TRULY SUB-CUBIC ALGORITHMS FOR LANGUAGE EDIT DISTANCE AND RNA FOLDING VIA FAST BOUNDED-DIFFERENCE MIN-PLUS PRODUCT*

KARL BRINGMANN†, FABRIZIO GRANDONI‡, BARNA SAHA§, AND VIRGINIA VASSILEVSKA WILLIAMS¶

Abstract. It is a major open problem whether 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, in particular since it is equivalent to the famous All-Pairs-Shortest-Paths problem (APSP) in n-vertex graphs. Some restrictions of the $(\min,+)$-product to special types of matrices are known to 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 (answering an open problem of Chan and Lewenstein).

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 (a major problem in bioinformatics) 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, Saarland Informatics Campus, 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 [18] 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, 46, 22]) and vertex centrality measures (e.g. [13, 2]).

While the $(\min,+)$-product of two $n \times n$ matrices has a trivial $O(n^3)$ time algorithm, 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. [19, 42, 15]), a relatively recent breakthrough of Williams [49] gave an $O(n^3/c^{\sqrt{\log n}})$ time algorithm for a constant $c > 1$. Note that despite this striking improvement, the running time is still not truly sub-cubic.

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

(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 [45, 21];
(2) when each row of matrix A has at most D distinct values, then the $(\min,+)$-product of A with an arbitrary matrix B can be computed in time $\tilde{O}(Dn^{(3+\omega)/2})$ [50].[2]

Among other applications, these restricted $(\min,+)$-products yield faster algorithms for special cases of APSP. E.g., the distance product of type (1) is used to compute APSP in both undirected [40, 41] and directed [52] 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 [50].

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*

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

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 2. *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 nonconstant 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:

Theorem 3. *Let B be arbitrary and assume either of the following:*

$$\textit{(i)}\ \forall i,j \in [n],\ |A_{i,j} - A_{i+1,j}| \leq W \quad \textit{or} \quad \textit{(ii)}\ \forall i,j \in [n],\ |A_{i,j} - A_{i,j+1}| \leq W$$

If $W \leq O(n^{3-\omega-\varepsilon})$ for any $\varepsilon > 0$, 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 $O(n^{2.9217})$.

The main obstacle towards achieving a truly sub-cubic algorithm for the $(\min,+)$-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

[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 D distinct values per column.

some constant $\varepsilon > 0$, while (2), without any other assumptions on A and B, achieves the same 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 $C = A \star B$.. For BD matrices it is quite easy to obtain an additive overestimate $\tilde{C}$ of C: Let us subdivide A and B into square blocks of size $\Delta \times \Delta$, for some small polynomial value $\Delta = n^{\delta}$. 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 *some* triple of indices (i, k, j) in order to obtain an overestimate of *all* the entries in $A' \star B'$ within an additive error of $O(\Delta W)$. This way in truly sub-cubic time we can compute an additive $O(\Delta W)$ overestimate $\tilde{C}$ of C.

Remark: It would seem that Phase 1 requires that the matrices are BD, and 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, given a very good approximation $\tilde{C}$, can compute the correct product C; this is exactly what the remaining phases do. To show this, we use a scaling approach à la Seidel [40]. Assume that the entries of A and B are nonnegative integers bounded by[3] M, 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 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, if all remaining phases (which compute the correct product C from the approximation $\tilde{C}$) could be made to work for arbitrary matrices, then Phase 1 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 result (1) from [6], 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 (for some d) after $\tilde{O}(n^{3d})$ rounds of perturbations and associated bounded entry $(\min, +)$-products, there are at most $O(n^{3-d})$ triples (i, k, j) for which (a) $|A_{i,k} + B_{k,j} - \tilde{C}_{i,j}| \leq c' \cdot \Delta W$ for some $c' = O(1)$ (i.e. k is a potential witness for $C_{i,j}$) and (b) none of the perturbations had both $A^r_{i,k}$ and $B^r_{k,j}$ finite.

Interestingly, our proof of correctness of Phase 2 relies on an extremal graph theoretical lemma that bounds from below 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 the "bad" triples that are not covered by the perturbation steps, we could simply iterate over them in a brute-force way, fixing $\tilde{C}$ to the correct product C. Since Phases 1 and 2 do 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 the $(\min, +)$-matrix product!

Phase 3: Finding and fixing the bad triples.. To fix the bad triples, one could try to keep track of the triples covered in each perturbation iteration. For arbitrary matrices A and B this would not give a truly sub-cubic algorithm as the number of triples is already n^3. For BD matrices, however, we do not need to keep track of all triples, but it suffices to consider the 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 at least n^3 time, we obtain an algorithm spending $O(\rho \cdot (n/\Delta)^3)$ time, where ρ is the number of perturbation rounds (from Phase 2). After finding the bad block representative triples, we can iterate over 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 is $O(n^{3-d})$ as this is the total number of bad triples.

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

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

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 some 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 a 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 we will not explicitly mention the dependency of running times on the grammar size.[4] We will also assume that G is given in Chomsky Normal Form (CNF)[5]. In a breakthrough result Valiant [44] 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 non-negative integer score (or cost) $s(p)$. The goal is 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 scored parsing problem, 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 [39]). In particular, applying Williams' algorithm for the $(\min,+)$-product [49], 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 string σ from X (where the grammar G is assumed to be clear from the context). Let us define a bounded difference notion for CFGs. Intuitively, we require that adding or deleting a terminal at one endpoint of a string does not change the corresponding score by much.

Definition 4. *A CFG G is a W-bounded differences (W-BD) grammar if, for any non-terminal X, terminal x, and non-empty string of terminals σ, 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 Valiant's parser, we are able to show that the scored parsing problem on BD grammars can be reduced to the $(\min,+)$-product of BD matrices (see Section 3).

Theorem 5. *Let $O(n^\alpha)$ be the time needed to perform one $(\min,+)$-product of two $n \times n$ BD matrices. Then the scored parsing problem on BD grammars in CNF can be solved in time $\tilde{O}(n^\alpha)$.*

Corollary 6. *The scored parsing problem on BD grammars in CNF can be solved in randomized time $\tilde{O}(n^{2.8244})$ and deterministic time $\tilde{O}(n^{2.8603})$.*

BD grammars appear naturally in relevant applications. Consider for example the well-studied Language Edit Distance problem (LED) [4, 32, 28, 38, 39, 1, 36]. 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. Indeed, 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. We show that, by performing the above steps carefully, the resulting scored grammar is BD, leading to a truly sub-cubic algorithm for LED via Corollary 6 (see Section 5.2). 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 the parsing

[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.

[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.

problem for the parenthesis grammar has recently received considerable attention [9, 38, 26, 14, 29, 34]. 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 defined by Nussinov and Jacobson in 1980 [33]. They 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 $\Sigma' = \{c' \mid c \in \Sigma\}$ be the set of "matching" letters, such that for every letter $c \in \Sigma$ the pair c, c' matches. 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. In particular, 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., $i < j < k < l$. (In nature, there are 4 types of nucleotides in an RNA molecule, with matching pairs A, U and C, G, i.e., $|\Sigma| = 2$.) We can rephrase RNA-folding as follows. We are given the CFG with productions $S \to SS \mid \varepsilon$ and $S \to \sigma S \sigma' \mid \sigma' S \sigma$ for any $\sigma \in \Sigma$ with matching $\sigma' \in \Sigma'$. The goal is to find the minimum number of *insertions* and *deletions* of symbols on a given string σ that will generate a string σ' consistent with the above grammar. This is essentially a variant of LED where only insertions and deletions (and no substitutions) are allowed. Despite considerable efforts (e.g. [47, 5, 51, 33]), 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 6.

As a final application, consider the Optimum Stack Generation problem (OSG) described by Tarjan in [43]. 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*, ending in an empty stack. For example, the string $BCCAB$ can be printed via the following sequence of operations: $push(B)$, $emit(B)$, $push(C)$, $emit(C)$, $emit(C)$, $pop(C)$, $push(A)$, $emit(A)$, $pop(A)$, $emit(B)$, $pop(B)$. While there is a simple $O(n^3)$ time algorithm for OSG, Tarjan suspected this could be improved. In Section 5.3, 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 7. *LED, RNA-folding, and OSG can be solved in randomized time $\tilde{O}(n^{2.8244})$ and deterministic time $\tilde{O}(n^{2.8603})$ (on constant-size grammars or alphabet, respectively).*

Moreover, 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 $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ the $(\min, +)$-convolution of a and b is the vector $c = (c_1, \ldots, c_n)$ 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 differences along the rows, while if b has bounded differences, then B has bounded differences along the columns. Theorem 3 now allows us 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 the $(\min, +)$-convolution over bounded monotone sequences is equivalent to computing it over bounded difference sequences.

We envision other applications of our BD $(\min, +)$-product algorithm 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, 32, 28, 38, 39, 1, 36]. 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 time $O(|G|^2 n^3)$ [4], which was improved to $O(|G| n^3)$ by Myers in 1985 [32]. These algorithms are based on the popular CYK parsing algorithm [3] with the observation that LED can be reduced to a scored parsing problem [4]. This implies the previous best running time of $O(n^3/2^{\Theta(\sqrt{\log n})})$. In a recent paper [39], 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 [28, 1], LED cannot be approximated within any multiplicative factor in time $o(n^\omega)$ (unless cliques can be found

faster). Interestingly, if we only allow insertions as edit operations, then [39] also showed that a truly sub-cubic exact algorithm is unlikely due to a reduction from APSP [46]. In contrast, here we show that with insertions and deletions (and possibly substitutions) as edit operations, LED is solvable in truly sub-cubic time. LED provides a very generic framework for modeling problems with many applications (e.g. [25, 24, 48, 31, 37, 35, 23]). A fast exact algorithm for it is likely to have tangible impact.

RNA-Folding.. Computational approaches to finding the secondary structure of RNA molecules are used extensively in bioinformatics applications. Since the seminal work of Nussinov and Jacobson [33], a multitude of sophisticated RNA-folding algorithms with complex objectives and softwares have been developed[7], but the basic dynamic programming algorithm of Nussinov and Jacobson remains at the heart of all of these. Despite much effort, only mild improvements in running time have been achieved so far [47, 5, 51], 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 for the Clique problem. Moreover, their results imply that any truly sub-cubic algorithm for RNA-folding must use fast matrix multiplication, unless there are fast algorithms for Clique that do not use fast matrix multiplication. Their results hold for alphabet Σ of size 13, which was recently improved to $|\Sigma| = 2$ [17].

Dyck LED.. A problem closely related to RNA-folding is Dyck language edit distance, which is LED for the grammar of well-balanced parentheses. For example, [()] belongs to the Dyck language, but [) or)[do not. (The RNA grammar is often referred to as "two-sided Dyck", where)[is also a valid match.) Dyck edit distance with insertion and deletion generalizes the widely-studied string edit distance problem [30, 27, 10, 11, 8, 7]. When approximation is allowed, a near-linear time $O(\text{poly}\log n)$-approximation algorithm was developed by Saha [38]. Moreover, a $(1+\epsilon)$-approximation in $O(n^{\omega})$ time was shown in [39] 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 $t(n)$ with high probability.

Matrix Multiplication.. As is typical, we denote by $\omega < 2.3729$ [45, 21] 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 is 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 $M(a,b,c)$ [20]. As in (1) above we have the following:

LEMMA 8 ([6]). *Let A, B be $a \times b$ and $b \times c$ matrices with entries in $\{-M, -M+1 \ldots, M\} \cup \{\infty\}$. 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 $\tilde{O}(Mn^{\omega})$.*

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 productions, and $S \in N$ is the start symbol. We recall that a production rule p is of the form $X \to \alpha$, with $X \in N$ and[8] $\alpha \in (N \cup T)^*$, 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. For $\alpha, \beta \in (N \cup T)^*$, we write $\alpha \to \beta$ if β can be generated from α by applying one production rule, and we write $\alpha \to^* \beta$ ("β can be derived from α") if there is a sequence of productions generating β from α. The language $L(X)$ generated by a non-terminal $X \in N$ is the set of strings $\sigma \in T^*$ that can be derived from X. We also let $L(G) := L(S)$ denote the language generated by G.

At many places in this paper we may assume that G is given in Chomsky normal form (CNF). Specifically, all productions are of the form $Z \to XY$, $Z \to c$, and $S \to \epsilon$, where $X, Y \in N \setminus \{S\}$, $Z \in N$, $c \in T$, and ϵ denotes the empty string.

A *scored grammar* is a CFG G where each production rule $p \in P$ is associated with a non-negative integer score $s(p)$. Intuitively, applying production p has a cost $s(p)$. The total score of any derivation is simply the sum of all scores of productions used in the derivation. For any $X \in N$ and $\sigma \in T^*$, we define $s_G(X, \sigma) = s(X, \sigma)$ as the minimum total score of any derivation $X \to^* \sigma$, or as ∞ if $\sigma \notin L(X)$. The scored

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

[8] Given a set of symbols U, by U^* we denote as usual any, possibly empty, string of elements from U.

language generated by $X \in N$ is the set $\{(\sigma, s(X,\sigma)) \mid \sigma \in L(X)\}$, and the scored language generated by G is the scored language generated by the start symbol S.

In the *scored parsing problem* on grammar G, we are given a string σ of length n, and we wish to compute $s(S,\sigma)$.

Organization.. In Section 2 we give our main technical result, a truly sub-cubic algorithm for the $(\min,+)$-product of BD matrices. In Section 3, we show how bounded difference scored parsing can be solved asymptotically in the same time as computing a single BD $(\min,+)$ product. In Section 4, we show how to further reduce the running time, how to derandomize our algorithm, and some generalizations of our approach. Section 5 is 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. 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 4. 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 the 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[9] 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 are 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 9. *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 first (1). 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, we can move from $A_{i',k}$ to $A_{i',k'}$. The overall absolute change is therefore at most $2\Delta W$. The proof of (2) is analogous.

For (3), 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 $C_{i',j'} \leq C_{i,j} + 2\Delta W$.

It remains to prove (4). 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). ∎

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 8. 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 $A \star B$.

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

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.

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 us to compute $C^r = A^r \star B^r$ efficiently using Lemma 8. 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 10). 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 10. Similarly, we need a weak version of being covered.

Definition 10. *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. Finally, a triple is strongly (resp., weakly) r-covered if it is not strongly (resp., weakly) r-uncovered.

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

Lemma 11. *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 in round r. ∎

We will crucially exploit the following extremal graph-theoretic result. This relatively simple result might be known, but we were not able to find an explicit reference and so we prove it for the sake of completeness.

Lemma 12. *Let $G = (U \cup V, E)$ be a bipartite graph with $|U| = |V| = n$ nodes per partition and $|E| = m$ edges. Let C be the number of 4-cycles of G. If $m \geq 2n^{3/2}$, then $C \geq m^4/(32n^4)$.*

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

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

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

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

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

LEMMA 13. *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 weakly r-uncovered.

Let $z = c(n^2/\rho)\ln n$ for any constant $c > 3$. 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 11, (i,k,j) will be r-covered and thus (i,j) will be removed from G_k.

Thus, if in any round r the indices 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, then (i,j) is in G_k at the end of all ρ rounds. For a particular edge (i,j) with at least z 4-cycles in a particular G_k, the probability that 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, over all i,j,k we obtain an error probability of at most $1/n^{c-3}$, which is $1/\mathrm{poly}(n)$ as we picked $c > 3$. Hence, at the end of all ρ rounds, with high probability 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, 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 12, $C_k \geq (m_k/n)^4/32$. Thus,

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

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 19. 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 14. *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* $1 \le r \le \rho$ *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 15. *Any strongly relevant triple is also approximately relevant. Any approximately relevant triple is also weakly relevant. The 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 9, 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 $|A_{i,k} - A_{i',k'}| \le 2\Delta W$, $|B_{k,j} - B_{k',j'}| \le 2\Delta W$, and $|C_{i,j} - \tilde{C}_{i',j'}| = |C_{i,j} - \tilde{C}_{i,j}| \le 4\Delta W$. Thus, if $A_{i,k} + B_{k,j} = C_{i,j}$ (i.e., (i,k,j) is strongly relevant), then $|A_{i',k'} + B_{k',j'} - \tilde{C}_{i',j'}| \le 8\Delta W$ (i.e., (i,k,j) is approximately relevant). On the other hand, if (i,k,j) is approximately relevant, then $|A_{i,k} + B_{k,j} - C_{i,j}| \le 16\Delta W$ (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 9, 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 $|A^r_{i,k} - A^r_{i',k'}| \le 4\Delta W$. 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, $|B^r_{k,j} - B^r_{k',j'}| \le 4\Delta W$. The statements on "$r$-uncovered" follow immediately from these inequalities. ∎

In our algorithm, we enumerate every triple (i',k',j') whose indices 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 15 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 8 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 15. 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 19. Thus, w.h.p. Phase 3 takes total time $\tilde{O}((n/\Delta)^3\rho + n^3/\rho^{1/3} + n^{2.5})$. 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 $\omega \le 2.3729$ [45, 21] this running time evaluates to $O(n^{2.8825})$.

3. Fast Scored Parsing. In this section, we prove Theorem 5 that reduces the scored parsing problem for BD grammars to the $(\min,+)$ product for BD matrices. For a square matrix M, we let $n(M)$ denote its number of rows and columns.

We will exploit a generalization of Valiant's parser [44]. We start by describing Valiant's classic approach in Section 3.1. Then in Section 3.2 we show how to modify Valiant's parser to solve the scored parsing problem, thereby replacing Boolean matrix multiplications by $(\min,+)$ products. Finally, in Section 3.3 we show that all $(\min,+)$ products in our scored parser involve BD matrices.

3.1. 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 $\sigma \in L(G)$. In a breakthrough paper [44], Valiant presented a reduction from parsing to Boolean matrix multiplication, which we describe in the following (for a more detailed description,

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 $I(q) = \{q - \Delta + 1, \ldots, q\}$.

▷ *Phase 1: compute entry-wise additive $4\Delta W$-approximation $\tilde{C}$ of $A \star B$*
for any i', j' divisible by Δ **do**
  $\tilde{C}_{i',j'} := \min\{A_{i',k'} + B_{k',j'} \mid k' \text{ divisible by } \Delta\}$
  for any $i \in I(i')$, $j \in I(j')$ **do**
    $\tilde{C}_{i,j} := \tilde{C}_{i',j'}$
▷ *Phase 2: randomized reduction to $(\min,+)$-product with small entries*
initialize all entries of $\hat{C}$ with ∞
for $1 \le r \le \rho$ **do**
  pick i^r and j^r independently and uniformly at random from $[n]$
  for all i, k **do**
    set $A^r_{i,k} := A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r}$
    if $A^r_{i,k} \notin [-48\Delta W, 48\Delta W]$ then set $A^r_{i,k} := \infty$
  for all k, j **do**
    set $B^r_{k,j} := B_{k,j} - B_{k,j^r} + \tilde{C}_{i^r,j^r} - \tilde{C}_{i^r,j}$
    if $B^r_{k,j} \notin [-48\Delta W, 48\Delta W]$ then set $B^r_{k,j} := \infty$
  compute $C^r := A^r \star B^r$ using Lemma 8
  for all i, j **do** $\hat{C}_{i,j} := \min\{\hat{C}_{i,j}, C^r_{i,j} + \tilde{C}_{i,j^r} - \tilde{C}_{i^r,j^r} + \tilde{C}_{i^r,j}\}$
▷ *Phase 3: exhaustive search over all relevant uncovered triples of indices*
for all i', k', j' divisible by Δ **do**
  if $|A_{i',k'} + B_{k',j'} - \tilde{C}_{i',j'}| \le 8\Delta W$ **then**
    if for all r we have $|A^r_{i',k'}| > 44\Delta W$ or $|B^r_{k',j'}| > 44\Delta W$ **then**
      for all $i \in I(i'), k \in I(k'), j \in I(j')$ **do**
        $\hat{C}_{i,j} := \min\{\hat{C}_{i,j}, A_{i,k} + B_{k,j}\}$
return $\hat{C}$

see [44]). 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 $C_{i,j} = \bigcup_{k=1}^{b} A_{i,k}.B_{k,j}$. Observe that this "." operator can be reduced to the computation of a constant[10] number of standard Boolean matrix multiplications. Indeed, for a matrix M and non-terminal X, we let $M(X)$ be the 0-1 matrix with the same dimensions as X and entries $M(X)_{i,j} = 1$ iff $X \in M_{i,j}$. In order to compute the product $C = A.B$, we initialize matrix C 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 the 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

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

where

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

Here unions are taken component-wise.

[10] Here we ignore the (polynomial) dependence on the size of the grammar G, as we assume for simplicity that G has constant size.

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

Suppose that, for two given $n \times n$ matrices, the "." operation can be performed in time $O(n^\alpha)$ for some $2 \leq \alpha \leq 3$, and note that the "$\cup$" operation can be performed in time $O(n^2)$. Crucially, we cannot simply use the usual squaring technique to compute A^+ in time $\tilde{O}(n^\alpha)$, due to the fact that "." is not associative. However, Valiant describes a more sophisticated approach to achieve the same running time. It then follows that the parsing problem can be solved in time $\tilde{O}(n^\omega)$, where $2 \leq \omega < 2.373$ is the exponent of fast Boolean matrix multiplication [45, 21] ($O(n^\omega)$ if $\omega > 2$).

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 $j \in J$.

Parse(B):

1: **if** $n(B) > 1$ **then**
2: Parse($B^{[1,n(B)/2]}_{[1,n(B)/2]}$)
3: Parse($B^{[n(B)/2+1,n(B)]}_{[n(B)/2+1,n(B)]}$)
4: Parse$_2(B)$

Parse$_2(B)$:

1: **if** $n(B) > 2$ **then**
2: Parse$_2$($B^{[n(B)/4+1,3n(B)/4]}_{[n(B)/4+1,3n(B)/4]}$)
3: Parse$_3$($B^{[1,3n(B)/4]}_{[1,3n(B)/4]}$)
4: Parse$_3$($B^{[n(B)/4+1,n(B)]}_{[n(B)/4+1,n(B)]}$)
5: Parse$_4(B)$

Parse$_3(B)$:

1: Let $I_1 = [1, n(B)/3]$, $I_2 = [n(B)/3+1, 2n(B)/3]$, and $I_3 = [2n(B)/3+1, n(B)]$
2: $B^{I_3}_{I_1} \leftarrow B^{I_3}_{I_1} \cup (B^{I_2}_{I_1} \,.\, B^{I_3}_{I_2})$
3: $C \leftarrow$ matrix obtained from B by deleting row/column indices in I_2
4: Parse$_2(C)$
5: $B \leftarrow$ matrix obtained from C by reintroducing the rows and columns deleted in Step 3

Parse$_4(B)$:

1: Let $I_1 = [1, n(B)/4]$, $I_2 = [n(B)/4+1, 2n(B)/4]$, $I_3 = [2n(B)/4+1, 3n(B)/4]$, and $I_4 = [3n(B)/4+1, n(B)]$
2: $B^{I_4}_{I_1} \leftarrow B^{I_4}_{I_1} \cup (B^{I_2}_{I_1} \,.\, B^{I_4}_{I_2}) \cup (B^{I_3}_{I_1} \,.\, B^{I_4}_{I_3})$
3: $C \leftarrow$ matrix obtained from B by deleting row/column indices in $I_2 \cup I_3$
4: Parse$_2(C)$
5: $B \leftarrow$ matrix obtained from C by reintroducing the rows and columns deleted in Step 3

For the sake of simplicity we assume that $n+1$ is a power of 2. This way we can avoid the use of ceilings and floors in the definition of some indices. It is not hard to handle the general case either by introducing ceilings and floors or by introducing dummy entries (with a mild adaptation of some definitions). Valiant's fast procedure to compute the transitive closure of a given matrix is described in Algorithm 2. In this algorithm, we use the following notation: 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 as input an $n(B) \times n(B)$ matrix B, and the result of the computation is stored in B (i.e., B is passed by reference). Assuming that $n+1$ is a power of 2, it is easy to see that the sizes of the input matrices to Parse and Parse$_2$ are all powers of 2. This guarantees that all the indices used in the algorithm are integers (this way we can avoid ceilings and floors as mentioned earlier).

The running time bound $\tilde{O}(n^\omega)$ follows by standard arguments. For the (subtle) correctness argument we refer to [44], but the argument is also implicit in Lemma 17 below.

3.2. Scored Parser and $(\min,+)$ **Products.** We can adapt Valiant's approach to scored parsing as follows[11]. Let $G=(N,T,P,S)$ be a scored grammar with score function s (mapping productions to non-negative integers). Let us consider the set $\mathcal{F}_N$ of all functions $F\colon N\to\mathbb{N}_{\geq 0}\cup\{\infty\}$. We interpret $F(X)$ as the score of non-terminal $X\in N$, and thus F is a score function on the non-terminals. We write $\overline{\infty}$ for the score function mapping each $X\in N$ to ∞. Let $F_1,F_2\in\mathcal{F}_N$. We redefine the operator "$\cup$" as pointwise minimum:

$$(F_1\cup F_2)(X):=\min\{F_1(X),F_2(X)\}.$$

We also redefine the operator "." as follows (where the minimum is ∞ if the set is empty):

$$(F_1.F_2)(X)=\min\{s(X\to YZ)+F_1(Y)+F_2(Z)\mid (X\to YZ)\in P\}.$$

Given the above operations "." and "$\cup$", we can define the product of two matrices whose entries are in $\mathcal{F}_N$ as well as the transitive closure of one such square matrix in the same way as before, i.e., $C=A.B$ is defined via $C_{i,j}=\bigcup_k A_{i,k}.B_{k,j}$, and for an $m\times m$-matrix A we have $A^+=\bigcup_{i=1}^m A^{(i)}$ where $A^{(1)}=A$ and $A^{(i)}=\bigcup_{j=1}^{i-1}A^{(j)}.A^{(i-j)}$. We can then solve the scored parsing problem as follows. For a given string σ of length n, we define a $(n+1)\times(n+1)$ matrix A whose entries are in $\mathcal{F}_N$, where

$$A_{i,i+1}(X)=\min\{s(X\to\sigma_i)\mid (X\to\sigma_i)\in P\},$$

for $i=1,\ldots,n$ and $A_{i,j}=\overline{\infty}$ for $j\neq i+1$. Then by the definition of the operator "." it follows that $(A^+)_{i,j}$ evaluated at X equals the score $s(X,\sigma_i\ldots\sigma_{j-1})$. Hence, the solution to the scored parsing problem is $(A^+)_{1,n+1}$ evaluated at the starting symbol $S\in N$.

Crucially for our goals, the "." operator can be implemented with a reduction to a constant (for constant grammar size) 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 $\mathcal{F}_N$ and for $X\in N$, let $M(X)$ be the matrix with the same dimension as M and having $M(X)_{i,j}=M_{i,j}(X)$. With the same notation as before, in order to compute the product $C=A.B$ we initialize matrix C with $\overline{\infty}$ entries. Then we consider each production rule $Z\to XY$ separately, and we compute $C'(Z)=A(X)\star B(Y)$. Then we set $C_{i,j}(Z)=\min\{C_{i,j}(Z),s(Z\to XY)+C'(Z)_{i,j}\}$ for all i,j. This computes $C=A.B$.

With the above modifications, the same Algorithm 2 computes A^+ in the scored setting. This is proven formally in the next section.

3.3. Reduction to Bounded Differences $(\min,+)$ **Product.** In this section, we show that Algorithm 2 also works in the scored setting. More importantly, we prove that the matrix products $B_I^J.B_{I'}^{J'}$ called by this algorithm can be implemented using $(\min,+)$ matrix products of W-BD matrices, if the scored grammar is W-BD (recall Definition 4). This allows us to use our main result to obtain a good running time bound for Algorithm 2 and thus for scored parsing of BD grammars.

We start by proving the correctness of Algorithm 2 in the scored setting, see Lemma 17 below. Some properties that we show along the way will be also crucial for the BD property and running time analysis.

We first prove a technical lemma that relates the indices of the input square matrices B in the various procedures to the indices of the original matrix A. Note that each such matrix B corresponds to some submatrix of A, however indices of B might map discontinuously to indices of A (i.e., the latter indices do not form one interval). This is due to Step 3 of $\mathrm{Parse}_3(B)$ and $\mathrm{Parse}_4(B)$ that constructs a matrix C by removing central rows and columns of B. Note also that by construction the row indices of A associated to B are equal to the corresponding column indices (since the mentioned step removes the same set of rows and columns). We 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 $\{map_B(i)\}_{i=1,\ldots,n(B)}=\{map_B(1),map_B(1)+1,\ldots,map_B(1)+n(B)-1\}$. In other words, the indices of A corresponding to B form an interval of contiguous indices. We say that B has a *discontinuity* at index $1<a<n(B)$ if B is not contiguous but the submatrices $B^{[1,a]}_{[1,a]}$ and $B^{[a+1,n(B)]}_{[a+1,n(B)]}$ are contiguous. We call the indices $J=\{map_B(a)+1,\ldots,map_B(a+1)-1\}$ the *missing indices* of B.

LEMMA 16. *Any input matrix B considered by the procedures in the scored parser:*

1. *is contiguous if it is the input to Parse;*

[11] This has already been done in [39], but we give details here for the sake of completeness.

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

Proof. We prove the claims by induction on the partial order induced by the recursion tree.

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 $B = D^{[n(D)/2+1,n(D)]}_{[n(D)/2+1,n(D)]}$. The claim follows by inductive hypothesis on Parse(D).

Parse$_4(B)$ is called by Parse$_2(B)$. The claim 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 contiguous or has a discontinuity at $n(D)/2$. Hence B, if not contiguous, has a discontinuity at $2n(B)/3$ in case (i) and at $n(B)/3$ in case (ii). The claim follows.

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

The following lemma proves the correctness of our algorithm, and is also crucial to analyse its running time.

LEMMA 17. *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 we have the* input property

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

Furthermore, let J be the missing indices in B if B has a discontinuity, and let $J = \emptyset$ if B is contiguous. Then for all $i \in [1, n(B)/k]$ and $j \in [n(B) - n(B)/k + 1, n(B)]$ we have the additional input property

$$B_{i,j} = A_{map_B(i),map_B(j)} \cup \bigcup_{k \in J} (A^+)_{map_B(i),k}.(A^+)_{k,map_B(j)} \tag{2}$$

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

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

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$_2(B)$. Suppose Parse$_2(B)$ is called in Step 4 of Parse(B). Let $I_1 = [1, n(B)/2]$ and $I_2 = [n(B)/2 + 1, n(B)]$. The input property (1) follows by the output property of Parse$_2(B^{I_1}_{I_1})$ and Parse$_2(B^{I_2}_{I_2})$ called in Steps 2 and 3. Since B is contiguous, we have $J = \emptyset$, and thus input property (2) requires $B_{i,j} = A_{map_B(i),map_B(j)}$ for all $i \in I_1$ and $j \in I_2$. Since in the calls of Parse(B) the input top-right quadrant is as in the initial matrix A, and this is not affected by Steps 2 and 3, input property (2) is satisfied.

If Parse$_2(B)$ is called in Step 2 of Parse$_2(D)$ for some D, the input property follows by inductive hypothesis on the input property of Parse$_2(D)$. Otherwise Parse$_2(B)$ is called in Step 4 of Parse$_k(D)$, for some D and $k \in \{3,4\}$. Input property (1) directly follows from the input property of Parse$_k(D)$. It remains to show that input property (2) holds. Let $S = [1, n(D)/k]$, $M = [n(D)/k + 1, n(D) - n(D)/k]$, and $L = [n(D) - n(D)/k + 1, n(D)]$. We need to show that B^L_S has the desired property. Let J be the missing indices in D (or $\emptyset$, if D is contiguous). Observe that, by Step 3 of Parse$_k(D)$, the missing indices in B will be $J' = M \cup J$. By the input property (2) of D, we have

$$D_{i,j} = A_{map_D(i),map_D(j)} \cup \bigcup_{k \in J} (A^+)_{map_D(i),k}.(A^+)_{k,map_D(j)} \quad \forall i \in S, \forall j \in L$$

Therefore at the end of Step 2 of $\text{Parse}_k(D)$ one has, for all $i \in S$ and $j \in L$,

$$\begin{aligned} D_{i,j} &= A_{map_D(i),map_D(j)} \cup \Big(\bigcup_{k \in J} (A^+)_{map_D(i),k} . (A^+)_{k,map_D(j)} \Big) \cup \bigcup_{k \in M} D_{i,k} . D_{k,j} \\ &= A_{map_D(i),map_D(j)} \cup \Big(\bigcup_{k \in J} (A^+)_{map_D(i),k} . (A^+)_{k,map_D(j)} \Big) \cup \bigcup_{k \in M} (A^+)_{map_D(i),k} . (A^+)_{k,map_D(j)} \\ &= A_{map_D(i),map_D(j)} \cup \bigcup_{k \in J \cup M} (A^+)_{map_D(i),k} . (A^+)_{k,map_D(j)}, \end{aligned}$$

where in the second equality we used the input property (1) of $\text{Parse}_k(D)$. This implies input property (2) for $\text{Parse}_2(B)$.

Let us consider the input property of some call $\text{Parse}_3(B)$. Note that $\text{Parse}_3(B)$ is called either in Step 3 or in Step 4 of $\text{Parse}_2(D)$ for some D. Consider the first case, the second one is analogous. Let $I_1 = [1, n(D)/4]$, $I_2 = [n(D)/4 + 1, 2n(D)/4]$, $I_3 = [2n(D)/4 + 1, 3n(D)/4]$, and $I_4 = [3n(D)/4 + 1, n(D)]$. By the input property of D and the output property of $\text{Parse}_2(D^{I_2 \cup I_3}_{I_2 \cup I_3})$, the input property (1) of B follows. The input property (2) of B follows directly from the input property (2) of D since the missing indices in D and B are the same.

Finally consider the input property of $\text{Parse}_4(B)$. Note that $\text{Parse}_4(B)$ is called in Step 5 of $\text{Parse}_2(B)$. With the same notation as above, the input property (1) of B follows directly from the output property of $\text{Parse}_3(B^{I_1 \cup I_2 \cup I_3}_{I_1 \cup I_2 \cup I_3})$ and of $\text{Parse}_3(B^{I_2 \cup I_3 \cup I_4}_{I_2 \cup I_3 \cup I_4})$. The input property (2) of B follows directly from the input property (2) of B at the beginning of $\text{Parse}_2(B)$ since $B^{I_4}_{I_1}$ is not modified by Steps 2-4.

It remains to discuss the output properties. Let us start with the output property of $\text{Parse}(B)$. The base case is $n(B) = 1$. In this case the unique entry $B_{1,1}$ corresponds to an entry in the main diagonal of the input matrix, and these entries are never updated by the algorithm. In other words, $B_{1,1} = A_{map_B(1),map_B(1)}$ where A is the input matrix (in particular, this is a trivial entry with all ∞ values). This is the correct answer since trivially $(A^+)_{i,i} = A_{i,i} = \overline{\infty}$ for all $i = 1, \ldots, n + 1$. For $n(B) \geq 2$, the output property follows from the output property of $\text{Parse}_2(B)$.

Consider next the output property of $\text{Parse}_2(B)$. The base case is $n(B) = 2$. In this case the input property of B coincides with its output property. More precisely, let J be the indices strictly between $i = map_B(1)$ and $j = map_B(2)$. Then the entry $B_{1,2}$ satisfies

$$B_{1,2} = A_{i,j} \cup \bigcup_{k \in J} (A^+)_{i,k} . (A^+)_{k,j} = (A^+)_{i,j},$$

where the last equality follows from the definition of A^+. For $n(B) > 2$, the output property follows from the output property of $\text{Parse}_4(B)$.

Consider the output property of $\text{Parse}_3(B)$. Let $I_1 = [1, n(B)/3]$, $I_2 = [n(B)/3 + 1, 2n(B)/3]$, and $I_3 = [2n(B)/3+1, n(B)]$. By the input property (1) of B, at the beginning of the procedure the only part of B which might not satisfy the output property is $B^{I_3}_{I_1}$. This property is enforced on $B^{I_3}_{I_1}$ at the end of Step 4 due to the output property of $\text{Parse}_2(C)$. The output property of $\text{Parse}_4(B)$ can be shown analogously: Here the part of B that needs to fixed is $B^{I_4}_{I_1}$, with $I_1 = [1, n(B)/4]$ and $I_4 = [3n(B)/4 + 1, n(B)]$. This is done in Step 4 due to the output property of $\text{Parse}_2(C)$. ∎

It remains to argue that all (explicit) matrix products performed by the scored parser can be implemented using $(\min, +)$ products of BD matrices. Recall that in the scored parser the only explicit matrix products that we perform are of type $B^J_I . B^K_J$ in procedures $\text{Parse}_k(B)$, $k \in \{3, 4\}$. Recall that in order to implement a product $B^J_I . B^K_J$ we consider each production rule $Z \to XY$ (the number of such rules is constant), we derive integer matrices $B^J_I(X)$ and $B^K_J(Y)$, and then we compute the $(\min, +)$ product $B^J_I(X) \star B^K_J(Y)$. In the next corollary we prove that each such product involves two BD matrices. Therefore we can perform it in time $O(m^\alpha)$ for some $2 \leq \alpha < 3$ using our faster algorithm for BD $(\min, +)$ product. It follows from the previous discussion that the overall running time of our scored parsers is $\tilde{O}(n^\alpha)$ (or $O(n^\alpha)$ if $\alpha > 2$).

LEMMA 18. *If the scored grammar G is W-BD, then the products in Step 2 of $Parse_k$, $k \in \{3, 4\}$, involve W-BD submatrices.*

Proof. Consider first $\text{Parse}_3(B)$. Recall that we perform the product $B^{I_2}_{I_1} . B^{I_3}_{I_2}$, where $I_1 = [1, n(B)/3]$, $I_2 = [n(B)/3+1, 2n(B)/3]$ and $I_3 = [2n(B)/3+1, n(B)]$. By Lemma 16, B is contiguous or has a discontinuity

at $n(B)/3$ or $2n(B)/3$. Thus each such I_j forms a contiguous set of indices w.r.t. the input matrix A. By Lemma 17 (input property), the submatrices $B_{I_1}^{I_2}$ and $B_{I_2}^{I_3}$ are equal to the corresponding contiguous submatrices of A^+. Since the scored grammar is W-BD, and since $(A^+)_{i,j}$ evaluated at non-terminal X equals the score $s(X, \sigma_i \dots \sigma_{j-1})$, the matrix $A^+(X)$ is W-BD for any $X \in N$. It follows that also $B_{I_1}^{I_2}(X)$ and $B_{I_2}^{I_3}(X)$ are W-BD for any $X \in N$.

The proof in the case of $\text{Parse}_4(B)$ is analogous. Recall that we perform the products $B_{I_1}^{I_2} \,.\, B_{I_2}^{I_4}$ and $B_{I_1}^{I_3} \,.\, B_{I_3}^{I_4}$, where $I_1 = [1, n(B)/4]$, $I_2 = [n(B)/4 + 1, 2n(B)/4]$, $I_3 = [2n(B)/4 + 1, 3n(B)/4]$, and $I_4 = [3n(B)/4 + 1, n(B)]$. By Lemma 16, B is contiguous or has a discontinuity at $n(B)/2$. Thus each such I_j forms a contiguous set of indices w.r.t. the input matrix A. By Lemma 17 (input property), the submatrices $B_{I_1}^{I_2}$, $B_{I_1}^{I_3}$, $B_{I_2}^{I_4}$, and $B_{I_3}^{I_4}$ are equal to contiguous submatrices of A^+. Since $A^+(X)$ is W-BD for any $X \in N$, also $B_{I_1}^{I_2}(X)$, $B_{I_1}^{I_3}(X)$, $B_{I_2}^{I_4}(X)$, and $B_{I_3}^{I_4}(X)$ are W-BD for any $X \in N$. ∎

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

In this section, we prove Theorem 2 by improving on the running time from Section 2.

4.1. Speeding Up Phase 2.

We begin with a more refined version of Lemma 13. Recall that ρ is the maximum number of iterations in Phase 2.

LEMMA 19. *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 as a function of r the number of edges $m_{r,k}$ drops by a constant factor after $O(\alpha_{r,k}^{-3} \log(n))$ rounds w.h.p., as long as the density is large enough.

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

Now fix a round r. For $r \le r'$, we call $\{i, j\}$ *r'-heavy* if $C_{r',k}(i, j) \ge 2^{-8}\alpha_{r,k}^3 n^2$. 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 $C_{r',k}(i, j)/n^2 = \Omega(\alpha_{r,k}^3)$ we choose $i^{r'}, j^{r'}$ as the remaining vertices in one of the 4-cycles containing $\{i, j\}$. In this case, Lemma 11 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 12 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 $\alpha_{r^*,k} \le \max\{\alpha_{r,k}/2, 2/\sqrt{n}\}$. 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 $m_{r+1,k}$ 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 19 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 19 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 8) we obtain the following improved running time.

LEMMA 20. *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 8, 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 $M(n, x, n) \le O((1 + x/y)M(n, y, n))$, by

splitting columns and rows of length x into $\lceil x/y \rceil \leq 1+x/y$ blocks. Hence, we can bound the running time by $\tilde{O}\big(\sum_{r=1}^{\rho} \Delta W \cdot (1+s_r\rho^{1/3}/n) \cdot M(n, n/\rho^{1/3}, n)\big)$. 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 $\sum_{r=1}^{\rho} s_r \leq \tilde{O}(n\rho^{2/3})$ holds 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 19. 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 $s_r \leq n$ allows us 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 $\sum_{r=1}^{\rho} \mathbb{E}[s_r] \leq \tilde{O}(n \sum_{r=1}^{\rho} r^{-1/3}) \leq \tilde{O}(n\rho^{2/3})$.

For concentration, fix r^* as any power of two and consider $s_{r^*} + s_{r^*+1} + \ldots + s_{2r^*-1}$. For any $r^* \leq r < 2r^*$ denote by $\bar{s}_r$ the number of triples i^r, k, j^r that are weakly relevant and weakly r^*-uncovered, and note that $s_r \leq \bar{s}_r$. Again we have $\mathbb{E}[\bar{s}_r] \leq \tilde{O}(n/r^{1/3})$. Moreover, conditioned on the choices up to round r^*, the numbers $\bar{s}_r$, $r^* \leq r < 2r^*$, are independent. Hence, a Chernoff bound (Lemma 21 below) on variables $\bar{s}_r/n \in [0,1]$ shows that w.h.p.

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

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

LEMMA 21. *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 $c \geq 1$ we have*

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

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

4.2. Speeding Up Phase 3.

Enumerating approximately uncovered blocks. In line 17 of Algorithm 1 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 $U^{k'}_{xr} := [|A^r_{x\Delta,k'}| \leq 44\Delta W]$ 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 of Algorithm 1, 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.

4.3. 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

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

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 [20]. Specifically, we set $\delta := 0.0772$ and $s := 0.4863$ to obtain a bound of $O(n^{2.8244})$, which proves part of Theorem 2. 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 [20] 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.

4.4. 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 19, 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 19 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 19 (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 19 still holds, after replacing "weakly" by "approximately" relevant and uncovered: *For any $1 \leq r \leq \rho$ 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 $\tilde{O}(\rho\Delta W n^\omega + \rho(n/\Delta)^{1+\omega} + n^3/\rho^{1/3} + n^{2.5})$. 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 $\tilde{O}(n^{3-(5+\omega-\omega^2)/(4\omega+8)} W^{(\omega+1)/(4\omega+8)}) \leq O(n^{2.9004} W^{0.1929})$, using the current bound of $\omega \leq 2.3728639$ [21]. In particular, the algorithm has truly sub-cubic running time whenever $W \leq O(n^{2-\omega+3/(\omega+1)-\varepsilon}) \approx O(n^{0.5165-\varepsilon})$ for any $\varepsilon > 0$.

For $W = O(1)$, adapting the improved algorithm from Section 4.3 yields

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

which is $O(n^{2.8603})$ for $\delta := 0.2463$ and $s := 0.3159$, finishing the proof of Theorem 2. Here we use the bounds $M(m, m^{1-s/3}, m) \leq M(m, m^{0.90}, m) \leq O(m^{2.298048})$ and $M(m, m^{s/(1-\delta)}, m) \leq M(m, m^{0.45}, m) \leq O(m^{2.027102})$ by Le Gall [20] for $m = n$ and $m = n^{1-\delta}$, respectively.

4.5. Generalizations. In this section we study generalizations of Theorem 2. 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 22. *Let B be arbitrary and assume either of the following:*

(1) for an appropriately chosen $1 \leq \Delta \leq n$ we are given a partitioning $[n] = I_1 \cup \ldots \cup I_{n/\Delta}$ such that $\max_{i \in I_\ell} A_{i,k} - \min_{i \in I_\ell} A_{i,k} \leq \Delta W$ for all k, ℓ, or
(2) for an appropriately chosen $1 \leq \Delta \leq n$ we are given a partitioning $[n] = K_1 \cup \ldots \cup K_{n/\Delta}$ such that $\max_{k \in K_\ell} A_{i,k} - \min_{k \in K_\ell} A_{i,k} \leq \Delta W$ for all i, ℓ.

If $W \leq O(n^{3-\omega-\varepsilon})$, 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 ($|A_{i+1,k} - A_{i,k}| \leq W$ for all i, k) or only along the rows ($|A_{i,k+1} - A_{i,k}| \leq W$ for all i, k). In these cases the assumption is indeed satisfied, since we can choose each I_ℓ or K_ℓ as a contiguous 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 9 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., $|A_{i^*k} + B_{kj} - \tilde{C}_{i^*j}| \leq 8\Delta W$ 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 15 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 $\tilde{O}(n^{3-(3-\omega)/8}) \leq O(n^{2.9217})$. The latter bound can be slightly improved by incorporating the improvements from Section 4, 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 $1 \leq \Delta \leq n$ we are given a partitioning $[n] = K_1 \cup \ldots \cup K_{n/\Delta}$ such that $\max_{k \in K_\ell} A_{i,k} - \min_{k \in K_\ell} A_{i,k} \leq \Delta W$ for all i, ℓ and $\max_{k \in K_\ell} B_{kj} - \min_{k \in K_\ell} B_{kj} \leq \Delta W$ 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 9 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., $|A_{ik^*} + B_{k^*j} - \tilde{C}_{ij}| \leq 8\Delta W$ 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 15 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 $A_{ik} + (v(\ell, j) + 2\Delta W) \geq A_{ik^*} + B_{k^*j} + \Delta W > C_{ij}$, 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 $\max_{k \in K_\ell} B_{kj} - \min_{k \in K_\ell} B_{kj} \leq 2\Delta W$ 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. ∎

5. Applications. We show that LED, RNA-folding, and OSG can be cast as a scored parsing problems on BD grammars. To apply Theorem 5 we also have to make sure that the grammars are in CNF. To relax the latter condition, we first show that it suffices to obtain grammars that are "almost CNF", as is made precise in the following section.

Recall that a scored grammar G is W-BD if for any non-terminal X, terminal x, and string of terminals $\sigma \neq \varepsilon$ the following holds:

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

5.1. From Almost-CNF to CNF.

Definition 23. *We call a (scored) grammar G* almost-CNF *if every production is of the form*

- $X \to YZ$ *for non-terminals* X, Y, Z,
- $X \to c$ *for a non-terminal* X *and a terminal* c,
- $X \to \varepsilon$ *for a non-terminal* X, *or*
- $X \to Y$ *for non-terminals* X, Y.

That is, we relax CNF by allowing (1) ε-productions for all non-terminals, (2) unit productions $X \to Y$, and (3) the starting symbol to appear on the right-hand-side.

We show that any scored grammar that is almost-CNF can be transformed into a scored grammar in CNF, keeping BD properties. Hence, for our applications it suffices to design almost-CNF grammars.

Lemma 24. *Let G be a scored grammar G that is almost-CNF. In time $O(\text{poly}(|G|))$ we can compute a scored grammar G' in CNF generating the same scored language as G, i.e., for the start symbols S, S' of G, G', respectively, and any string of terminals σ we have $s_G(S,\sigma) = s_{G'}(S',\sigma)$. Moreover, if G is W-BD then G' is also W-BD.*

The remainder of this section is devoted to the proof of this lemma. We follow the standard conversion of context-free grammars into CNF, but we can skip some steps since G is already almost-CNF. In this conversion, whenever we add a new production $X \to \alpha$ with score s, if there already exists the production $X \to \alpha$ with score s', then we only keep the production with lowest cost $\min\{s, s'\}$. We denote the set of non-terminals of G by N and the set of productions by P. The size $|G|$ is equal to $|N| + |P|$ up to constant factors.

Eliminating ε-Productions. We eliminate productions of the form $X \to \varepsilon$ as follows.

(Step 1) For any non-terminal X from which we can derive the empty string ε, let s be the lowest score of any derivation $X \to^* \varepsilon$. We add the production $X \to \varepsilon$ with score s.

(Step 2) For any production p of the form $X \to YZ$ or $X \to ZY$ where $Y \to \varepsilon$ is a production in the current grammar and $X \neq Z$, add a new production $X \to Z$ with a score of $s(X \to Z) = s(p) + s(Y \to \varepsilon)$.

(Step 3) Delete all productions of the form $X \to \varepsilon$.

Note that this does not change the set of non-terminals[12]. Call the resulting grammar G_1. We claim that any non-terminal generates the same scored language in G and G_1, except that we delete the empty string from this language. In particular, the BD property is not affected, as it ignores the empty string. To prove the claim, consider any derivation $X \to^* \sigma$ in G, where $\sigma \neq \varepsilon$ is a string of terminals. Consider any non-terminal Y that appears in the derivation and generates the empty string ε, such that Y was derived from a production p of the form $A \to BY \mid YB$. Then we can replace the use of p by the newly added production $A \to B$, while not increasing the score. Iterating this eventually yields a derivation not using any ε-production, i.e., a derivation in G_1. For the other direction, by construction we can replace any newly added production in G_1 by a derivation in G with the same score.

Efficient Implementation. Steps 2 and 3 clearly run in linear time. To efficiently implement Step 1, we use a Bellman-Ford-like algorithm. For each non-terminal X initialize s_X as the score of the production $X \to \varepsilon$, or as ∞, if no such production exists. At the end of the algorithm, s_X will hold the minimal cost of any derivation $X \to^* \varepsilon$, or ∞ if there is no such derivation. Repeat the following for $|N|$ rounds. For each production of the form $X \to YZ$ with score s, set $s_X := \min\{s_X, s + s_Y + s_Z\}$. For each production of the form $X \to Y$ with score s, set $s_X := \min\{s_X, s + s_Y\}$.

[12] Some non-terminals might not appear in any productions anymore. In this case we still keep them in the set of non-terminals N.

The running time of this algorithm is clearly $O(|N| \cdot |P|) \leq O(|G|^2)$. Correctness is implied by the following claim, asserting that we can restrict our attention to derivations of depth at most $|N|$, where the depth of a derivation is to be understood as the depth of the corresponding parse tree. Observe that all such derivations are incorporated in the output of our algorithm.

Claim 1. *For any non-terminal X such that there exists a derivation $X \to^* \varepsilon$, let s be the minimal score of any such derivation. Then there exists a derivation $X \to^* \varepsilon$ of score s and depth at most $|N|$.*

Proof. Among the derivations $X \to^* \varepsilon$ of (minimal) score s, consider one of minimum length. In this derivation, the non-terminal X cannot appear anywhere (except for the first step), as any appearance of X would give rise to another derivation $X \to^* \varepsilon$, with score at most s and smaller length, which is a contradiction.

We can now argue inductively. If the first production is $X \to YZ$, then the remaining derivations $Y \to^* \varepsilon$ and $Z \to^* \varepsilon$ without loss of generality only use non-terminals in $N \setminus \{X\}$, and thus inductively they have depth at most $|N| - 1$. This yields depth at most $|N|$ for the derivation $X \to^* \varepsilon$. ∎

Eliminating the Start Symbol from the Right-Hand-Side. Let S be the start symbol of the grammar G_1 resulting from the last step. We introduce a new non-terminal S' and add the production $S' \to S$, making S' the new start symbol. This does not change the generated language and eliminates the start symbol from the right-hand-side of all productions. Moreover, since S' generates the same language as S, it inherits the BD property, so the resulting grammar has the same BD properties as G.

If the original grammar G can generate the empty string, then we add the production $S' \to \varepsilon$. Since G_1 generates the same language as G except that we delete the empty string, the resulting grammar G_2 generates exactly the same language as G. Moreover, since the BD property ignores the empty string, it is not affected by this change.

Eliminating Unit Productions. We now eliminate productions of the form $X \to Y$. Interpret any production $X \to Y$ with score s as an edge from vertex X to vertex Y with weight s, and compute all-pairs-shortest-paths on the resulting graph. Using Dijkstra, this runs in time $\tilde{O}(|N| \cdot |P|) \leq \tilde{O}(|G|^2)$. Iterate over all productions $X \to \alpha$ with α of the form YZ (for non-terminals Y, Z) or of the form c (for a terminal c). Iterate over all non-terminals W, and let s be the shortest path length from W to X. If $s < \infty$, add the production $W \to \alpha$ with score $s(W \to \alpha) = s + s(X \to \alpha)$. Finally, delete all productions of the form $X \to Y$.

It is easy to see that this procedure for eliminating unit productions runs in time $\tilde{O}(|N| \cdot |P|) \leq \tilde{O}(|G|^2)$ (note that up to the construction of G_1 we increased the sizes of N and P at most by constant factors). We claim that in the resulting grammar G', any non-terminal generates the same scored language as in G_2. Hence, BD properties are again not affected by this change. To prove the claim, consider any derivation $X \to^* \sigma$ in G_2, where $\sigma \neq \varepsilon$ is a string of terminals. In this derivation, replace any maximal sequence of unit productions followed by a non-unit production by the corresponding newly added production in G'. This yields a derivation $X \to \sigma$ in G', while not increasing the score. For the other direction, note that any newly added production in G' by construction can be replaced by productions in G_2 with the same score.

Observe that the resulting grammar G' is indeed in CNF. Thus, the above steps prove Lemma 24.

5.2. From LED and RNA-folding to Scored Parsing. We show that LED can be reduced to scored parsing on BD grammars. Recall that in LED we are given a context-free grammar G in CNF and a string σ of terminals, and we want to compute the smallest edit distance of σ to a string σ' in the language generated by G. The possible edit operations are insertions, deletions, and substitions, and all have cost 1. However, our construction also works if we only allow insertions and deletions (and no substitution).

Recall from the introduction that RNA-folding can be cast as an LED problem without substitutions, where the grammar is given by the productions $S \to SS \mid \varepsilon$ and $S \to \sigma S \sigma' \mid \sigma' S \sigma$ for any symbol $\sigma \in \Sigma$ with matching symbol $\sigma' \in \Sigma'$. Then if d is the edit distance (using only insertions and deletions) of a given string σ to this RNA grammar, then $(|\sigma| - d)/2$ is the maximum number of bases that can be paired in the corresponding RNA sequence. Therefore, RNA folding is covered by our construction for LED without substitutions.

We assume that we are given a scored grammar $G = (N, T, P, S)$ in CNF. In the following we describe how to adapt this grammar. In this procedure, whenever we add a new production $X \to \alpha$ with score s, if there already exists the production $X \to \alpha$ with score s', then we only keep the production with lowest cost $\min\{s, s'\}$. Initially, all productions in G get score 0.

Modeling Substitutions. (For the LED problem without substitutions, simply ignore this paragraph.) To model substitutions, for any production of the form $X \to c$ (for a non-terminal X and a terminal c) in the original grammar, and for each terminal $c' \in T$, we add a production $X \to c'$ with score 1. Note that this allows us to substitute any terminal at a cost of 1 in any derivation $X \to^* \sigma$. In other words, in the resulting scored grammar $\hat{G}$ the score of any string of terminals σ is the minimal number of substitutions to transform σ into a string σ' in the language generated by G. Note that $\hat{G}$ is still in CNF. This transformation increases the number of productions by at most $|N| \cdot |T|$.

Modeling Insertions. Without loss of generality we can assume that for each terminal $a \in T$, there exists a non-terminal X_a and the production $X_a \to a$ with score 0, and this is the only production with X_a on the left-hand-side (if not, we introduce a new non-terminal X_a and the corresponding production, this does not change the generated language or the fact that the grammar is in CNF). In order to model insertions, we create a new non-terminal I, and add the following productions:

$$I \to X_a I\,(\text{score} = 1) \mid I X_a\,(\text{score} = 1) \mid \varepsilon\,(\text{score} = 0), \quad \text{for every } a \in T$$
$$X \to XI\,(\text{score} = 0) \mid IX\,(\text{score} = 0), \quad \text{for every nonterminal } X \in N$$

Observe that I can generate any string of terminals, and the associated score is the length of the string. Moreover, I can be inserted at any point of a string generated by any non-terminal.

Modeling Insertions. In order to model deletions, for any non-terminal X, if there exists a production of the form $X \to c$ with terminal c, then we add the production

$$X \to \varepsilon\,(\text{score} = 1).$$

This production allows us, in any derivation where X produces a single terminal, to delete this terminal at a cost of 1.

This creates an augmented grammar G' of size polynomial in $|G|$. It has been shown in [4] that the LED problem on grammar G is equivalent to the scored parsing problem on G'. Since G' is almost-CNF by construction, we can use Lemma 24 to obtain an equivalent scored grammar G'' in CNF, again with size polynomial in $|G|$. In order to use Corollary 6 for solving the scored parsing problem on G'', it only remains to show that G' (and thus also G'') is a BD grammar.

Claim 2. *G' is a 1-BD grammar.*

Proof. Consider any non-terminal $X \in N$ and string of terminals σ, and let $s := s_{G'}(X, \sigma)$. Then for any terminal x, $X \to IX \to X_x IX \to X_x X \to xX \to^* x\sigma$ is a valid derivation of $x\sigma$ with score $s + 1$. For $X = I$ we similarly use the derivation $I \to X_x I \to xI \to^* x\sigma$.

For the other direction, consider a derivation $X \to^* x\sigma$ with total score s'. In this derivation, the first terminal x must be generated using a production of the form $Y \to x$. By replacing this production with $Y \to \varepsilon$ we obtain a derivation of the string σ, while increasing the score by at most 1. In total, we obtain $|s_{G'}(X, \sigma) - s_{G'}(X, x\sigma)| \leq 1$. The other condition $|s_{G'}(X, \sigma) - s_{G'}(X, \sigma x)| \leq 1$ can be shown symmetrically. □

Proposition 25. *LED and RNA-folding can be reduced to scored parsing problems of 1-BD grammars. The blow-up in the grammar size is polynomial, and the input string is not changed by the reduction.*

5.3. From Optimal Stack Generation to Scored Parsing. We show that OSG can be reduced to a scored parsing problem on a 3-BD grammar in almost-CNF. Recall that in OSG we are given a string σ over an alphabet Σ, and we want 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*, ending with an empty stack.

We model this problem as a scored parsing problem as follows. We have a start symbol 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". Note that this grammar is already almost-CNF.

$$
\begin{array}{llllll}
S & \to \varepsilon & \text{(score 0)} & \text{end of string} & & \\
S & \to X_c S & \text{(score 1)} & \text{push } c & \text{for any } c \in \Sigma \\
X_c & \to N_c X_c & \text{(score 0)} & \text{pre-emit } c & \text{for any } c \in \Sigma \\
X_c & \to X_{c'} X_c & \text{(score 1)} & \text{push } c' & \text{for any } c, c' \in \Sigma \\
X_c & \to \varepsilon & \text{(score 1)} & \text{pop } c, & \text{for any } c \in \Sigma \\
N_c & \to c & \text{(score 1)} & \text{emit } c & \text{for any } c \in \Sigma
\end{array}
$$

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. It is immediate that the scored parsing problem on this grammar is equivalent to OSG.

For an example, consider the string *bccab*. This string can be generated as follows, where we always resolve the leftmost non-terminal. Note that the suffix of non-terminals always corresponds to the current content of the stack.

$$
\begin{aligned}
&S \to X_b S \to N_b X_b S \to b X_b S \to b X_c X_b S \to b N_c X_c X_b S \to bc X_c X_b S \to bc N_c X_c X_b S \\
&\to bcc X_c X_b S \to bcc X_b S \to bcc X_a X_b S \to bcc N_a X_a X_b S \to bcca X_a X_b S \to bcca X_b S \\
&\to bcca N_b X_b S \to bccab X_b S \to bccab S \to bccab
\end{aligned}
$$

Bounded Differences. In order to obtain a BD grammar, we slightly change the above grammar by adding the following productions:

$$
N_c \to X_{c'} \quad \text{(score 1)} \quad \text{helper for BD} \qquad \text{for any } c, c' \in \Sigma
$$

This does not change the scored language generated by the grammar. Indeed, whenever N_c appears in a derivation starting from S, then it was produced by an application of the rule $X_c \to N_c X_c$. Using the new production $N_c \to X_{c'}$ thus results in $X_c \to N_c X_c \to X_{c'} X_c$, with total score 1. However, this derivation can be performed directly using the productions modeling push operations, with the same score. As this is the only way to use the newly added productions in any derivation starting from S, the generated language of the grammar is not changed (in fact, only the scored language generated by N_c is changed).

Call the resulting grammar G. Note that G is still almost-CNF. We show that it is also BD.

Claim 3. *G is a 5-BD grammar.*

Proof. Consider a string $\sigma \neq \varepsilon$ over Σ and a symbol $x \in \Sigma$. We have to show that for any non-terminal X of G,

$$
\big|s(X, \sigma) - s(X, \sigma x)\big| \leq 5 \qquad \text{and} \qquad \big|s(X, \sigma) - s(X, x\sigma)\big| \leq 5.
$$

Consider a derivation $X \to^* x\sigma$. At some point we produce the first terminal x, via the production $N_x \to x$. We change the derivation by instead using $N_x \to X_x \to \varepsilon$, obtaining a derivation of σ. This increases the score by 1 (as the scores of $N_x \to x$, $N_x \to X_x$, and $X_x \to \varepsilon$ are all 1). Hence, $s(X, \sigma) \leq s(X, x\sigma) + 1$. The inequality $s(X, \sigma) \leq s(X, \sigma x) + 1$ can be shown symmetrically.

For the other direction, first consider a non-terminal X in $\{S\} \cup \{X_c \mid c \in \Sigma\}$. Consider a derivation $X \to^* \sigma$. Then the adapted derivation $X \to X_x X \to N_x X_x X \to x X_x X \to xX \to^* x\sigma$ increases the score by 3 and generates $x\sigma$.

Similarly, to generate σx, note that X is always the rightmost symbol during the whole derivation, until we delete it with the rule $X \to \varepsilon$. At the point in the derivation $X \to^* \sigma$ where we delete X via $X \to \varepsilon$, instead use the derivation $X \to X_x X \to N_x X_x X \to x X_x X \to xX \to x$, to produce x at a cost of 3. Then the adapted derivation generates σx.

For non-terminals $X = N_c$ (for any $c \in \Sigma$) we argue as follows. If the first step of the derivation is $N_c \to X_{c'}$, then we can instead argue about $X_{c'}$, which we have done above. Otherwise, the derivation is $N_c \to c$, and $\sigma = c$. Then to produce xc we instead use the derivation

$$
N_c \to X_c \to X_x X_c \to N_x X_x X_c \to x X_x X_c \to x X_c \to x N_c X_c \to xc X_c \to xc,
$$

at a cost of 6, increasing the score of $N_c \to c$ by 5. The case σx is symmetric.

In all cases, the scores of σ and $x\sigma$ (or σx) differ by at most 5. ∎

Together with Lemma 24 we now obtain the following.

PROPOSITION 26. *OSG can be reduced to a scored parsing problem of a BD grammar. The size of the grammar is polynomial in $|\Sigma|$, and the input string is not changed by the reduction.*

REFERENCES

[1] A. Abboud, A. Backurs, and V. Vassilevska Williams, *If the current clique algorithms are optimal, so is Valiant's parser*, in 56th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 2015, pp. 98–117.

[2] A. Abboud, F. Grandoni, and V. Vassilevska Williams, *Subcubic equivalences between graph centrality problems, APSP and diameter*, in 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, 2015, pp. 1681–1697.

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

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

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

[6] N. Alon, Z. Galil, and O. Margalit, *On the exponent of the all pairs shortest path problem*, J. Comput. Syst. Sci., 54 (1997), pp. 255–262.

[7] A. Andoni, R. Krauthgamer, and K. Onak, *Polylogarithmic approximation for edit distance and the asymmetric query complexity*, in 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, Las Vegas, Nevada, USA, 2010, pp. 377–386.

[8] A. Andoni and K. Onak, *Approximating edit distance in near-linear time*, in 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, 2009, pp. 199–204.

[9] A. Backurs and K. Onak, *Fast algorithms for parsing sequences of parentheses with few errors*, in 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, 2016, pp. 477–488.

[10] Z. Bar-Yossef, T. S. Jayram, R. Krauthgamer, and R. Kumar, *Approximating edit distance efficiently*, in 45th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2004, Rome, Italy, 2004, pp. 550–559.

[11] T. Batu, F. Ergün, and S. C. Sahinalp, *Oblivious string embeddings and edit distance approximations*, in 17th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, 2006, pp. 792–801.

[12] A. Bernstein and D. R. Karger, *A nearly optimal oracle for avoiding failed vertices and edges*, in 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, 2009, pp. 101–110.

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

[14] A. Chakrabarti, G. Cormode, R. Kondapally, and A. McGregor, *Information cost tradeoffs for augmented index and streaming language recognition*, in 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, Las Vegas, Nevada, USA, 2010, pp. 387–396.

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

[16] T. M. Chan and M. Lewenstein, *Clustered integer 3SUM via additive combinatorics*, in 47th Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, 2015, pp. 31–40.

[17] Y. Chang, *Hardness of RNA folding problem with four symbols*, in 27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, Tel Aviv, Israel, 2016, pp. 13:1–13:12.

[18] M. J. Fischer and A. R. Meyer, *Boolean matrix multiplication and transitive closure*, in 12th Annual Symposium on Switching and Automata Theory, East Lansing, Michigan, USA, 1971, pp. 129–131.

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

[20] F. L. Gall, *Faster algorithms for rectangular matrix multiplication*, in 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, 2012, pp. 514–523.

[21] F. L. Gall, *Powers of tensors and fast matrix multiplication*, in 39th International Symposium on Symbolic and Algebraic Computation, ISSAC 2014, Kobe, Japan, 2014, pp. 296–303.

[22] F. Grandoni and V. 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, 2012, pp. 748–757.

[23] R. Gutell, J. Cannone, Z. Shang, Y. Du, and M. Serra, *A story: unpaired adenosine bases in ribosomal RNAs.*, in Journal of Molecular Biology, vol. 304, 2010, pp. 335–354.

[24] M. Johnson, *PCFGs, topic models, adaptor grammars and learning topical collocations and the structure of proper names*, in 48th Annual Meeting of the Association for Computational Linguistics, ACL 2010, Uppsala, Sweden, 2010, pp. 1148–1157.

[25] F. Korn, B. Saha, D. Srivastava, and S. Ying, *On repairing structural problems in semi-structured data*, Proceedings of the VLDB Endowment, 6 (2013), pp. 601–612.

[26] A. Krebs, N. Limaye, and S. Srinivasan, *Streaming algorithms for recognizing nearly well-parenthesized expressions*, in 36th International Symposium on Mathematical Foundations of Computer Science, MFCS 2011, Warsaw, Poland, 2011, pp. 412–423.

[27] G. M. Landau, E. W. Myers, and J. P. Schmidt, *Incremental string comparison*, SIAM J. Comput., 27 (1998), pp. 557–582.

[28] L. Lee, *Fast context-free grammar parsing requires fast boolean matrix multiplication*, J. ACM, 49 (2002), pp. 1–15.

[29] F. Magniez, C. Mathieu, and A. Nayak, *Recognizing well-parenthesized expressions in the streaming model*, in 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 2010, pp. 261–270.
[30] W. J. Masek and M. S. Paterson, *A faster algorithm computing string edit distances*, Journal of Computer and System Sciences, 20 (1980), pp. 18–31.
[31] D. Moore and I. Essa, *Recognizing multitasked activities from video using stochastic context-free grammar*, in 18th National Conference on Artificial Intelligence, NCAI 2002, Edmonton, Alberta, Canada, 2002, pp. 770–776.
[32] G. Myers, *Approximately matching context-free languages*, Information Processing Letters, 54 (1995), pp. 85–92.
[33] R. Nussinov and A. 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 (1980), pp. 6309–6313.
[34] M. Parnas, D. Ron, and R. Rubinfeld, *Testing membership in parenthesis languages*, Random Struct. Algorithms, 22 (2003), pp. 98–138.
[35] G. K. Pullum and G. Gazdar, *Natural languages and context-free languages*, Linguistics and Philosophy, 4 (1982), pp. 471–504.
[36] S. Rajasekaran and M. Nicolae, *An error correcting parser for context free grammars that takes less than cubic time*, Manuscript, (2014).
[37] A. Rosani, N. Conci, and F. G. De Natale, *Human behavior recognition using a context-free grammar*, Journal of Electronic Imaging, 23 (2014).
[38] B. Saha, *The Dyck language edit distance problem in near-linear time*, in 55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, 2014, pp. 611–620.
[39] B. Saha, *Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems*, in 56th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 2015, pp. 118–135.
[40] R. Seidel, *On the all-pairs-shortest-path problem in unweighted undirected graphs*, J. Comput. Syst. Sci., 51 (1995), pp. 400–403.
[41] A. Shoshan and U. Zwick, *All pairs shortest paths in undirected graphs with integer weights*, in 40th Annual IEEE Symposium on Foundations of Computer Science, FOCS 1999, New York, NY, USA, 1999, pp. 605–615.
[42] T. Takaoka, *Subcubic cost algorithms for the all pairs shortest path problem*, Algorithmica, 20 (1998), pp. 309–318.
[43] R. E. Tarjan, *Problems in data structures and algorithms*, in Graph Theory, Combinatorics and Algorithms, Springer, 2005, pp. 17–39.
[44] L. G. Valiant, *General context-free recognition in less than cubic time*, J. Comput. Syst. Sci., 10 (1975), pp. 308–315.
[45] V. Vassilevska Williams, *Multiplying matrices faster than Coppersmith-Winograd*, in 44th Annual ACM Symposium on Theory of Computing, STOC 2012, New York, NY, USA, 2012, pp. 887–898.
[46] V. Vassilevska Williams and R. Williams, *Subcubic equivalences between path, matrix and triangle problems*, in 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, Las Vegas, Nevada, USA, 2010, pp. 645–654.
[47] B. Venkatachalam, D. Gusfield, and Y. Frid, *Faster algorithms for RNA-folding using the four-russians method*, Algorithms for Molecular Biology, 9 (2014), pp. 1–12.
[48] Y.-Y. Wang, M. Mahajan, and X. Huang, *A unified context-free grammar and n-gram model for spoken language processing*, in IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2000, Istanbul, Turkey, 2000, pp. 1639–1642.
[49] R. Williams, *Faster all-pairs shortest paths via circuit complexity*, in 46th Annual ACM Symposium on Theory of Computing, STOC 2014, New York, NY, USA, 2014, pp. 664–673.
[50] R. Yuster, *Efficient algorithms on sets of permutations, dominance, and real-weighted APSP*, in 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, 2009, pp. 950–957.
[51] S. Zakov, D. Tsur, and M. Ziv-Ukelson, *Reducing the worst case running times of a family of RNA and CFG problems, using Valiant's approach*, Algorithms for Molecular Biology, 6 (2011), pp. 1–22.
[52] U. Zwick, *All pairs shortest paths using bridging sets and rectangular matrix multiplication*, J. ACM, 49 (2002), pp. 289–317.