Truly Sub-cubic Algorithms for Language Edit Distance and RNA Folding via Fast Bounded-Difference Min-Plus Product[*]

Karl Bringmann[†], Fabrizio Grandoni[‡], Barna Saha[§], Virginia Vassilevska Williams[¶]

April 5, 2016

Abstract

It is a major open problem whether the the $(\min, +)$-product of two $n \times n$ matrices has a truly sub-cubic (i.e. $O(n^{3-\varepsilon})$ for $\varepsilon > 0$) time algorithm, as it is equivalent to the famous All-Pairs-Shortest-Paths problem (APSP) in n-vertex graphs. There are a few restrictions of the $(\min, +)$-product to special types of matrices that admit truly sub-cubic algorithms, each giving rise to a special case of APSP that can be solved faster. In this paper we consider a new, different and powerful restriction in which one matrix can be arbitrary, as long as the other matrix has "bounded differences" in either its columns or rows, i.e. any two consecutive entries differ by only a small amount. We obtain the first truly sub-cubic algorithm for this Bounded Differences $(\min, +)$-product.

Our new algorithm, combined with a strengthening of an approach of L. Valiant for solving context-free grammar parsing with matrix multiplication, yields the first truly sub-cubic algorithms for the following problems: Language Edit Distance (a major problem in the parsing community), RNA-folding (answering an open problem of Chan and Lewenstein) and Optimum Stack Generation (answering an open problem of Tarjan).

[*]This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing.

[†]Max Planck Institute for Informatics, Saarbrücken, Germany, `kbringma@mpi-inf.mpg.de`.

[‡]IDSIA, University of Lugano, `fabrizio@idsia.ch`. This work was partially supported by the ERC StG project NEWNET no. 279352 and the SNSF project APPROXNET no. 200021_159697/1

[§]University of Massachusetts Amherst, College of Information and Computer Science, Amherst, MA. `barna@cs.umass.edu`. This work is partially supported by a NSF CCF 1464310 grant, a Yahoo ACE Award and a Google Faculty Research Award.

[¶]Stanford University, `virgi@cs.stanford.edu`. Partially supported by NSF Grants CCF-1417238, CCF-1528078 and CCF-1514339, and BSF Grant BSF:2012338.

1 Introduction

The $(\min,+)$-product (also called *min-plus* or *distance* product) of two integer matrices A and B is the matrix $C = A \star B$ such that $C_{i,j} = \min_k\{A_{i,k} + B_{k,j}\}$.[1] Computing a $(\min,+)$-product is a basic primitive used in solving many other problems. For instance, Fischer and Meyer [17] showed that the $(\min,+)$-product of two $n \times n$ matrices has essentially the same time complexity as that of the All Pairs Shortest Paths problem (APSP) in n node graphs, one of the most basic problems in graph algorithms. APSP itself has a multitude of applications, from computing graph parameters such as the diameter, radius and girth, to computing replacement paths and distance sensitivity oracles (e.g. [12, 45, 21]) and vertex centrality measures (e.g. [13, 2]).

The $(\min,+)$-product of two $n \times n$ matrices has a trivial $O(n^3)$ time algorithm, and it is a major open problem whether there is a *truly sub-cubic* algorithm for this problem, i.e. an $O(n^{3-\varepsilon})$ time algorithm for some constant $\varepsilon > 0$. Following a multitude of polylogarithmic improvements over n^3 (e.g. [18, 41, 15]), a relatively recent breakthrough of Williams [48] obtained an $O(n^3/c^{\sqrt{\log n}})$ time algorithm for a constant $c > 1$. Note that this striking improvement is still sub-polynomial.

For restricted types of matrices, there are truly sub-cubic algorithms. The probably most relevant examples include:

(1) when all matrix entries are integers bounded in absolute value by M, then the problem can be solved in $\tilde{O}(Mn^\omega)$ time [6], where $\omega < 2.373$ is the matrix multiplication exponent [44, 20];

(2) when each row of matrix A has at most C distinct values, then the $(\min,+)$-product of A with an arbitrary matrix B can be computed in time $\tilde{O}(Cn^{(3+\omega)/2})$ [49].[2]

Each of these restricted $(\min,+)$-products have applications. E.g., the distance product of type (1) is used to compute APSP in both undirected [39, 40] and directed [51] graphs with bounded edge weights, while the distance product of type (2) is used to compute APSP in graphs in which each vertex has a bounded number of distinct edge weights on its incident edges [49].

1.1 Our Result

In this paper we significantly extend the family of matrices for which a $(\min,+)$-product can be computed in truly sub-cubic time to include the following class.

Definition 1. *A matrix X with integer entries is a W-*bounded differences *(W-BD) matrix if for every row i and every column j, the following holds*

$$|X_{i,j} - X_{i,j+1}| \le W \quad \text{and} \quad |X_{i,j} - X_{i+1,j}| \le W$$

When $W = O(1)$, we will refer to X as a bounded differences (BD) matrix.

In this paper we present the first truly sub-cubic algorithm for $(\min,+)$-product of BD matrices, answering a question of Chan and Lewenstein [16].

Theorem 1. *There is an $\tilde{O}(n^{2.8244})$ time randomized algorithm and an $\tilde{O}(n^{2.8603})$ time deterministic algorithm that computes the $(\min,+)$-product of any two $n \times n$ BD matrices.*

Indeed, our algorithm produces a truly sub-cubic running time for W-BD matrices for non-constant values of W as well, as long as $W = O(n^{3-\omega-\varepsilon})$ for some constant $\varepsilon > 0$. In fact, we are able to prove an even more general result: suppose that matrix A only has bounded differences in its rows or its columns (and not necessarily both). Then, A can be $(\min,+)$-multiplied by an arbitrary matrix B in truly sub-cubic time:

[1]By $M_{i,j}$ we will denote the entry in row i and column j of matrix M.

[2]The same holds if A is arbitrary and B has at most C distinct values per column.

Theorem 2. *Let B be arbitrary and assume either of the following:*

$$\text{(1) } \forall i,j \in [n],\ |A_{i,j} - A_{i+1,j}| \le W \quad \text{or} \quad \text{(2) } \forall i,j \in [n],\ |A_{i,j} - A_{i,j+1}| \le W$$

If $W \le O(n^{3-\omega-\varepsilon})$ for any $\varepsilon > 0$, then $A \star B$ can be computed in randomized $O(n^{3-\Omega(\varepsilon)})$ time. If $W = O(1)$, then $A \star B$ can be computed in randomized time $O(n^{2.9217})$.

The main obstacle towards achieving a truly sub-cubic algorithm for $(\min,+)$ product in general is the presence of entries of large absolute value. In order to compare our result with (1) and (2) from that point of view, assume for a moment that $\omega = 2$ (as conjectured by many). Then (1) can perform a $(\min,+)$ product in truly sub-cubic time if *both* A and B have entries of absolute value at most $M = O(n^{1-\varepsilon})$ for some constant $\varepsilon > 0$, while (2), without any other assumptions on A and B, can do that if *at least* one of A and B has entries of absolute value at most $M = O(n^{1/2-\varepsilon})$. We can do the same when *at least one* of A and B has entries of absolute value at most $M = O(n^{1-\varepsilon})$.

1.2 Our Approach

Our approach has three phases.

Phase 1: additive approximation $\tilde{C}$ of the product $C = A \star B$. It turns out that for BD matrices it is quite easy to obtain an additive overestimate $\tilde{C}$ of C: Let us subdivide A and B in square blocks of size $\Delta \times \Delta$, for some small polynomial value Δ. Thus the overall product reduces to the multiplication of $O((n/\Delta)^3)$ pairs of blocks (A', B'). By the bounded differences property, it is sufficient to compute $A'_{i,k} + B'_{k,j}$ for any triple of indexes (i,k,j) in order to obtain an overestimate of all the entries in $A' \star B'$ within an additive error in $O(\Delta W)$. This way in truly sub-cubic time we can compute an $O(\Delta W)$ additive overestimate $\tilde{C}$ of C.

It would seem that Phase 1 requires that the matrices are BD, so that one would not be able to use the same approach to attack the $(\min,+)$ product of general matrices. We note that this is NOT the case: Phase 1 can be performed for *arbitrary* integer matrices A and B as well, provided one has an algorithm that can *fix* $\tilde{C}$ and obtain C from it: We use a scaling approach ala Seidel [39]. Assume that the entries of A and B are nonnegative integers bounded by M^3, and obtain A' and B' by setting $A'_{i,j} = \lceil A_{i,j}/2 \rceil$ and $B'_{i,j} = \lceil B_{i,j}/2 \rceil$. Recursively compute $A' \star B'$, where the depth of the recursion is $\log M$ and the base case is when the entries of A and B are bounded by a constant in which case $A' \star B'$ can be computed in $O(n^\omega)$ time. Then, in each recursive step we can set $\tilde{C}_{i,j} = 2C_{i,j}$ for all i,j. This gives an overestimate that errs by at most an additive 2 in each entry. Thus, as long as the rest of the phases can be made to work for arbitrary matrices to fix up $\tilde{C}$, this phase would also work.

Phase 2: Correcting $\tilde{C}$ up to a few bad triples. The heart of our approach comes at this point. We perform a (non-trivial) perturbation of A and B, and then set to ∞ the entries of absolute value larger than $c \cdot \Delta W$ for an appropriate constant c. The perturbation consists of adding the same vector V_A^r (resp., V_B^r) to each column of A (resp., row of B). Here V_A^r and V_B^r are random vectors derived from the estimate $\tilde{C}$. Let A^r and B^r be the resulting matrices. Using (1) we can compute $C^r = A^r \star B^r$ in truly sub-cubic time $O(\Delta W n^\omega)$ for sufficiently small W and Δ. The perturbation is such that it is possible to derive from $(C^r)_{i,j}$ the corresponding value $(A \star B)_{i,j} = A_{i,k} + B_{k,j}$ *unless* one of the entries $A^r_{i,k}$ or $B^r_{k,j}$ was *rounded to* ∞.

The crux of our analysis is to show that if we do $\tilde{O}(n^{3d})$ perturbations (for some d) and associated bounded entry $(\min,+)$-products, then there is only a small number ($O(n^{3-d})$) of triples (i,k,j) for which (a) $|A_{i,k} + B_{k,j} - \tilde{C}_{i,j}| \le c' \cdot \Delta W$ for some $c' = O(1)$ (i.e. k is a potential witness for $C_{i,j}$) and (b) none of the perturbations had both $A^r_{i,j}$ and $B^r_{i,j}$ finite.

[3]We can assume that M is a power of 2.

Interestingly, our proof of the correctness of Phase 2 relies on an extremal graph theoretical lemma that lower bounds the number of 4-cycles in sufficiently dense bipartite graphs.

In a sense Phase 1 and 2 only leave $O(n^{3-d})$ work to be done: if we knew these "bad" triples that are not covered by the perturbation steps, we could just go through them in a brute-force way, fixing $\tilde{C}$. Since Phase 2 does not use the fact that A and B are BD, if we could find the bad triples efficiently, we would obtain a truly sub-cubic algorithm for $(\min, +)$-matrix product!

Phase 3: Finding and fixing the bad triples. To fix the bad triples, one could try to keep track of the triples handled in each perturbation iteration. For arbitrary matrices A and B this wouldn't give a truly sub-cubic algorithm as the number of triples is already n^3. For BD-matrices however one doesn't need to keep track of all triples, but rather only of triples formed by the upper-most left-most entries of the blocks from Phase 1 since these entries are good additive approximations of all block entries. The number of these block representative triples is only $O(n/\Delta)^3)$ where Δ is the block size (from Phase 1). Thus, instead of spending more than n^3 time, one can get away with spending $O(\rho \cdot (n/\Delta)^3)$ time where ρ is the number of perturbation iterations (from Phase 2). After we find the bad block representative triples, we can go through their blocks in a brute-force manner to fix $\tilde{C}$ and compute C. Since each triple in the blocks of a bad block representative triple must also be bad, the total number of triples considered by the brute-force procedure must be $O(n^{3-d})$ as this is the total number of bad triples.

We reiterate that this is the *only* part of the algorithm that does not work for arbitrary A and B.

1.3 Applications

The notion of BD matrices is quite natural and has several applications. Indeed, our original motivation for studying the $(\min, +)$-product of such matrices came from a natural scored version of the classical Context-Free Grammar (CFG) parsing problem. It turns out that a fast algorithm for a bounded difference version of scored parsing implies the first truly sub-cubic algorithms for a few well-studied problems such as Language Edit Distance, RNA-Folding and Optimum Stack Generation.

Recall that in the *parsing* problem we are given a CFG G and string $\sigma = \sigma_1 \dots \sigma_n$ of n terminals. Our goal is to determine whether σ belongs to the language L generated by G. For ease of presentation and since this covers most applications, we will assume unless differently stated that the size of the grammar is $|G| = O(1)$, and will not explicitly mention the dependency of running times on grammar size.[4] We will also assume G is given in the Chomsky Normal Form (CNF)[5]. In a breakthrough result [43] Valiant proved a reduction from parsing to Boolean matrix multiplication: the parsing problem can be solved in $O(n^\omega)$ time.

One can naturally define a scored generalization of the parsing problem (see, e.g., [4]). Here each production rule p in G has an associated integer score (or cost) $s(p)$. Our goal is now to find a sequence of production rules of minimum total score that generates a given string σ. It is relatively easy to adapt Valiant's parser to this goal, the main difference being that Boolean matrix multiplications are replaced by $(\min, +)$-products. It follows that scored parsing can be solved up to logarithmic factors in the time needed to perform one $(\min, +)$-product (see also [38]). In particular, applying Williams' algorithm for $(\min, +)$-product [48], one can solve scored parsing in $O(n^3/2^{\Theta(\sqrt{\log n})})$ time, which is the current best running time for this problem.

For a nonterminal X let $s(X, \sigma)$ be the minimum total score needed to generate σ from X (where G is assumed to be clear from the context). Let us define a bounded difference notion

[4]Our approach also works when $|G|$ is a sufficiently small polynomial.

[5] Note that it is well-known that any context free grammar can be transformed into an equivalent CNF grammar.

for CFGs: Intuitively, this means that adding or deleting a terminal at one endpoint of a string does not change the corresponding score by much.

Definition 2. *A CFG G is a W-bounded differences (W-BD) grammar if, for any non-terminal X, terminal x, and string σ of terminals, the following holds:*

$$|s(X,\sigma) - s(X,\sigma x)| \leq W \quad \text{and} \quad |s(X,\sigma) - s(X,x\sigma)| \leq W$$

When $W = O(1)$, we will refer to G as a bounded differences (BD) grammar.

Via a simple but very careful analysis of the scored version of Valian't parser, we are able to show that the scored parsing problem on BD grammars can be reduced to $(\min,+)$-product of BD matrices (see Section A).

Theorem 3. *Let $O(n^\alpha)$ be the time needed to perform one $(\min,+)$-product between two $n \times n$ BD matrices. Then the scored parsing problem on BD grammars in CNF can be solved in $\tilde{O}(n^\alpha)$ time.*

Corollary 1. *The scored parsing problem on BD grammars in CNF can be solved in $\tilde{O}(n^{2.8244})$ randomized time and $\tilde{O}(n^{2.8603})$ deterministic time.*

BD grammars appear naturally in relevant applications. Consider for example the well-studied Language Edit Distance problem (LED) [4, 31, 27, 37, 38, 1, 35]. Here we are given a CFG G and a string σ of terminals. We are allowed to *edit* σ by *inserting*, *deleting* and *substituting* terminals. Our goal is to find a sequence of such edit operations of minimum length so that the resulting string σ' belongs to the language L generated by G[6]. As already observed by Aho and Peterson in 1972 [4], LED can be reduced to scored parsing. In more detail, it is sufficient to assign score zero to the production rules of the input grammar, and then augment the grammar with production rules of score 0 and 1 that model edit operations. By performing the above steps carefully, the resulting scored grammar is BD, leading to a truly sub-cubic algorithm for LED via Corollary 1 (see Appendix C). We remark that finding a truly-sub-cubic algorithm for LED was wide open even for very restricted cases. For example, consider *Dyck LED*, where the underlying CFG represents well-balanced strings of parentheses. Developing fast algorithms for Dyck LED, and understanding membership in parenthesis language has recently received considerable attention [9, 37, 25, 14, 28, 33]. Even for such restricted grammars, no truly sub-cubic exact algorithm was known prior to this work.

Another relevant application is related to *RNA-folding*, a central problem in bioinformatics since its definition by Nussinov and Jackobson in 1980 [32]. We can rephrase RNA folding as follows. We are given the CFG $S \to SS \mid aSu \mid uSa \mid cSg \mid gSc \mid \epsilon$. Here terminals $\{a, u, c, g\}$ represent the 4 types of nucleotides in an RNA molecule. We have to find the minimum number of *insertions* and *deletions* of nucleotides on the RNA molecule that will generate a string consistent with the above grammar. This is essentially a variant of LED where only insertions and deletions (and no substitutions) are allowed. Nussinov and Jackobson proposed a simple $O(n^3)$ time algorithm to solve RNA-folding. Despite considerable efforts [46, 5, 50, 32], no truly sub-cubic algorithm for RNA folding was known prior to our work. By essentially the same argument as for LED, it is easy to obtain a BD scored grammar modeling RNA folding. Thus we immediately obtain a truly sub-cubic algorithm to solve this problem via Corollary 1.

As a final application, consider the Optimum Stack Generation problem (OSG) described by Tarjan in [42]. Here, we are given a finite alphabet Σ, a stack S, and a string $\sigma \in \Sigma^*$. We would like to print σ by a minimum length sequence of three stack operations: *push()*, *emit* (i.e., print the top character in the stack), and *pop*. For example, the string $BCCAB$ can be printed via the sequence of operations: *push*(B), *emit*(B), *push*(C), *emit*(C), *emit*(C) *pop*(C), *push*(A),

[6]In some variants of the problem each edit operation has some integer cost upper bounded by a constant. Our approach clearly works also in that case.

emit(A) *pop*(A),*emit*(B), *pop*(B). While, there exists a simple $O(n^3)$ time algorithm for OSG, Tarjan suspected this could be improved. In Appendix D, we show that OSG can be reduced to scored parsing on BD grammars. This leads to the first truly sub-cubic algorithm for OSG.

Let us summarize the mentioned applications of our approach.

Theorem 4. *LED, RNA-folding, and OSG can be solved in $\tilde{O}(n^{2.8244})$ randomized time and $\tilde{O}(n^{2.8603})$ deterministic time.*

We remark that our techniques also lead to a truly subquadratic algorithm for bounded monotone $(\min, +)$ convolution. A subquadratic algorithm was already and very recently achieved in a breakthrough result by Chan and Lewenstein [16], however with very different techniques. For two sequences $a = (a_1, \dots, a_n)$ and $b = (b_1, \dots, b_n)$ the $(\min, +)$-convolution of a and b is the vector $c = (c_1, \dots, c_n)$ with $c_k = \min_i\{a_i + b_{k-i}\}$. Assume $n = m^2$. A standard reduction from $(\min, +)$ convolution to the $(\min, +)$ matrix product constructs the $m \times m$ matrices A^r with $A^r_{i,k} = a_{rm+i+k}$ (for $1 \le r \le m$) and B with $B_{k,j} = b_{jm-k}$. Then from the products $A^r \star B$ we can infer the $(\min, +)$-convolution of a and b in time $O(n^{3/2})$. Note that if a has bounded differences, then the matrices A^r have bounded difference along the rows, while B has bounded difference along the columns. In Section B, we discuss how to obtain truly sub-cubic algorithms for $(\min, +)$-products when only one matrix has bounded differences, and even just along one dimension. This allows to compute the m $(\min, +)$-products in time $O(m \cdot m^{2.9217}) = O(n^{1.961})$, obtaining a subquadratic algorithm for BD $(\min, +)$-convolution. As observed by Chan and Lewenstein, computing $(\min, +)$ convolution over bounded monotone sequences is equivalent to computing over two bounded difference sequences.

We envision other applications of BD $(\min, +)$-product to come in the future.

1.4 Related Work

Language Edit Distance. LED is among the most fundamental and best studied problems related to strings and grammars [4, 31, 27, 37, 38, 1, 35]. It generalizes two basic problems in computer science: parsing and string edit distance computation. In 1972, Aho and Peterson presented a dynamic programming algorithm for LED that runs in $O(|G|^2 n^3)$ time [4], which was improved to $O(|G|n^3)$ by Myers in 1985 [31]. These algorithms are based on popular CYK parsing algorithm [3] with the observation that LED can be reduced to a scored parsing problem [4]. This implied the previous best running time in $O(n^3/2^{\Theta(\sqrt{\log n})})$. In a recent paper [38], Saha showed that LED can be solved in $O(\frac{n^\omega}{\text{poly}(\epsilon)})$ time if we allow to approximate the exact edit distance by a $(1+\epsilon)$-factor. Due to known conditional lower bound results for parsing [27, 1], LED cannot be approximated within any multiplicative factor in time $o(n^\omega)$. Interestingly, if we only allow insertion as edits, then [38] also showed that a sub-cubic exact algorithm is unlikely due to a reduction to APSP on weighted graphs [45]. Whereas, here we show with insertions and deletions (and possibly substitutions) as edits LED is solvable in truly sub-cubic time. LED provides a very generic framework for modeling problems with vast applications [24, 23, 47, 30, 36, 34, 22]. A fast exact algorithm for it is likely to have tangible impact.

RNA-Folding. Computational approaches to find the secondary structure of RNA molecules are used extensively in bioinformatics applications. In 1980, Nussinov and Jackobson [32] proposed the following optimization problem, and a simple $O(n^3)$ dynamic programming solution to obtain the optimal folding. Let Σ be a set of letters and let $\Sigma' = \{\sigma' \mid \sigma \in \Sigma\}$ be the set of "matching" letters, such that for every letter $\sigma \in \Sigma$ the pair σ, σ' match. Given a sequence of n letters over $\Sigma \cup \Sigma'$, the RNA-folding problem asks for the maximum number of non-crossing pairs $\{i, j\}$ such that the ith and jth letter in the sequence match, i.e., if letters in positions i and j are paired and if letters in positions k and l are paired, and $i < k$ then either they are nested, i.e., $i < k < l < j$ or they are non-intersecting, i.e., $i < j < k < l$. The objective is to maximize the number of pairings under these constraints.

Since the seminal work of [32], multitude of sophisticated RNA-folding algorithms with complex objectives and softwares have been developed[7], but the basic dynamic programming algorithm of Nussinov and Jackobson remains at the heart of all of these. Despite much effort, only mild improvements in running time has been reported so far [46, 5, 50], and obtaining a truly sub-cubic algorithm for RNA-folding has remained open till this work.

Abboud et al. [1] showed that obtaining an algorithm for RNA-folding that runs in $O(n^{\omega-\varepsilon})$ time for any $\varepsilon > 0$ would result in a breakthrough in our algorithmic knowledge of the Clique problem. Moreover, the work of [1] implies that RNA-folding problem needs fast matrix multiplication to be solved in truly sub-cubic time, unless there are fast algorithms for Clique that do not use matrix multiplication.

Dyck LED. A closely related problem is Dyck Edit distance. In Dyck LED, a parentheses string must be well-balanced by matching open parentheses with corresponding closed parentheses in a non-crossing way. For example, [()] belongs to the Dyck language, but [) or][do not. The RNA grammar is often referred to as the two-sided Dyck where][is also a valid match. Dyck edit distance with insertion and deletion generalizes the widely-studied string edit distance problem [29, 26, 10, 11, 8, 7]. When approximation is allowed, a near-linear time $O(\text{poly}\log n)$-approximation algorithm was developed by Saha [37]. A $(1+\epsilon)$-approximation in $O(n^{\omega})$ time was also shown in [38] for any constant $\epsilon > 0$. Abboud et al. [1] related the Dyck LED problem to Clique with the same implications as for RNA-folding. Thus, up to a breakthrough in Clique algorithms, truly sub-cubic Dyck LED requires fast matrix multiplication. Prior to our work, no sub-cubic exact algorithm was known for Dyck LED.

1.5 Preliminaries and Notation

In this paper, by "randomized time $t(n)$" we mean a zero-error randomized algorithm running in time $t(n)$ in expectation, and also with high probability.

As is typical, we denote by $\omega < 2.3729$ [44, 20] the exponent of square matrix multiplication, i.e. ω is the infimum over all reals such that $n \times n$ matrix multiplication over the complex numbers can be computed in $n^{\omega+o(1)}$ time. For ease of notation and as typical in the literature, we shall omit the $o(1)$ term and write $O(n^{\omega})$ instead. We denote the running time to multiply an $a \times b$ matrix with a $b \times c$ matrix by $M(a, b, c)$ [19]. As in (1) above we have the following:

Lemma 1. *[6] Let A, B be $a \times b$ and $b \times c$ matrices with entries in $\{-M, -M+1 \dots, M\} \cup \{\infty\}$. Then $A \star B$ can be computed in time $\tilde{O}(M \cdot M(a, b, c))$. In particular, for $a = b = c = n$ this running time is $\tilde{O}(Mn^{\omega})$.*

Organization. In Section 2 we give our main technical result, a truly sub-cubic algorithm for $(\min, +)$-product with BD property. In Section A, we show how bounded difference scored parsing can be solved asymptotically in the same time as computing a single BD $(\min, +)$ product. In Section B, we show how to further reduce the running time, how to derandomize our algorithm, and some generalizations of our approach. Sections C and D are devoted to prove reductions from LED, RNA-folding, and OSG to scored parsing on BD grammars.

2 Fast Bounded-Differences $(\min, +)$ Product

In this section we present our fast $(\min, +)$ product algorithm for BD matrices. For ease of presentation, we will focus here only on the case that both input matrices A and B are BD.

[7]see `https://en.wikipedia.org/wiki/List_of_RNA_structure_prediction_software`

Furthermore, we will present a simplified randomized algorithm which is still truly sub-cubic. Refinements of the running time, derandomization, and generalizations are discussed in Section B. Let A and B be $n \times n$ matrices with W-bounded differences. We write $C = A \star B$ for the desired output and denote by $\hat{C}$ the result computed by our algorithm. Our algorithm consists of following three main phases (see also Algorithm 1).

2.1 Phase 1: Computing an approximation

Let Δ be a positive integer that we later fix as a small polynomial[8] in n. We partition $[n]$ into blocks of length Δ by setting $I(i') := \{i \in [n] \mid i' - \Delta < i \leq i'\}$ for any i' divisible by Δ. From now on by i, k, j we denote indices in the matrices A, B, and C and by i', k', j' we denote numbers divisible by Δ, i.e., indices of blocks.

The first step of our algorithm is to compute an entry-wise additive $O(\Delta W)$-approximation $\tilde{C}$ of $A \star B$. Since A and B have W-BD, it suffices to approximately evaluate $A \star B$ only for indices i', k', j' divisible by Δ. Specifically, we compute $\tilde{C}_{i',j'} = \min\{A_{i',k'} + B_{k',j'} \mid k' \text{ divisible by } \Delta\}$, and set $\tilde{C}_{i,j} := \tilde{C}_{i',j'}$ for any $i \in I(i'), j \in I(j')$, see lines 1-3 of Algorithm 1.

The next lemma shows that $\tilde{C}$ is a good approximation of C.

Lemma 2. *For any i', k', j' divisible by Δ and any $(i, k, j) \in I(i') \times I(k') \times I(j')$ we have*

$$(1)\ |A_{i,k} - A_{i',k'}| \leq 2\Delta W \qquad (2)\ |B_{k,j} - B_{k',j'}| \leq 2\Delta W;$$
$$(3)\ |C_{i,j} - C_{i',j'}| \leq 2\Delta W; \qquad (4)\ |C_{i,j} - \tilde{C}_{i,j}| \leq 4\Delta W$$

Proof. Consider the first statement. Observe that we can move from $A_{i,k}$ to $A_{i',k}$ in $i' - i \leq \Delta$ steps each time changing the absolute value by at most W, hence $|A_{i,k} - A_{i',k}| \leq \Delta W$. Similarly from $A_{i',k}$ to $A_{i',k'}$. The overall absolute change is therefore at most $2\Delta W$. The proof of the second claim is analogous.

For the third statement, let k be such that $C_{i,j} = A_{i,k} + B_{k,j}$. Then $C_{i',j'} \leq A_{i',k} + B_{k,j'} \leq A_{i,k} + B_{k,j} + 2\Delta W = C_{i,j} + 2\Delta W$. In the second inequality we used the fact that $A_{i',k} \leq A_{i,k} + \Delta W$ and $B_{k,j'} \leq B_{k,j} + \Delta W$ from the same argument as above. Symmetrically, we obtain $C_{i',j'} \leq C_{i,j} + 2\Delta W$.

For the last statement, note that $\tilde{C}_{i,j} = \tilde{C}_{i',j'}$ by construction. Let k' be divisible by Δ and such that $\tilde{C}_{i',j'} = A_{i',k'} + B_{k',j'}$. Then $C_{i,j} \leq A_{i,k'} + B_{k',j} \leq A_{i',k'} + B_{k',j'} + 2\Delta W = \tilde{C}_{i',j'} + 2\Delta W$, where again the second inequality exploits the above observation. For the other direction, let k be such that $C_{i,j} = A_{i,k} + B_{k,j}$, and consider k' with $k \in I(k')$. Then $\tilde{C}_{i',j'} \leq A_{i',k'} + B_{k',j'} \leq A_{i,k} + B_{k,j} + 4\Delta W = C_{i,j} + 4\Delta W$, where in the second inequality we exploited (1) and (2). $\square$

2.2 Phase 2: Randomized reduction to (min,+)-product with small entries

The second step of our algorithm is the most involved one. The goal of this step is to change A and B in a randomized way to obtain matrices where each entry is ∞ or has small absolute value, thus reducing the problem to Lemma 1. This step will cover most triples i, k, j, but not all: the third step of the algorithm will cover the remaining triples by exhaustive search. We remark that Phase 2 works with arbitrary matrices A and B (assuming we know an approximate answer $\tilde{C}$ as computed in Phase 1).

The following observation is the heart of our argument. For any vector $F = (F_1, \ldots, F_n)$, adding F_k to every entry $A_{i,k}$ $(\forall i)$ and subtracting F_k from every entry $B_{k,j}$ $(\forall j)$ does not change the product $A \star B$. Similarly, for n-dimension vectors X and Y, adding X_i to every entry $A_{i,k}$ and adding Y_j to every entry $B_{k,j}$ changes the entry $(A \star B)_{i,j}$ by $+X_i + Y_j$, which we can cancel after computing the product.

[8] We can assume that both n and Δ are powers of two, so in particular we can assume that Δ divides n.

Specifically, we may fix indices i^r, j^r and consider the matrices A^r with $A^r_{i,k} := A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r}$ and B^r with $B^r_{k,j} := B_{k,j} - B_{k,j^r} + \tilde{C}_{i^r,j^r} - \tilde{C}_{i^r,j}$. Then from $C^r := A^r \star B^r$ we can infer $C = A \star B$ via the equation $C_{i,j} = C^r_{i,j} + \tilde{C}_{i,j^r} - \tilde{C}_{i^r,j^r} + \tilde{C}_{i^r,j}$.

We will set an entry of A^r or B^r to ∞ if its absolute value is more than $48\Delta W$. This allows to compute $C^r = A^r \star B^r$ efficiently using Lemma 1. However, it does not correctly compute $C = A \star B$. Instead, we obtain values $\hat{C}^r_{i,j} := C^r_{i,j} + \tilde{C}_{i,j^r} - \tilde{C}_{i^r,j^r} + \tilde{C}_{i^r,j}$ that fulfill $\hat{C}^r_{i,j} \geq C_{i,j}$. Moreover, if neither $A^r_{i,k}$ nor $B^r_{k,j}$ was set to ∞ then $\hat{C}^r_{i,j} \leq A_{i,k} + B_{k,j}$; in this case the contribution of i, k, j to $C_{i,j}$ is incorporated in $\hat{C}^r_{ij}$ (and we say that i, k, j is "covered" by A^r, B^r, see Definition 3). We repeat this procedure with independently and uniformly random $i^r, j^r \in [n]$ for $r = 1, \ldots, \rho$ many rounds, where $1 \leq \rho \leq n$ is a small polynomial in n to be fixed later. Then $\hat{C}$ is set to the entry-wise minimum over all $\hat{C}^r$. This finishes the description of Phase 2, see lines 4–14 of Algorithm 1.

In the analysis of this step of the algorithm, we want to show that w.h.p. most of the "relevant" triples i, k, j get covered: in particular, all triples with $A_{i,k} + B_{k,j} = C_{i,j}$ are relevant, as these triples define the output. However, since this definition would depend on the output $C_{i,j}$, we can only (approximately) check a weak version of relevance, see Definition 3. Similarly, we need a weak version of being covered.

Definition 3. *We call a triple* (i, k, j)

- strongly relevant *if* $A_{i,k} + B_{k,j} = C_{i,j}$,
- weakly relevant *if* $|A_{i,k} + B_{k,j} - C_{i,j}| \leq 16\Delta W$,
- strongly r-uncovered *if for all* $1 \leq r' \leq r$ *we have* $|A^{r'}_{i,k}| > 48\Delta W$ *or* $|B^{r'}_{k,j}| > 48\Delta W$, *and*
- weakly r-uncovered *if for all* $1 \leq r' \leq r$ *we have* $|A^{r'}_{i,k}| > 40\Delta W$ *or* $|B^{r'}_{k,j}| > 40\Delta W$.

A triple is strongly (resp., weakly) uncovered if it is strongly (resp., weakly) ρ-uncovered.

Next lemma gives a sufficient condition for not being weakly r-uncovered.

Lemma 3. *For any i, k, j and i^r, j^r, if all triples (i, k, j^r), (i^r, k, j^r), (i^r, k, j) are weakly relevant then (i, k, j) is not weakly r-uncovered.*

Proof. From the assumption and $\tilde{C}$ being an additive $4\Delta W$-approximation of C, we obtain

$$|A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r}| \leq |A_{i,k} + B_{k,j^r} - C_{i,j^r}| + |\tilde{C}_{i,j^r} - C_{i,j^r}| \leq 16\Delta W + 4\Delta W = 20\Delta W.$$

Similarly, we also have $|A_{i^r,k} + B_{k,j^r} - \tilde{C}_{i^r,j^r}| \leq 20\Delta W$ and $|A_{i^r,k} + B_{k,j} - \tilde{C}_{i^r,j}| \leq 20\Delta W$.

Recall that in the algorithm we set $A^r_{i,k} := A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r}$ and $B^r_{k,j} := B_{k,j} - B_{k,j^r} + \tilde{C}_{i^r,j^r} - \tilde{C}_{i^r,j}$ (and then reset them to ∞ if their absolute value is more than $48\Delta W$). From the above inequalities, we have $|A^r_{i,k}| \leq 20\Delta W$. Moreover, we can write $B^r_{k,j}$ as $(A_{i^r,k} + B_{k,j} - \tilde{C}_{i^r,j}) - (A_{i^r,k} + B_{k,j^r} - \tilde{C}_{i^r,j^r})$, where both terms in brackets have absolute value bounded by $20\Delta W$, and thus $|B^r_{k,j}| \leq 40\Delta W$. It follows that the triple i, k, j gets weakly covered within round r. □

We will crucially exploit the following simple extremal graph-theoretic result.

Lemma 4. *Let $G = (U \cup V, E)$ be a bipartite graph with $|U| = |V| = n$ nodes per partition and $|E| = m$ edges. Let C_4 be the number of 4-cycles of G. If $m \geq 2n^{3/2}$, then $C_4 \geq m^4/(32n^4)$.*

Proof. For any pair of nodes $v, v' \in V$, let $N(v, v')$ be the number of common neighbors $\{u \in U \mid \{u, v\}, \{u, v'\} \in E\}$, and let $N = \sum_{\{v,v'\} \in \binom{V}{2}} N(v, v')$. By $d(w)$ we denote the degree of node w in G. By convexity of $\binom{x}{2} = \frac{x(x-1)}{2}$ and Jensen's inequality, we have

$$N = \sum_{\{v,v'\} \in \binom{V}{2}} N(v, v') = \sum_{u \in U} \binom{d(u)}{2} \geq n \cdot \binom{\sum_{u \in U} d(u)/n}{2} = n\binom{m/n}{2} = \frac{m^2}{2n} - \frac{m}{2} \geq \frac{m^2}{2n} - n^2.$$

Since $m \geq 2n^{3/2}$ by assumption, we derive $\frac{m^2}{2n} \geq 2n^2$ and thus we obtain $N \geq n^2 > 2\binom{n}{2}$ as well as $N \geq m^2/(4n)$.

By the same convexity argument as above, we also have

$$C_4 = \sum_{\{v,v'\} \in \binom{V}{2}} \binom{N(v,v')}{2} \geq \binom{n}{2} \cdot \binom{N/\binom{n}{2}}{2} = \left(N - \binom{n}{2}\right) \frac{N}{n(n-1)} \geq \frac{N^2}{2n^2},$$

where in the last inequality above we used the fact that $N \geq 2\binom{n}{2}$. Altogether, this yields

$$C_4 \geq \frac{N^2}{2n^2} \geq \frac{m^4/(16n^2)}{2n^2} = \frac{m^4}{32n^4}.$$

□

We are now ready to lower bound the progress made by the algorithm at each round.

Lemma 5. *W.h.p for any $\rho \geq 1$ the number of weakly relevant, weakly uncovered triples is $\tilde{O}(n^{2.5} + n^3/\rho^{1/3})$.*

Proof. Fix $k \in [n]$. We construct a bipartite graph G_k on $n + n$ vertices (we denote vertices in the left vertex set by i or i^r and vertices in the right vertex set by j or j^r). We add edge $\{i, j\}$ to G_k if the triple (i, k, j) is weakly relevant.

In each of the ρ rounds of our algorithm we select i^r and j^r uniformly at random. Round r covers some triples (i, k, j). For any such weakly r-covered triple (i, k, j), if (i, j) is in G_k, we remove it from G_k. Thus, after round r, G_k contains (i, j) if and only if (i, k, j) is weakly relevant and r-weakly uncovered.

Let $z = c(n^2/\rho) \ln n$ for any constant $c > 4$. Consider an edge (i, j) in G_k that is contained in at least z 4-cycles in G_k before any of the rounds of Phase 2 are performed. Now consider each round r in turn and let $i \to \ell \to p \to j \to i$ be a 4-cycle containing (i, j) whose edges are still in G_k. If $i^r = p$ and $j^r = \ell$ are selected, then since by the definition of G_k $(i, k, \ell), (p, k, \ell)$ and (p, k, j) are weakly uncovered, by Lemma 3, (i, k, j) will be r-covered and thus (i, j) will be removed from G_k.

Thus, if in any of the ρ rounds r, i^r, j^r are selected to be among the at least z choices of vertices that complete (i, j) to a 4-cycle in G_k, (i, j) will not be in G_k at the end of all ρ iterations. The probability that for a particular edge (i, j) with at least z 4-cycles in a particular G_k, i^r, j^r are *never* picked to form a 4-cycle with (i, j) is

$$\leq \left(1 - \frac{z}{n^2}\right)^{\rho} = \left(1 - \frac{z}{n^2}\right)^{c(n^2/z) \ln n} \leq \frac{1}{n^c}.$$

By a union bound we get that the probability for some i, j, k this happens is $\leq 1/n^{c-3}$ which is $1/\text{poly}(n)$ as we picked $c > 4$. We thus get that with high probability, at the end of all ρ iterations, every edge in every G_k is contained in less than z 4-cycles.

Let m_k denote the number of edges of G_k. Now we will bound $\sum_k m_k$, as this is exactly the number of weakly relevant, weakly uncovered triples. First, let's note that $\sum_{\{k \mid m_k < 2n^{3/2}\}} m_k < 2n^{2.5}$, and so it suffices to compute the sum for those k for which $m_k \geq 2n^{3/2}$. Fix one such G_k. Since every edge in G_k is contained in less than z 4-cycles w.h.p., the number of 4-cycles C_k of G_k is less than $m_k z$. On the other hand, by Lemma 4, $C_k \geq (m_k/n)^4/32$. Thus,

$$(m_k/n)^4 < 32 m_k z \implies m_k^3 < 32n^4 z \implies m_k < \left(32c(n^6/\rho) \ln n\right)^{1/3} \implies m_k \leq \tilde{O}(n^2/\rho^{1/3}).$$

The total number of weakly uncovered, weakly relevant triples at the end of the ρ iterations is thus w.h.p. $\tilde{O}(n^{2.5} + n^3/\rho^{1/3})$. □

2.3 Phase 3: Exhaustive search over all relevant uncovered triples of indices

In the third and last phase we make sure to fix all strongly relevant, strongly uncovered triples by exhaustive search, as these are the triples defining the output matrix whose contribution is not yet incorporated in $\hat{C}$. We are allowed to scan all weakly relevant, weakly uncovered triples, as we know that their number is small by Lemma 11. This is the only phase that requires that A and B are BD.

We use the following definitions of being *approximately* relevant or uncovered, since they are identical for all triples (i,k,j) in a block i',k',j' and thus can be checked efficiently.

Definition 4. *We call a triple* $(i,k,j) \in I(i') \times I(k') \times I(j')$

- approximately relevant *if* $|A_{i',k'} + B_{k',j'} - \tilde{C}_{i',j'}| \leq 8\Delta W$*, and*
- approximately uncovered *if for all* $1 \leq r \leq \rho$ *we have* $|A^r_{i',k'}| > 44\Delta W$ *or* $|B^r_{k',j'}| > 44\Delta W$.

The notions of being strongly, weakly, and approximately relevant/uncovered are related as follows.

Lemma 6. *Any strongly relevant triple is also approximately relevant. Any approximately relevant triple is also weakly relevant. Then same statements hold with "relevant" replaced by "r-uncovered".*

Proof. Let $(i,k,j) \in I(i') \times I(k') \times I(j')$. Using Lemma 2, we can bound the absolute difference between $A_{i,k}+B_{k,j}-C_{i,j}$ and $A_{i',k'}+B_{k',j'}-\tilde{C}_{i',j'}$ by the three contributions $|A_{i,k}-A_{i',k'}| \leq 2\Delta W$, $|B_{k,j} - B_{k',j'}| \leq 2\Delta W$, and $|C_{i,j} - \tilde{C}_{i',j'}| = |C_{i,j} - \tilde{C}_{i,j}| \leq 4\Delta W$. Thus, if $A_{i,k} + B_{k,j} = C_{i,j}$ (i.e., (i,k,j) is strongly relevant), then $|A_{i',k'} + B_{k',j'} - \tilde{C}_{i',j'}| \leq 8\Delta W$ (i.e., (i,k,j) is approximately relevant). On the other hand, if (i,k,j) is approximately relevant, then $|A_{i,k} + B_{k,j} - C_{i,j}| \leq 16\Delta W$ (i.e., (i,k,j) is weakly relevant).

For the notion of being r'-uncovered, for any r we bound the absolute differences $|A^r_{i,k} - A^r_{i',k'}|$ and $|B^r_{k,j} - B^r_{k',j'}|$. Recall that we set $A^r_{i,j} := A_{i,j} + B_{k,j^r} - \tilde{C}_{i,j^r}$. Again using Lemma 2, we bound both $|A_{i,j} - A_{i',j'}|$ and $|B_{k,j^r} - B_{k',j^r}|$ by $2\Delta W$. Since we have $\tilde{C}_{i,j^r} = \tilde{C}_{i',j^r}$ by definition, in total we obtain $|A^r_{i,k} - A^r_{i',k'}| \leq 4\Delta W$. Similarly, recall that we set $B^r_{k,j} := B_{k,j} - B_{k,j^r} + \tilde{C}_{i^r,j^r} - \tilde{C}_{i^r,j}$. The first two terms both contribute at most $2\Delta W$, while the latter two terms are equal for $B^r_{k,j}$ and $B^r_{k',j'}$. Thus, $|B^r_{k,j} - B^r_{k',j'}| \leq 4\Delta W$. The statements on "$r'$-uncovered" follow immediately from these inequalities. □

In our algorithm, we enumerate every triple (i',k',j') whose indexes are divisible by Δ, and check whether that triple is approximately relevant. Then we check whether it is approximately uncovered. If so, we perform an exhaustive search over the block i',k',j': We iterate over all $(i,k,j) \in I(i') \times I(k') \times I(j')$ and update $\hat{C}_{i,j} := \min\{\hat{C}_{i,j}, A_{i,k} + B_{k,j}\}$, see lines 15-19 of Algorithm 1.

Note that i',k',j' is approximately relevant (resp., approximately uncovered) if and only if all $(i,k,j) \in I(i') \times I(k') \times I(j')$ are approximately relevant (resp., approximately uncovered). Hence, we indeed enumerate all approximately relevant, approximately uncovered triples, and by Lemma 6 this is a superset of all strongly relevant, strongly uncovered triples. Thus, every strongly relevant triple (i,k,j) contributes to $\hat{C}_{i,j}$ in Phase 2 or Phase 3. This proves correctness of the output matrix $\hat{C}$.

2.4 Running Time

The running time of Phase 1 is $O((n/\Delta)^3 + n^2)$ using brute-force. The running time of Phase 2 is $\tilde{O}(\rho \Delta W n^\omega)$, since there are ρ invocations of Lemma 1 on matrices whose finite entries have absolute value $O(\Delta W)$. It remains to consider Phase 3. Enumerating all blocks i',k',j'

and checking whether they are approximately relevant and approximately uncovered takes time $O((n/\Delta)^3\rho)$. The approximately relevant and approximately uncovered triples form a subset of the weakly relevant and weakly uncovered triples by Lemma 6. The number of the latter triples is upper bounded by $\tilde{O}(n^{2.5}+n^3/\rho^{1/3})$ w.h.p. by Lemma 11. In total Phase 3 takes time $\tilde{O}((n/\Delta)^3\rho+n^3/\rho^{1/3}+n^{2.5})$ w.h.p. In total, the running time of Algorithm 1 is w.h.p.

$$\tilde{O}((n/\Delta)^3+n^2+\rho\Delta W n^\omega+(n/\Delta)^3\rho+n^3/\rho^{1/3}+n^{2.5}).$$

A quick check shows that for appropriately chosen ρ and Δ (say $\rho:=\Delta:=n^{0.1}$) and for sufficiently small W this running time is truly sub-cubic. We optimize by setting $\rho:=(n^{3-\omega}/W)^{9/16}$ and $\Delta:=(n^{3-\omega}/W)^{1/4}$, obtaining time $\tilde{O}(W^{3/16}n^{(39+3\omega)/16})$, which is truly sub-cubic for $W\le O(n^{3-\omega-\varepsilon})$. For $W=O(1)$ using $\omega\le 2.3729$ [44, 20] this running time evaluates to $O(n^{2.8825})$. Note that even with bad random choices the running time of our algorithm is bounded by $O(n^3)$. In particular, this implies that our w.h.p. time bounds also hold in expectation.

References

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 98–117, 2015.

[2] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697, 2015.

[3] Alfred V. Aho and John E. Hopcroft. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1974.

[4] Alfred V. Aho and Thomas G. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM J. Comput.*, 1(4), 1972.

[5] Tatsuya Akutsu. Approximation and exact algorithms for RNA secondary structure prediction and recognition of stochastic context-free languages. *Journal of Combinatorial Optimization*, 3(2-3):321–336, 1999.

[6] Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, April 1997.

[7] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 377–386, 2010.

[8] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 199–204, 2009.

[9] Arturs Backurs and Krzysztof Onak. Fast algorithms for parsing sequences of parentheses with few errors. In *PODS*, page to appear, 2016.

[10] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *FOCS*, pages 550–559, 2004.

Algorithm 1 $(\min,+)$-product $A \star B$ for $n \times n$ matrices A, B with W-bounded differences. Here Δ and ρ are carefully chosen polynomial values. Also $I(q) = \{q - \Delta + 1, \dots, q\}$.

```
    ▷ Phase 1: compute entry-wise additive $4\Delta W$-approximation $\tilde{C}$ of $A \star B$
 1: for any $i', j'$ divisible by $\Delta$ do
 2:     $\tilde{C}_{i',j'} := \min\{A_{i',k'} + B_{k',j'} \mid k'$ divisible by $\Delta\}$
 3:     for any $i \in I(i')$, $j \in I(j')$ do
 4:         $\tilde{C}_{i,j} := \tilde{C}_{i',j'}$
 5:     end for
 6: end for
    ▷ Phase 2: randomized reduction to $(\min,+)$-product with small entries
 7: initialize all entries of $\hat{C}$ with $\infty$
 8: for $1 \le r \le \rho$ do
 9:     pick $i^r$ and $j^r$ independently and uniformly at random from $[n]$
10:     for all $i, k$ do
11:         set $A^r_{i,k} := A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r}$
12:         if $A^r_{i,k} \notin [-48\Delta W, 48\Delta W]$ then set $A^r_{i,k} := \infty$
13:     end for
14:     for all $k, j$ do
15:         set $B^r_{k,j} := B_{k,j} - B_{k,j^r} + \tilde{C}_{i^r,j^r} - \tilde{C}_{i^r,j}$
16:         if $B^r_{k,j} \notin [-48\Delta W, 48\Delta W]$ then set $B^r_{k,j} := \infty$
17:     end for
18:     compute $C^r := A^r \star B^r$ using Lemma 1
19:     for all $i, j$ do $\hat{C}_{i,j} := \min\{\hat{C}_{i,j}, C^r_{i,j} + \tilde{C}_{i,j^r} - \tilde{C}_{i^r,j^r} + \tilde{C}_{i^r,j}\}$
20:     end for
21: end for
    ▷ Phase 3: exhaustive search over all relevant uncovered triples of indices
22: for all $i', k', j'$ divisible by $\Delta$ do
23:     if $|A_{i',k'} + B_{k',j'} - \tilde{C}_{i',j'}| \le 8\Delta W$ then
24:         if for all $r$ we have $|A^r_{i',k'}| > 44\Delta W$ or $|B^r_{k',j'}| > 44\Delta W$ then
25:             for all $i \in I(i'), k \in I(k'), j \in I(j')$ do
26:                 $\hat{C}_{i,j} := \min\{\hat{C}_{i,j}, A_{i,k} + B_{k,j}\}$
27:             end for
28:         end if
29:     end if
30: end for
31: return $\hat{C}$
```

[11] Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *SODA*, pages 792–801, 2006.

[12] Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 101–110, 2009.

[13] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[14] Amit Chakrabarti, Graham Cormode, Ranganath Kondapally, and Andrew McGregor. Information cost tradeoffs for augmented index and streaming language recognition. In *FOCS*, 2010.

[15] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010.

[16] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 31–40, 2015.

[17] Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *12th Annual Symposium on Switching and Automata Theory, East Lansing, Michigan, USA, October 13-15, 1971*, pages 129–131, 1971.

[18] Michael L Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5(1), 1976.

[19] François Le Gall. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 514–523, 2012.

[20] François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

[21] Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 748–757, 2012.

[22] R.R Gutell, J.J. Cannone, Z Shang, Y Du, and M.J Serra. A story: unpaired adenosine bases in ribosomal RNAs. In *Journal of Mol Biology*, 2010.

[23] Mark Johnson. PCFGs, Topic Models, Adaptor Grammars and Learning Topical Collocations and the Structure of Proper Names. In *ACL*, pages 1148–1157, 2010.

[24] Flip Korn, Barna Saha, Divesh Srivastava, and Shanshan Ying. On repairing structural problems in semi-structured data. In *VLDB*, 2013.

[25] Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming algorithms for recognizing nearly well-parenthesized expressions. In *MFCS*, 2011.

[26] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2), April 1998.

[27] Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1), January 2002.

[28] Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. In *STOC*, 2010.

[29] William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18 – 31, 1980.

[30] Darnell Moore and Irfan Essa. Recognizing multitasked activities from video using stochastic context-free grammar. In *NCAI*, pages 770–776, 2002.

[31] Gene Myers. Approximately matching context-free languages. *Information Processing Letters*, 54, 1995.

[32] Ruth Nussinov and Ann B.Jacobson. Fast algorithm for predicting the secondary structure of single-stranded rna. *Proceedings of the National Academy of Sciences of the United States of America*, 77(11):6309–6313, 1980.

[33] Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Testing membership in parenthesis languages. *Random Struct. Algorithms*, 22(1), January 2003.

[34] Geoffrey K Pullum and Gerald Gazdar. Natural languages and context-free languages. *Linguistics and Philosophy*, 4(4), 1982.

[35] Sanguthevar Rajasekaran and Marius Nicolae. An error correcting parser for context free grammars that takes less than cubic time. *Manuscript*, 2014.

[36] Andrea Rosani, Nicola Conci, and Francesco G. De Natale. Human behavior recognition using a context-free grammar. *Journal of Electronic Imaging*, 23(3), 2014.

[37] Barna Saha. The dyck language edit distance problem in near-linear time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 611–620, 2014.

[38] Barna Saha. Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 118–135, 2015.

[39] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.

[40] Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 605–615, 1999.

[41] Tadao Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20(3):309–318, 1998.

[42] Robert E Tarjan. Problems in data structures and algorithms. In *Graph Theory, Combinatorics and Algorithms*, pages 17–39. Springer, 2005.

[43] Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975.

[44] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898, 2012.

[45] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654, 2010.

[46] Balaji Venkatachalam, Dan Gusfield, and Yelena Frid. Faster algorithms for RNA-folding using the four-russians method. In *WABI*, 2013.

[47] Ye-Yi Wang, Milind Mahajan, and Xuedong Huang. A unified context-free grammar and n-gram model for spoken language processing. In *ICASP*, pages 1639–1642, 2000.

[48] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *STOC*, pages 664–673, 2014.

[49] Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 950–957, 2009.

[50] Shay Zakov, Dekel Tsur, and Michal Ziv-Ukelson. Reducing the worst case running times of a family of RNA and CFG problems, using Valiant's approach. In *WABI*, 2010.

[51] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

A Fast Scored Parsing

In this section we prove Theorem 3 that reduces the scored parsing problem over BD grammars to $(\min, +)$ product over BD matrices. For a square matrix M, we let $n(M)$ denote its number of rows and columns.

Context Free Grammars and Scored Parsing. Let $G = (N, T, P, S)$ be a Context Free Grammar (CFG), where N and T are the (disjoint) sets of non-terminals and terminals, respectively, P is the set of production rules, and S is the starting non-terminal. We recall that a production rule is of the form $p = (X \to \alpha)$, with $X \in N$ and $\alpha \in (N \cup T)^*$, and applying p to (some instance of) $X \in N$ in a string $\sigma \in (N \cup T)^*$ generates the string σ' where X is replaced by α.[9] The language $L(S)$ generated by G is the set of strings $\sigma \in T^*$ that can be derived from S by applying a finite sequence of production rules. We also let $L(X)$ denote the set of strings that can be generated from the non-terminal $X \in N$.

For the sake of simplicity and w.l.o.g., we assume that G is given in Chomsky normal form (CNF). In particular, productions rules are of the form $(Z \to XY)$, $(Z \to x)$, and $(S \to \epsilon)$, where $X, Y \in N \setminus \{S\}$, $Z \in N$, $x \in T$, and ϵ denotes the empty string.

In the *scored parsing* problem, each production rule $p \in P$ is associated with an integer score $s(p)$. Intuitively, applying a given production rule has a cost. Given a string σ of length n, we wish to compute a sequence of production rules of minimum total score $s(S, \sigma)$ that derives σ from S (we assume $s(S, \sigma) = +\infty$ if $\sigma \notin L(S)$). Let us define $s(X, \sigma)$ similarly for a non-terminal X.

We will exploit a generalization of Valiant's parser [43]: in Section A.1 we start by describing it. Then in Section A.2 we describe how to modify Valiant's parser to solve the scored parsing problem: here we replace Boolean matrix multiplications with $(\min, +)$ products. Finally, in Section A.3 we show that all the involved $(\min, +)$ products can be reduced to $(\min, +)$ product on W-bounded differences matrices.

[9] Given a set of symbols U, U^* as usual denotes any, possibly empty, string of elements from U.

A.1 Valiant's Parser

Given a CFG $G = (N, T, P, S)$ and a string $\sigma = \sigma_1\sigma_2....\sigma_n \in T^*$, the *parsing* problem is to determine whether $\sigma \in L(S)$. In a breakthrough paper [43], Valiant described a reduction from parsing to Boolean matrix multiplication (for a more detailed description, please see [43]). Let us define a (product) operator "." as follows. For $N_1, N_2 \subseteq N$,

$$N_1.N_2 = \{Z \in N : \exists X \in N_1, \exists Y \in N_2 : (Z \to XY) \in P\}.$$

Note the above operator is *not associative* in general, namely $(N_1.N_2).N_3$ might be different from $N_1.(N_2.N_3)$.

Given a $a \times b$ matrix A and a $b \times c$ matrix B, whose entries are subsets of N, we can naturally define a matrix product $C = A.B$, where $C_{i,j} = \cup_{k=1}^{b} A_{i,k}.B_{k,j}$. Observe that the "." operator can be reduced to the computation of a constant number of standard Boolean matrix multiplications. In more detail, for a matrix M and non-terminal X, we let $M(X)$ be the 0-1 matrix with the same dimension of X and having $M(X)_{i,j} = 1$ iff $X \in M_{i,j}$. Consider the product $C = A.B$. Matrix C is initialized with empty entries. Then we consider each production rule $(Z \to XY)$ separately, and we compute $C'(Z) = A(X) \cdot B(Y)$ where $\cdot$ is standard Boolean matrix multiplication. Then, for all i, j, we add Z to the set $C_{i,j}$ if $C'(Z)_{i,j} = 1$.

The transitive closure A^+ of an $m \times m$ matrix A of the above kind is defined as

$$A^+ = \cup_{i=1}^{m} A^{(i)},$$

where

$$A^{(1)} = A \quad \text{and} \quad A^{(i)} = \cup_{j=1}^{i-1} A^{(j)}.A^{(i-j)}.$$

Here unions are taken component-wise.

Given the above definitions we can formulate the parsing problem as follows. We construct an $(n+1) \times (n+1)$ matrix A where $A_{i,i+1} = \{X \in N : (X \to \sigma_i) \in P\}$ and $A_{i,j} = \emptyset$ for $j \neq i+1$. Then by the definition of "." it turns out that $X \in A^+_{i,j}$ iff $\sigma_i \dots \sigma_{j-1} \in L(X)$. Hence one can solve the parsing problem by computing A^+ and checking whether $S \in A^+_{1,n+1}$.

Suppose that, for two given $m \times m$ matrices, the "." operation can be performed in $O(m^\alpha)$ time for some $2 \leq \alpha \leq 3$, and assume w.l.o.g. that the $\cup$ operation can be performed in $O(m^2)$ time[10]. In this case we cannot use the usual squaring technique to compute A^+ in $O(n^\alpha)$ time due to the fact that "." is not associative. Valiant describes a more sophisticated approach to achieve the same running time. By the above discussion, it follows that the parsing problem can be solved in time $O(n^\omega)$, where $\omega < 2.373$ is fast Boolean matrix multiplication exponent [44, 20].

The fast procedure to compute the transitive closure of a given matrix is described in Algorithm 2. For the sake of simplicity assume that $n+1$ is a power of 4: this way all the indexes in the recursive calls are integer.[11] By Q we denote a sufficiently large constant so that $(Q/2)^\alpha$ is smaller than Q^α by a sufficiently large constant multiplicative factor (recall that $O(m^\alpha)$ is the time needed to perform one "." product on two $m \times m$ matrices). For two sets of indices I and J, by B_I^J we denote the submatrix of B given by entries $B_{i,j}$, with $i \in I$ and $j \in J$.

The algorithm involves 4 recursive procedures: Parse, Parse$_2$, Parse$_3$, and Parse$_4$. Each one of them receives in input a $n(B) \times n(B)$ matrix B, and the result of the computation is stored in B (B is passed by reference).

The running time bound follows by standard arguments, observing that each recursive call involves the call of a constant number of procedures on submatrices of at most half the size of the input matrix.

[10]Here we exploit the fact that G has constant size.

[11]This can be enforced by expanding the matrix with dummy entries.

Algorithm 2 Valiant's parser. In all the subroutines the input is a $n(B) \times n(B)$ matrix B, which is passed by reference. By B_I^J we denote the submatrix of B having entries $B_{i,j}$, with $i \in I$ and $j \in J$. Q is a sufficiently large constant.

```
Parse(B)
1: if $n(B) \leq Q = O(1)$ then
2:     compute $B^+$ with the trivial algorithm and set $B \leftarrow B^+$
3: else
4:     Parse($B_{[1,n(B)/2]}^{[1,n(B)/2]}$)
5:     Parse($B_{[n(B)/2+1,n(B)]}^{[n(B)/2+1,n(B)]}$)
6:     Parse$_2$($B$)
7: end if
Parse$_2$($B$)
1: if $n(B) \leq Q = O(1)$ then
2:     compute $B^+$ with the trivial algorithm and set $B \leftarrow B^+$
3: else
4:     Parse$_2$($B_{[n(B)/4+1,3n(B)/4]}^{[n(B)/4+1,3n(B)/4]}$)
5:     Parse$_3$($B_{[1,3n(B)/4]}^{[1,3n(B)/4]}$)
6:     Parse$_3$($B_{[n(B)/4+1,n(B)]}^{[n(B)/4+1,n(B)]}$)
7:     Parse$_4$($B$)
8: end if
Parse$_3$($B$)
1: $B \leftarrow B \cup B.B$
2: $C \leftarrow$ matrix obtained from $B$ by deleting row/column indices in $[n(B)/3+1, 2n(B)/3]$
3: Parse$_2$($C$)
4: $B \leftarrow$ matrix obtained from $C$ by reintroducing the rows and columns deleted in Step 2
Parse$_4$($B$)
1: $B \leftarrow B \cup B.B$
2: $C \leftarrow$ matrix obtained from $B$ by deleting row/column indices in $[n(B)/4+1, 3n(B)/4]$
3: Parse$_2$($C$)
4: $B \leftarrow$ matrix obtained from $C$ by reintroducing the rows and columns deleted in Step 2
```

A.2 Scored Parser and $(\min,+)$ Products

We can adapt Valiant's approach to scored parsing as follows. This has already been done in [38], but we give details here for the sake of completeness. Let us consider pairs of the form $X^s = (X, s_X)$, where $X \in N$ and s_X is a non-negative integer (that can be interpreted as a score). We denote by N^s the set of such pairs. Let $N_1, N_2 \subseteq N^s$, where each non terminal X appears in at most one pair in N_1 and N_2. We redefine the operator "." as follows: $N_1.N_2$ is the set of pairs $(Z, s_Z) \in N^s$ such that there exist $(X, s_X) \in N_1$ and $(Y, s_Y) \in N_2$ such that $(Z \to XY) \in P$ and $s_Z = s_X + s_Y + s(Z \to XY)$, and furthermore s_Z is the minimum possible value of the mentioned type for each considered non-terminal Z. We also redefine the "$\cup$" operator on N^s as follows: we first compute the classical set union operation and then, in case of multiples entries of type $(X, s_1), \ldots, (X, s_k)$ (all sharing the same non-terminal X), we replace all such entries with one single entry $(X, \min\{s_1, \ldots, s_k\})$.

Given the above operations "." and "$\cup$", we can define the product of two matrices whose entries are subsets of N^s and the transitive closure of one such square matrix in the same way as before. We can then solve the scored parsing problem as follows. For a string σ on length n and a scored grammar (G, s), we define a $(n+1) \times (n+1)$ matrix A whose entries are subsets of N^s, where

$$A_{i,i+1} = \{(X, s_X) \in N^s : (X \to \sigma_i) \in P, s(X \to \sigma_i) = s_X\}$$

for $i = 1, \ldots, n$ and $A_{i,j} = \emptyset$ for $j \neq i+1$. It then follows that the solution to the scored parsing problem is s if $(S, s) \in A^+_{1,n+1}$ ($+\infty$ if there is no entry of type (S, s) in $A^+_{1,n+1}$).

Crucially for our goals, the "." operator can be implemented with a reduction to a constant number of $(\min,+)$ products $\star$, with a natural adaptation of the previously described reduction to Boolean matrix multiplication. For a matrix M with entries in N^s and for $X \in N$, let $M(X)$ be the matrix with the same dimension of M and having $M(X)_{i,j} = s_X$ if some pair $(X, s_X) \in M_{i,j}$ and $M(X)_{i,j} = +\infty$ otherwise. Consider the product $C = A.B$. Matrix C is initialized with empty entries. Then we consider each production rule $(Z \to XY)$ sequentially and compute the $(\min,+)$ product $C'(Z) = A(X) \star B(Y)$. Then we construct a matrix $C''(Z)$ with $C''(Z)_{i,j} = \{(Z, s(Z \to XY) + C'(Z)_{i,j}\}$ and set $M \leftarrow M \cup C''(Z)$ (recall that $\cup$, in case of multiple entries with the same non-terminal, keeps only the entry with minimum score).

Suppose that we can perform a $(\min,+)$ product of the mentioned type on $m \times m$ matrices in time $O(m^\alpha)$ for some $2 \leq \alpha \leq 3$. Then exactly the same analysis as in [43] implies that the scored parsing problem (hence LED) can be solved in time $O(n^\alpha)$.

A.3 Reduction to Bounded Differences $(\min,+)$ Product

In this section we show that, given a $O(n^\alpha)$ time algorithm for W-BD $(\min,+)$ product, then we can perform the products in Step 1 of Parse$_3(B)$ and Parse$_4(B)$ in $O(n(B)^\alpha)$ time.

We will have to perform $(\min,+)$ products between matrices that not only contain integers, but also ∞. For this reason it is convenient to consider the following relaxed notion of W-BD.

Definition 5. *An $n \times n$ matrix M with entries in $\mathbb{Z} \cup \{\infty\}$ has* relaxed W-BD *if (i) M has W-BD, or (ii) M contains only ∞, or (iii) $M_{i,j} = \infty$ for $i \geq j$, and for any two indexes i, j, $i < j < n$, one has*

$$|M_{i,j} - M_{i,j+1}| \leq W \quad \text{and} \quad |M_{i,j} - M_{i+1,j}| \leq W.$$

An $n \times n$ matrix M with entries in N^s has relaxed W-BD if $M(X)$ has this property for all non-terminals X.

Observe that A^+ satisfies condition (iii) in the above definition since by assumption the input grammar is W-BD.

Next lemma shows a reduction from $(\min,+)$ product of relaxed W-BD matrices to $(\min,+)$ product of (standard) W-BD matrices.

Lemma 7. *Assume we are given a $O(n^\alpha)$ time algorithm to compute the* $(\min,+)$ *product of two $n \times n$ W-BD matrices, for some W and $2 \leq \alpha \leq 3$. Then it is possible to compute the* $(\min,+)$ *product of two $n \times n$ relaxed W-BD matrices in time $\tilde{O}(n^\alpha)$ ($O(n^\alpha)$ for $\alpha > 2$).*

Proof. Consider the product $C = A \star B$, where A and B are $n \times n$ matrices with relaxed W-BD. The claim is trivially true if at least one of A and B satisfies case (ii) in the definition (C only contains ∞), or if both A and B satisfy case (i). Hence assume that at least one of A and B satisfies case (iii), say A. W.l.o.g. we can assume that also B satisfies (iii): indeed, it is sufficient to set to ∞ the entries $B_{i,j}$ with $i \geq j$; this doesn't change the value of the product.

We initialize a matrix $\hat{C}$ with ∞, and then update it to achieve $\hat{C} = C$. This will be done by performing $(\min,+)$ products between submatrices taken from the upper triangular part of A and B (main diagonal excluded), which have therefore W-BD in the standard sense.

Recall that $C_{i,j} = \min_{k=1,\dots,n}\{A_{i,k} + B_{k,j}\}$. Trivially $C_{i,j} = \infty$ for $i \leq j$. Furthermore $A_{i,k} = B_{k,j} = \infty$ for $i \geq k$ and $k \geq j$. Therefore, in order to compute any finite value $C_{i,j}$ it is sufficient to consider the *relevant* triples of indexes (i,k,j) with $i < k < j$.

We partition the computation into levels $\ell = 1, \dots, \log_2 n$. At level ℓ we consider a partition of $[n]$ into intervals $I(\ell, a) = (an/2^\ell, (a+1)n/2^\ell]$, $a = 0, \dots, 2^\ell - 1$. Consider first the following algorithm: for each level ℓ, and each $a, b, c \in \{0, \dots, 2^\ell - 1\}$, $a < b < c$, we perform all the $(\min,+)$ products $C^\ell_{a,b,c} = A^{I(\ell,b)}_{I(\ell,a)} \star B^{I(\ell,c)}_{I(\ell,b)}$. Then we update the entries in $\hat{C}^{I(\ell,c)}_{I(\ell,a)}$ by taking the entry-wise minimum with $C^\ell_{a,b,c}$. Observe that all the considered products are among submatrices of A and B with W-BD. Furthermore all the relevant triples (i,k,j), $i < k < j$, are considered at least once. Thus the algorithm computes the correct solution.

The above algorithm can be sped up by considering, for a given level $\ell > 1$, only triples (a,b,c) with $b = a+1$ or $b+1 = c$. Indeed, whenever $a+1 < b < c-1$, one has $I(\ell,a) \times I(\ell,b) \times I(\ell,c) \subseteq I(\ell-1,a') \times I(\ell-1,b') \times I(\ell-1,c')$ for proper indexes $a', b', c' \in \{0, \dots, 2^{\ell-1} - 1\}$, $a' < b' < c'$. Thus any relevant triple $(i,k,j) \in I(\ell,a) \times I(\ell,b) \times I(\ell,c)$ is anyway considered at level $\ell - 1$.

It remains to bound the running time. The number of triples (a,b,c) considered at level ℓ is at most $2^{2\ell}$, and each corresponding $(\min,+)$ product can be performed in time $O((\frac{n}{2^\ell})^\alpha)$. Thus the total cost of the computation is $O(\sum_\ell 2^{2\ell} \cdot (\frac{n}{2^\ell})^\alpha)$. This quantity is $O(n^\alpha)$ for $\alpha > 2$, and $O(n^2 \log n)$ otherwise. $\square$

Recall that in the scored parser the only $(\min,+)$ products that we perform, inside procedures $\text{Parse}_k(B)$, $k \in \{3,4\}$, are of type $B.B$. Let us focus on one such product. Recall also that in order to implement such products we consider each production rule $(Z \to XY)$, we derive integer matrices $B(X)$ and $B(Y)$, and then we compute $B(X) \star B(Y)$.

It is possible that B does not have relaxed W-BD. We next show that we can reduce the product $B.B$ to a constant number of $(\min,+)$ products over smaller square matrices which have that property. In order to do that we need to analyze the behavior of the algorithm at intermediate steps.

We start by proving a technical lemma that relates the indexes of the input square matrices B in the various procedures to the indexes of the original matrix A. Note that each such matrix B corresponds to some submatrix of A, however indexes of B might map discontinuously to indexes of A (i.e., the latter indexes do not form one interval). This is due to Step 2 of $\text{Parse}_3(B)$ and $\text{Parse}_4(B)$ that constructs a matrix C by removing central rows and columns of B. Note also that by construction the row indexes of A associated to B are equal to the corresponding column indexes (since the mentioned step removes the same set of rows and columns). We next denote by $map_B(i)$ the row/column index of A corresponding to row/column index i of B. We say that B is *contiguous* if $\{map_B(i)\}_{i=1,\dots,n(B)} = \{map_B(1), map_B(1)+1, \dots, map_B(1)+n(B)-1\}$. In other words, the indexes of A corresponding to B form an interval of consecutive indexes. We say that B has a *discontinuity* at index $1 < a < n(B)$ if B is not continuous but the submatrices $B^{[1,a]}_{[1,a]}$ and $B^{[a+1,n(B)]}_{[a+1,n(B)]}$ are continuous.

Lemma 8. *Any input matrix B considered by the procedures in the scored parser:*

1. *is continuous if it is the input to Parse;*
2. *is continuous or has a discontinuity at $n(B)/2$ if it is the input to Parse$_2$ or Parse$_4$;*
3. *is continuous or has a discontinuity at $n(B)/3$ or $2n(B)/3$ if it is the input to Parse$_3$.*

Proof. We prove the claim by induction on the partial order induced by the recursion tree from the root to the leaves.

Parse(B) satisfies the claim in the starting call with $B = A$. In the remaining cases Parse(B) is called by Parse(D) with $B = D^{[1,n(D)/2]}_{[1,n(D)/2]}$ or $B = D^{[n(D)/2+1,n(D)]}_{[n(D)/2+1,n(D)]}$. Claim (1) follows by inductive hypothesis on Parse(D).

Parse$_4$(B) is called by Parse$_2$(B). Claim (2) follows by inductive hypothesis on Parse$_2$(B).

Parse$_3$(B) is called by Parse$_2$(D), with (i) $B = D^{[1,3n(D)/4]}_{[1,3n(D)/4]}$ or (ii) $B = D^{[n(D)/4+1,n(D)]}_{[n(D)/4+1,n(D)]}$. By inductive hypothesis D is continuous or has a discontinuity at $n(D)/2$. Hence B, if not continuous, has a discontinuity at $2n(B)/3$ in case (i) and at $n(B)/3$ in case (ii). Claim (3) follows.

Finally consider Parse$_2$(B). If it is called by Parse(B), Claim (4) follows by inductive hypothesis on Parse. If it is called by Parse$_2$(D) with $B = D^{[n(D)/4+1,3n(D)/4]}_{[n(D)/4+1,3n(D)/4]}$, Claim (4) follows by inductive hypothesis on Parse$_2$. Suppose it is called by Parse$_4$(D). Then D has size $n(D) = 2n(B)$, and is continuous or has a discontinuity at $n(D)/2$ by inductive hypothesis. In this case B is obtained by removing the $n(D)/2$ central columns and rows of D. Therefore B has a discontinuity at $n(B)/2$. The remaining case is that Parse$_2$(B) is called by Parse$_4$(D), where D has size $n(D) = 4n(B)/3$. In this case B is obtained by removing the $n(D)/3$ central columns and rows of D. Since D is continuous or has a discontinuity at $n(D)/3$ or $2n(D)/3$ by inductive hypothesis on Parse$_3$, B has a discontinuity at $n(B)/2$. □

The following two lemmas upper and lower bound the progress made by the scored parser in intermediate steps of the algorithm. Indeed, this is the lemma which follows from the correctness of Valiant's algorithm. It shows that any submatrix which is input to some $Parse_k$ is transitively closed after that call of $Parse_k$ completes.

Lemma 9. *Let A be the input matrix and B be any submatrix in input to some call to Parse$_k$, $k \in \{2, 3, 4\}$. Then one has the following* input property*:*

$$B_{i,j} = (A^+)_{map_B(i),map_B(j)} \quad \forall i, j \in [1, n(B) - n(B)/k] \tag{1}$$

$$B_{i,j} = (A^+)_{map_B(i),map_B(j)} \quad \forall i, j \in [n(B)/k + 1, n(B)] \tag{2}$$

The matrix B at the end of the procedure has the following output property

$$B_{i,j} = (A^+)_{map_B(i),map_B(j)} \quad \forall i, j \in [1, n(B)] \tag{3}$$

The same output property holds for procedure Parse.

Proof. We prove the claim by induction on the total order defined by the beginning and the end of each procedure during the execution of the algorithm starting from Parse(A).

Consider the input property of some call Parse$_2$(B). If Parse$_2$(B) is called in Step 6 of Parse, the property follows by inductive hypothesis on the output property of Parse. If Parse$_2$(B) is called in Step 4 of Parse$_2$, the claim follows by inductive hypothesis on the input property of Parse$_2$. Otherwise Parse$_2$(B) is called in Step 3 of Parse$_k$, $k \in \{3, 4\}$: the claim follows by inductive hypothesis on the input property of Parse$_k$.

Let us consider the input property of some call Parse$_3$(B). Note that Parse$_3$(B) is called either in Step 5 or in Step 6 of Parse$_2$. In the first case the claim follows by inductive hypothesis on the input and output property of Parse$_2$. In the second case it follows from the input property of Parse$_2$ and the output property of Parse$_3$. The input property of Parse$_4$ holds by a similar argument.

The output property of Parse follows immediately by the correctness of the trivial algorithm used in Step 2 and by inductive hypothesis on the output property of Parse$_2$. Similarly, the output property of Parse$_2$ follows by the correctness of the trivial algorithm used in Step 2 and by inductive hypothesis on the output property of Parse$_4$.

The output property of Parse$_k$, $k \in \{3, 4\}$, follows by inductive hypothesis on the input property of Parse$_k$ and on the output property of Parse$_2$. □

Lemma 10. *Let B be some input matrix to Parse$_k$, $k \in \{2, 3, 4\}$, and define $S = [1, n(B)/k]$ and $L = [n(B) - n(B)/k + 1, n(B)]$. Then submatrix B^S_L contains only empty entries. Furthermore, submatrix B^L_S has relaxed W-bounded differences for $k \in \{3, 4\}$, and for $k = 2$ it has relaxed W-BD or contains only empty entries except for entry $B_{n(B)/2,n(B)/2+1}$.*

Proof. We prove the claim by induction on the partial order induced by the recursion tree from root to leaves.

Consider first B^S_L. By Lemma 8 it is easy to see that the entries of B^S_L always correspond to entries in the lower triangular part of A. Observe that this portion of A contains only empty entries both at the beginning of the algorithm (by construction) and at its end (by the properties of A^+). Since the algorithm never turns a non-empty entry into an empty one, it follows that those entries are always empty during the execution of the algorithm. The claim on B^S_L follows.

Consider next B^L_S. We start by considering Parse$_3(B)$. This procedure is called on Step 5 or 6 of Parse$_2(D)$ for some matrix D. In both cases the claim follows by inductive hypothesis on Parse$_2$ since Step 4 in the first case, and Steps 4-5 in the second case do not update B^L_S. Note that in both cases the entry $D_{n(D)/2,n(D)/2+1}$ is not contained in B^L_S. The case of Parse$_4(B)$ is analogous: the procedure is called in Step 7 of Parse$_2(D)$ and the portion of D corresponding to B^L_S is not updated in Steps 4-6.

It remains to consider Parse$_2$. Suppose that Parse$_2(B)$ is called by Parse. In this case B^L_S is always taken from the upper-triangular part of the initialization matrix A (which contains only empty entries with the exception of the entries above the main diagonal). The claim follows. If Parse$_2(B)$ is called in Step 4 of Parse$_2$, the claim follows by inductive hypothesis on Parse$_2$.

Otherwise Parse$_2$ is called in Step 3 of Parse$_k(D)$, $k \in \{3, 4\}$. Consider first the case $k = 3$. Let $S' = [1, n(D)/3]$, $M' = [n(D)/3 + 1, 2n(D)/3]$ and $L' = [2n(D)/3 + 1, n(D)]$. Note that $B^L_S = D^{L'}_{S'}$. By inductive hypothesis we have that at the beginning of the call to Parse$_3$ matrix $D^{L'}_{S'}$ has relaxed W-BD. In Step 1 of Parse$_3$ we execute the update

$$D^{L'}_{S'} \leftarrow D^{L'}_{S'} \cup D^{S'}_{S'}.D^{L'}_{S'} \cup D^{M'}_{S'}.D^{L'}_{M'} \cup D^{L'}_{S'}.D^{L'}_{L'}$$

Observe that the union and the product of two square matrices with relaxed W-BD has relaxed W-BD. Since $D^{L'}_{S'}$ has this property by inductive hypothesis, it is sufficient to show that the same holds for $D^{S'}_{S'}$, $D^{M'}_{S'}$, $D^{L'}_{M'}$, and $D^{L'}_{L'}$. By Lemma 8, D is continuous or has a discontinuity at $n(D)/3$ or $2n(D)/3$. Combining this with Lemma 9, one obtains the desired property.

The case of Parse$_4$ is analogous. Let us define $S' = [1, n(D)/4]$, $M' = [n(D)/4 + 1, n(D)/2]$, $M'' = [n(D)/2 + 1, 3n(D)/4]$, and $L' = [3n(D)/4 + 1, n(D)]$. Also in this case $B^L_S = D^{L'}_{S'}$, and at the beginning of Parse$_4(D)$ matrix $D^{L'}_{S'}$ has relaxed W-BD by inductive hypothesis. The update in this case can be written as:

$$D^{L'}_{S'} \leftarrow D^{L'}_{S'} \cup D^{S'}_{S'}.D^{L'}_{S'} \cup D^{M'}_{S'}.D^{L'}_{M'} \cup D^{M''}_{S'}.D^{L'}_{M''} \cup D^{L'}_{S'}.D^{L'}_{L'}$$

By Lemma 8, D is continuous or has a discontinuity at $n(D)/2$. Combining this with Lemma 9, one obtains that $D^{S'}_{S'}$, $D^{M'}_{S'}$, $D^{L'}_{M'}$, $D^{M''}_{S'}$, $D^{L'}_{M''}$, and $D^{L'}_{L'}$ all have relaxed W-BD. The claim on $D^{L'}_{S'}$ (hence on B^L_S) follows. □

The following corollary, in combination with Lemma 7 and our fast bounded-differences $(\min,+)$ product algorithm, implies a fast implementation of the scored parser.

Corollary 2. *The product in Step 1 of Parse$_k$, $k \in \{3,4\}$, can be reduced to a constant number of products between smaller square submatrices with relaxed W-BD.*

Proof. Consider first Parse$_3(B)$. Let $S = [1, n(B)/3]$, $M = [n(B)/3+1, 2n(B)/3]$ and $L = [2n(B)/3+1, n(B)]$. It is sufficient to show that all the submatrices B_I^J, $I, J \in \{S, M, L\}$, have relaxed W-BD. By Lemma 8, B is continuous or has a discontinuity at $n(D)/3$ or $2n(D)/3$. It follows by Lemma 9, that submatrices B_S^S, B_M^M, B_L^L, B_S^M, B_M^L, B_M^S, and B_L^M are equal to square blocks of A^+. In more detail, the first three such matrices correspond to blocks along the main diagonal of A^+, the second two such matrices to blocks in the upper-triangular part of A^+, and the last two such matrices to blocks in the lower-triangular part of A^+. Therefore all such matrices have relaxed W-BD. By Lemma 10, also B_S^L and B_L^S have this property.

The proof for Parse$_4(B)$ is analogous. Here we consider the partition of B into blocks B_I^J, $I, J \in \{S, M', M'', L\}$, with $S = [1, n(B)/4]$, $M' = [n(B)/4+1, n(B)/2]$, $M'' = [n(B)/2+1, 3n(B)/4]$, and $L = [3n(B)/4+1, n(B)]$. All these blocks have relaxed W-BD by Lemmas 8, 9, and 10 similarly to the previous case. $\square$

B Bounded-Differences $(\min,+)$ Product: Improvement, Derandomization, and Generalization

B.1 Speeding Up Phase 2

We begin with a more refined version of Lemma 5.

Lemma 11. *W.h.p for any $1 \le r \le \rho$ the number of weakly relevant, weakly r-uncovered triples is $\tilde{O}(n^{2.5} + n^3/r^{1/3})$.*

Proof. Fix $k \in [n]$. For any $1 \le r \le \rho+1$, we construct a bipartite graph $G_{r,k}$ on $n+n$ vertices (we denote vertices in the left vertex set by i or i^r and vertices in the right vertex set by j or j^r). We add edge $\{i,j\}$ to $G_{r,k}$ if the triple (i,k,j) is weakly relevant and weakly $(r-1)$-uncovered. Note that $E(G_{r,k}) \supseteq E(G_{r',k})$ for $r \le r'$. Denote the number of edges in $G_{r,k}$ by $m_{r,k}$ and its density by $\alpha_{r,k} = m_{r,k}/n^2$. In the following we show that w.h.p. $m_{r,k}$ drops by a constant factor after $O(\alpha_{r,k}^{-3}\log(n))$ rounds.

We denote by $Y_{r,k}(i,j)$ the number of 4-cycles in $G_{r,k}$ containing edge $\{i,j\}$. (If $\{i,j\}$ is not an edge, we set $Y_{r,k}(i,j) = 0$.) Observe that $Y_{r,k}(i,j) \ge Y_{r',k}(i,j)$ for $r \le r'$.

Now fix a round r. For $r' \ge r$, we call $\{i,j\}$ *r'-heavy* if $Y_{r',k}(i,j) \ge 2^{-8}\alpha_{r,k}^3 n^2$. Let r^* be a round with $r^* - r = \Theta(\alpha_{r,k}^{-3}\log n)$ (with sufficiently large hidden constant). We claim that w.h.p. no $\{i,j\}$ is r^*-heavy. Indeed, in any round $r \le r' < r^*$, either $\{i,j\}$ is not r'-heavy, say because some of the edges in its 4-cycles got covered in the last round, but then we are done. Or $\{i,j\}$ is r'-heavy, but then with probability $Y_{r',k}(i,j)/n^2 = \Omega(\alpha_{r,k}^3)$ we choose $i^{r'}, j^{r'}$ as the remaining vertices in one of the 4-cycles containing $\{i,j\}$. In this case, Lemma 3 shows that (i,k,j) will get weakly covered in round r', so in particular $\{i,j\}$ is not $(r'+1)$-heavy. Over $r^* - r = \Theta(\alpha_{r,k}^{-3}\log n)$ rounds, this event happens with high probability.

Now we know that w.h.p. no $\{i,j\}$ is r^*-heavy. Thus, each of the $\alpha_{r^*,k}n^2$ edges of $G_{r^*,k}$ is contained in less than $2^{-8}\alpha_{r,k}^3 n^2$ 4-cycles, so that the total number of 4-cycles in $G_{r^*,k}$ is at most $2^{-8}\alpha_{r^*,k}\alpha_{r,k}^3 n^4$. On the other hand, Lemma 4 shows that the number of 4-cycles is at least $(\alpha_{r^*,k}n^2)^4/(32n^4)$ if $\alpha_{r^*,k} \ge 2/\sqrt{n}$. Altogether, we obtain $\alpha_{r^*,k} \le \max\{\alpha_{r,k}/2, 2/\sqrt{n}\}$, i.e., after $O(\alpha_{r,k}^{-3}\log n)$ rounds the density of $G_{r,k}$ drops by a factor 1/2. In particular, w.h.p. in round $r = O(\sum_{i=0}^{t} 2^{3i}\log n) = O(2^{3t}\log n)$ the density of $G_{r,k}$ is at most 2^{-t}, as long as $2^{-t} \ge 2/\sqrt{n}$. In other words, w.h.p. the density of $G_{r,k}$ is $O((\log(n)/r)^{1/3} + n^{-1/2})$, and $m_{r,k} \le$

$O(n^2(\log(n)/r)^{1/3} + n^{3/2})$. Since we have $r \le \rho \le n$, the dominating term is $O(n^2(\log(n)/r)^{1/3})$. Since $m_{r+1,k}$ counts the weakly relevant, weakly r-uncovered triples (i,k,j) for fixed k, summing over all $k \in [n]$ yields the claim. □

Inspection of the proof of Lemma 11 shows that we only count triples i,k,j that get covered in round r if the triple i^r, k, j^r is weakly relevant and weakly $(r-1)$-uncovered. Hence, after line 12 of Algorithm 1 we can remove all columns k from A^r and all rows k from B^r for which i^r, k, j^r is not weakly relevant or not weakly $(r-1)$-uncovered. Then Lemma 11 still holds, so the other steps are not affected. Note that checking this property for i^r, k, j^r takes time $O(\rho)$ for each k and each round r, and thus in total incurs cost $O(n\rho^2) \le O(\rho n^2)$, which is dominated by the remaining running time of Phase 2. Using rectangular matrix multiplication to compute $A^r * B^r$ (Lemma 1) we obtain the following improved running time.

Lemma 12. *W.h.p. the improved Step 2 takes time $\tilde{O}(\rho\Delta W \cdot M(n, n/\rho^{1/3}, n))$.*

Proof. Let s_r denote the number of surviving k's in round r, i.e., the number of k such that i^r, k, j^r is weakly relevant, weakly $(r-1)$-uncovered. Using Lemma 1, the running time of Step 2 is bounded by $\tilde{O}\big(\sum_{r=1}^{\rho} \Delta W \cdot M(n, s_r, n)\big)$. Note that for any x, y, we have $M(n,x,y) \le O((1+x/y)M(n,y,n))$, by splitting columns and rows of length x into $\lfloor x/y \rfloor \le 1 + x/y$ blocks. Hence, we can bound the running time by $\tilde{O}\big(\sum_{r=1}^{\rho} \Delta W (1 + s_r \rho^{1/3}/n) \cdot M(n, n/\rho^{1/3}, n)\big)$. Thus, to show the desired bound of $\tilde{O}(\rho\Delta W \cdot M(n, n/\rho^{1/3}, n))$, it suffices to show that $\sum_{r=1}^{\rho} s_r \le \tilde{O}(n\rho^{2/3})$ (in expectation and w.h.p.).

W.h.p. the number of weakly relevant, weakly $(r-1)$-uncovered triples is $\tilde{O}(n^3/r^{1/3})$, by Lemma 11. Thus, for a random k the probability that i^r, k, j^r is weakly relevant, weakly $(r-1)$-uncovered is $\tilde{O}(r^{-1/3})$. Summing over all k we obtain $\mathbb{E}[s_r] = \tilde{O}(n/r^{1/3})$ (note that the inequality $s_r \le n$ allows to condition on any w.h.p. event for evaluating the expected value). This yields the desired bound for the expectation of the running time, since $\sum_{r=1}^{\rho} \mathbb{E}[s_r] \le \tilde{O}(n \sum_{r=1}^{\rho} r^{-1/3}) \le \tilde{O}(n\rho^{2/3})$.

For concentration, fix r^* as any power of two and consider $s_{r^*} + s_{r^*+1} + \ldots + s_{2r^*-1}$. For any $r^* \le r < 2r^*$ denote by $\bar{s}_r$ the number of i^r, k, j^r that are weakly relevant and weakly r^*-uncovered, and note that $s_r \le \bar{s}_r$. Again we have $\mathbb{E}[\bar{s}_r] \le \tilde{O}(n/r^{1/3})$. Moreover, conditioned on the choices up to round r^*, the numbers $\bar{s}_r$, $r^* \le r < 2r^*$, are independent. Hence, a Chernoff bound (Lemma 13) on variables $\bar{s}_r/n \in [0,1]$ shows that w.h.p.

$$\bar{s}_{r^*} + \bar{s}_{r^*+1} + \ldots + \bar{s}_{2r^*-1} \le O\big(\mathbb{E}[\bar{s}_{r^*} + \bar{s}_{r^*+1} + \ldots + \bar{s}_{2r^*-1}] + n \log n\big).$$

Hence, w.h.p. $\sum_{r=1}^{\rho} s_r \le \sum_{r=1}^{\rho} \bar{s}_r \le O\big(n \log(n) \log(\rho) + \sum_{r=1}^{\rho} \mathbb{E}[\bar{s}_r]\big)$. Using our bound on $\mathbb{E}[\bar{s}_r]$, we obtain $\sum_{r=1}^{\rho} s_r \le \tilde{O}(n + n\rho^{2/3}) \le \tilde{O}(n\rho^{2/3})$ as desired. □

Lemma 13. *Let $X_1, \ldots, X_n$ be independent random variables taking values in $[0,1]$, and set $X := \sum_{i=1}^{n} X_i$. Then for any $c \ge 1$ we have*

$$\Pr[X > (1 + 6ec)\mathbb{E}[X] + c \log n] \le n^{-c}.$$

Proof. If $\mathbb{E}[X] < \log(n)/2e$ we use the standard Chernoff bound $\Pr[X > t] \le 2^{-t}$ for $t > 2e\mathbb{E}[X]$ with $t := c \log n$. Otherwise, we use the standard Chernoff bound $\Pr[X > (1+\delta)\mathbb{E}[X]] \le \exp(-\delta\mathbb{E}[X]/3)$ for $\delta \ge 1$ with $\delta := 6ec$. □

B.2 Speeding Up Phase 3

Enumerating approximately uncovered blocks In line 17 we check for each block i', k', j' of approximately relevant triples whether it consists of approximately uncovered triples. This step can be improved using rectangular matrix multiplication as follows. For each block k' we construct a $(n/\Delta) \times \rho$ matrix $U^{k'}$ and a $\rho \times (n/\Delta)$ matrix $V^{k'}$ with entries $U^{k'}_{xr} := [|A^r_{x\Delta,k'}| \le$

$44\Delta W]$ and $V^{k'}_{ry} := [|B^r_{k',y\Delta}| \leq 44\Delta W]$. Then from the Boolean matrix product $U^{k'} \cdot V^{k'}$ we can infer for any block i', k', j' whether it consists of approximately uncovered triples by checking $(U^{k'} \cdot V^{k'})_{i'/\Delta, j'/\Delta} = 1$. Hence, enumerating the approximately relevant, approximately uncovered triples i', k', j' can be done in time $O((n/\Delta) \cdot M(n/\Delta, \rho, n/\Delta))$.

Recursion In the exhaustive search in Step 3, see lines 18-19, we essentially compute the $(\min, +)$-product of the matrices $(A_{ik})_{i \in I(i'), k \in I(k')}$ and $(B_{kj})_{k \in I(k'), j \in I(j')}$. These matrices again have W-BD, so we can use Algorithm 1 recursively to compute their product. Writing $T(n, W)$ for the running time of our algorithm, this reduces the time complexity of one invocation of lines 18-19 from $O(\Delta^3)$ to $T(\Delta, W)$, which in total reduces the running time of the exhaustive search from $\tilde{O}(n^3/\rho^{1/3})$ to $\tilde{O}((T(\Delta, W)/\Delta^3) \cdot n^3/\rho^{1/3})$ w.h.p.

B.3 Total running time

Recall that Step 1 takes time $O((n/\Delta)^3 + n^2)$, Step 2 now runs in $\tilde{O}(\rho \Delta W \cdot M(n, n/\rho^{1/3}, n))$ w.h.p., and Step 3 now runs in $\tilde{O}((n/\Delta) \cdot M(n/\Delta, \rho, n/\Delta) + (T(\Delta, W)/\Delta^3) \cdot n^3/\rho^{1/3})$ w.h.p. This yields the complicated recursion

$$T(n, W) \leq \tilde{O}\Big(\rho \Delta W \cdot M\Big(n, \frac{n}{\rho^{1/3}}, n\Big) + \frac{n}{\Delta} \cdot M\Big(\frac{n}{\Delta}, \rho, \frac{n}{\Delta}\Big) + \frac{T(\Delta, W)}{\Delta^3} \cdot \frac{n^3}{\rho^{1/3}}\Big),$$

while the trivial algorithm yields $T(n, W) \leq O(n^3)$.

In the remainder, we focus on the case $W = O(1)$, so that $T(n, W) = T(n, O(1)) =: T(n)$. Setting $\Delta := n^\delta$ and $\rho := n^s \log^c n$ for constants $\delta, s \in (0, 1)$ and sufficiently large $c > 0$, and using $M(a, \tilde{O}(b), c) \leq \tilde{O}(M(a, b, c))$, we obtain

$$T(n) \leq \tilde{O}\big(n^{\delta+s} M(n, n^{1-s/3}, n) + n^{1-\delta} M(n^{1-\delta}, n^s, n^{1-\delta})\big) + n^{3-3\delta-s/3} T(n^\delta).$$

This is a recursion of the form $T(n) \leq \tilde{O}(n^\alpha) + n^\beta T(n^\gamma)$, which solves to $T(n) \leq \tilde{O}(n^\alpha + n^{\beta/(1-\gamma)})$, by an argument similar to the master theorem. Hence, we obtain

$$T(n) \leq \tilde{O}\big(n^{\delta+s} M(n, n^{1-s/3}, n) + n^{1-\delta} M(n^{1-\delta}, n^s, n^{1-\delta}) + n^{(3-3\delta-s/3)/(1-\delta)}\big).$$

We optimize this expression using the bounds on rectangular matrix multiplication by Le Gall [19]. Specifically, we set $\delta := 0.0772$ and $s := 0.4863$ to obtain a bound of $O(n^{2.8244})$, which proves part of Theorem 1. Here we use the bounds $M(m, m^{1-s/3}, m) \leq M(m, m^{0.85}, m) \leq O(m^{2.260830})$ and $M(m, m^{s/(1-\delta)}, m) \leq M(m, m^{0.5302}, m) \leq O(m^{2.060396})$ by Le Gall [19] for $m = n$ and $m = n^{1-\delta}$, respectively.

We remark that if perfect rectangular matrix multiplication exists, i.e., $M(a, b, c) = \tilde{O}(ab + bc + ac)$, then our running time becomes $T(n) \leq \tilde{O}(n^{2+\delta+s} + n^{3-3\delta} + n^{(3-3\delta-s/3)/(1-\delta)})$, which is optimized for $\delta = (13 - \sqrt{133})/18$ and $s = (2\sqrt{133} - 17)/9$, yielding an exponent of $(5 + \sqrt{133})/6 \approx 2.7554$. This seems to be a barrier for our approach.

B.4 Derandomization

The only random choice in Algorithm 1 is to pick i^r, j^r uniformly at random from $[n]$. In the following we show how to derandomize this choice, at the cost of increasing the running time of Step 2 by $O(\rho (n/\Delta)^{1+\omega})$. At the end of this section we then show that we still obtain a truly sub-cubic total running time.

Fix round r. Similar to the proof of Lemma 11, for any k' divisible by Δ we construct a bipartite graph $G'_{r,k'}$ with vertex sets $\{\Delta, 2\Delta, \ldots, n\}$ and $\{\Delta, 2\Delta, \ldots, n\}$ (we denote vertices in the left vertex set by i' or i^r and vertices in the right vertex set by j' or j^r). We connect i', j' by an edge in $G'_{r,k'}$ if i', k', j' is approximately relevant and approximately $(r-1)$-uncovered.

In $G'_{r,k'}$ we count the number of 3-paths between any i', j'. Now we pick i^r, j^r as the pair i', j' maximizing the sum over all k' of the number of 3-paths in $G'_{r,k'}$ containing i', j'. This finishes the description of the adapted algorithm.

It is easy to see that this adaptation of the algorithm increases the running time of Step 2 by at most $O(\rho(n/\Delta)^{\omega+1})$. Indeed, constructing all graphs $G'_{r,k'}$ over the ρ rounds takes time $O(\rho(n/\Delta)^3)$, and computing the number of 3-paths between any pair of vertices can be done in $O(|V(G'_{r,k'})|^\omega)$, which over all r and k' incurs a total cost of $O(\rho(n/\Delta)^{\omega+1})$.

It remains to argue that an analog of Lemma 11 still holds. Note that the number of 3-paths in $G'_{r,k'}$ containing i^r, j^r counts the number of i', j' such that $(i', k', j'), (i^r, k', j'), (i', k', j^r), (i^r, k', j^r)$ are all approximately relevant and approximately $(r-1)$-uncovered. For any such (i', k', j'), any $(i, k, j) \in I(i') \times I(k') \times I(j')$ gets covered in round r, in fact, these are the triples counted in Lemma 11 (after replacing "weakly" by "approximately" relevant and uncovered). As we maximize this number, we cover at least as many new triples as in expectation, so that Lemma 11 still holds, after replacing "weakly" by "approximately" relevant and uncovered: *For any* $1 \le r \le \rho$ *the number of approximately relevant, approximately* r*-uncovered triples is* $\tilde{O}(n^3/r^{1/3})$. Since this is sufficient for the analysis of Step 3, we obtain the same running time bound as for the randomized algorithm, except that Step 2 takes additional time $O(\rho(n/\Delta)^{1+\omega})$.

Total running time Adapting the basic Algorithm 1 yields, as in Section 2.4, a running time of $\tilde{O}(\rho\Delta W n^\omega + \rho(n/\Delta)^{1+\omega} + n^3/\rho^{1/3})$. We optimize this by setting $\Delta := (n/W)^{1/(\omega+2)}$ and $\rho := n^{3(5+\omega-\omega^2)/(4\omega+8)} W^{-3(\omega+1)/(4\omega+8)}$. This yields running time $\tilde{O}(n^{3-(5+\omega-\omega^2)/(4\omega+8)} W^{(\omega+1)/(4\omega+8)}) \le O(n^{2.9004} W^{0.1929})$, using the current bound of $\omega \le 2.3728639$ [20]. In particular, the algorithm has truly sub-cubic running time whenever $W \le O(n^{2-\omega+3/(\omega+1)-\varepsilon}) \approx O(n^{0.5165-\varepsilon})$ for any $\varepsilon > 0$.

For $W = O(1)$, adapting the improved algorithm from Section B.3 yields

$$T(n) \le \tilde{O}\big(n^{\delta+s} M(n, n^{1-s/3}, n) + n^{1-\delta} M(n^{1-\delta}, n^s, n^{1-\delta}) + n^{(3-3\delta-s/3)/(1-\delta)} + n^{(1+\omega)(1-\delta)+s}\big),$$

which is $O(n^{2.8603})$ for $\delta := 0.2463$ and $s := 0.3159$, finishing the proof of Theorem 1. Here we use the bounds $M(m, m^{1-s/3}, m) \le M(m, m^{0.90}, m) \le O(m^{2.298048})$ and $M(m, m^{s/(1-\delta)}, m) \le M(m, m^{0.45}, m) \le O(m^{2.027102})$ by Le Gall [19] for $m = n$ and $m = n^{1-\delta}$, respectively.

B.5 Generalizations

In this section we study generalizations of Theorem 1. In particular, we will see that it suffices if A has bounded differences along either the columns or the rows, while B may be arbitrary. Since $A \star B = (B^T \star A^T)^T$, a symmetric algorithm works if A is arbitrary and B has bounded differences along either its columns or its rows.

Theorem 5. *Let* B *be arbitrary and assume either of the following:*

(1) for an appropriately chosen $1 \le \Delta \le n$ *we are given a partitioning* $[n] = I_1 \cup \ldots \cup I_{n/\Delta}$ *such that* $\max_{i \in I_\ell} A_{i,k} - \min_{i \in I_\ell} A_{i,k} \le \Delta W$ *for all* k, ℓ*, or*

(2) for an appropriately chosen $1 \le \Delta \le n$ *we are given a partitioning* $[n] = K_1 \cup \ldots \cup K_{n/\Delta}$ *such that* $\max_{k \in K_\ell} A_{i,k} - \min_{k \in K_\ell} A_{i,k} \le \Delta W$ *for all* i, ℓ*.*

If $W \le O(n^{3-\omega-\varepsilon})$*, then* $A \star B$ *can be computed in randomized time* $O(n^{3-\Omega(\varepsilon)})$*. If* $W = O(1)$*, then* $A \star B$ *can be computed in randomized time* $O(n^{2.9217})$*.*

Important special cases of the above theorem are that A has W-BD only along columns ($|A_{i+1,k} - A_{i,k}| \le W$ for all i, k) or only along the rows ($|A_{i,k+1} - A_{i,k}| \le W$ for all i, k). In these cases the assumption is indeed satisfied, since we can choose each I_ℓ or K_ℓ as a continuous subset of Δ elements of $[n]$, thus amounting to a total difference of at most ΔW.

Proof. (1) For the first assumption, adapting Algorithm 1 is straight-forward. Instead of blocks $I(I') \times I(k') \times I(j')$ we now consider blocks $I_\ell \times \{k\} \times \{j\}$, for any $\ell \in [n/\Delta]$, $k, j \in [n]$. Within any such block, $A_{i,k}$ varies by at most ΔW by assumption. Moreover, B_{kj} does not vary at all, since k, j are fixed. We adapt Step 1 by computing for each block $I_\ell \times \{k\} \times \{j\}$ one entry $\tilde{C}_{i^*j} = (A \star B)_{i^*j}$ exactly, for some $i^* \in I_\ell$, and setting $\tilde{C}_{ij} := \tilde{C}_{i^*j}$ for all other $i \in I_\ell$. It is easy to see that Lemma 2 still holds. Note that Step 1 now runs in time $O(n^3/\Delta)$.

Step 2 does not have to be adapted at all, since as we remarked in Section 2.2 it works for arbitrary matrices.

For Step 3, we have analogous notions of being approximately relevant or uncovered, by replacing the notion of "blocks". Thus, we now iterate over every ℓ, k, j, check whether it is approximately relevant (i.e., $|A_{i^*k} + B_{kj} - \tilde{C}_{i^*j}| \le 8\Delta W$ for some $i^* \in I_\ell$), check whether it is approximately uncovered (i.e., for all rounds r we have $|A^r_{i^*k}| > 44\Delta W$ or $|B^r_{kj}| > 44\Delta W$), and if so we exhaustively search over all $i \in I_\ell$, setting $\hat{C}_{ij} := \min\{\hat{C}_{ij}, A_{ik} + B_{kj}\}$. Then Lemma 6 still holds and correctness and running time analysis hold almost verbatim. Step 3 now runs in time $\tilde{O}(\rho n^3/\Delta + n^3/\rho^{1/3})$ w.h.p.

The total running time is w.h.p. $\tilde{O}(\rho \Delta W n^\omega + \rho n^3/\Delta + n^3/\rho^{1/3})$. We optimize this by setting $\Delta := n^{(3-\omega)/2}/W^{1/2}$ and $\rho := n^{3(3-\omega)/8}/W^{3/8}$, obtaining time $\tilde{O}(n^{3-(3-\omega)/8} W^{1/8})$. As desired, this is $n^{3-\Omega(\varepsilon)}$ for $W = O(n^{3-\omega-\varepsilon})$, while for $W = O(1)$ it evaluates to $\tilde{O}(n^{3-(3-\omega)/8}) \le O(n^{2.9217})$. The latter bound can be slightly improved by incorporating the improvements from Section B, we omit the details.

(2') Before we consider the second assumption, we first discuss a stronger assumption where also B is nice along the columns: Assume that for an appropriately chosen $1 \le \Delta \le n$ we are given a partitioning $[n] = K_1 \cup \ldots \cup K_{n/\Delta}$ such that $\max_{k \in K_\ell} A_{i,k} - \min_{k \in K_\ell} A_{i,k} \le \Delta W$ for all i, ℓ and $\max_{k \in K_\ell} B_{kj} - \min_{k \in K_\ell} B_{kj} \le \Delta W$ for all ℓ, j.

In this case, adapting Algorithm 1 is straight-forward and similar to the last case. Instead of blocks $I_\ell \times \{k\} \times \{j\}$ we now consider blocks $\{i\} \times I_\ell \times \{j\}$, for any $\ell \in [n/\Delta]$, $i, j \in [n]$. Within any such block, A and B vary by at most ΔW by assumption. We adapt Step 1 by computing for each i, ℓ, j for some value $k^* \in K_\ell$ the sum $A_{ik^*} + B_{k^*j}$. We set $\tilde{C}_{ij}$ as the minimum over all ℓ of the computed value. It is easy to see that Lemma 2 still holds. Step 1 now runs in time $O(n^3/\Delta)$.

Step 2 does not have to be adapted at all, since as we remarked in Section 2.2 it works for arbitrary matrices.

For Step 3, we now iterate over every i, ℓ, j, check whether it is approximately relevant (i.e., $|A_{ik^*} + B_{k^*j} - \tilde{C}_{ij}| \le 8\Delta W$ for some $k^* \in K_\ell$), check whether it is approximately uncovered (i.e., for all rounds r we have $|A^r_{ik^*}| > 44\Delta W$ or $|B^r_{k^*j}| > 44\Delta W$), and if so we exhaustively search over all $k \in K_\ell$, setting $\hat{C}_{ij} := \min\{\hat{C}_{ij}, A_{ik} + B_{kj}\}$. Then Lemma 6 still holds and correctness and running time analysis hold almost verbatim. Step 3 now runs in time $\tilde{O}(\rho n^3/\Delta + n^3/\rho^{1/3})$ w.h.p.

We obtain the same running time as in the last case.

(2) For the second assumption, compute for all ℓ, j the value $v(\ell, j) := \min\{B_{kj} \mid k \in K_\ell\}$, and consider a matrix B' with $B'_{kj} := \min\{B_{kj}, v(\ell, j) + 2\Delta W\}$, where $k \in K_\ell$. Note that for any i, k, j with $k \in K_\ell$ and $k^* \in K_\ell$ such that $B_{k^*j} = v(\ell, j)$, we have $A_{ik} + (v(\ell, j) + 2\Delta W) \ge A_{ik^*} + B_{k^*j} + \Delta W > C_{ij}$, since A varies by at most ΔW. Hence, no entry $B_{kj} = v(\ell, j) + 2\Delta W$ is strongly relevant, which implies $A \star B' = A \star B$. Note that B' satisfies $\max_{k \in K_\ell} B_{kj} - \min_{k \in K_\ell} B_{kj} \le 2\Delta W$ for all ℓ, j, so we can use case (2') to compute $A \star B'$. Since B' can be computed in time $O(n^2)$, the result follows. □

C From LED and RNA-folding to Scored Parsing

In this section, we show that LED can be reduced to scored parsing on BD grammars. The same construction works also in the case of insertions and deletions only. Observe that RNA-folding can be seen as a special case of LED with only insertion and deletion as edits. Indeed, if d is the optimum distance of a sequence using only insertions and deletions from the RNA grammar, then the maximum number of bases that can be paired in the corresponding RNA sequence is simply $n - d$, where n is the length of the sequence. Therefore the mentioned reduction for LED implies the same for RNA-folding.

Recall that a CFG G is a W-BD grammar if for any nonterminal X, terminal x, and string σ, the following holds:

$$\left|s(X,\sigma) - s(X,\sigma x)\right| \le W \qquad \text{and} \qquad \left|s(X,\sigma) - s(X,x\sigma)\right| \le W$$

.

We assume that G is given in the Chomsky normal form (CNF).

Creating an Augmented Grammar The first step of solving LED is to create an augmented grammar as follows. We are given a CNF context-free grammar $G = (N, T, P, S)$, and a string σ as the input to LED. We next use $N(G)$ to denote the non-terminals of G. W.l.o.g. we can assume that, for each terminal a, there exists at least one non-terminal X_a such that $(X \to a) \in P$ (if not, we can create one new such non-terminal and a corresponding production rule without changing the language generated by G).

Now we turn G into a scored grammar G'. Each production rule of P gets a score of 0. We then add new, costly production rules and one new non-terminals in order to model insertions, deletions and substitutions.

In order to model insertions, we create a new non-terminal I, and add the following scored production rules:

$$I \to X_a I \,(\text{score} = 1) \mid I X_a \,(\text{score} = 1) \mid \varepsilon \,(\text{score} = 0), \text{ for every } a \in T$$

$$X \to XI \,(\text{score} = 0) \mid IX \,(\text{score} = 0) \quad \text{for every nonterminal } X \in N$$

Observe that I can generate any string of terminals, and the associated score is the length of the string. Then I can be inserted at any point of a string generated by the original grammar.

In order to model substitutions and deletions we add the productions

$$X \to a \,(\text{score} = 1) \mid \varepsilon \,(\text{score} = 1) \text{ for every } X \in N \text{ and } a \in T$$

$X \to b$ allows us to substitute b in place of any character generated by X, and similarly $X \to \varepsilon$ allows to delete any such character. Note that we might have two identical production rules with different scores: let us keep only the one with minimum score.

This creates an augmented grammar $G' = (N \cup I, T, P', S)$ where P' included the new production rules and the original rules of P.

It has been shown in [4] that with the augmented grammar G', LED is equivalent to find the minimum scored parsing of σ according to G'.

Claim 1. *G' is a 1-BD grammar.*

Proof. Consider any nonterminal $X \in N \cup I$. Assume that X produces σ with score ℓ, then $X \to IX \to X_x X \to xX$ is a valid parsing of $x\sigma$ with score $\ell + 1$.

Now assume that X produces $x\sigma$ with score ℓ. There must be some production rule of type $(Y \to x)$ that produces the first terminal of $x\sigma$. By replacing that rule with $(Y \to \varepsilon)$ we obtain the string σ while increasing the cost at most by one.

The other condition $\left|s(X,\sigma) - s(X,x\sigma)\right| \le 1$ can be proved similarly. □

Note that, having or not having the substitution rules do not affect the 1-BD property of G'. However, G' is not in CNF form, which is required for our algorithms to work. We therefore convert G' to CNF, and show that even after conversion, the BD property holds. Again, we only need the rules for insertions and deletions. Hence, this also shows the BD property required for RNA-folding.

Conversion to Chomsky normal form. Note that except for the rules $X \to \varepsilon$, $X \in N' \backslash \{S\}$, all the other productions are in the required CNF form.

We next execute the standard steps to convert G' into an equivalent CNF grammar G'', with the extra care of propagating the scores during this process in a natural way (when we *compose* two rules to create a new rule, we sum the respective scores). By construction any sequence of production rules in G'' that produces a string σ with some score s corresponds to a sequence of production rules in G' that produces σ with the same score s and vice versa. Therefore, following the proof of Claim 1 one obtains that G'' is 1-BD.

In some more details, consider any nonterminal $X \in N(G'') = N(G')$, if X produces σ with score ℓ, then $X \to IX \to X_x X \to xX$ is a valid parsing of $x\sigma$ with score $\ell + 1$ in G''. Suppose now that X produces $x\sigma$ with score ℓ. Consider the parsing of $x\sigma$ where the left-most nonterminal is always expanded first. Consider the steps when a production rule of the form $Y_1 \to Y_2 Z_1$ and $Y_2 \to X_x Z$ are applied. Then in G'' we have a production $Y_1 \to Z Z_1$ with a score of $score(Y_2 \to Z) = score(X_x \to \varepsilon) = 1$ such that if we use it instead of $Y_1 \to Y_2 Z_1$ then X produces σ with a score of $\ell + 1$. If there is no such Y_1, that is $S \to X_x Z$, then if in the original grammar, there was a production of the form $Z \to Z_1 Z_2$ (or $Z \to a$ where a is either a terminal or ε) then after the conversion, if the conversion is correct, we must also have $S \to Z_1 Z_2$ (or $S \to a$) that is the scores for producing σ and $x\sigma$ are off by at most 1.

We therefore obtain the desired property.

Proposition 1. *LED and RNA-folding problem can be reduced to scored parsing problem over* 1-*BD grammars.*

D From Optimal Stack Generation to Scored Parsing

Recall that a context-free grammar G is a W-bounded difference grammar if for any nonterminal X, terminal x, and string $\sigma \neq \varepsilon$, the following holds:

$$\left| s(X, \sigma) - s(X, \sigma x) \right| \leq W$$

and

$$\left| s(X, \sigma) - s(X, x\sigma) \right| \leq W$$

.

We now show that OSG can be reduced to a scored parsing problem on 3-bounded difference grammar in Chomsky Normal Form (CNF).

We first show a scored grammar G that is not yet in CNF or 3-BD. It has a non-terminal S representing that the stack is empty, and a non-terminal X_c for any $c \in \Sigma$ representing that the topmost symbol on the stack is c. Moreover, we use a symbol N_c for emitting symbol c, and call a production producing N_c a "pre-emit". To obtain bounded differences, we also allow N_c to "change it's mind" and not emit any character.

S	$\to$	ε	(score 0)	end of string
S	$\to$	$X_c S$	(score 1)	push c, for any $c \in \Sigma$
X_c	$\to$	$N_c X_c$	(score 0)	pre-emit c, for any $c \in \Sigma$
X_c	$\to$	$X_{c'} X_c$	(score 1)	push c', for any $c, c' \in \Sigma$
X_c	$\to$	ε	(score 1)	pop c, for any $c \in \Sigma$
N_c	$\to$	c	(score 1)	emit c, for any $c \in \Sigma$
N_c	$\to$	ε	(score 0)	do not emit c, for any $c \in \Sigma$

Indeed, these productions model that from an empty stack the only possible operation is to push some symbol c, while if the topmost symbol is c then we may (pre-)emit c, or push another symbol c', or pop c.

Consider the example string $bccab$. This can be produced as follows, where we always resolve the leftmost non-terminal. Note that the suffix of non-terminals always corresponds to the content of the stack.

$$\begin{aligned} S &\to X_bS \to N_bX_bS \to bX_bS \to bX_cX_bS \to bN_cX_cX_bS \to bcX_cX_bS \to bcN_cX_cX_bS \\ &\to bccX_cX_bS \to bccX_bS \to bccX_aX_bS \to bccN_aX_aX_bS \to bccaX_aX_bS \to bccaX_bS \\ &\to bccaN_bX_bS \to bccabX_bS \to bccabS \to bccab \end{aligned}$$

Bounded Differences We change the above grammar G by adding the following productions, thus obtaining a grammar G'. For any non-terminal X we introduce the productions

$$X \;\to\; N_cX \text{ (score} = 3) \mid XN_c \text{ (score} = 3), \quad \text{for any } c \in \Sigma$$

Moreover, for any $c \in \Sigma$ we introduce the productions

$$N_c \;\to\; c' \text{ (score} = 3), \quad \text{for any } c' \in \Sigma$$

The following claims show that G' is 3-BD and correctly models the ODG problem. However, it is not yet in CNF.

Claim 2. *G' is a 3-bounded difference grammar.*

Proof. Consider any nonterminal X of G'. If X produces σ with score ℓ, then $X \to N_cX \to cX \to^* c\sigma$ is a valid parsing of $c\sigma$ with score $\ell + 3$.

On the other hand, if X produces $c\sigma$ with score ℓ, then consider the parsing of $c\sigma$ where the left-most nonterminal is always expanded first. Consider the step in the parsing where c is produced from the terminal producing rule $N_c \to c$. Replace it with $N_c \to \varepsilon$, decreasing the score by 1. Hence, X produces σ with a score of at most $\ell - 1$.

Similarly, the other condition $\big|s(X,\sigma) - s(X,\sigma c)\big| \le 3$ can be verified. □

Claim 3. *G' describes the same scored language as G, i.e., $s_G(S,\sigma) = s_{G'}(S,\sigma)$ for the starting symbol S and any string σ.*

Proof. We will use the following property of OSG (*): If string $\sigma = \sigma'\sigma''$ can be generated with cost s, then for any symbol $c \in \Sigma$ the string $\sigma'c\sigma''$ can be generated with cost at most $s+3$. Indeed, for any sequence of push, emit, and pop that generates σ, at the point where we have exactly emitted σ' we can insert the sequence "push(c), emit, pop", then the adapted sequence produces $\sigma'c\sigma''$ and has length $s+3$.

Note that if the old grammar G produces string σ at cost s, and we insert $k \ge 0$ new symbols into σ, obtaining a string σ', then the new productions allow to generate σ' at cost $s+3k$, i.e., $s_{G'}(S_0,\sigma') \le s+3k$ (and this is all that changes). However, the generation cost of σ' is anyways bounded by $s+3k$ by the above property (*), even without using the new productions, i.e., we have $s_G(S_0,\sigma') \le s+3k$ by (*). This shows $s_G(S_0,\sigma) \le s_{G'}(S_0,\sigma)$ for any string σ. Since we extended the grammar, we clearly also have $s_G(S_0,\sigma) \ge s_{G'}(S_0,\sigma)$. Thus, the new productions do not change the generation cost $s(S_0,\sigma)$. □

Conversion to Chomsky normal form Note that except for the rules $X \to \varepsilon$ for all non-terminals X all the other productions are in the required CNF form.

The conversion works as follows. Consider every rule p of the form $Y \to XZ \mid ZX$ where $X \to \varepsilon \in G'$. Add a new rule $Y \to Z$ with a score of $s(Y \to Z) = s(p) + s(X \to \varepsilon)$ if $Z \ne Y$.

Now consider every rule in G' where Y appears on the right, and add a new rule by replacing Y with Z and adding $s(Y \to Z)$ to its score. This process continues recursively until no new rule is formed. If the same rule is generated with multiple scores, then we simply need to keep the minimum score. After this process, we can delete all productions of the form $X \to \varepsilon$ or $X \to Y$ to obtain a CNF grammar G''. This is exactly how ϵ productions are eliminated while converting a non-CNF grammar to CNF. We just additionally maintain the scores. We remark that we ignored the handling of the empty string here; to be precise one has to add a new starting symbol S_0 with the production $S_0 \to \varepsilon$, and for each production $S \to u$ add $S_0 \to u$.

By construction, G'' is in CNF and generates the same scored language as G' and thus correctly models the OSG problem. It remains to prove that G'' is 3-bounded.

Claim 4. *G'' is a 3-bounded difference grammar.*

Proof. Again consider any nonterminal $X \in N(G'') = N(G')$. If X produces σ with score ℓ, then $X \to N_c X \to cX \to^* c\sigma$ is a valid parsing of $c\sigma$ with score $\ell + 3$ in G''. Let us now consider X that produces $c\sigma$ with score ℓ. Consider the parsing of $c\sigma$ where the left-most nonterminal is always expanded first. Consider the steps when a production rule of the form $Y_1 \to Y_2 Z_1$ and $Y_2 \to N_c Z$ are applied. Then in G'' we have a production $Y_1 \to Z Z_1$ with a score of $s(Y_2 \to Z) = s(Y_2 \to N_c Z) + s(N_c \to \varepsilon) = s(Y_2 \to N_c Z)$ such that if we use it instead of $Y_1 \to Y_2 Z_1$ then X produces σ with the same score ℓ. Hence, $s_{G''}(X, \sigma) \leq s_{G''}(X, c\sigma)$.

The other condition $\left| s_{G''}(X, \sigma) - s_{G''}(X, \sigma c) \right| \leq 3$ can be verified similarly. $\square$