2015 IEEE 56th Annual Symposium on Foundations of Computer Science

Language Edit Distance & Maximum Likelihood Parsing of Stochastic Grammars: Faster Algorithms & Connection to Fundamental Graph Problems

Barna Saha
College of Information and Computer Science
University of Massachusetts Amherst
Amherst, MA 01003, USA
barna@cs.umass.edu

Abstract

Given a context free language G over alphabet Σ and a string $s \in \Sigma^*$, *the language edit distance* problem seeks the minimum number of edits (insertions, deletions and substitutions) required to convert s into a valid member of the language $\mathcal{L}(\mathcal{G})$. The well-known dynamic programming algorithm solves this problem in $O(n^3)$ time (ignoring grammar size) where n is the string length [Aho, Peterson 1972, Myers 1985]. Despite its numerous applications, to date there exists no algorithm that computes the exact or approximate language edit distance problem in true subcubic time.

In this paper we give the first such algorithm that computes language edit distance almost optimally. For any arbitrary $\epsilon > 0$, our algorithm runs in $\tilde{O}(\frac{n^\omega}{poly(\epsilon)})$ time and returns an estimate within a multiplicative approximation factor of $(1+\epsilon)$ with high probability, where ω is the exponent of ordinary matrix multiplication of n dimensional square matrices. It also computes the actual edits required. We further solve the local alignment problem; for all substrings of s, we can estimate their language edit distance within $(1 \pm \epsilon)$ factor in $\tilde{O}(\frac{n^\omega}{poly(\epsilon)})$ time with high probability. Next, we design the very first subcubic ($\tilde{O}(n^\omega)$) algorithm that given an arbitrary *stochastic context free grammar*, and a string returns a nearly-optimal maximum likelihood parsing of that string. Stochastic context free grammars significantly generalize hidden Markov models; they lie at the foundation of statistical natural language processing, and have found widespread applications in many other fields.

To complement our upper bound result, we show that exact computation of maximum likelihood parsing of stochastic grammars or language edit distance in true subcubic time will imply a truly subcubic algorithm for all-pairs shortest paths, a long-standing open question. This will result in a breakthrough for a large range of problems in graphs and matrices due to subcubic equivalence. By a known lower bound result [Lee 2002], and a recent development [Abboud et al. 2015] even the much simpler problem of parsing a context free grammar requires $\Omega(n^\omega)$ time. Therefore any nontrivial multiplicative approximation algorithms for either of the two problems in time $o(n^\omega)$ are unlikely to exist.

I. Introduction

Given a model for data semantics and structures, estimating how well a dataset fits the model is a core question in large-scale data analysis. Formal languages (e.g., regular language, context free language) provide a generic technique for data modeling. The deviation from a model is measured by the least changes required in data to perfectly fit the model. Aho and Peterson studied this basic question more than forty years back to design error-correcting parsers for programming languages. Given a context free grammar (CFG) G over alphabet Σ and a string $s \in \Sigma^*$, they proposed *the language edit distance* problem, which determines the fewest number of edits (insertions, deletions and substitutions) along with an edit script to convert s into a valid member of $\mathcal{L}(G)$. Due to its fundamental nature, the language edit distance problem has found many applications in compiler optimization (error-correcting parser [4], [30]), data mining and data management (anomaly detection, mining and repairing data quality problems [16], [22], [33]), computational biology (biological structure prediction [18], [43]), machine learning (learning topic and behavioral models [20], [29], [35]), and signal processing (video and speech analysis [44]).

Often it is natural to consider a probabilistic generative model, and ask for the most probable derivation/explanation of the observed data according to the model. The stochastic context free grammar (SCFG) model is an example of that. It associates a conditional probability to each production rule of a CFG (see definition in Section I-A) which reflects the likelihood of applying it to generate members of the language. SCFGs significantly generalize hidden Markov models, and several stochastic processes such as Latent Dirichlet Allocation [20] and Galton-Watson branching process [28]. They lie at the foundation of statistical natural language processing, and have found wide-spread applications as a rich framework for modeling complex phenomena [14], [19], [40], [50]. The most probable

0272-5428/15 $31.00 © 2015 IEEE
DOI 10.1109/FOCS.2015.17

derivation of a string according to a stochastic context free grammar is known as the *maximum likelihood parsing of SCFG*, or simply as *SCFG parsing*.

The well-known Cocke-Younger-Kasami (CYK) algorithm for context free grammar parsing can be easily modified to solve the SCFG parsing problem in $O(|G|n^3)$ time, where $|G|$ is the grammar size and n is the length of the input sequence. After half a century since the proposal of CYK algorithm, there still does not exist a "true" subcubic algorithm for SCFG parsing that runs in $O(n^{3-\gamma})$ time in the string length for some $\gamma > 0$. For ordinary CFG parsing, Valiant provided a truly subcubic $O(|G|^2 n^\omega)$ algorithm [42]. For the language edit distance problem, the proposed algorithm by Aho and Peterson has a running time of $O(|G|^2 n^3)$ [4]. The dependency on grammar size in run time was later improved by Myers to $O(|G|n^3)$ [30]. Naturally, the cubic time-complexity on string length is a major bottleneck for most applications. Except minor polylogarithmic improvements over $O(n^3)$ [34], [43], [49], to date true subcubic algorithms for the language edit distance problem, or SCFG parsing have not been found.

In this paper, we make several contributions.

Upper Bound.

1) *We give the first true subcubic algorithm to approximate language edit distance for arbitrary context free languages almost optimally.* Our algorithm runs in $\tilde{O}(|G|^2 \frac{n^\omega}{\mathrm{poly}(\epsilon)})$[1] time and computes the language edit distance within a multiplicative approximation factor of $(1+\epsilon)$ for any $\epsilon > 0$, where ω is the exponent of ordinary matrix multiplication over $(\times,+)$-ring [25] (Section II). Therefore, if d is the optimum distance, the computed distance is in $[d, d(1+\epsilon)]$. The best known bound for $\omega < 2.373$ [46]. In fact the running time can be further improved to $\tilde{O}\left(\frac{1}{\mathrm{poly}(\epsilon)}\left(|G|n^\omega + \omega(n|G|, n, n|G|) + |G|^2 n^2\right)\right)$ where $\omega(n|G|, n, n|G|)$ is the running time of fast rectangular matrix multiplication of a $n|G| \times n$ matrix with a $n \times n|G|$ matrix. The algorithm also computes an edit script.

The above result is obtained by casting the problem as an *algebraic closure computation* problem [3]. All-pairs shortest path problem, and many other variants of path problems on graphs can be viewed as computing closure over an algebra which is a semiring. *However, for the language edit distance computation, the underlying algebra is not a semiring; the corresponding "multiplication" operation is neither commutative, nor associative.* This poses significant difficulties, and requires new techniques. Being more generic, our approximation algorithm can be employed to obtain alternate approximation schemes for a large variety of problems such as all-pairs shortest paths, minimum weight triangle/cycle detection, computing diameter, radius and many others in $\tilde{O}(m^\omega \log W)$ time over m-node graphs where W is the maximum absolute weight of any edge.

We note that by a lower bound result of Lee [26], it is known that a faster context free grammar parsing (distinguishing between 0 and nonzero edit distance) leads to a faster algorithm for boolean matrix multiplication. Therefore, obtaining any multiplicative approximation factor for language edit distance in $o(n^\omega)$ time is unlikely.

Our algorithm also solves the *local alignment problem* in $\tilde{O}(|G|^2 \frac{n^\omega}{\mathrm{poly}(\epsilon)})$ time where given $\mathcal{L}(G)$ and $\sigma \in \Sigma^*$, for all substrings σ' of σ, we need to compute their language edit distance (Section II). Such local alignment problems are studied extensively under string edit distance computation for approximate pattern matching, for the simpler problem of context free parsing etc. [2], [17].

2) *We design the first subcubic algorithm for SCFG parsing near-optimally* (Section III). Given a SCFG, which is a pair of CFG and a probability assignment on each production, $(G, \mathbf{p})$, and a string s, $|s| = n$, we give an $\tilde{O}(|G|^2 n^\omega \log\log \frac{1}{p_{min}} \frac{1}{\epsilon^4})$ algorithm to compute a parse $\pi'(s)$ such that

$$|\log \Pr[\pi'(s)]| \geqslant (1-\epsilon)|\log \Pr[\pi(s)]|$$

where $\pi(s)$ is the most likely parse of s with the highest probability $\Pr[\pi(s)]$, and p_{min} is the minimum probability of any production. To the best of our knowledge, prior to our work, no true subcubic algorithm for arbitrary SCFG parsing was known.

Lower Bound. In a pursuit to explain the difficulty in obtaining exact subcubic algorithms, *we show that a subcubic algorithm for SCFG parsing or computing language edit distance (with insertion and substitution) will culminate in*

[1] $\tilde{O}$ as standard notational practice includes $\mathrm{polylog}$ factors.

a breakthrough result for several fundamental graph problems. This will lead to a subcubic algorithm for all-pairs shortest path problem (APSP), and many others. Specifically we show

Theorem 1. *Given a stochastic context-free grammar* $(G, \mathbf{p})$*, and a string* $s \in \Sigma^*$*,* $|s| = n$*, if the SCFG parsing problem can be solved in* $O(|G|n^{3-\delta} \max \log_{p \in \mathbf{p}} \frac{1}{p}))$ *time then that implies an algorithm with running time* $\tilde{O}(m^{3-\delta/3} \log W)$ *for APSP on* m*-node digraphs with maximum edge weight* W*.*

Theorem 2. *Given a context-free grammar* G*, and a string* $s \in \Sigma^*$*,* $|s| = n$*, if the language edit distance problem with only insertion and substitution as edits can be solved in* $O(|G|n^{3-\delta})$ *time then that implies an* $\tilde{O}(m^{3-\delta/3} \log W)$ *algorithm for APSP on* m*-node digraphs with maximum edge weight* W*.*

Our lower bound results build upon a construction given by Lee [26] who showed a faster algorithm for CFG parsing implies a faster algorithm for boolean matrix multiplication. For SCFG parsing, by suitably modifying his construction, we show a subcubic SCFG parser implies a subcubic $(\min, \times)$ matrix product computation [2]. Next, we prove a subcubic algorithm for $(\min, \times)$ matrix product leads to a subcubic algorithm for negative triangle detection in weighted graphs. The negative triangle detection is one of the many problems known to be subcubic equivalent with the all-pairs shortest path problem [47]. Our second reduction interestingly also shows computing $(\min, +)$ product of matrices with real weights bounded by $\log n$ in subcubic time is unlikely to exist. In contrast, $(\min, +)$ product of matrices with integer weights bounded by $\log n$ can be done fast in $O(n^\omega \log n)$ time. Our lower bound result for language edit distance also uses similar construction and builds upon it to additionally handle edit distance. Note that the later reduction only works when insertion and substitution are allowed as edits. However, it is extremely unlikely that computing language edit distance becomes easier when insertion, substitution and deletion are allowed instead of only insertion and substitution.

By subcubic equivalence [1], [47], a subcubic algorithm for language edit distance or SCFG parsing implies a subcubic algorithm for a large number of graph problems, e.g. detecting *minimum weight triangle*, *minimum weight cycle*, checking *metricity*, finding *second shortest path*, *replacement path*, *radius problem*.

Language edit distance computation is much harder than language recognition (or parsing). A beautiful result by Valiant first broke the barrier of cubic running time for context free recognition, and provided an $\tilde{O}(|G|^2 n^\omega)$ algorithm [42]. Our result surprisingly indicates that parsing time is enough to compute a near-optimal result for the harder language edit distance problem. In our prior work on Dyck language edit distance we obtained a polylogarithmic approximation guarantee in near parsing time (linear time for Dyck language) [37]. The Dyck language edit distance generalizes string edit distance, for which obtaining a better than polylog approximation in near-linear time is an outstanding open question [7]. Understanding this relation between parsing time, and language edit distance remains a big challenge.

Our current algorithm due to its algebraic nature is not practical. Obtaining a combinatorial subcubic algorithm remains a major open problem. The dependency on grammar size of our algorithm is quadratic. Even for parsing, no subcubic algorithm with sub-quadratic dependency on grammar size is known till date. Improving the dependency on grammar size and $\frac{1}{\epsilon}$ to linear will be important. On the other hand, the lower bound proofs require linear dependency on grammar size. Can we strengthen the result to consider super-linear dependency on grammar size?

Grammars are a versatile method to encode problem structures. Lower bounds with grammar based distance computation may shed light into deriving unconditional lower bounds for polynomial time solvable problems, for which our understandings are still lacking.

Related Works.: The language edit distance problem is a significant generalization of the widely-studied *string edit distance problem* where two strings need to be matched with minimum number of edits. The string edit distance problem can be exactly computed in quadratic time. Despite many efforts, a sub-quadratic exact algorithm for string edit distance does not exist. A recent result by Backurs and Indyk explains this difficulty by showing a subquadratic algorithm for string edit distance implies the strongly exponential time hypothesis (SETH) is false [9]. Our lower bound results are in a similar spirit which connects subcubic algorithm for language edit distance, and SCFG parsing to graph problems for which obtaining exact subcubic algorithms are long-standing open questions. Recently after this work, new lower bound results connecting Valiant's CFG parsing algorithm to detecting cliques in graphs have

[2] $(\min, \times)$ matrix product C of two matrices A and B of suitable dimensions is defined as $C[i,j] = \min_k A[i,k] \times B[k,j]$.

been shown [1]. For approximate string edit distance computation, there is a series of works that tried to lower the running time to near-linear [7], [8], [10], [11], [13], [24], [39].

Language recognition and parsing problems have been studied for variety of languages under different models for decades, including streaming model, sub-linear query access model etc. [6], [12], [23], [27], [32]. The early works of $O(n^3)$ time algorithm for parsing context free grammars (such as the Cocke-Younger-Kasami algorithm (CYK or CKY algorihtm)) was improved by an elegant work of Valiant [42]. However, for stochastic grammar parsing and language edit distance computation, till date there does not exist any true subcubic algorithm, except for minor polylogarithmic improvements in the running time [34], [43], [49].

A. Preliminaries

Grammars & Derivations. A context-free grammar (grammar for short) is a 4-tuple $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ where $\mathcal{N}$ and Σ are finite disjoint collection of nonterminals and terminals respectively. $\mathcal{P}$ is the set of productions of the form $A \to \alpha$ where $A \in \mathcal{N}$ and $\alpha \in (\mathcal{N} \cup \Sigma)^*$. S is a distinguished *start* symbol in $\mathcal{N}$.

For two strings $\alpha, \beta \in (\mathcal{N} \cup \Sigma)^*$, we say α directly derives β, written as $\alpha \Rightarrow \beta$, if one can write $\alpha = \alpha_1 A \alpha_2$ and $\beta = \alpha_1 \gamma \alpha_2$ such that $A \to \gamma \in \mathcal{P}$. Thus, β is a result of applying the production $A \to \gamma$ to α.

$\mathcal{L}(G)$ is the context-free language generated by grammar G, i.e., $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} w\}$, where $\stackrel{*}{\Rightarrow}$ implies that w can be derived from S using one or more production rules.

Chomsky Normal Form (CNF). We consider the CNF representation of G. This implies every production is either of type (i) $A \to BC$, $A, B, C \in \mathcal{N}$, or (ii) $A \to x$, $x \in \Sigma$ or (iii) $S \to \varepsilon$ if $\varepsilon \in \mathcal{L}(G)$. It is well-known that every context-free grammar has a CNF representation. CNF representation is popularly used in many algorithms, including CYK and Earley's algorithm for CFG parsing [21]. Prior works on cubic algorithms for language edit distance computation use CNF representation as well [4], [30]

Definition 1 (Language Edit Distance)**.** *Given a grammar $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ and $s \in \Sigma^*$, the language edit distance between G and s is defined as $d_G(G, s) = \min_{z:z\in\mathcal{L}(G)} d_{ed}(s, z)$ where d_{ed} is the standard edit distance (insertion, deletion and substitution) between s and z. If this minimum is attained by considering $z \in \mathcal{L}(G)$, then z serves as an witness for $d_G(G, s)$.*

We will often omit the subscript from d_G and d_{ed} and use d to represent both language and string edit distance when that is clear from the context. We assume $\mathcal{L}(G) \neq \phi$ and $\varepsilon \in \mathcal{L}(G)$ so that $d_G(G, s) \leqslant |s|$.

Definition 2. *A stochastic context free grammar (SCFG) is a pair $(G, \mathbf{p})$ where $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ is a context free grammar, and we additionally have a parameter $p(\alpha \to \beta)$ where $\alpha \in \mathcal{N}, \alpha \to \beta \in \mathcal{P}$ for every production in $\mathcal{P}$ such that (1) $p(\alpha \to \beta) > 0$ for all $\alpha \to \beta \in \mathcal{P}$, and (2) $\sum_{\alpha\to\beta\in\mathcal{P}:\alpha=X} p(\alpha \to \beta) = 1$ for all $X \in \mathcal{N}$. We can view $p(\alpha \to \beta)$ as the conditional probability of applying the rule $\alpha \to \beta$ given that the current nonterminal being expanded in a derivation is α.*

Given a string $s \in \Sigma^*$ and a parse $\pi(s)$ where π applies the productions $P_1 P_2 ... P_l$ successively to derive s, probability of $\pi(s)$ under SCFG is $\Pr[\pi(s)] \prod_{i=1}^{l} p(P_i)$.

A basic question regarding SCFG is parsing, where given a string $s \in \Sigma^*$, we want to find the most likely parse of s:"$\arg\max_{\pi(s)} \Pr[\pi(s) \mid s, (G, p)]$".

II. A Near-Optimal $\tilde{O}(|G|^2 n^\omega)$ Algorithm for Language Edit Distance

In this section, we derive the following theorem.

Theorem 3. *Given any arbitrary context-free grammar $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$, a string $s = s_1 s_2 ... s_n \in \Sigma^*$, and any $\epsilon > 0$, there exists an algorithm that runs in $\tilde{O}(\frac{1}{\epsilon^4}|G|^2 n^\omega)$ time and with probability at least $(1 - \frac{1}{n})$ returns the followings.*

- *An estimate $e(G, s)$ for $d(G, s)$ such that $d(G, s) \leqslant e(G, s) \leqslant (1 + \epsilon) d(G, s)$ along with a parsing of s within distance $e(G, s)$.*
- *An estimate $e(G, s_i^j)$ for every substring $s_i^j = s_i s_{i+1} ... s_j$ $i, j \in \{1, 2, ..., n\}$ of s such that $(1 - \epsilon) d(G, s_i^j) \leqslant e(G, s_i^j) \leqslant (1 + \epsilon) d(G, s_i^j)$*

Moreover for every substring s_i^j its parsing information can be retrieved in time $\tilde{O}(j - i)$ time.

This is the first subcubic algorithm on string length for computing language edit distance near optimally.

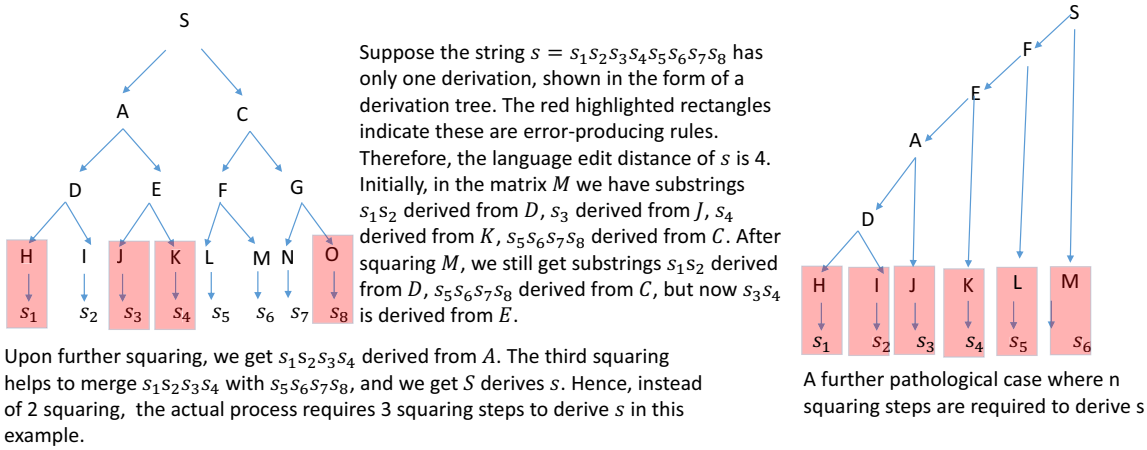

Fig. 1: Example

Overview. The very first starting point of our algorithm is an elegant work by Valiant where given a context-free grammar $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ and a string $s \in \Sigma^*$, $|s| = n$, one can test in $O(n^\omega)$ time whether $s \in \mathcal{L}(G)$ [42]. This was done by reducing the context free grammar parsing problem to a transitive closure computation where each matrix multiplication requires boolean matrix multiplication time. We extend this framework by augmenting $\mathcal{P}$ with error-producing rules (insertions, deletions, substitutions), such that if parsing s must use k such productions, $d_G(G, s) = k$. Our goal is to use fewest such rules during parsing. This trick of adding error-producing rules was first used by Aho and Peterson for their cubic dynamic programming algorithm [4]. Instead of a single bit, whether a substring parses or not, we now need to keep a "distance" count on the minimum such productions needed to parse the substring. Using the augmented grammar, we can suitably modify Valiant's approach which replaces boolean matrix multiplication with distance product computation. However, to date, the best-known bound $T(n)$ to compute distance product of n-dimensional square matrices A and B: $(A \odot B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$, is $\tilde{O}(\frac{n^3}{2^{Omega(\sqrt{\log n})}})$ by a breakthrough result of Williams [45]. Hence, this reduction does not provide any true subcubic algorithm for distance product computation.

Indeed directly using the augmented grammar by Aho and Peterson, followed by a modification of Valiant's method leads to a $O(T(n)|G|^4)$ algorithm for language edit distance computation. In a recent independent result Rajsekaran and Nicolae obtain this bound [34][3]. The dependency on grammar size blows up to at least $O(|G|^4)$. Addition of error-producing rules make the augmented grammar non-CNF; converting it back to CNF may blow up the grammar size to $O(|G|^2)$ with another factor 2 in the exponent coming from Valiant's method. Both works of [4], [42] must use CNF grammars. We overcome this difficulty by carefully dealing with the resultant non-CNF augmented grammar directly. The first step already gives an $O(|G|^2 T(n))$ time algorithm for exact computation of language edit distance.

When language edit distance is at most R, $T(n) = Rn^\omega$ by a result of Alon, Galil and Margalit [5] and Takaoka [41]. This trivially gives an $O(\frac{n}{R})$-approximation algorithm in $O(|G|^4 Rn^\omega)$ time [34], which provides no approximation guarantee in $O(n^\omega)$ time, and an $O(n^{.3729})$-approximation in $O(|G|^4 n^3)$ time with the current best bound of ω. We instead get a $(1+\epsilon)$-approximation in $O(|G|^2 n^\omega)$ time.

Suppose, we use $R = 1$ and identify all substrings that can be derived using at most one edit in $O(|G|^2 n^\omega)$ time. Let M be a matrix such that $M[i,j]$ contains information on how to derive substring s_i^j using at most one error-producing rule. We can then hope to define a suitable matrix-product such that squaring M identifies all substrings within edit distance 2 by concatenating two adjacent substrings each of which are within edit distance 1. We would then repeat this process at most $\lceil \log n \rceil$ times successively obtaining substrings with larger edit distance, e.g., $2, 2^2, 2^3, \ldots$ etc. *But, there is a fundamental difficulty in applying this approach.*

[3]The authors do not specify the dependency on grammar size, and dependency of $O(|G|^4)$ is the best possible. Their method might require even higher dependency because of more complex operations used to adapt Valiant's approach.

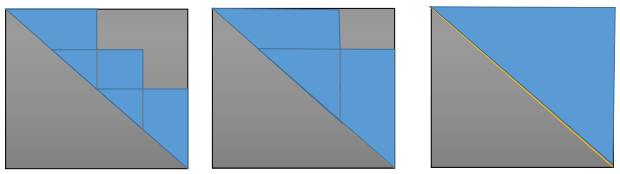

Fig. 2: Decomposition into overlapping parts. While Valiant's Method computes transitive closure in asymptotically same time as single matrix multiplication, there could be cells in the matrix that take part in $O(n)$ multiplication steps. Even if we employ approximate distance product computation for square matrices with dimension $\geqslant n^{1-\epsilon}$, the number of such multiplications could be $O(n^{\epsilon})$ all involving a single element. Finally, while our algorithm can be employed for approximate distance product computation, known approximate distance product computation algorithms neither can be applied here, nor they can be easily modified (probabilistically) maintaining concentration around true edit distance.

Consider a string $s_1 s_2 ... s_n$ where each symbol s_i needs to be substituted by t_i, and which has only one length n parsing tree. The start symbol derives t_1 and some $A \in \mathcal{N}$ which subsequently derives t_2 and another $B \in \mathcal{N}$, and so on (see Figure 1). Due to a strict derivation sequence, instead of $\lceil \log n \rceil$, n squaring steps are needed to compute the language edit distance.

This also illustrates replacing exact distance product computation by known "approximate" distance product algorithm will not be sufficient. If we use a $(1+\epsilon)$-approximation for distance product each time when multiplying matrices in Valiant's method, then the overall approximation can be as bad as $(1+\epsilon)^n$. With the best known bound for ϵ with $\tilde{O}(n^{\omega})$ running time [51], [52], the approximation factor becomes $O(\frac{n}{\mathrm{polylog} n})$. Lack of flexibility in recursively combining substrings is a major bottleneck. This arises because the underlying structure over which matrix product is defined is nonassociative (see Fig 2).

We are able to overcome this significant difficulty by introducing several new ideas. Randomization plays a critical role. We restrict the number of possible edit distances to few distinct values, and then use randomization carefully to maintain the language edit distance for each substring on expectation and show even for a long parse tree the variance does not blow up. In contrast, any deterministic rounding may add an $(1+\epsilon)$ multiplicative error at every level of a parsing tree, leading to an $(1+\epsilon)^n$-approximation in the worst case. An old theorem by Erdös and Turán [15], and a construction by Ruzsa [36] allow to map the distinct edit distances maintained by our algorithm to a narrow range of possible scores such that each matrix product can be computed cheaply. This also leads to the first subcubic algorithm for stochastic context free grammar parsing near-optimally (Section III).

Note. Proofs that are omitted from the extended abstract due to page limitation can be found in the full version [38].

A. Algorithm

1) Error Producing Productions & Assigning Scores.: The first step of our algorithm is to introduce new error-producing rules in $\mathcal{P}(G)$, and assign scores to them. We refer the augmented grammar as G_e.

Substitution Rule. If there exists rule $A \to x_1|x_2|...|x_l$, $x_i \in \Sigma, i = 1,2,...l$ for some $l \geqslant 1$, we (implicitly) add an error-producing rule $A \to y_1|y_2|...|y_{l'}$, $\{y_1, y_2, ..., y_{l'}\} = \Sigma \setminus \{x_1, x_2, ..., x_l\}$ with a score of 1. Explicitly, we maintain $A \to \$x_1 x_2 ... x_l$, where $\$$ is a special symbol not in Σ. This rule is not in CNF form, and only used to check if $A \to y_i$ is a substitution rule, maintaining the grammar size within a constant factor.

Insertion Rule. For each $x \in \Sigma$, we add an error-producing rule $I \to x$ which corresponds to a single insertion error with a score of 1. We also introduce for each nonterminal $A \in \mathcal{N}$, new rules $A \to IA$, $A \to AI$ to allow a single insertion right before and after a substring derivable from A. Further, we add a rule $I \to II$ to combine two insertion errors. These later rules have score 0.

Deletion Rule. For each $A \to x$, $x \in \Sigma$, we add a rule $A \to \varepsilon$ with a score of 1. Inclusion of these rules violate the CNF property, as in a CNF grammar only $S \to \varepsilon$ is allowed.

If a parsing of a string s requires applying productions $\pi = P_1, P_2, ..., P_n$ respectively then the parse has a score $\mathrm{score}(\pi) = \sum_{i=1}^{n} \mathrm{score}(P_i)$. The following lemma essentially follows from Aho and Peterson [4].

Lemma 1. *Given any string* $s \in \Sigma^*$*, there exists a nonterminal* $A \in \mathcal{N}(G)$ *such that* $A \stackrel{*}{\Rightarrow} s$ *with a parse of score* l *in* G_e *if and only if there exists a string* $s' \in \Sigma^*$ *such that* $A \stackrel{*}{\Rightarrow} s'$ *in* G *(and* G_e *with a parse of score* 0*) and* $d(s,s') = l$*. Thus if* l *is the minimum score for deriving* $A \stackrel{*}{\Rightarrow} s$ *in* G_e *and if* $A = S$*, then* $d(G,s) = l$*.*

2) Parsing with At most R *Errors.:* The next step of our algorithm is to compute a $(n+1) \times (n+1)$ upper triangular matrix M such that its (i,j)th entry contains all nonterminals that can derive the substring $s_i^{j-1} = s_i s_{i+1} ... s_{j-1}$ using at most $R \geqslant 1$ error-producing rules in $\tilde{O}(|G|^2 R n^\omega)$ time.

Definitions. We define a binary vector operation between (A,u) and (B,v) where $A,B \in \mathcal{N}$ and $u,v \in [0,1,2,...,n]$ as $(A,u) *_r (B,v) = (C,x)$ if $C \to AB \in \mathcal{P}$ and $x = u+v < r$, otherwise $(A,u) *_r (B,v) = \phi$. *Note that this operation is not associative.* We omit r from $*_r$ if it is clear from the context.

We next define Elem-Mult. Given $T_1 = \{(A_1,u_1),...,(A_k,u_k)\}$ and $T_2 = \{(B_1,v_1),...,(B_l,v_l)\}$ for some $k,l \in \mathbb{N} \cup \{0\}$, define $T = T_1.T_2 = \bigcup^{\min}_{\forall i=1,2,..,k, \text{ and} j=1,2,...,l} (A_i,u_i) * (B_j,v_j)$, where $\bigcup^{\min}$ implies if we have $(C,x_1),(C,x_2),....,(C,x_i)$ involving the same nonterminal C, we only keep $(C, \min\{x_1,x_2,..,x_i\})$.

We define matrix multiplication (Matrix-Mult) $a.b = c$ for suitable dimensions in terms of the above where matrix elements are collection of tuples $(A \in \mathcal{N}, u \in \{0,1,2,...,n\})$. $c(i,k) = \bigcup^{min}_{j=1}{}^{n} a(i,j).b(j,k)$.

The *transitive closure* of a square matrix is defined as $a^+ = a_1 \cup a_2 \cup ...$, where $a_1 = a$ and $a_i = \bigcup^{min}_{j=1}{}^{i-1} a_j.a_{i-j}$. It is typical to compute transitive closure a^+ from a by repeated squaring of a, so to compute a^n requires $\lceil \log n \rceil$ squaring. This property does not hold here, because the "." operation is non-associative.

3) Algorithm: Our algorithm has two steps. *First*, we generate sets $D_1, D_2, ..., D_n$ such that $(A,u) \in D_n$ if and only if $A \Rightarrow \varepsilon$ with a score of u, $u \leqslant n$. D_1 contains all nonterminals that derive ε with a score of at most 1. D_n is obtained recursively: $D_i = D_{i-1} \cup D_{i-1}.D_{i-1}$ in $O(n|\mathcal{P}(G_e)|)$ time (see full version).

Next, we compute a $(n+1) \times (n+1)$ upper triangular matrix M such that its (i,j)th entry contains all nonterminals that can derive the substring $s_i^{j-1} = s_i s_{i+1} ... s_{j-1}$ using at most R edit operations, along with their respective *minimum* scores for deriving s_i^{j-1}. We start with defining a matrix b^R:

$$b^R(i,i+1) = \{(A_k, \text{score}(A_k \to s_i)) \mid A_k \to s_i\} \text{ and } b^R(i,j) = \phi \quad \text{for } j \neq i+1.$$

If we allow only insertion and substitution error, then computing transitive closure M^R of b^R using Matrix-Mult as the matrix multiplication operation (with $*_R$) returns all the local alignment scores. To handle deletion errors, we need to multiply (Elem-Mult) D_n to each $b^R_{i,j}$ repeatedly up to $|G_e|$ times. For example, if $B \to s_i$ with a score of x_1, $A_0 \to \varepsilon$ with a score of y_1, and $C_0 \to A_0 B$, then $b^R(i,i+1) \cup D_n.b^R(i,i+1) \cup b^R(i,i+1).D_n$ will contain C_0 with score $x_1 + y_1$. Suppose instead, we have $C_0 \to A_0 C_1$ and $C_1 \to A_1 B$ with $A_1 \Rightarrow \varepsilon$. Since, we are interested in the minimum score associated with any nonterminal, we only care if $C_1 \neq C_0$. Then after two repeated multiplications by D_n, $b^R(i,i+1)$ contains both C_0 and C_1 with appropriate scores. We show repeating this operation $|G_e|$ times after every Matrix-Mult while computing transitive closure returns the desired M^R matrix (full version). We use $\text{TransitiveClosure}^+(b^R, R)$ to denote this augmented procedure.

Lemma 2. $(A_k,u) \in M^R(i,j)$*, if and only if* $A_k \stackrel{*}{\Rightarrow} s_i^{j-1}$ *in* G_e *with a parse score of* $u \leqslant R$*, and there does not exist any* $u' < u$ *such that* $A_k \stackrel{*}{\Rightarrow} s_i^{j-1}$ *with parse score of* u'*.*

Running Time. We now calculate the time needed to compute M. The analysis goes in two steps.

- Step 1. Show that time needed to compute M^R ($\text{TransitiveClosure}^+(b^R, R)$) is asymptotically same as the time needed to compute a single Matrix-Mult with $*_R$.
- Step 2. Show that time needed to compute a single Matrix-Mult is $O(\min\{R|G|^2 n^\omega, |G|^2 T(n)\})$, where $T(n)$ is the time required to compute distance product of two n dimensional square matrices.

The first step follows from Valiant's algorithm. Each multiplication by D_n requires checking each production and see if one of its two RHS nonterminal appears in the matrix entry and the other in D_n. Over $|G_e|$ iterations, it takes $O(|G_e|^2 m^2)$ time when considering a square submatrix of dimension $m \times m$, $m \leqslant n$, and is dominated by corresponding Matrix-Mult time. Initializing b^R may need to handle non-CNF substitution rules, but can still be handled in $O(|G_e|^2 n^2)$ time. Valiant gave an elegant method of *decomposing into overlapping subparts* to handle non-associativity. It is possible for some entries in the matrix to be multiplied $O(n)$ times, but overall a transitive closure is as cheap to compute as a single matrix multiplication.

To reduce $\mathtt{Matrix\text{-}Mult}(a,b)$ to distance product, we create two real matrices a' and b' of dimension $(n|G_e|,n)$ and $(n,n|G_e|)$ respectively. Assuming nonterminals are numbered $1,2,...,|G_e|$, the entry (A_k,u) in $a(i,j)$ is represented by setting $a'((i-1)|G_e|+k,j)=u$, and the entry (A_l,v) in $b(j,k)$ is represented by $b'(j,(k-1)|G_e|+l)=v$. Now, it is easy to verify that from the distance product of a' and b', we can retrieve $\mathtt{Matrix\text{-}Mult}(a,b)$. For details, see the full version.

Reduction from $\mathtt{Matrix\text{-}Mult}$ *to Ordinary Matrix Multiplication.:* $\mathtt{Matrix\text{-}Mult}$ can be done much faster in $O(|G|^2|R|n^\omega)$ time when we obtain parsing information for all substrings with score of at most R. This is precisely because distance product computation when matrix entries are all integers in $[-R,R]$ can be computed fast.

We use the reduction of distance product computation to ordinary matrix multiplication by Alon, Galil and Margalit [5] and Takaoka [41]. Given a and b, we first create the matrices a' and b' as above of dimension $nh\times n$ and $n\times nh$ respectively. Then if $a'(i,k)=x$, we set $a'(i,k)=(nh+1)^{M-x}$ and if $b(k,j)=y$, we set $b'(k,j)=(nh+1)^{M-y}$. We now calculate ordinary matrix product of a' and b' to obtain c' in time $O(h^2n^\omega)$. In fact the time taken is $O(R\omega(nh,n,nh))$ which represents the time to multiply two rectangular matrices of dimensions $nh\times n$ and $n\times nh$ respectively.

If $c'(i,j)=z$, then we set $c'(i,j)=2R-\lceil\log_{(nh+1)}z\rceil$. After that, we retrieve c from c' as before.

We maintain a list of all productions. To obtain $c(i,j)$, we go through this list and for each production $B\to A_xA_y$ we check appropriate cells for A_x and A_y to obtain the score for B. This takes at most $O(R)$ time. Hence $c(i,j)$ can be constructed in time $O(Rh)$ time. Therefore, total time to compute c' is $O(R\omega(nh,n,nh))$ and time to compute c from c' is $O(Rhn^2)$. That the reduction is correct follows directly from the correctness proof of reducing distance product to ordinary matrix product computation [5], [41], and the way the matrices are computed. The details are simple and left to the reader.

Therefore a single $\mathtt{Matrix\text{-}Mult}$ operation can be done in time $O(R\omega(nh,n,nh))$. Together this proves that parsing strings with score at most R (and the complete parsing information of all its substrings) can be obtained in $O(R\omega(nh,n,nh)+h^2n^2)$ time.

4) $\mathtt{Matrix\text{-}Mult}$ *with* R *Distinct Scores:* Before, we can describe the final algorithm, we need one more step. We saw when the values are bounded by R, $\mathtt{Matrix\text{-}Mult}$ can be solved in $O(|G|^2Rn^\omega+|G|^2Rn^2)$ time. We extend this to handle the case when there could be R distinct values in the matrix, but with arbitrary values.

We first give a simple construction by reducing the distance product computation with R distinct values to boolean matrix multiplication.

Reduction to boolean matrix multiplication: Given two matrices a and b each of dimension $n\times n$, we wish to compute c such that

$$c(i,j)=\bigcup_{k=1,2,..,n}^{\min} a(i,k).b(k,j)$$

There can be at most $R+1$ distinct integer entries $v_0=0<v_1<v_2<..<v_R<n$ in a and b.

To do so, we compute two boolean matrices a' of dimension $((R+1)hn,n)$ and b' of dimension $(n,(R+1)hn)$, from a and b respectively where $|h|=|\mathcal{N}(G_e)|$. Initially all the cells of a' and b' are zeros.

To construct a' from a, we reserve $(R+1)h$ consecutive rows for each row index p of a. For each $(A_i,v_x)\in a(p,q)$, $\forall p,q\in\{1,2,..,n+1\}$ we set $a'((p-1)h(R+1)+(i-1)(R+1)+(x+1),q)=1$.

To construct b' from b, we reserve $(R+1)h$ consecutive columns for each column index q of b. For each $(A_j,v_y)\in b(p,q)$, $\forall p,q\in\{1,2,..,n+1\}$ we set $b'(p,(q-1)h(R+1)+(j-1)(R+1)+(y+1))=1$.

We compute the boolean product c' of a' and b' in time $O(R^2h^2n^\omega)$.

To construct c from c', consider all the rows $(p-1)h(R+1)+1\leqslant i\leqslant ph(R+1)$, and all the columns $(q-1)h(R+1)+1\leqslant j\leqslant qh(R+1)$, if $c'(i,j)=1$ then if $i-(p-1)h(R+1)=(s-1)h+z$, $j-(q-1)h(R+1)=(t-1)h+x$, and $B\to A_sA_t\in\mathcal{P}(G_e)$, generate (B,v_z+v_w) as a candidate to include in $c(p,q)$. After generating all the candidates for each nonterminal B if $(B,u_1),(B,u_2),...,(B,u_i),u_1\leqslant u_2\leqslant ...u_i$ are all candidates–include only (B,u_1) in $c(p,q)$.

We maintain a list of all productions. To obtain $c(p,q)$, we go through this list and for each production $B\to A_sA_t$ we check if A_s and A_t are present in the appropriate cells of c'. This takes at most $O(R^2)$ time. Hence $c(p,q)$

can be constructed in time $O(R^2h)$ time. Therefore, total time to compute c' is $O(R^2h^2n^\omega)$ and time to compute c from c' is $O(R^2hn^2)$. Proving this construction is correct, is an easy exercise and left to the reader.

Thus, we can compute $a.b$ by first creating a', b', performing their boolean matrix multiplication c' and then obtaining c from it using the above relation. The total time taken is $O(n^2R^2h + n^\omega R^2h^2)$. □

We now discuss an alternate construction using which gives slightly worse dependency on $\frac{1}{\epsilon}$ in the final algorithm, but brings in a different perspective and may be useful in other contexts.

Sidon Sequences.: A Sidon sequence is a sequence $\mathcal{G} = (g_1, g_2, g_3, ...)$ of natural numbers in which all pairwise sums $g_i + g_j$, $i \leqslant j$ are different. The Hungarian mathematician Simon Sidon introduced the concept in his investigations of Fourier series in 1932. An early result by Erdös and Turán showed that the largest Sidon subset of $\{1, 2, ..., n\}$ has size $\sim \sqrt{n}$. There are several constructions known for Sidon sequences. A greedy algorithm gives a construction of size k Sidon sequence from $O(k^3)$ elements. Better constructions matching $O(k^2)$ bound are known due to Ruzsa, Bose, Singer (for comprehensive literature survey see [31]).

Our Construction.: Given R distinct values $v_1 < v_2 < v_3 < ... < v_R$ that may appear during Matrix-Mult, we create a size R Sidon sequence $\mathcal{G}_R = (g_1, g_2, ..., g_R)$, $g_1 < g_2 < ... < g_R$, and define map $f(v_i) = g_i$. By the property of Sidon sequences, given a sum $g = g_i + g_j$, we can uniquely detect g_i and g_j. We can keep a look-up table and do a binary search to find g_i and g_j given g in $O(\log R)$ time. By known construction $g_R = O(R^2)$. The property of Sidon sequence ensures that all pair-wise sums are disjoint. Therefore, if we define $f(v_i) = g_i$ by looking at the sum of $g_i + g_j$, we can identify v_i and v_j. It is possible that $v_i + v_j < v_k + v_l$ but $g_i + g_j > g_k + g_l$, and vice versa.

Our reduction of Matrix-Mult to ordinary matrix multiplication when values are bounded by R^2 uses an old construction by Alon, Galil, Margalit [5] and Takaoka [41] using which we can compute for each (i, j) all distinct $a_{i,k} + b_{k,j}$, $k = 1, 2, .., n$ in $O(R^4 n^\omega |G|^2)$ time. The construction maps $a'(i, j) = (nh + 1)^{R^2 - a_{i,j}}$ and $b'(j, k) = (nh + 1)^{R^2 - b_{j,k}}$. Hence if $c' = a' * b'$ where $*$ is ordinary matrix multiplication, then

$$c'(i, j) = \sum_{k=1}^{n} (nh + 1)^{2R^2 - (a_{i,k} + b_{k,j})}$$

We can now find all distinct sums $2R^2 - (a_{i,k} + b_{k,j})$ (and hence $a_{i,k} + b_{k,j}$), $k = 1, 2, .., n$ by repeatedly finding the largest exponent of $(nh + 1)$ contributing to $c'(i, j)$, that is by calculating $\lfloor \log_{(nh+1)} c'(i, j) \rfloor$ and setting $c'(i, j) = c'(i, j) - \left\lfloor \frac{c'(i,j)}{(nh+1)^{\lfloor \log_{(nh+1)} c'(i,j) \rfloor}} \right\rfloor (nh + 1)^{\lfloor \log_{(nh+1)} c'(i,j) \rfloor}$ for the next iteration. Since there are at most R^2 distinct sums, and operating on these large numbers require $O(R^2 \log R)$ time, the overall time required is $\tilde{O}(R^4 n^\omega)$. Therefore, from the mapping f, we only require that the sum $f(v_i) + f(v_j)$ uniquely identifies v_i and v_j. Then, from these at most R^2 distinct pairs, we can find the one with minimum sum in $O(R^2)$ time per entry. Using this alternate approach gives a dependency of $\frac{1}{\epsilon^8}$ in the final algorithm instead of $\frac{1}{\epsilon^4}$ that is obtained using reduction to boolean matrix multiplication.

We note that Yuster's construction to compute APSP in subcubic time when the number of distinct weight edges is not too many [48] may be applied here. But even for $R = O(1)$ their construction yields an $O(n^{2.5})$ algorithm (see Lemma 3.3 [48]), and hence provides much worse bound in our context.

5) Final Algorithm: Given an $\epsilon > 0$, let $\delta = \frac{\epsilon^2}{16}$. we set $R = \lceil \log_{(1+\delta)}(n) \rceil$. We start with b^C where C is some constant say 10, and compute $M^C = \mathit{TransitiveClosure}^+(b^C, C)$.

We now define a new operation $\mathit{TransitiveClosure}^{++}$ which is same as $\mathit{TransitiveClosure}^+$ except we do some further auxiliary processings before and after each Matrix-Mult (of possibly submatrices).

After every Matrix-Mult (possibly of submatrices) and also after multiplying by D_n each time, for every (A, y) appearing in the current matrix, if $y = (1 + \delta)^r + k$, where k is any real number in $(0, \delta(1 + \delta)^r)$, $r = 0, 1, 2, .., \lceil \log_{(1+\delta)}(n) \rceil$ then set

$$y = \begin{cases} (1+\delta)^r & \text{with probability } \frac{\delta(1+\delta)^r - k}{\delta(1+\delta)^r} \\ (1+\delta)^{r+1} & \text{with probability } \frac{k}{\delta(1+\delta)^r} \end{cases} \qquad \text{(Round)}$$

Clearly, the number of distinct parsing scores that can appear on any tuple (A, y) during the computation of $\mathit{TransitiveClosure}^{++}$ is bounded by $\lceil \log_{(1+\delta)} n \rceil$. Multiplying two submatrices of dimension $m \times m$ requires

time $\Omega(|G_e|^2 m^2)$, whereas this auxiliary $\mathtt{Round}$ operation can be performed in $O(|G_e|m^2)$ time. Hence overall the asymptotic time taken is not affected due to $\mathtt{Round}$.

To multiply any two submatrices of (say) dimension $m \times m$, with $R = \lceil \log_{1+\delta} n \rceil$ possible parsing scores use reduction to boolean matrix multiplication as discussed in Section II-A4. Hence, overall the time to compute $\mathtt{TransitiveClosure}^{++}(M^C, R)$ is same as $\mathtt{TransitiveClosure}^{+}(M^C, R^2)$ which is $\tilde{O}(\frac{1}{\epsilon^4}|G_e|^2 n^\omega)$.

Starting from M^C we repeat the process of computing $\mathtt{TransitiveClosure}^{++}(M^C, R)$ $\eta = 6 \log n$ times and obtain matrices $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_\eta$. For each substring s_i^{j-1} we consider the estimates given by $(S, d_k^{i:j}) \in \mathcal{M}_k(i,j)$, $k = 1, 2, ..., \eta$. Note that using the productions $S \to SI, I \to II$ and $I \to x, x \in \Sigma$, S can always derive s_i^{j-1} within edit distance $|j-i|$. Thus there always will be a tuple of the form $(S, d_k^{i:j}) \in \mathcal{M}_k(i,j)$ for all $\eta = 1, 2, .., k$ and $1 \leqslant i, j \leqslant (n+1)$. Let $d_{med}^{i:j}$ denote the median of these η estimates. We return $d_{med}^{i:j}$ as the estimated edit score for s_i^{j-1}.

The parsing information for any substring s_i^{j-1} can also be obtained in time $O(j-i)$ by small additional bookkeeping.

6) Analysis of Approximation Factor.: Consider any $\mathcal{M}_k$ and let us abuse notation and use $\mathcal{M}$ to denote it. Consider any substring $s' = s_i^{j-1}$. Given a parse tree $\mathcal{T}$, we say $\mathcal{M}$ retains $\mathcal{T}$ if for every intermediate nonterminal A of $\mathcal{T}$ deriving some substring $s_{i'}^{j'-1}$ the corresponding entry $(A, x) \in \mathcal{M}(i', j')$ contains the estimated value of parsing score of A in $\mathcal{T}$. Therefore, x is the actual estimate of parsing score of A in $\mathcal{T}$ if $\mathcal{T}$ is retained.

Let P be the parse tree corresponding to the estimate $\hat{e_P}$ returned by $\mathcal{M}[i,j]$ when e_P is the actual parsing score for it. Let O be the optimum parse tree for s' with minimum edit distance e_O. Since, each cell $\mathcal{M}(i', j')$ $1 \leqslant i', j' \leqslant (n+1)$ contains all nonterminals that can derive substring $s_{i'}^{j'-1}$, the reason we do not return O is simply because if we had retained O throughout, its estimated score $\hat{e_O} \geqslant \hat{e_P}$ [4].

We show that $\hat{e_P} \in [(1-\epsilon)e_P, (1+\epsilon)e_P]$ and similarly $\hat{e_O} \in [(1-\epsilon)e_O, (1+\epsilon)e_O]$ with high probability.

$$\text{Therefore, } e_O \leqslant e_P \leqslant (1+\epsilon)\hat{e_P} \leqslant (1+\epsilon)\hat{e_O} \leqslant (1+\epsilon)^2 e_O \approx (1+2\epsilon)e_O \qquad \text{(ApproxEstimate)}$$

Consider the parse tree P and let P' denote the truncated parse tree after the algorithm completes execution of $\mathtt{TransitiveClosure}^{+}(b^C, C)$. The leaves of P' denote the exact parsing scores corresponding to the associated nonterminals by Lemma 2 and due to exact computation of D_n. Hence, if $l_1, l_2, ..., l_m$ are the leaves of P' and $\mathtt{score}(l_i)$ is the score computed by $\mathtt{TransitiveClosure}^{+}(b^C, C)$ or D_n, then $e_P = \sum_i \mathtt{score}(l_i)$. Associate a random variable $\mathcal{X}_v$ with each node v (intermediate and leaves) of P'.

Lemma 3. $\mathsf{E}[\hat{e_P}] = e_P$.

Proof. Consider the parse tree P and let P' denote the truncated parse tree after the algorithm completes execution of $\mathtt{TransitiveClosure}^{+}(b^C, C)$. The leaves of P' denote the exact parsing scores corresponding to the associated nonterminals by Lemma 2 and due to exact computation of D_n. Hence, if $l_1, l_2, ..., l_m$ are the leaves of P' and $\mathtt{score}(l_i)$ is the score computed by $\mathtt{TransitiveClosure}^{+}(b^C, C)$ or D_n, then $e_P = \sum_i \mathtt{score}(l_i)$.

Associate a random variable $\mathcal{X}_v$ with each node v (intermediate and leaves) of P'. First consider the case when v is a leaf node of P'. Let $\mathtt{score}(v) = (1+\delta)^r + k$ for some $r \in \{0, 1, 2, ..., \lceil \log_{(1+\delta)} n \rceil\}$ and $k \in [0, \delta(1+\delta)^r)$. Then

$$\begin{aligned} \mathsf{E}[\mathcal{X}_v] &= (1+\delta)^r \frac{\delta(1+\delta)^r - k}{\delta(1+\delta)^r} + (1+\delta)^{r+1} \frac{k}{\delta(1+\delta)^r} \\ &= (1+\delta)^r - \frac{k}{\delta} + (1+\delta)\frac{k}{\delta} = (1+\delta)^r + k = \mathtt{score}(v) \end{aligned} \qquad (1)$$

Now consider the case when v is an intermediate node. Every intermediate node has two children. So let v_1 and v_2 be its two children. Let $\mathcal{T}_v, \mathcal{T}_{v_1}, \mathcal{T}_{v_2}$ be the subtrees of P' rooted at v, v_1 and v_2 respectively. Let $L(v), L(v_1), L(v_2)$ be the leaf nodes in $\mathcal{T}(v), \mathcal{T}(v_1), \mathcal{T}(v_2)$ respectively.

[4]Note that we do not retain O possibly because at some intermediate node A of O, the estimated score for A by O is higher than some other parse tree with a different estimate for A. Therefore, the estimates shown for various nodes of O in $\mathcal{M}$ is only lower than the actual estimates if indeed O was retained by $\mathcal{M}$. If $\hat{e_O}$ is the estimate shown by $\mathcal{M}$ for O at nonterminal S (root) and $\hat{e_O}$ is the estimate for O if $\mathcal{M}$ retained all the actual estimates of O then we must have $\hat{e_P} \leqslant \hat{e_O} \leqslant \hat{e_O}$

Claim 1. $\mathsf{E}[\mathcal{X}_v] = \mathsf{E}[\mathcal{X}_{v_1}] + \mathsf{E}[\mathcal{X}_{v_2}]$.

The proof is by induction. The base case for leaves is already proven. By induction hypothesis,

$$\mathsf{E}[\mathcal{X}_{v_1}] = \sum_{l \in L(V_1)} \mathrm{score}(l)$$

$$\mathsf{E}[\mathcal{X}_{v_2}] = \sum_{l \in L(V_2)} \mathrm{score}(l)$$

For given any real y, we let $\lfloor y \rfloor_\delta$ denote $(1+\delta)^{\lfloor \log_{(1+\delta)} y \rfloor}$, and $\lceil y \rceil_\delta$ denote $(1+\delta)^{\lceil \log_{(1+\delta)} y \rceil}$.
We have

$$\begin{aligned}
\mathsf{E}[\mathcal{X}_v] &= \sum_x x \mathsf{Pr}[X_v = x] \\
&= \sum_y \sum_x x \mathsf{Pr}\left[\mathcal{X}_v = x \mid \mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \\
&= \sum_{y = \lfloor y \rfloor_\delta} y \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \\
&+ \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \left(\lfloor y \rfloor_\epsilon \mathsf{Pr}\left[\mathcal{X}_v = \lfloor y \rfloor_\epsilon \mid \mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] + \lceil y \rceil_\epsilon \mathsf{Pr}\left[\mathcal{X}_v = \lceil y \rceil_\epsilon \mid \mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \right) \\
&= \sum_y y \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \quad \text{By same calculation as Eqn. 1} \\
&= \mathsf{E}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2}\right] = \mathsf{E}\left[\mathcal{X}_{v_1}\right] + \mathsf{E}\left[\mathcal{X}_{v_2}\right] \quad \text{By linearity of expectation} \\
&= \sum_{l \in L(v)} \mathrm{score}(l) \quad \text{since } L(v_1) \text{ and } L(v_2) \text{ are obviously disjoint} \quad \square
\end{aligned}$$

Therefore, we get the desired result $\mathsf{E}[\hat{e_p}] = e_p$. $\square$

Corollary 1. $\mathsf{E}[\hat{e_O}] = e_O$.

Proof. Follows using the same argument as in Lemma 3. $\square$

We now use second moment method to bound the deviation of our estimate from expectation. Variance calculation is complicated and needs to be done with care.

Lemma 4. $\mathsf{Var}[\mathcal{X}_v] \leqslant \mathsf{Var}[\mathcal{X}_{v_1}] + \mathsf{Var}[\mathcal{X}_{v_2}] + \delta\left(\sum_{g \in L(v_1)} \mathrm{score}(g)\right)\left(\sum_{h \in L(v_2)} \mathrm{score}(h)\right)$ *if v is an intermediate node. If v is a leaf then* $\mathsf{Var}[\mathcal{X}_v] = (\mathrm{score}(v) - \lfloor \mathrm{score}(v) \rfloor_\delta)(\lceil \mathrm{score}(v) \rceil_\delta - \mathrm{score}(v))$.

Proof. First consider the case when v is a leaf node. Let $\mathrm{score}(v) = (1+\delta)^r + k$ for some $r \in \{0, 1, 2, ..., \lceil \log_{(1+\delta)} n \rceil\}$ and $k \in [0, \delta(1+\delta)^r)$. Then

$$\begin{aligned}
\mathsf{E}[\mathcal{X}_v^2] &= (1+\delta)^{2r} \frac{\delta(1+\delta)^r - k}{\delta(1+\delta)^r} + (1+\delta)^{2(r+1)} \frac{k}{\delta(1+\delta)^r} \\
&= (1+\delta)^{2r} - (1+\delta)^r \frac{k}{\delta} + (1+\delta)^{(r+2)} \frac{k}{\delta} \\
&= (1+\delta)^{2r} - (1+\delta)^r \frac{k}{\delta}(-1 + 1 + \delta^2 + 2\delta) \\
&= (1+\delta)^r((1+\delta)^r + k(2+\delta))
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathsf{Var}[\mathcal{X}_v] &= \mathsf{E}\left[\mathcal{X}_v^2\right] - (\mathsf{E}[\mathcal{X}_v])^2 \\
&= (1+\delta)^r((1+\delta)^r + k(2+\delta)) - (1+\delta)^{2r} - k^2 - 2k(1+\delta)^r \\
&= k(\delta(1+\delta)^r - k) \\
&= (\mathsf{score}(v) - \lfloor \mathsf{score}(v) \rfloor_\delta)(\lceil \mathsf{score}(v) \rceil_\delta - \mathsf{score}(v)) \qquad (2)
\end{aligned}$$

Now let us consider the case when v is not a leaf node and v has two children v_1 and v_2.

$$\begin{aligned}
\mathsf{E}\left[\mathcal{X}_v^2\right] &= \sum_x x^2 \mathsf{Pr}[\mathcal{X}_v = x] \\
&= \sum_y \sum_x x^2 \mathsf{Pr}\left[\mathcal{X}_v = x \mid \mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \\
&= \sum_{y = \lfloor y \rfloor_\delta} y^2 \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] + \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \Big(\lfloor y \rfloor_\delta^2 \mathsf{Pr}\left[\mathcal{X}_v = \lfloor y \rfloor_\delta \mid \mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \\
&\qquad + \lceil y \rceil_\delta^2 \mathsf{Pr}\left[\mathcal{X}_v = \lceil y \rceil_\delta \mid \mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \Big) \\
&= \sum_{y = \lfloor y \rfloor_\delta} y^2 \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] + \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \Big(\lfloor y \rfloor_\delta^2 \frac{\lceil y \rceil_\delta - y}{\lceil y \rceil_\delta - \lfloor y \rfloor_\delta} + \lceil y \rceil_\delta^2 \frac{y - \lfloor y \rfloor_\delta}{\lceil y \rceil_\delta - \lfloor y \rfloor_\delta} \Big)
\end{aligned}$$

Now let us use the fact that when $y \neq \lfloor y \rfloor_\delta$ then $\lceil y \rceil_\delta - \lfloor y \rfloor_\delta = \delta \lfloor y \rfloor_\delta = \frac{\delta}{(1+\delta)} \lceil y \rceil_\delta$ to get

$$\begin{aligned}
&= \sum_{y = \lfloor y \rfloor_\delta} y^2 \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \\
&\quad + \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \Big(\frac{\lfloor y \rfloor_\delta \lceil y \rceil_\delta}{\delta} - \frac{\lfloor y \rfloor_\delta y}{\delta} + \frac{(1+\delta)\lceil y \rceil_\delta y}{\delta} - \frac{(1+\delta)\lceil y \rceil_\delta \lfloor y \rfloor_\delta}{\delta} \Big) \\
&= \sum_{y = \lfloor y \rfloor_\delta} y^2 \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] + \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \Big(\frac{(1+\delta)^2 - 1}{\delta} \lfloor y \rfloor_\delta y - \lfloor y \rfloor_\delta \lceil y \rceil_\delta \Big) \\
&= \sum_{y = \lfloor y \rfloor_\delta} y^2 \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] + \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \Big((2+\delta) \lfloor y \rfloor_\delta y - \lfloor y \rfloor_\delta \lceil y \rceil_\delta \Big)
\end{aligned}$$

Therefore,

$$
\begin{aligned}
&\mathsf{Var}[\mathcal{X}_v] \\
&= \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \left((2+\delta)\lfloor y \rfloor_\delta y - \lfloor y \rfloor_\delta \lceil y \rceil_\delta - y^2\right) + \sum_y y^2 \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] - (\mathsf{E}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2}\right])^2 \\
&= \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \left((2+\delta)\lfloor y \rfloor_\delta y - \lfloor y \rfloor_\delta \lceil y \rceil_\delta - y^2\right) + \mathsf{E}\left[(\mathcal{X}_{v_1} + \mathcal{X}_{v_2})^2\right] - (\mathsf{E}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2}\right])^2 \\
&= \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \left((2+\delta)\lfloor y \rfloor_\delta y - \lfloor y \rfloor_\delta \lceil y \rceil_\delta - y^2\right) + \mathsf{Var}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2}\right] \\
&= \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \left((2+\delta)\lfloor y \rfloor_\delta y - \lfloor y \rfloor_\delta \lceil y \rceil_\delta - y^2\right) + \mathsf{Var}\left[\mathcal{X}_{v_1}\right] + \mathsf{Var}\left[\mathcal{X}_{v_2}\right] \\
&\qquad\qquad \text{Since } \mathcal{X}_{v_1} \text{ and } \mathcal{X}_{v_2} \text{ are independent} \\
&= \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \left((2+\delta)\lfloor y \rfloor_\delta y - (1+\delta)\lfloor y \rfloor_\delta^2 - y^2\right) + \mathsf{Var}\left[\mathcal{X}_{v_1}\right] + \mathsf{Var}\left[\mathcal{X}_{v_2}\right] \\
&= \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] ((1+\delta)\lfloor y \rfloor_\delta - y)(y - \lfloor y \rfloor_\delta) + \mathsf{Var}\left[\mathcal{X}_{v_1}\right] + \mathsf{Var}\left[\mathcal{X}_{v_2}\right] \\
&= \Big(\sum_{y \neq \lfloor y \rfloor_\delta} (\lceil y \rceil_\delta - y)(y - \lfloor y \rfloor_\delta) \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \Big) + \mathsf{Var}\left[\mathcal{X}_{v_1}\right] + \mathsf{Var}\left[\mathcal{X}_{v_2}\right] \qquad \text{(a)}
\end{aligned}
$$

Now we use the fact that for any vertex u, $\mathcal{X}_u$ can only take values that is a power of $(1+\delta)$. That is, $\mathcal{X}_u$ can only take values from $\{1, (1+\delta), (1+\delta)^2, ..., (1+\delta)^{\lceil \log_{(1+\delta)} n \rceil}\}$.

Therefore, $\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y$ and $y \neq \lfloor y \rfloor_\delta$ can be broken in two possible ways, (i) $\mathcal{X}_{v_1} = \lfloor y \rfloor_\delta$ and $\mathcal{X}_{v_2} = y - \lfloor y \rfloor_\delta$, or (ii) $\mathcal{X}_{v_2} = \lfloor y \rfloor_\delta$ and $\mathcal{X}_{v_2} = y - \lfloor y \rfloor_\delta$. Also $\mathcal{X}_{v_1}$ and $\mathcal{X}_{v_2}$ are independent random variables.

Therefore,

$$
\begin{aligned}
&\sum_{y \neq \lfloor y \rfloor_\delta} (\lceil y \rceil_\delta - y)(y - \lfloor y \rfloor_\delta) \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \\
&= \sum_{y \neq \lfloor y \rfloor_\delta} (\lceil y \rceil_\delta - y)(y - \lfloor y \rfloor_\delta) \Big(\mathsf{Pr}\left[\mathcal{X}_{v_1} = \lfloor y \rfloor_\delta\right] \mathsf{Pr}\left[\mathcal{X}_{v_2} = y - \lfloor y \rfloor_\delta\right] + \mathsf{Pr}\left[\mathcal{X}_{v_1} = y - \lfloor y \rfloor_\delta\right] \mathsf{Pr}\left[\mathcal{X}_{v_2} = \lfloor y \rfloor_\delta\right] \Big) \\
&\leqslant \sum_{y \neq \lfloor y \rfloor_\delta} (\lceil y \rceil_\delta - \lfloor y \rfloor_\delta)(y - \lfloor y \rfloor_\delta) \Big(\mathsf{Pr}\left[\mathcal{X}_{v_1} = \lfloor y \rfloor_\delta\right] \mathsf{Pr}\left[\mathcal{X}_{v_2} = y - \lfloor y \rfloor_\delta\right] + \mathsf{Pr}\left[\mathcal{X}_{v_1} = y - \lfloor y \rfloor_\delta\right] \mathsf{Pr}\left[\mathcal{X}_{v_2} = \lfloor y \rfloor_\delta\right] \Big) \\
&= \delta \sum_{y \neq \lfloor y \rfloor_\delta} \lfloor y \rfloor_\delta (y - \lfloor y \rfloor_\delta) \Big(\mathsf{Pr}\left[\mathcal{X}_{v_1} = \lfloor y \rfloor_\delta\right] \mathsf{Pr}\left[\mathcal{X}_{v_2} = y - \lfloor y \rfloor_\delta\right] + \mathsf{Pr}\left[\mathcal{X}_{v_1} = y - \lfloor y \rfloor_\delta\right] \mathsf{Pr}\left[\mathcal{X}_{v_2} = \lfloor y \rfloor_\delta\right] \Big) \\
&\leqslant \delta \Big(\sum_{y_1} y_1 \mathsf{Pr}\left[\mathcal{X}_{v_1} = y_1\right] \Big) \Big(\sum_{y_2} y_2 \mathsf{Pr}\left[\mathcal{X}_{v_2} = y_2\right] \Big) = \delta \mathsf{E}\left[\mathcal{X}_{v_1}\right] \mathsf{E}\left[\mathcal{X}_{v_2}\right] \qquad \text{(b)}
\end{aligned}
$$

Hence, combining (a) and (b) we get

$$\mathsf{Var}[\mathcal{X}_v] \leqslant \mathsf{Var}\left[\mathcal{X}_{v_1}\right] + \mathsf{Var}\left[\mathcal{X}_{v_2}\right] + \delta \mathsf{E}\left[\mathcal{X}_{v_1}\right] \mathsf{E}\left[\mathcal{X}_{v_2}\right]$$

Now using Lemma 3 we get

$$\mathsf{Var}[\mathcal{X}_v] \leqslant \mathsf{Var}\left[\mathcal{X}_{v_1}\right] + \mathsf{Var}\left[\mathcal{X}_{v_2}\right] + \delta \Big(\sum_{g \in L(v_1)} \mathrm{score}(g) \Big) \Big(\sum_{h \in L(v_2)} \mathrm{score}(h) \Big)$$

This establishes the lemma.

□

Lemma 5. $\mathsf{Var}[\mathcal{X}_v] \leqslant \sum_{l \in L(v)} \mathsf{Var}[\mathcal{X}_l] + \delta \Big(\sum_{g,h \in L(v), g<h} \mathrm{score}(g)\mathrm{score}(h) \Big)$.

Proof. We use the tree $\mathcal{T}(v)$ rooted at v. We start from v and Lemma 4

$$\mathsf{Var}[\mathcal{X}_v] \leqslant \mathsf{Var}\left[\mathcal{X}_{v_1}\right] + \mathsf{Var}\left[\mathcal{X}_{v_2}\right] + \delta\Big(\sum_{g\in L(v_1)} \mathrm{score}(g)\Big)\Big(\sum_{h\in L(v_2)} \mathrm{score}(h)\Big)$$

and go down the tree $\mathcal{T}(v)$ successively opening up the expressions for $\mathsf{Var}\left[\mathcal{X}_{v_1}\right]$ and $\mathsf{Var}\left[\mathcal{X}_{v_2}\right]$. For every non-leaf node u, let u_1 and u_2 denote its left and right child respectively. Then we get,

$$\mathsf{Var}[\mathcal{X}_v] \leqslant \sum_{l\in L(v)} \mathsf{Var}[\mathcal{X}_l] + \delta \sum_{u\in \mathcal{T}(v)\setminus L(v)} \left\{\Big(\sum_{g\in L(u_1)} \mathrm{score}(g)\Big)\Big(\sum_{h\in L(u_2)} \mathrm{score}(h)\Big)\right\}. \tag{A}$$

We now bound the second term of the above equation.

To do so, we calculate $\mathsf{E}\left[\left(\sum_{l\in L(v)} \mathcal{X}_l\right)^2\right]$.

$$\mathsf{E}\left[\Big(\sum_{l\in L(v)} \mathcal{X}_l\Big)^2\right] = \mathsf{E}\left[\Big(\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big) + \Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)\Big)^2\right]$$

$$= \mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)^2 + \Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)^2 + 2\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)\right]$$

$$= \mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)^2\right] + \mathsf{E}\left[\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)^2\right] + 2\mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)\right]$$

by linearity of expectation

$$= \mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)^2\right] + \mathsf{E}\left[\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)^2\right] + 2\mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)\right]\mathsf{E}\left[\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)\right]$$

since $\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)$ and $\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)$ are independent

$$= \mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)^2\right] + \mathsf{E}\left[\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)^2\right] + 2\Big(\sum_{g\in L(v_1)} \mathsf{E}[\mathcal{X}_g]\Big)\Big(\sum_{h\in L(v_2)} \mathsf{E}[\mathcal{X}_h]\Big)$$

again by linearity of expectation

$$= \mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)^2\right] + \mathsf{E}\left[\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)^2\right] + 2\Big(\sum_{g\in L(v_1)} \mathrm{score}(g)\Big)\Big(\sum_{h\in L(v_2)} \mathrm{score}(h)\Big) \text{ by Lemma 3}$$

Now by successively opening up the expressions for $\mathsf{E}\left[\left(\sum_{g\in L(v_1)} \mathcal{X}_g\right)^2\right]$ and $\mathsf{E}\left[\left(\sum_{h\in L(v_2)} \mathcal{X}_h\right)^2\right]$, we get

$$\mathsf{E}\left[\Big(\sum_{l\in L(v)} \mathcal{X}_l\Big)^2\right] = \sum_{l\in L(v)} \mathsf{E}\left[\mathcal{X}_l^2\right] + 2\sum_{u\in \mathcal{T}(v)\setminus L(v)} \left\{\Big(\sum_{g\in L(u_1)} \mathrm{score}(g)\Big)\Big(\sum_{h\in L(u_2)} \mathrm{score}(h)\Big)\right\} \tag{B}$$

Therefore, from (A) and (B) we get

$$\begin{aligned}\mathsf{Var}[\mathcal{X}_v] &= \sum_{l\in L(v)} \mathsf{Var}[\mathcal{X}_l] + \frac{\delta}{2}\left(\mathsf{E}\left[\Big(\sum_{l\in L(v)} \mathcal{X}_l\Big)^2\right] - \sum_{l\in L(v)} \mathsf{E}[\mathcal{X}_l^2]\right)\\ &= \sum_{l\in L(v)} \mathsf{Var}[\mathcal{X}_l] + \delta\Big(\sum_{g,h\in L(v) g<h} \mathsf{E}[\mathcal{X}_g\mathcal{X}_h]\Big)\end{aligned}$$

Now using the fact that whenever $g \neq h$, $g,h \in L(v)$, $\mathcal{X}_g$ and $\mathcal{X}_h$ are independent, and from Lemma 3 we get

$$\mathsf{Var}[\mathcal{X}_v] = \sum_{l\in L(v)} \mathsf{Var}[\mathcal{X}_l] + \delta\Big(\sum_{g,h\in L(v), g<h} \mathrm{score}(g)\mathrm{score}(h)\Big)$$

□

Finally, we could bound the deviation.

Lemma 6. *Let* $\delta' \geqslant 2\sqrt{\delta}$, $\mathsf{Pr}[|e_P - \hat{e_P}| > \delta'] \leqslant \frac{1}{4}$.

Proof. From Lemma 4 for any leaf node l

$$\begin{aligned}\mathsf{Var}[\mathcal{X}_l] &= (\mathrm{score}(v) - \lfloor \mathrm{score}(v)\rfloor_\delta)(\lceil \mathrm{score}(v)\rceil_\delta - \mathrm{score}(v)) \text{ (see Eqn. 2)}\\ &\leqslant \mathrm{score}(v)(\lceil \mathrm{score}(v)\rceil_\delta - \lfloor \mathrm{score}(v)\rfloor_\delta) = \delta\,\mathrm{score}(v)\lfloor \mathrm{score}(v)\rfloor_\delta \leqslant \delta\,\mathrm{score}(v)^2\end{aligned}$$

Therefore, by Lemma 5

$$\begin{aligned}\mathsf{Var}[\mathcal{X}_v] &\leqslant \delta\left(\sum_{g\in L(v)} \mathrm{score}(g)^2 + \sum_{g,h\in L(v), g<h} \mathrm{score}(g)\mathrm{score}(h)\right)\\ &< \delta\Big(\mathsf{E}[\mathcal{X}_v]\Big)^2\end{aligned}$$

Hence, by Chebyshev's inequality

$$\mathsf{Pr}\left[|e_P - \hat{e_P}| > \delta' \hat{e_P}\right] \leqslant \frac{\mathsf{Var}[e_P]}{\delta'^2\Big(\mathsf{E}[e_P]\Big)^2} < \frac{\delta}{\delta'^2} \leqslant \frac{1}{4}$$

□

Corollary 2. *Let* $\delta' \geqslant 2\sqrt{\delta}$, $\mathsf{Pr}[|e_O - \hat{e_O}| > \delta'] \leqslant \frac{1}{4}$.

Proof. By argument same as Lemma 6. □

Lemma 7. $\mathsf{Pr}\left[\frac{e_O}{(1+\epsilon/2)} \leqslant \hat{e_P} \leqslant (1+\epsilon)e_O\right] > \frac{1}{2}$.

Proof. We have $\delta = \epsilon^2/16$. Hence $\delta' = \epsilon/2$, or $\epsilon = 2\delta'$. Now the lemma follows from Eqn. (ApproxEstimate), Lemma 6 and Corollary 2. □

Considering medians of the estimates boost the probability, and return estimates for all substrings as required for local alignment.

Lemma 8. *For all strings* s_i^{j-1}, $i = 1,2,...,n, j = 2,3,...,n+1$, $d_{med}^{i:j} \in [(1-\epsilon)d(G,s_i^{j-1}),(1+\epsilon)d(G,s_i^{j-1})]$ *with probability* $\geqslant (1-\frac{1}{n})$ *for all* $i \in \{1,2,...,n\}, j \in \{2,3,...,(n+1)\}$.

Proof. Since we take $6\log n$ estimates, if $d_{med}^{i:j} > (1+\delta')d(G,s_i^{j-1})$ that implies at least $3\log n$ estimates all are higher than $(1+\epsilon)d(G,s_i^{j-1})$ which happens with probability $< \frac{1}{n^3}$. Similarly, if $d_{med}^{i:j} < (1-\epsilon)d(G,s_i^{j-1})$ that implies at least $3\log n$ estimates all are lower than $(1-\epsilon)d(G,s_i^{j-1})$ which happens with probability $< \frac{1}{n^3}$. Therefore, probability that either of the two pathological cases happen for $d_{med}^{i:j}$ is at most $\frac{2}{n^3}$. There are $\binom{n}{2}$

possible substrings. So either of the two pathological cases happen for at least one substring is at most $\frac{1}{n}$. Hence, with probability at least $(1-\frac{1}{n})$, all the median estimates returned are correct within $(1\pm\epsilon)$ factor. □

This leads to Theorem 3.

III. Stochastic Context Free Grammar Parsing

For SCFG parsing we prove the following theorem.

Theorem 4. *Given a stochastic context free grammar* $(G=(\mathcal{N},\Sigma,\mathcal{P},S),\mathbf{p})$ *and a string* $s\in\Sigma^*$ *and any* $\epsilon>0$*, there exists an algorithm that runs in* $\tilde{O}(|G|^2\frac{n^\omega}{\epsilon^4})$ *time and with probability at least* $(1-\frac{1}{n})$ *returns the followings.*

- *A parsing* $\pi'(s)$ *such that* $|\log \mathsf{Pr}[\pi'(s)]|\geqslant(1-\epsilon)|\log \mathsf{Pr}[\pi(s)]|$ *where* $\pi(s)$ *is the most likely parse of* s.
- *For every substring* s_i^j $i,j\in\{1,2,...,n\}$ *of* s*, and estimate* $e(G,s_i^j)$ *such that* $(1-\epsilon)|\log \mathsf{Pr}\left[\pi(s_i^j)\right]|\leqslant e(G,s_i^j)\leqslant(1+\epsilon)|\log \mathsf{Pr}\left[\pi(s_i^j)\right]|$ *where* $\pi(s_i^j)$ *is the most likely parse of* s_i^j.

For each production $P\in\mathcal{P}(G)$, we assign a score, $\mathtt{score}(P)=\log\frac{1}{p(P)}$. Then if $\pi(s)$ maximizes $\mathsf{Pr}[\pi(s)]$, it must minimize $\mathtt{score}(\pi(s))=\sum_{P\in\pi(s)}\mathtt{score}(P)$. The theorem then essentially follows from Theorem 3.We do not have to deal with error-producing rules, or generating sets D_n. On the other hand, all rules, and hence all nodes in a parse tree (and not only leaves) have an associated score. Still, the entire analysis from Section II-A6 applies almost unchanged to derive Theorem 4. See full version for details.

IV. Lower Bound

In this section, we give a high-level description of Theorem 1. For details, and the proof of Theorem 2. see full version. The reduction takes the following steps.

1) Reduce $(\min,\times)$-matrix product over positive real-weighted matrices to SCFG parsing, i.e. show an $O(|G|n^{3-\epsilon}\max_{p\in\mathbf{p}}\log\frac{1}{p})$ algorithm for SCFG parsing implies an $\tilde{O}(n^{3-\beta}\log W)$ algorithm for $(\min,\times)$-matrix product over $\mathbb{R}^+$, $\epsilon,\beta>0$ with maximum weight W.
2) We show an $\tilde{O}(n^{3-\beta}\log W)$ algorithm for $(\min,\times)$-matrix product with entries in $(0,W)$, $\beta>0$, implies an $\tilde{O}(n^{3-\beta}\log W)$ algorithm for detecting negative weight triangle in n-node graphs with edge weights in $[-W,W]$.
3) Finally, use subcubic equivalence between negative weight triangle detection and weighted APSP [47].

We define the output of a SCFG parsing algorithm as an oracle $\mathcal{F}_{G,s}$ which given any nonterminal $A\in\mathcal{N}(G)$ and substring s_i^j returns q in constant time under the following conditions. (1) A derives s_i^j with maximum probably q, and (2) there is a derivation sequence $S\overset{*}{\Rightarrow}s_1^{i-1}As_{j+1}^n$, that is the full string can be derived using A.We say A consistently derives s_i^j. In this definition we maintain only parsing information for substrings from which the full string can be derived. All algorithms known even for simple parsing maintain this information, along with the lower bound result of Lee [26].

Step 1: Reducing $(\min,\times)$*-matrix product to SCFG parsing:* We are given two matrices a and b with entries from $\mathbb{R}^+$, and want to compute their $(\min,\times)$-product $c=a.b:c_{i,j}=\min_{1\leqslant k\leqslant m}(a_{i,k}.b_{k,j})$.

Input string. Let us take $d=\lceil m^{1/3}\rceil$, and we set $\delta=d+2$. Our universe of terminals is $\Sigma=\{s_1,s_2,...,s_{3d+6},x\}$. Our input string s is of length 3δ and is simply $s_1s_2....s_{d+2}s_{d+3}...s_{2d+4}s_{2d+5}....s_{3d+6}$. **Grammar construction.** Consider a matrix index i, $1\leqslant i\leqslant m\leqslant d^3$. Let $f_1(i)=\lfloor i/d\rfloor$ and $f_2(i)=(i \bmod d)+2$. Hence $f_1(i)\in[1,d^2]$, and $f_2(i)\in[2,d+1]$. We can uniquely obtain i from $f_1(i)$ and $f_2(i)$. For notational simplicity we use i_1 to denote $f_1(i)$ and i_2 to denote $f_2(i)$. If we decompose s into three consecutive equal parts of size $d+2$ each, then i_2, $i_2+\delta$, and $i_2+2\delta$ belong to first, second and third halves respectively. We now create the grammar $(G,\mathbf{p})=(\{\mathcal{N},\Sigma,\mathcal{P},S\},\mathbf{p})$. Start from $\mathcal{N}=\{S\}$ and $\mathcal{P}=\phi$.

- We add nonterminal W and productions to generate arbitrary non-empty substrings from $\Sigma\setminus\{x\}$. $W\longrightarrow s_lW|s_l$, $l\in[1,3d+6]$ each rule having probability $\frac{1}{2(3d+6)}$ (W-Rule). The probabilities add up to 1.
- We encode the entries of A and B in our grammar. We add nonterminals $A_{p,q}:1\leqslant p,q,\leqslant d^2$, and $B_{p,q}:1\leqslant p,q\leqslant d^2$. Let $\mathtt{Count}_A(i_1,j_1)=\sum_{i,j:x=f_1(i),y=f_1(j)}\frac{1}{a_{i,j}}$, and $\mathtt{Count}_B(i_1,j_1)=\sum_{i,j:x=f_1(i),y=f_1(j)}\frac{1}{b_{i,j}}$. Set $\mathtt{MaxCount}_A=\max_{r,s,1\leqslant r,s\leqslant d^2}\mathtt{Count}_A(r,s)$, and $\mathtt{MaxCount}_B=\max_{r,s,1\leqslant r,s\leqslant d^2}\mathtt{Count}_B(r,s)$. For each entry $a_{i,j},b_{i,j}$, $i=(i_1,i_2)$, and $j=(j_1,j_2)$ add productions (A-Rule): $A_{i_1,j_1}\longrightarrow s_{i_2}Ws_{j_2+\delta}$ with prob. $\frac{1}{a_{i,j}\mathtt{MaxCount}_A}$, if $\mathtt{MaxCount}_A>\mathtt{Count}_A(i_1,j_1)$, then add a

dummy rule $A_{i_1,j_1} \longrightarrow x$ with prob. $\frac{MaxCount_A - Count_A(i_1,j_1)}{MaxCount_A}$. Add productions (B-Rule) by replacing every "A" and "a" in (A-Rule) with "B" and "b" respectively.

- We add nonterminals $\{C_{p,q} : 1 \leqslant p,q \leqslant d^2\}$ and the productions for all r, $1 \leqslant r \leqslant d^2$ $C_{p,q} \longrightarrow A_{p,r}B_{r,q}$ with prob. $\frac{1}{d^2}$ (C-Rule)
- Finally, we add the production for the start symbol S for all p,q, $1 \leqslant p,q \leqslant d^2$ $S \longrightarrow WC_{p,q}W$ with prob. $\frac{1}{d^4}$ (S-Rule)

It can be verified that probabilities of all rules with same nonterminal on the LHS add up to 1. Hence the constructed grammar is a SCFG. The following lemma suggests that by looking at $\mathcal{F}_{G,s}(C_{i_1,j_1}, s_{i_2}^{j_2+2\delta})$ we can derive $c(i,j)$ where $i = (i_1,i_2)$ and (j_1,j_2). Then noting that the grammar size is $O(m^2)$, and string length $O(m^{1/3})$, we get the desired subcubic equivalence between SCFG parsing and $(\min,\times)$ matrix product. Note that this non-CNF grammar can easily be converted into a CNF representation with constant factor blow-up in size.

Lemma 9. *For $1 \leqslant i,j \leqslant m$, the entry $c_{i,j} = l$, if and only if $C_{i_1j_1}$ consistently derives $s_{i_2}^{j_2+2\delta}$ with probability $1/(ld^2.MaxCount_A.MaxCount_B[2(3d+6)]^{j_2+2\delta-i_2-2})$.*

Step 2: Reducing Negative Triangle Detection to $(\min,\times)$-matrix product with positive real entries to [5]*:* We assume all weights are integers, and the maximum absolute weight is at least 3. Both of these can be achieved by appropriately scaling the edge weights. Let W be the maximum absolute weight on any edge e, $|W| \geqslant 3$. Set $a(i,j) = w_{i,j} + W^3$, and $b(i,j) = a(i,j)$. Therefore, all entries of a and b are > 0. Find the $(\min,\times)$ product of $c = a.b$. Let $c'(i,j) = c(i,j) - W^6 + W^3 w_{i,j} + 2W^2$.

If there exists a negative triangle $i \to k \to j$, then $w_{i,k} + w_{k,j} + w_{i,j} \leqslant -1$. Hence $W^3(w_{i,k} + w_{k,j} + w_{i,j}) \leqslant -W^3$ or, $W^3(w_{i,k} + w_{k,j} + w_{i,j}) + 2W^2 \leqslant -W^3 + 2W^2 \leqslant -W^2$. Now $(w_{i,k} + W^3)(w_{k,j} + W^3) = w_{i,k}w_{k,j} + W^3(w_{i,k} + w_{k,j}) + W^6$ Hence, $c'(i,j) = \min_k(w_{i,k}w_{k,j} + W^3(w_{i,k} + w_{k,j} + w_{i,j}) + 2W^2)$. Now $-W^2 \leqslant w_{i,k}w_{k,j} \leqslant W^2$. Therefore, if there is a negative triangle involving edge (i,j), then $c'(i,j) \leqslant W^2 + \min_k(W^3(w_{i,k} + w_{k,j} + w_{i,j})) + 2W^2 \leqslant 0$

On the other hand, if there is no negative triangle, then $c'(i,j) \geqslant -W^2 + 2W^2 = W^2 \geqslant 9$ for all $1 \leqslant i,j \leqslant n$. Thus by checking c' we can detect existence of negative weight triangle, completing the desired reduction. Theorem 1 now follows.

Acknowledgement. The author thanks Alfred Aho for asking whether language edit distance can be cast as an algebraic closure over a closed semiring, and anonymous reviewers for pointing out a glitch in the construction of Sidon sequence and for many helpful comments.

REFERENCES

[1] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, apsp and diameter. SODA, pages 1681–1697, 2015.
[2] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. ICALP, pages 39–51, 2014.
[3] Alfred V. Aho and John E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1974.
[4] Alfred V. Aho and Thomas G. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM J. Comput.*, 1(4), 1972.
[5] Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2), April 1997.
[6] Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM J. Comput.*, 30(6), December 2001.
[7] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. FOCS, 2010.
[8] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. STOC, 2009.
[9] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). STOC, 2015.
[10] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. FOCS, 2004.
[11] Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. SODA, 2006.
[12] Amit Chakrabarti, Graham Cormode, Ranganath Kondapally, and Andrew McGregor. Information cost tradeoffs for augmented index and streaming language recognition. FOCS, 2010.
[13] Richard Cole and Ramesh Hariharan. Approximate string matching: A simpler faster algorithm. *SIAM J. Comput.*, 31(6), June 2002.
[14] Richard Durbin. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[5] Reduction interestingly implies $(\min,+)$-product with *real* weights bounded by $O(\log n)$ is subcubic equivalent to APSP. In contrast when the weights are integer and bounded by $O(\log n)$, algorithms with $\tilde{O}(n^\omega)$ running time exist. Reducing $(\min,+)$ product to $(\min,\times)$ product is possibly a folklore. Here we show a reduction from negative triangle for the sake of completeness.

[15] P Erdös and Pál Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, 1(4).

[16] Steven Grijzenhout and Maarten Marx. The quality of the XML web. *Web Semant.*, 19, 2013.

[17] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.

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

[19] Yuri A. Ivanov and Aaron F. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), 2000.

[20] Mark Johnson. PCFGs, Topic Models, Adaptor Grammars and Learning Topical Collocations and the Structure of Proper Names. ACL, 2010.

[21] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. 1997.

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

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

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

[25] François Le Gall. Faster algorithms for rectangular matrix multiplication. FOCS, 2012.

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

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

[28] Michael I Miller and Joseph A O'Sullivan. Entropies and combinatorics of random branching processes and context-free languages. *IEEE Transactions on Information Theory,*, 38(4), 1992.

[29] Darnell Moore and Irfan Essa. Recognizing multitasked activities from video using stochastic context-free grammar. NCAI, 2002.

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

[31] Kevin OBryant. A complete annotated bibliography of work related to Sidon sequences. *Electron. J. Combin.*, 575, 2004.

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

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

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

[35] Andrea Rosani, Nicola Conci, and Francesco G. De Natale. Human behavior recognition using a context-free grammar. *Journal of Electronic Imaging*.

[36] Imre Z Ruzsa. An infinite Sidon sequence. *Journal of Number Theory*, 68(1), 1998.

[37] Barna Saha. The Dyck language edit distance problem in near-linear time. FOCS, 2014.

[38] Barna Saha. Faster language edit distance, connection to all-pairs shortest paths and related problems. *arXiv preprint arXiv:1411.7315*, 2014.

[39] S. C. Sahinalp and U. Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm. FOCS, 1996.

[40] Yasubumi Sakakibara. Grammatical inference in bioinformatics. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(7), 2005.

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

[42] Leslie G Valiant. General context-free recognition in less than cubic time. *Journal of computer and system sciences*, 10(2), 1975.

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

[44] Milind Mahajan Wang, Ye-Yi and Xuedong Huang. A unified context-free grammar and n-gram model for spoken language processing. ICASP, 2000.

[45] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. STOC, 2014.

[46] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. STOC, 2012.

[47] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. FOCS, 2010.

[48] Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. SODA, 2009.

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

[50] Weiling Zhu and Javier Garcia-Frias. Stochastic context-free grammars and hidden Markov models for modeling of bursty channels. *IEEE Transactions on Vehicular Technology*, 53(3), 2004.

[51] Uri Zwick. All pairs shortest paths in weighted directed graphs–exact and almost exact algorithms. FOCS, 1998.

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