $d_v \leq d_u$. Furthermore, based on the definition of a checkpoint, $d_t \leq d_q/2$ and for all vertices $w$ where $q \preccurlyeq w \prec t$, we have $d_w > d_q/2$; therefore, $d_v \leq d_t \leq d_w$. ◀

▶ **Lemma 8.** *For all nonleaf vertices $v \in C_i$, and for all vertices edges $(u, w)$ in $T$, with both endpoints in $D_i^v$ we have $\alpha(u, w) \geq \min(d_v/2, W(\Pi_w(C_{i+1})))$.*

**Proof.** First, for every edge $(u, w)$ with both ends in $D_i^v$, we prove

$$d_{h(u)} \geq \min(d_v/2, W(\Pi_w(C_{i+1}))). \tag{1}$$

If there exists a checkpoint $t$ such that $v \prec t \preccurlyeq u$, then for every vertex $q$ such that $t \preccurlyeq q \preccurlyeq w$, we have $d_q \geq W(\mathcal{C}(w))$. This is because thanks to the definition of $C_{i+1}$ and the fact that $q \preccurlyeq w$ we have $\Pi_t(C_{i+1}) = \mathcal{C}(t)$ and $\Pi_w(C_{i+1}) \subseteq \Pi_q(C_{i+1})$, and furthermore using Lemma 5, we know $\Pi_q(C_{i+1})$ is a min-cut of $T_q$ and $\Pi_w(C_{i+1})$ is a min-cut of $T_w$. Therefore,

$$d_q \geq W(\mathcal{C}(q)) = W(\Pi_q(C_{i+1})) \geq W(\Pi_w(C_{i+1})) = W(\mathcal{C}(w)).$$

Moreover, for every vertex $x$ where $v \preccurlyeq x \prec t$, we know $d_x \geq d_t$; therefore, we have $d_x \geq W(\mathcal{C}(w))$. Then, using Lemma 7, we can conclude $d_{h(u)} \geq W(\mathcal{C}(w)) = W(\Pi_w(C_{i+1}))$.

If there exists no checkpoint $t$ between $v$ and $u$, then based on the definition of a checkpoint and Lemma 7, we have $d_{h(u)} \geq d_v/2$. Thus, we have shown Eqn. (1).

Now, for every edge $(u, w)$ with both ends in $D_i^v$, we prove

$$W(\mathcal{C}^-(w)) \geq \min(d_v/2, W(\Pi_u(C_{i+1}))). \tag{2}$$

First, note that $W(\mathcal{C}^-(w)) \geq W(\mathcal{C}(w))$ because $\mathcal{C}(w)$ is the minimum over all cuts including the cut $\{w\}$ whereas $\mathcal{C}^-(w) \neq \{w\}$. Furthermore, note that it is either the case that there is a checkpoint between $v$ and every vertex in $\mathcal{C}(w)$ which implies $\mathcal{C}(w) = \Pi_w(C_{i+1})$, or $\mathcal{C}(w)$ has a vertex with dimension larger than $d_v/2$. Therefore, we have

$$W(\mathcal{C}^-(w)) \geq W(\mathcal{C}(w)) \geq \min(d_v/2, W(\Pi_w(C_{i+1}))).$$

Since $\alpha(u, w) = \min(d_{h(u)}, W(\mathcal{C}^-(w)))$, Eqn. (1) and (2) give the lemma. ◀

▶ **Lemma 9.** *Every vertex $v$ in $R$ that is not $r$, is the min-cut of both $T_v$ and $R_v$.*

**Proof.** The fact that $v$ is a min-cut of $T_v$ follows from the definition of the cuts $C_i$ and the definition of the Reduce algorithm. The fact that $v$ is a min-cut of $R_v$ follows from the fact that min-cuts of $R_v$ are feasible cuts for $T_v$. ◀

▶ **Lemma 10.** *For every edge $(u, v) \in R$ such that $u \neq r$ and $v$ is not a leaf, it must be the case that $d_u \geq 2d_v$.*

**Proof.** This is a direct consequence of the definition of the $C_i$'s. ◀

## 5.3 Reduce Analysis

▶ **Lemma 11.** *The cost incurred by the Reduce algorithm is at most* $8 \cdot \mathrm{Opt}$.

**Proof.** We divide the multiplications into 4 categories and analyse their costs separately. We will refer to these costs as categories.

1. The multiplications that involve the matrices between the root and the vertices in $C_1$.
2. The multiplications of the matrices with both ends in $U_i^v$ for some $v \in C_i$.
3. The multiplications involving a matrix $M_{u,w}$ where $(u, w)$ is an edge with $w$ being in $S_i^v \cup \Pi_v(C_{i+1})$ for some $v \in C_i$.
4. The multiplications of the form $M_{v,m}M_{m,u}$ where $m \in U_i^v$ and $u \in \Pi_v(C_{i+1})$.

Note that the above categories cover all the multiplications done by the Reduce algorithm. We use the lower bound in Lemma 2, and show that the cost of each multiplication in the first three categories is a constant factor of $d_u d_v \alpha(u, v)$ for some edge $(u, v)$ and no edge is charged more than once. Then we use the lower bound in Lemma 4 to bound the cost of the multiplications in the fourth category.

**Category 1:** Every matrix multiplication in this category has the form $M_{p(u)u} \cdot M_{u,v}$ where $v$ is a vertex in $C_1$ and $M_{u,v}$ is the result of the product of the matrices in path between $u$ and $v$ for some vertex $v \in C_1$. Therefore, the cost of all multiplications in this category is

$$\sum_{(u,v)\in E_1} \sum_{x\in\Pi_v(C_1)} d_u d_v d_x,$$

where $E_1$ is the set of edges between the root and $C_1$. This is because any matrix $M_{u,v}$ will be in one multiplication per each vertex of $\Pi_v(C_1)$. Note that since $C_1$ is a min-cut, for any subset $B$ of $C_1$, the value $\sum_{x\in B} d_x$ is smaller than the dimension of any of their common ancestors (otherwise, we could have got a smaller min-cut by replacing them with that ancestor). Therefore,

$$\sum_{(u,v)\in E_1} \sum_{x\in\Pi_v(C_1)} d_u d_v d_x = \sum_{(u,v)\in E_1} d_u d_v \alpha(u,v)$$

**Category 2:** Fix an integer $i$ and a vertex $v \in C_i$. For every edge $(u,w)$ with both ends in $U_i^v \setminus \{v\}$, the algorithm performs one multiplication of form $M_{v,u}M_{u,w}$ in top-down multiplication of $U_i^v$, and the cost for this multiplication is $d_v d_u d_w$. Using Lemma 8 and the definition of $U_i^v$, we know $\alpha(u,w) \geq \min(d_v/2, W(\Pi_w(C_{i+1}))) = d_v/2$. As a result the total cost of the multiplications in this category is bounded by

$$\sum_i \sum_{v\in C_i} \sum_{(u,w)\in E(U_i^v)} d_v d_u d_w \leq \sum_i \sum_{v\in C_i} \sum_{(u,w)\in E_2(v)} d_u d_w \alpha(u,w),$$

where $E_2(v)$ is the set of edges with both ends in $U_i^v$.

**Category 3:** Let $u$ be a vertex in $C_{i+1}$ and $v$ be its ancestor in $C_i$. Then the path between $u$ and $v$ can be divided into two sections such that the vertices of the upper section are all in $U_i^v \cup \{v\}$ and the vertices of the lower section are in $S_i^v \cup \{u\}$. Then on the path between $u$ and $v$, for every edge $(w,t)$ on this path for which $t$ is in $S_i^v \setminus \{u\}$, the algorithm performs the multiplication of form $M_{w,t}M_{t,u}$. Then if we sum over different vertices $u \in \Pi_t(C_{i+1})$, the total cost of the multiplications in this category that involve $M_{w,t}$ is $d_w d_t W(\Pi_t(C_{i+1}))$. Using the Lemma 8 and the definition of $S_i^v$, we have

$$2\alpha(w,t) \geq \min(d_v, 2W(\Pi_t(C_{i+1}))) \geq W(\Pi_t(C_{i+1})).$$

Therefore, the total cost of the multiplications in this category can be bounded by

$$\sum_i \sum_{v\in C_i} \sum_{(w,t)\in E_3(v)} d_w d_t W(\Pi_t(C_{i+1})) \leq \sum_i \sum_{v\in C_i} \sum_{(w,t)\in E_3(v)} 2 d_w d_t \alpha(w,t),$$

where $E_3(v)$ is the set of edges $(w,t)$ where $t$ is in $S_i^v$.

Since the edges that are above $C_1$, the edges that have both ends in $\bigcup_i \bigcup_{v\in C_i} U_i^v$, and the edges $(u,w)$ with $w$ being in $\bigcup_i \bigcup_{v\in C_i} S_i^v$ are disjoint, we have not double charged any edge. Therefore, using Lemma 2 we can conclude that the total cost of the multiplications in categories 1, 2, and 3 is at most 4Opt.

**Category 4:** Let $u$ be a vertex in $C_{i+1}$ and $v$ be its ancestor in $C_i$. Let $(q,w)$ be an edge on the path between $v$ and $u$ such that $q \in U_i^v$ and $w \in S_i^v$. The algorithm may make one multiplication of the form $M_{v,q}M_{q,u}$ for this path. Therefore, summing over all vertices $u \in \Pi_v(C_{i+1})$, we can derive the following total cost of all multiplications of this form:

$$\sum_i \sum_{v \in C_i} \sum_{(q,w) \in E_4(v)} d_v d_q W(\Pi_w(C_{i+1}))$$

in which $E_4(v)$ is the set of edges with one end in $U_i^v$ and one end in $S_i^v$.

For any edge $(q, w) \in E_4(v)$, using Lemma 6 and the fact that $w \in S_i^v$, we have

$$2W(\mathcal{C}(w)) \geq \min(d_v, 2W(\Pi_w(C_{i+1}))) \geq W(\Pi_w(C_{i+1})).$$

Furthermore, we know no checkpoint can be in $U_v^i$, because for any checkpoint $t$ we have

$$W(\Pi_t(C_{i+1})) = W(\mathcal{C}(t)) \leq d_t \leq d_v/2.$$

Therefore, using Lemma 7 and the fact that no checkpoint is in $U_v^i$, for any edge $(q, w) \in E_4(v)$ we have $h(q) \geq d_v/2$, and we get the following upperbound on the cost of the multiplications in this category:

$$\sum_i \sum_{v \in C_i} \sum_{(q,w) \in E_4(v)} d_v d_q W(\Pi_w(C_{i+1})) \leq \sum_i \sum_{v \in C_i} \sum_{(q,w) \in E_4(v)} 4 d_q d_{h(q)} W(\mathcal{C}(w)).$$

Then using Lemma 4, and taking the summation over all the edges that are not connected to the root we will get:

$$\sum_i \sum_{v \in C_i} \sum_{(q,w) \in E_4(v)} 4 d_q d_{h(q)} W(\mathcal{C}(w)) \leq 4 \sum_{u \in V} d_u \gamma(u) \leq 4\text{Opt}.$$ ◀

## 5.4 Top-Down Analysis

Our analysis of the Top-Down algorithm on the reduced tree $R$ is based on a charging argument. A few of the Top-Down multiplications will be charged to the root-leaf cut lower bound. However, most of the Top-Down matrix multiplications will be directly charged to various matrix multiplications in Opt. There are three different possible ways that the Top-Down matrix multiplications can be charged: leaf-charge, low-charge, and high-charge.

The charging is done independently for each root-to-leaf path $P$. Iteratively consider a fixed root-to-leaf path $P$ in $T$ ending in a leaf $\ell \in L$. Let $M_{r,u}M_{u,v}$ be a Top-Down matrix multiplication on $P$ that has not yet leaf-charged or low-charged any multiplication in optimal. If there is no checkpoint between $u$ and $v$ in $T$, note that we must have $v = \ell$ and the root-leaf cut lower bound is charged. We call this a leaf-charge. Otherwise, assume $u \in C_i$, $v \in C_{i+1}$ and note that there must exist a checkpoint $t$ strictly between $u$ and $v$ on $P$ (and thus $t$ can not be either the root $r$ nor a child of the root $r$ in $T$). Let $M_{r,a}M_{a,b}$ be an arbitrary matrix multiplication in the optimal solution such that $r \prec a \prec t \preccurlyeq b \preccurlyeq \ell$. We will show such a matrix multiplication must exist in the optimal solution in Lemma 12. If $b \preccurlyeq v$ then the optimal multiplication $M_{r,a}M_{a,b}$ is charged $d_r d_u d_v$, the cost of this Top-Down multiplication. Call this a low-charge. If $v \prec b$ then the optimal multiplication $M_{r,a}M_{a,b}$ is charged $d_r d_u d_b$, which is a fraction of the cost of this Top-Down multiplication. Call this a high-charge.

▶ **Lemma 12.** *For each root-to-leaf path $P$ in $T$ and for each vertex $t$ on $P$ that is neither the root nor a child of the root, at least one multiplication $M_{r,a}M_{a,b}$ is in the optimal solution for $T$ such that $r \prec a \prec t \preccurlyeq b \preccurlyeq \ell$ where $\ell$ is the leaf in $P$.*

**Proof.** Let $x_1 \prec \ldots \prec x_k = \ell$ be all the vertices in $P$ such that the optimal solution contains a multiplication of the form $M_{r,x_i} M_{x_i,x_{i+1}}$. Note that it must be the case $k \geq 2$ since the optimal solution is feasible and it needs to compute $M_{r,x_k}$. The claim is there exists $j \in [1, k-1]$ such that $x_j \preccurlyeq t \preccurlyeq x_{j+1}$, and one can take $a = x_j$ and $b = x_{j+1}$.

To prove this claim, note that $x_1$ must be a child of root. This is because, otherwise, the optimal solution needs to perform another multiplication to compute $M_{r,x_1}$ since it uses it in $M_{r,x_1} M_{x_1,x_2}$. Let $M_{r,x_0} M_{x_0,x_1}$ be that multiplication, then $x_0$ is on $P$ which contradicts with the definition of $x_1, \ldots, x_k$. Therefore, since $t$ is not the root or its children, we have $x_1 \prec t$. Also we know $x_k$ is the leaf and $t \preccurlyeq x_k$. Therefore, there exists a $j \in [1, k-1]$ such that $x_j \preccurlyeq t \preccurlyeq x_{j+1}$. ◀

▶ **Lemma 13.** *The aggregate amount of root-leaf cut charges is at most twice the root-leaf cut lower bound, and therefore at most* $2 \cdot \mathrm{Opt}$.

**Proof.** For each leaf $\ell$ there can be at most one matrix multiplication, say $M_{r,u} M_{u,\ell}$ charged to it. From Lemma 7 and the fact that there is no check point be between $u$ and $\ell$ one can conclude that $\beta(\ell) \leq 2d_u$. ◀

▶ **Lemma 14.** *Every Top-Down matrix multiplication $M = M_{r,u} M_{u,v}$ charges at least $d_r d_u d_v$ to the multiplications in the optimal solution.*

**Proof.** If $M$ was charged via a high charge, this is obvious. Otherwise assume $M$ was only charged via low charges. Let $(a_1, b_1), \ldots (a_k, b_k)$ be the collection of multiplications in optimal that $M$ was charged to via low charges. By the feasibility of the optimal solution $\{b_1, \ldots, b_k\}$ must be a cut of $T_v$. Thus by Lemma 9 it must be the case that $\sum_{i=1}^{k} d_{b_i} \geq d_v$. Thus the aggregate amount of low charges is at least $d_r d_u d_v$. ◀

▶ **Lemma 15.** *Every matrix multiplication $M_{r,a} M_{a,b}$ in optimal is charged at most $2 d_r d_a d_b$ by low charges.*

**Proof.** Let $M_{r,u} M_{u,v}$ with $u \in C_i$ and $v \in C_{i+1}$ be one of the multiplications of top-down that low-charges $M_{r,a} M_{a,b}$. Then we know there exists a checkpoint $t$ in $T$ such that $u \prec t \preccurlyeq v$ and $a \prec t \preccurlyeq b \preccurlyeq v$. Then the claim is that the only multiplications of top-down that may low-charge the multiplication $M_{r,a} M_{a,b}$ in optimal are the ones of the form $M_{r,u} M_{u,w}$ where $w \in \Pi_b(C_{i+1})$.

To see the reason for the above claim, consider any multiplication $M_{r,p} M_{p,q}$ that can low-charge $M_{r,a} M_{a,b}$. Based on the definition of low-charge, we have $p \in C_j$ is an ancestor of $b$, and $w \in C_{j+1}$ such that $b \preccurlyeq w$. If $j + 1 \leq i$, then no vertex in $C_{j+1}$ can be an descendent of $u$, and therefore, we cannot have $b \preccurlyeq w$ because that would imply $u \prec b \preccurlyeq w$. Furthermore, we cannot have $j \geq i + 1$ because we already know that $v$ is in $C_{i+1}$ and $b \preccurlyeq v$; therefore, no vertex in $C_j$ can be an ancestor of $b$, meaning we cannot have $p \prec b$; because that would mean $p \prec u$. Thus, the only possibility is $j = i$, which means $p = u$. Then the only vertices in $C_{i+1}$ that are descendent of $b$ are by definition $\Pi_b(C_{i+1})$.

Using this claim, we can conclude the maximum amount low-charged to a multiplication $M_{r,a} M_{a,b}$ is $d_r d_u W(\Pi_b(C_{i+1}))$. Note that $a \prec t$, therefore, $d_a > d_u/2$; that is because, all the vertices between $t$ and $u$ have dimensions larger than $d_u/2$, and using Lemma 7, we know the dimension of all the ancestors of $u$ is at least $d_u$. Furthermore, note that $\Pi_t(C_{i+1})$ is the min-cut of $T_t$; therefore, using lemma 5 and the fact that $t \preccurlyeq b \preccurlyeq v$, we can conclude $W(\Pi_b(C_{i+1})) < d_b$. Therefore, we can get the following upperbound for the total cost lower-charged to a multiplication $M_{r,a} M_{a,b}$ of optimal: $d_r d_u W(\Pi_b(C_{i+1})) \leq 2 d_r d_a d_b$. ◀

▶ **Lemma 16.** *Every matrix multiplication $M_{r,a} M_{a,b}$ in optimal is charged at most $4 d_r d_a d_b$ by high-charges.*

**Proof.** Consider a multiplication $M_{r,u}M_{u,v}$ in top-down that high-charges the multiplication $M_{r,a}M_{a,b}$ in optimal. We have $a \prec t \preccurlyeq v \preccurlyeq b$ where $t$ is the checkpoint between $u$ and $v$, and $M_{r,u}M_{u,v}$ high-charges the cost $d_r d_u d_b$. Note that both $t$ and $v$ are on the path between $a$ and $b$ in $T$. Let $(u_1, u_2), (u_2, u_3), \dots, (u_{k-1}, u_k)$ be the edges in $R$, for which $u_2, \dots, u_k$ are all on the path between $a$ and $b$ in $T$, and the checkpoint between $u_i, u_{i+1}$ for all $i$ is also on this path. Using the definition of high-charge, the only multiplications in Top-Down that can high charge $M_{r,a}M_{a,b}$ are the multiplications of the form $M_{r,u_i}M_{u_i,u_{i+1}}$ for $i \in [1, k-1]$. Therefore, the total cost high-charged to $M_{r,a}M_{a,b}$ is at most: $d_r d_b \sum_{i=1}^{k} d_{u_i}$ Using Lemma 10, we know $d_{u_{i+1}} \le d_{u_i}/2$. Furthermore, using the definition of the checkpoint and Lemma 7, we have $d_{u_1} \le 2d_a$ because $a$ is the ancestor of the checkpoint between $u_1$ and $u_2$ in $T$. Therefore, the total cost can be upper bounded as follows: $d_r d_b \sum_{i=1}^{k} d_{u_i} \le d_r d_b d_{u_1} \sum_{i=0}^{\infty} 1/2^i \le 4 d_r d_b d_a$ ◀

We now can prove the main theroem.

▶ **Theorem 17.** *The Cut Contraction Algorithm for the Tree Matrix Multiplication problem is* 15 *approximate.*

**Proof.** Using Lemma 11, we can conclude the cost of Reduce multiplications is 8Opt, and using Lemmas 13, 14, 15, and 16, we can conclude the cost of the multiplications performed in TopDown phase is 7Opt which gives us total cost of 15Opt. ◀

## 6 Conclusions

In this paper we studied a natural extension of the matrix chain problem where multiples chains are overlaid forming a tree. Currently, we do not know if the problem is NP-hard although we believe so. The obvious open question is to show that the problem is indeed NP-hard. Further, we do not know how to obtain a better approximation using any polynomial time algorithms. Improving the approximation ratio would be another interesting direction. Finally, it would be very interesting to study the more general setting where the chains form an arbitrary DAG. The main challenge in such an extension seems to lie in discovering lower bounds different from what we used in this paper.

## References

1 Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: A survey. *J. Mach. Learn. Res.*, 18(1):5595–5637, 2017.

2 A. K. Chandra. Computing matrix chain products in near optimal time. *IBM Research Report*, RC 5625(24393), 1975. IBM T.J. Watson Research Center.

3 Francis Y. L. Chin. An o(n) algorithm for determining a near-optimal computation order of matrix chain products. *Communications of the ACM*, 21(7):544–549, 1978.

4 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009.

5 Andreas Griewank and Andrea Walther. *Evaluating Derivatives.* Society for Industrial and Applied Mathematics, second edition, 2008. doi:10.1137/1.9780898717761.

6 T. C. Hu and M. T. Shing. An o(n) algorithm to find a near-optimum partition of a convex polygon. *Journal of Algorithms*, 2(2):122–138, 1981.

7 T. C. Hu and M. T. Shing. Computation of matrix chain products. part I. *SIAM Journal of Computing*, 11(2):362–373, 1982.

8 T. C. Hu and M. T. Shing. Computation of matrix chain products. part II. *SIAM Journal of Computing*, 13(2):228–251, 1984.

9 Mike Steel. *Phylogeny: Discrete and Random Processes in Evolution.* SIAM-Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2016.