2016 IEEE 57th Annual Symposium on Foundations of Computer Science

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§
*Max Planck Institute for Informatics, Saarbrücken, Germany, Email: kbringma@mpi-inf.mpg.de
†IDSIA, University of Lugano, Switzerland, Email: fabrizio@idsia.ch
‡University of Massachusetts Amherst, College of Information and Computer Science, Email: barna@cs.umass.edu
§Stanford University, California, Email: virgi@cs.stanford.edu

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

***Keywords*-min-plus matrix multiplication; bounded differences; language edit distance; RNA folding; truly sub-cubic algorithm; fast matrix multiplication**

I. 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 [1] 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. [2], [3], [4]) and vertex centrality measures (e.g. [5], [6]).

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. [7], [8], [9]), a relatively recent breakthrough of Williams [10] gave an $O(n^3/c^{\sqrt{\log n}})$ time algorithm for a constant $c > 1$. Despite this striking improvement, the known running times for computing the $(\min,+)$-product are 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 [11], where $\omega < 2.373$ is the matrix multiplication exponent [12], [13];
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 $\tilde{O}(Dn^{(3+\omega)/2})$ [14].[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 [15], [16] and directed [17] 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 [14].

A. 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

This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing. FG was partially supported by the ERC StG project NEWNET no. 279352 and the SNSF project APPROX-NET no. 200021_159697/1. BS was partially supported by a NSF CCF 1464310 grant, a Yahoo ACE Award and a Google Faculty Research Award. VVW was partially supported by NSF Grants CCF-1417238, CCF-1528078 and CCF-1514339, and BSF Grant BSF:2012338.

[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.

0272-5428/16 $31.00 © 2016 IEEE
DOI 10.1109/FOCS.2016.48

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 [18].

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

Indeed, our algorithm produces a truly sub-cubic running time for W-BD matrices for 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 2: Let B be arbitrary and assume either of the following:

i) for all $i, j \in [n]$, $|A_{i,j} - A_{i+1,j}| \leq W$, or
ii) for all $i, j \in [n]$, $|A_{i,j} - A_{i,j+1}| \leq W$.

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

The main obstacle towards achieving a truly sub-cubic algorithm for 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 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 $M = O(n^{1-\varepsilon})$.

B. 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 [15]. 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^r_A (resp., V^r_B) to each column of A (resp., row of B). Here V^r_A and V^r_B are random vectors derived from the estimate $\tilde{C}$. Let A^r and B^r be the resulting matrices. Using (1) we can compute $C^r = A^r \star B^r$ in truly sub-cubic time $O(\Delta W n^{\omega})$ for sufficiently small W and Δ. The perturbation is such that it is possible to derive from $(C^r)_{i,j}$ the corresponding value $(A \star B)_{i,j} = A_{i,k} + B_{k,j}$ *unless* one of the entries $A^r_{i,k}$ or $B^r_{k,j}$ was *rounded to* ∞.

The crux of our analysis is to show that (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

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

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.

C. 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 [19] 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., [20]). Here each production rule p in G has an associated 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 [21]). In particular, applying Williams' algorithm for the $(\min, +)$-product [10], 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 2: A CFG G is a W-bounded differences (W-BD) grammar if, for any non-terminal X, terminal x, and string σ of terminals, the following holds:

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

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

Via a simple but very careful analysis of the scored version of 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 (the proof is deferred to the full version).

Theorem 3: 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 $\tilde{O}(n^\alpha)$ time.

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

BD grammars appear naturally in relevant applications. Consider for example the well-studied Language Edit Distance problem (LED) [20], [22], [23], [24], [21], [25], [26]. 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 [20], 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 1 (to appear in the full version). 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

[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.

for Dyck LED and understanding the parsing problem for the parenthesis grammar has recently received considerable attention [27], [24], [28], [29], [30], [31]. 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 [32]. 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 $S \to \sigma S \sigma' \mid S \to \sigma' S \sigma$ 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. [33], [34], [35], [32]), no truly sub-cubic algorithm for RNA-folding was known prior to our work. By essentially the same argument as for LED, it is easy to obtain a BD scored grammar modeling RNA-folding. Thus we immediately obtain a truly sub-cubic algorithm to solve this problem via Corollary 1.

As a final application, consider the Optimum Stack Generation problem (OSG) described by Tarjan in [36]. Here, we are given a finite alphabet Σ, a stack S, and a string $\sigma \in \Sigma^*$. We would like to print σ by a minimum length sequence of three stack operations: *push()*, *emit* (i.e., print the top character in the stack), and *pop*. For example, the string $BCCAB$ can be printed via the 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 the full version, we show that OSG can be reduced to scored parsing on BD grammars. This leads to the first truly sub-cubic algorithm for OSG.

Let us summarize the mentioned applications of our approach.

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

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 [18], 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 2 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.

D. Related Work

Language Edit Distance: LED is among the most fundamental and best studied problems related to strings and grammars [20], [22], [23], [24], [21], [25], [26]. It generalizes two basic problems in computer science: parsing and string edit distance computation. In 1972, Aho and Peterson presented a dynamic programming algorithm for LED that runs in $O(|G|^2 n^3)$ time [20], which was improved to $O(|G| n^3)$ by Myers in 1985 [22]. These algorithms are based on the popular CYK parsing algorithm [37] with the observation that LED can be reduced to a scored parsing problem [20]. This implies the previous best running time of $O(n^3/2^{\Theta(\sqrt{\log n})})$. In a recent paper [21], 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 [23], [25], 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 [21] also showed that a truly sub-cubic exact algorithm is unlikely due to a reduction from APSP [3]. 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. [38], [39], [40], [41], [42], [43], [44]). 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 [32], 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 [33], [34], [35], and obtaining a truly sub-cubic algorithm for RNA-folding has remained open till this work.

Abboud et al. [25] 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 an alphabet Σ of size 13, which was recently improved to $|\Sigma| = 2$ [45].

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 [46], [47], [48], [49], [50], [51]. When approximation is allowed, a near-linear time $O(\text{poly}\log n)$-approximation algorithm was developed by Saha [24]. Moreover, a $(1+\epsilon)$-approximation in $O(n^\omega)$ time was shown in [21] for any constant $\epsilon > 0$. Abboud et al. [25] 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.

E. Preliminaries and Notation

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

As is typical, we denote by $\omega < 2.3729$ [12], [13] 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 $M(a,b,c)$ [52]. As in (1) above we have the following:

Lemma 1 ([11]): 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 $\tilde{O}(Mn^\omega)$.

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

Organization: In the remainder of this extended abstract we give a full proof of our main technical result, a truly sub-cubic algorithm for the $(\min,+)$-product of BD matrices. In the full version, we show how bounded difference scored parsing can be solved asymptotically in the same time as computing a single BD $(\min,+)$-product, and we present reductions from LED, RNA-folding, and OSG to scored parsing on BD grammars. Also in the full version we show how to further reduce the running time of our BD $(\min,+)$-product algorithm, and how to derandomize and generalize it, thus proving Theorems 1 and 2.

II. 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 the full version of this paper. 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).

A. Phase 1: Computing an approximation

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

The first step of our algorithm is to compute an entry-wise additive $O(\Delta W)$-approximation $\tilde{C}$ of $A \star B$. Since A and B 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 2: For any i', k', j' divisible by Δ and any $(i,k,j) \in I(i') \times I(k') \times I(j')$ we have

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

and Jensen's inequality, we have

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

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

By the same convexity argument as above, we also have

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

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

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

■

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

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

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

In each of the ρ rounds of our algorithm we select i^r and j^r uniformly at random. Round r covers some triples (i,k,j). For any such weakly r-covered triple (i,k,j), if (i,j) is in G_k, we remove it from G_k. Thus, after round r, G_k contains (i,j) if and only if (i,k,j) is weakly relevant and 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 3, (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 $1/\text{poly}(n)$ 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 4, $C_k \geq (m_k/n)^4/32$. Thus, we have $(m_k/n)^4 < 32 m_k z$, which is equivalent to $m_k^3 < 32n^4 z$. By choice of z we obtain $m_k < \left(32c(n^6/\rho)\ln n\right)^{1/3}$, which yields $m_k \leq \tilde{O}(n^2/\rho^{1/3})$.

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

C. 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 $\tilde{C}$. We are allowed to scan all weakly relevant, weakly uncovered triples, as we know that their number is small by Lemma 5. This is the only phase that requires that A and B are BD.

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

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

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

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

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

Proof: Let $(i,k,j) \in I(i') \times I(k') \times I(j')$. Using Lemma 2, we can bound the absolute difference between $A_{i,k} + B_{k,j} - C_{i,j}$ and $A_{i',k'} + B_{k',j'} - \tilde{C}_{i',j'}$ by the three contributions $|A_{i,k} - A_{i',k'}| \leq 2\Delta W$, $|B_{k,j} - B_{k',j'}| \leq 2\Delta W$, and $|C_{i,j} - \tilde{C}_{i',j'}| = |C_{i,j} - \tilde{C}_{i,j}| \leq 4\Delta W$. Thus,

if $A_{i,k} + B_{k,j} = C_{i,j}$ (i.e., (i,k,j) is strongly relevant), then $|A_{i',k'} + B_{k',j'} - \tilde{C}_{i',j'}| \leq 8\Delta W$ (i.e., (i,k,j) is approximately relevant). On the other hand, if (i,k,j) is approximately relevant, then $|A_{i,k} + B_{k,j} - C_{i,j}| \leq 16\Delta W$ (i.e., (i,k,j) is weakly relevant).

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

In our algorithm, we enumerate every triple (i',k',j') whose 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 6 this is a superset of all strongly relevant, strongly uncovered triples. Thus, every strongly relevant triple (i,k,j) contributes to $\hat{C}_{i,j}$ in Phase 2 or Phase 3. This proves correctness of the output matrix $\hat{C}$.

D. Running Time

The running time of Phase 1 is $O((n/\Delta)^3 + n^2)$ using brute-force. The running time of Phase 2 is $\tilde{O}(\rho\Delta W n^\omega)$, since there are ρ invocations of Lemma 1 on matrices whose finite entries have absolute value $O(\Delta W)$. It remains to consider Phase 3. Enumerating all blocks i',k',j' and checking whether they are approximately relevant and approximately uncovered takes time $O((n/\Delta)^3\rho)$. The approximately relevant and approximately uncovered triples form a subset of the weakly relevant and weakly uncovered triples by Lemma 6. The number of the latter triples is upper bounded by $\tilde{O}(n^{2.5} + n^3/\rho^{1/3})$ w.h.p. by Lemma 5. 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 $W \leq O(n^{3-\omega-\varepsilon})$. For $W = O(1)$ using $\omega \leq 2.3729$ [12], [13] this running time evaluates to $O(n^{2.8825})$. Note that even with bad random choices the running time of our algorithm is bounded by $O(n^3)$. In particular, this implies that our w.h.p. time bounds also hold in expectation.

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 $A \star B$*
1: **for** any i', j' divisible by Δ **do**
2: $\quad \tilde{C}_{i',j'} := \min\{A_{i',k'} + B_{k',j'} \mid k' \text{ divisible by } \Delta\}$
3: $\quad$ **for** any $i \in I(i')$, $j \in I(j')$ **do**
4: $\quad\quad \tilde{C}_{i,j} := \tilde{C}_{i',j'}$

▷ *Phase 2: randomized reduction to $(\min,+)$-product with small entries*
5: initialize all entries of $\hat{C}$ with ∞
6: **for** $1 \leq r \leq \rho$ **do**
7: $\quad$ pick i^r and j^r uniformly at random from $[n]$
8: $\quad$ **for** all i, k **do**
9: $\quad\quad$ set $A^r_{i,k} := A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r}$
10: $\quad\quad$ if $A^r_{i,k} \notin [-48\Delta W, 48\Delta W]$ then set $A^r_{i,k} := \infty$
11: $\quad$ **for** all k, j **do**
12: $\quad\quad$ set $B^r_{k,j} := B_{k,j} - B_{k,j^r} + \tilde{C}_{i^r,j^r} - \tilde{C}_{i^r,j}$
13: $\quad\quad$ if $B^r_{k,j} \notin [-48\Delta W, 48\Delta W]$ then set $B^r_{k,j} := \infty$
14: $\quad$ compute $C^r := A^r \star B^r$ using Lemma 1
15: $\quad$ **for** all i, j **do**
16: $\quad\quad \hat{C}_{i,j} := \min\{\hat{C}_{i,j}, C^r_{i,j} + \tilde{C}_{i,j^r} - \tilde{C}_{i^r,j^r} + \tilde{C}_{i^r,j}\}$

▷ *Phase 3: exhaustive search over all relevant uncovered triples of indices*
17: **for** all i', k', j' divisible by Δ **do**
18: $\quad$ **if** $|A_{i',k'} + B_{k',j'} - \tilde{C}_{i',j'}| \leq 8\Delta W$ **then**
19: $\quad\quad$ **if** for all r we have $|A^r_{i',k'}| > 44\Delta W$ or $|B^r_{k',j'}| > 44\Delta W$ **then**
20: $\quad\quad\quad$ **for** all $i \in I(i'), k \in I(k'), j \in I(j')$ **do**
21: $\quad\quad\quad\quad \hat{C}_{i,j} := \min\{\hat{C}_{i,j}, A_{i,k} + B_{k,j}\}$
22: **return** $\hat{C}$

Acknowledgments: The authors are indebted to Uri Zwick for his insightful ideas during the early stages of this work, and would like to thank the anonymous reviewers for their useful comments.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[6] 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.

[7] M. L. Fredman, "New bounds on the complexity of the shortest path problem," *SIAM Journal on Computing*, vol. 5, no. 1, 1976.

[8] T. Takaoka, "Subcubic cost algorithms for the all pairs shortest path problem," *Algorithmica*, vol. 20, no. 3, pp. 309–318, 1998.

[9] T. M. Chan, "More algorithms for all-pairs shortest paths in weighted graphs," *SIAM J. Comput.*, vol. 39, no. 5, pp. 2075–2089, 2010.

[10] 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.

[11] N. Alon, Z. Galil, and O. Margalit, "On the exponent of the all pairs shortest path problem," *J. Comput. Syst. Sci.*, vol. 54, no. 2, pp. 255–262, Apr. 1997.

[12] 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.

[13] 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.

[14] 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.

[15] R. Seidel, "On the all-pairs-shortest-path problem in unweighted undirected graphs," *J. Comput. Syst. Sci.*, vol. 51, no. 3, pp. 400–403, 1995.

[16] 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.

[17] U. Zwick, "All pairs shortest paths using bridging sets and rectangular matrix multiplication," *J. ACM*, vol. 49, no. 3, pp. 289–317, 2002.

[18] 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.

[19] L. G. Valiant, "General context-free recognition in less than cubic time," *J. Comput. Syst. Sci.*, vol. 10, no. 2, pp. 308–315, 1975.

[20] A. V. Aho and T. G. Peterson, "A minimum distance error-correcting parser for context-free languages," *SIAM J. Comput.*, vol. 1, no. 4, 1972.

[21] 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.

[22] G. Myers, "Approximately matching context-free languages," *Information Processing Letters*, vol. 54, no. 2, pp. 85–92, 1995.

[23] L. Lee, "Fast context-free grammar parsing requires fast boolean matrix multiplication," *J. ACM*, vol. 49, no. 1, pp. 1–15, 2002.

[24] 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.

[25] 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.

[26] S. Rajasekaran and M. Nicolae, "An error correcting parser for context free grammars that takes less than cubic time," *Manuscript*, 2014.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] M. Parnas, D. Ron, and R. Rubinfeld, "Testing membership in parenthesis languages," *Random Struct. Algorithms*, vol. 22, no. 1, pp. 98–138, 2003.

[32] 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*, vol. 77, no. 11, pp. 6309–6313, 1980.

[33] B. Venkatachalam, D. Gusfield, and Y. Frid, "Faster algorithms for RNA-folding using the four-russians method," *Algorithms for Molecular Biology*, vol. 9, no. 1, pp. 1–12, 2014.

[34] T. Akutsu, "Approximation and exact algorithms for RNA secondary structure prediction and recognition of stochastic context-free languages," *Journal of Combinatorial Optimization*, vol. 3, no. 2-3, pp. 321–336, 1999.

[35] 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*, vol. 6, no. 1, pp. 1–22, 2011.

[36] R. E. Tarjan, "Problems in data structures and algorithms," in *Graph Theory, Combinatorics and Algorithms*. Springer, 2005, pp. 17–39.

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

[38] F. Korn, B. Saha, D. Srivastava, and S. Ying, "On repairing structural problems in semi-structured data," *Proceedings of the VLDB Endowment*, vol. 6, no. 9, pp. 601–612, 2013.

[39] 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.

[40] 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.

[41] 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.

[42] A. Rosani, N. Conci, and F. G. De Natale, "Human behavior recognition using a context-free grammar," *Journal of Electronic Imaging*, vol. 23, no. 3, 2014.

[43] G. K. Pullum and G. Gazdar, "Natural languages and context-free languages," *Linguistics and Philosophy*, vol. 4, no. 4, pp. 471–504, 1982.

[44] 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, no. 3, 2010, pp. 335–354.

[45] 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.

[46] W. J. Masek and M. S. Paterson, "A faster algorithm computing string edit distances," *Journal of Computer and System Sciences*, vol. 20, no. 1, pp. 18–31, 1980.

[47] G. M. Landau, E. W. Myers, and J. P. Schmidt, "Incremental string comparison," *SIAM J. Comput.*, vol. 27, no. 2, pp. 557–582, 1998.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.