Faster Language Edit Distance, Connection to All-pairs Shortest Paths and Related Problems

Barna Saha*

Abstract

Given a context free language $\mathcal{L}(\mathcal{G})$ 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 $\mathcal{L}(\mathcal{G})$. 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 in data management, machine learning, compiler optimization, computational biology, computer vision and linguistics, there is no algorithm known till date that computes or approximates language edit distance problem in true sub-cubic 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 $\tilde{O}(\frac{n^\omega}{\text{poly}(\epsilon)})$ 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 edit script. 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 $\tilde{O}(\frac{n^\omega}{\text{poly}(\epsilon)})$ time with high probability. An independent work [Rajasekaran, Nicolae 2014], recently provided an $O(\frac{n}{R})$-approximation algorithm for language edit distance that runs in $O(Rn^\omega)$ time. Our results significantly surpass theirs for both exact and approximate computations. We also design the very first sub-cubic ($\tilde{O}(n^\omega)$) algorithm to handle arbitrary *stochastic context free grammar parsing*. Stochastic context free grammars lie the foundation of statistical natural language processing, they generalize hidden Markov models, and have found widespread applications in many fields.

To complement our upper bound result, we show that exact computation of language edit distance in true sub-cubic time will imply a truly sub-cubic 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 sub-cubic equivalence. By a known lower bound result [Lee 2002], even parsing a context free grammar is as hard as boolean matrix multiplication. Therefore obtaining any nontrivial multiplicative approximation algorithm for language edit distance problem in time $o(n^\omega)$ is (almost) not possible.

1 Introduction

"What should a compiler do when it discovers an error in the source program?" In a pursuit to answer this question, Aho and Peterson introduced the study of *language edit distance problem* more than four decades back. Given a context free grammar (CFG) $\mathcal{L}(G)$ over alphabet Σ and a string $s \in \Sigma^*$, *the language edit distance* determines the fewest number of edits (insertions, deletions and substitutions) along with an edit script to convert s into a valid member of $\mathcal{L}(G)$. Since its introduction, the *language edit distance problem* has found wide applicability in diverse domains. Designing error-correcting parsers [2, 37], detecting

*University of Massachusetts Amherst, Amherst, MA 01002. Email: barna@cs.umass.edu.

anomalies, mining and repairing data quality problems [19, 28, 40], biological structure prediction [21, 49], learning topic and behavioral models [25, 36, 42], video and speech processing [50], are among its many applications. Context free languages play a pivotal role in the development of formal language theory, and have surprising connections to many branches of mathematics [17, 23].

The proposed algorithm by Aho and Peterson for language edit distance problem has a running time of $O(|G|^2n^3)$ where $|G|$ is the grammar size and n is the length of the input sequence [2]. The dependency on grammar size in run time was later improved by Myers to $O(|G|n^3)$ [37]. In most applications, the cubic time-complexity on string length is a major bottleneck. Except minor polylogarithmic improvement over $O(n^3)$ [41, 49], till date a "true" sub-cubic algorithm, one that runs in time $O(n^{3-\gamma})$ for some $\gamma > 0$, does not exist. Despite significant efforts to design faster algorithms, only heuristics with no provable guarantees are known for specific grammars [15, 16, 26, 28, 34]. Their performance could be arbitrarily bad with multiplicative $\Theta(n)$ gap in terms of number of edits. An exception is our previous work [44], where a near-linear time algorithm with running time $\tilde{O}(n^{1+\epsilon})$ irrespective of alphabet size and poly-logarithmic approximation factor has been developed for *Dyck* language edit distance. The edit distance to *Dyck* language significantly generalizes the well-studied string edit distance problem [6–9, 12, 30, 45]. It represents the grammar of well-balanced parentheses and forms a core class of context free grammars with many applications [28, 49].

Stochastic context free grammars (SCFG) extend context free grammars where each production rule has a probability, indicating the likelihood of applying that rule to generate members of the language. SCFG has found widespread applications as a rich framework for modeling complex phenomena [13, 24, 46, 57]. They generalize hidden Markov model, and many stochastic branching processes such as Galton-Watson branching process [35]. A pertinent question in this domain is *stochastic context free grammar parsing*. Given a string $s \in \Sigma^*$, we want to find the most likely way to parse s, that is among all its possible derivations, we want to detect the one with maximum probability of occurrence. If s does not belong to the language, then the probability is 0. All known algorithms for SCFG parsing, e.g., CYK, Inside-outside algorithm, run in $O(n^3)$ time. Prior to our work, no sub-cubic algorithm existed for either exact or approximate computation of arbitrary SCFG parsing.

In this paper, we make several important contributions.

1. *We give the first true sub-cubic algorithm to approximate language edit distance for arbitrary context free languages almost optimally.* Our algorithm runs in $\tilde{O}(|G|^2 \frac{n^{\omega}}{\mathtt{poly}(\epsilon)})$[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 [31] (Section 3). Therefore, if d is the optimum distance, the computed distance is in $[d, d(1+\epsilon)]$. The best known bound for $\omega < 2.373$ [53]. In fact the running time can be further improved to $\tilde{O}\left(\frac{1}{\mathtt{poly}(\epsilon)}\left(|G|n^{\omega} + \omega(n|G|, n, n|G|) + |G|^2n^2\right)\right)$ where $\omega(n|G|, n, n|G|)$ is the running time for 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.

As a first step of our proposed algorithm, an $O(|G|^2T(n))$ time algorithm for exact computation of language edit distance falls off directly, where $T(n)$ is the time required to compute distance product (a.k.a. min-plus product, tropical matrix product) of two $n \times n$ matrices. A breakthrough result of Williams [52] implies $T(n) = \tilde{O}(\frac{n^3}{\log^k n})$ for any constant $k > 0$. Recently, in an independent work Rajsekaran and Nicolae obtained an $O(|G|^4T(n))$ algorithm for exact computation of language edit distance problem [41]. The authors do not specify the dependency on grammar size in running time, but even assuming the best case running times for their procedures, their dependency is at least $O(|G|^4)$. When language edit distance is at

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

most R, $T(n) = Rn^{\omega}$ by an old result of Alon, Galil and Margalit [4] and Takaoka [47]. This directly gives an $O(\frac{n}{R})$-approximation algorithm in $O(|G|^4 Rn^{\omega})$ time [41]. Note that, this provides no approximation guarantee in $O(n^{\omega})$ time, and only an $O(n^{.3727})$-approximation in $O(|G|^4 n^3)$ time with the current best bound of ω. Our results surpass them by a big margin.

A question may be raised here, whether an approximate distance product computation, for which much better result is known [58], can be used to give an approximate answer to language edit distance problem. The answer, unfortunately, is no–there is a fundamental bottleneck which we will elaborate later. The known algorithms for approximate distance product computation [58] only lead to an $O(\frac{n}{\log n})$-approximation for language edit distance problem. On the other hand, our approximation algorithm can be used to give an alternate approximation scheme for a large variety of problems such as all-pairs shortest paths, minimum weight triangle/cycle detection, computing diameter, radius and many others in $\tilde{O}(n^{\omega} \log W)$ time where W is the maximum absolute weight of any edge.

2. Our algorithm solves the *local alignment problem* where given $\mathcal{L}(G)$ and $\sigma \in \Sigma^*$, for all substrings σ' of σ, we need to compute their language edit distance (Section 3). Such local alignment problems are studied extensively under string edit distance computation for approximate pattern matching, for parsing etc. [1,20]. Since, there are $\Theta(n^2)$ different substrings and at least linear time per substring is required to return an edit script, $\Theta(n^3)$ time is unavoidable for reporting all edit scripts. Our algorithm that runs in $\tilde{O}(|G|^2 \frac{n^{\omega}}{poly(\epsilon)})$ time, also provides an $(1 \pm \epsilon)$-approximation for the local alignment problem with high probability (prob $> 1 - \frac{1}{n}$). For any substring σ', if its size is m and optimum language edit distance d, then the returned estimate is within $[(1-\epsilon)d, (1+\epsilon)d]$. Further, its edit script can be computed in just an additional $O(m)$ time.

3. *We design the first sub-cubic algorithm for SCFG parsing near-optimally* (Section 4). 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 $\tilde{O}(|G|^2 n^{\omega} \log\log \frac{1}{p_{max}})$ 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_{max} is the maximum probability of any production. To the best of our knowledge, prior to our work, no true sub-cubic algorithm for arbitrary SCFG parsing was known.

The upper bound results naturally raise the question whether the running time can be further improved. First, we note that by a known lower bound result of Lee [32], even context free grammar parsing is as hard as boolean matrix multiplication. Therefore, obtaining any multiplicative approximation factor for language edit distance in $o(n^{\omega})$ time is (almost) impossible. Given that, our result is optimal in terms of string length. Our current dependency on ϵ is $O(\frac{1}{\epsilon^4})$. We conjecture this can be improved to $O(\frac{1}{\epsilon^2})$. Can we improve the grammar dependency to linear? Even for parsing, no sub-cubic algorithm with linear dependency on grammar size is known till date.

4. (i) For exact computation, we show (Section 5)

Any sub-cubic algorithm for context free language edit distance problem will lead to a sub-cubic algorithm for all-pairs shortest paths, a long-standing open question.

This will automatically imply all the following problems have sub-cubic running times as well due to sub-cubic equivalence [1,54].

(a) Minimum weight triangle: Given an n-node graph with real edge weights, compute u, v, w such that $(u, v), (v, w), (w, u)$ are edges and the sum of edge weights is minimized.

(b) Negative weight triangle: Given an n-node graph with real edge weights, compute u, v, w such that $(u, v), (v, w), (w, u)$ are edges and the sum of edge weights is negative.

(c) Metricity: Determine whether an $n \times n$ matrix over $\mathbb{R}$ defines a metric space on n points.

(d) Minimum cycle: Given an n-node graph with real positive edge weights, find a cycle of minimum total edge weight.

(e) Second shortest paths: Given an n-node directed graph with real positive edge weights and two nodes s and t, determine the second shortest simple path from s to t.

(f) Replacement paths: Given an n-node directed graph with real positive edge weights and a shortest path P from node s to node t, determine for each edge $e \in P$ the shortest path from s to t in the graph with e removed.

(g) Radius problem: Given an n-node weighted graph with real positive edge weights, determine the minimum distance r such that there is a vertex v with all other vertices within distance r from v.

(ii) *Further, any sub-cubic algorithm for exact stochastic context free grammar parsing will also lead to a sub-cubic algorithm for all-pairs shortest paths.*

The lower bound proofs require linear dependency on grammar size. Improving the lower bound to consider super-linear dependency on grammar size will be very interesting. 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 problems in P, for which our understanding still remains primitive.

Language edit distance computation is much harder than language recognition (or parsing) which only determines if $s \in \mathcal{L}(G)$ or not, that is it distinguishes between 0 and non-zero language edit distance. A beautiful result by Valiant was the first to overcome the barrier of cubic running time, and gave an $\tilde{O}(|G|^2 n^{\omega})$ algorithm for general context free parsing [48]. Our result surprisingly indicates that parsing time is enough to compute a near-optimal result for the much harder language edit distance problem. Is this result generally true? For example, Dyck language can be parsed in linear time; a linear time almost optimal Dyck language edit distance problem will be a major breakthrough giving a near-optimal linear time algorithm for string edit distance computation [44]. Understanding the relation between parsing time, and language edit distance computation remains a big open question.

Finally, the $O(n^3)$ dynamic programming algorithm for language edit distance computation is a prototypical example of a dynamic programming that runs in cubic time. Can we provide a generic framework to take any cubic time dynamic programming algorithm and create an $\tilde{O}(n^{\omega})$ time near-optimal algorithm out of it? This will go a long way in our endeavor to develop a systematic methodology for scalable algorithm design.

1.1 Other 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. The last decade has seen a remarkable development on lowering the running time from quadratic to near-linear for string edit distance computation in expense of *approximation* [6–9, 12, 30, 45]. An $O(n + d^2)$ algorithm was first developed by Landau that guarantees an $O(\sqrt{n})$ approximation [30], followed by a series of works that improved the bound successively to

$O(n^{\frac{3}{7}})$ [8], then to $O(n^{\frac{1}{3}+o(1)})$ [9], to $O(2^{\sqrt{\log n \log\log n}})$ (runs in time $O(n2^{\sqrt{\log n \log\log n}})$) [7]. The current best known result is by a beautiful work of Andoni, Karuthgamer and Onak that obtains a polylogarithmic approximation for string edit distance in near-linear time. Even the *Dyck language edit distance* is a generalization of string edit distance problem, that is given an α-approximation for Dyck language edit distance in $O(n^{\beta})$ time $\alpha \geqslant 1, \beta \geqslant 0$ implies an α-approximation for string edit distance in $O(n^{\beta})$ time [44].

Language recognition and parsing problems have been studied for variety of languages under different models for decades [5, 10, 29, 33, 39]. The works of [10, 29, 33] study complexity of recognizing Dyck language in space-restricted streaming model. Alon, Krivelevich, Newman and Szegedy consider testing regular language and Dyck language recognition problem using sub-linear queries [5], followed by improved bounds in works of [39]. 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 who obtained the first sub-cubic algorithm for context free grammar parsing [48].

The applicability of context free grammars, and language edit distance computation is huge. From detecting and repairing data/program errors, finding anomalies and interesting structural properties, discovering topic models from text, video and speech analysis; the number of potential fields where this generic problem has been and can be applied is ever expanding [19, 21, 25, 28, 36, 40, 42, 49, 50]. Our results on language edit distance computation, and on stochastic grammars will impact many of these fields directly. In this paper, for the first time, we establish their connection to many fundamental graph and matrix problems, which will potentially lead to even more applications of this area.

2 Preliminaries and Overview

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 symbol in $\mathcal{N}$ known as the *start* symbol.

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., $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$, where $\overset{*}{\Rightarrow}$ implies that w can be derived from S using one or more production rules. If we always first expand the leftmost nonterminal during derivation, we have a leftmost derivation. Similarly, one can have a rightmost derivation. If $s \in \mathcal{L}(G)$ it is always possible to have a leftmost (rightmost) derivation of s from S.

We only consider grammars for which all the nonterminals are reachable, that is each of them is included in at least one derivation of a string in the language from S. Any unreachable nonterminal can be easily detected and removed decreasing the grammar size.

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, for example, in CYK algorithm, Earley's algorithm for parsing context-free grammars [27]. All prior known cubic algorithms for language edit distance also uses CNF representation [2, 37]

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*

$$d_G(G, s) = \min_{z : \in \mathcal{L}(G)} d_{ed}(s, z)$$

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.

Definition 2 (t-approximation for Language Edit Distance)**.** *Given a grammar* $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ *and* $s \in \Sigma^*$*, a* t*-approximation algorithm for language edit distance problem,* $t \geqslant 1$*, returns a string* s' *such that* $s' \in \mathcal{L}(G)$ *and* $d_G(G, s) \leqslant d_{ed}(s', s) \leqslant t * d_G(G, s)$*.*

2.1 Contributions and Techniques

Upper Bound. For any $\epsilon > 0$, we give a randomized $(1 + \epsilon)$-approximation algorithm for the language edit distance problem in $\tilde{O}(\frac{1}{\epsilon^4}|G|^2 n^{\omega})$ time where ω is the exponent of boolean matrix multiplication of n dimensional square matrices. Moreover, the same algorithm solves the local alignment problem within $(1 \pm \epsilon)$ factor with the same time bound. All the estimates are obtained with probability $> 1 - \frac{1}{n}$.

Theorem 1. *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 true sub-cubic algorithm on string length for computing language edit distance near optimally.

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 $s \in \Sigma^*$, one can test in $O(n^{\omega})$ time whether $s \in \mathcal{L}(G)$. This was done by reducing the context free grammar parsing problem to boolean matrix multiplication. To extend this framework in our context, we augment $\mathcal{P}$ with error-producing rules, such that if parsing s must use k such productions, $d_G(G, s) = k$. The goal is therefore to obtain a parsing using minimum such error-producing rules. Aho and Peterson more than forty years back used error-producing productions, and proposed a dynamic programming algorithm to keep count on the minimum such productions required for parsing any substring. We can consider the augmented grammar, and modify Valiant's approach suitably to keep a count on the minimum number of error-producing productions required for each substring. Therefore instead of a single bit of information whether a substring parses or not, we keep a "distance" count, and the problem reduces to distance product computation. We recall that a distance-product between two $n \times n$ dimensional matrices A and B is defined as

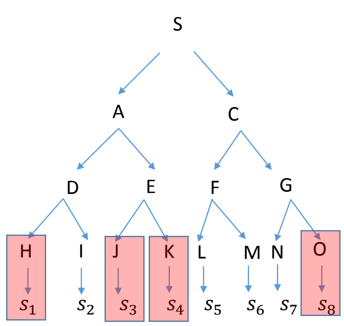
$$(A \odot B)[i, j] = \min_k \{A[i, k] + B[k, j]\}.$$

If indeed we use the augmented grammar by Aho and Peterson [2], followed by a modification of Valiant's method [48], we get a $O(T(n)|G|^4)$ algorithm for language edit distance computation. Recall

$T(n)$ is the time required for distance product computation of two n dimensional square matrices. The dependency on grammar size blows up to at least $O(|G|^4)$–this is the bound that a recent independent result of Rajsekaran and Nicolae obtains [41][2] Addition of error-producing rules make the augmented grammar non-CNF–and 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 [2, 48] must use CNF grammars. We overcome this difficulty by carefully constructing the error-producing rules, and directly dealing with the resultant non-CNF augmented grammar. The first step already gives an $O(|G|^2 T(n))$ time algorithm for exact computation of language edit distance and $O(|G|^2 R n^{\omega})$ algorithm for partially computing language edit distance for all those strings whose distance is bounded by R.

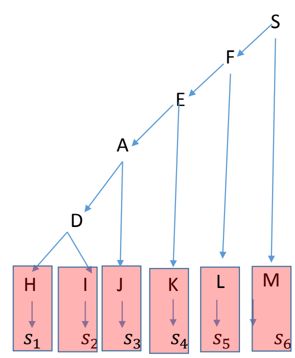
Suppose, we use $R = 1$ and identify all substrings that can be derived using at most one edit in $O(n^{\omega})$ time. Let M be a matrix such that $M[i, j]$ contains information on how to derive substring s_i^j using at most one error-producing rule. It is then possible to define a suitable matrix-product operation such that if we square M, we will be able to identify all substrings within edit distance 2 by clubbing two adjacent substrings each of which are within edit distance 1. We may then hope to repeat this process at most $\lceil \log n \rceil$ times successively obtaining substrings with larger edit distance, e.g., $2, 2^2, 2^3, \ldots$. This then leads to a 2-approximation algorithm for language edit distance computation. And, in fact, if this squaring process works, then language edit distance can be computed exactly in time $O(n^{\omega} \log n)$ time. This is reminiscent of transitive closure computation cheaply given a single matrix product. **However, this method just does not work.** There is a fundamental difficulty in applying this approach. It is best to illustrate the difficulty with an example.

Consider the following examples.



Suppose the string $s = s_1s_2s_3s_4s_5s_6s_7s_8$ has only one derivation, shown in the form of a derivation tree. The red highlighted rectangles indicate these are error-producing rules. Therefore, the language edit distance of s is 4. Initially, in the matrix M we have substrings s_1s_2 derived from D, s_3 derived from J, s_4 derived from K, $s_5s_6s_7s_8$ derived from C. After squaring M, we still get substrings s_1s_2 derived from D, $s_5s_6s_7s_8$ derived from C, but now s_3s_4 is derived from E.

Upon further squaring, we get $s_1s_2s_3s_4$ derived from A. The third squaring helps to merge $s_1s_2s_3s_4$ with $s_5s_6s_7s_8$, and we get S derives s. Hence, instead of 2 squaring, the actual process requires 3 squaring steps to derive s in this example.



A further pathological case where n squaring steps are required to derive s

This also illustrates why no good "approximation" algorithm for distance product computation may be sufficient. On the right figure, if we use a $(1 + \epsilon)$-approximation for distance product each time, then the overall approximation is as bad as $(1 + \epsilon)^n$. With the best known bound for ϵ with $\tilde{O}(n^{\omega})$ running time [58], the approximation factor can be as worse as $O(\frac{n}{\mathrm{polylog} n})$. Lack of flexibility of recursively combining substrings is a major bottleneck. The issue seems even more fundamental. As we will see when designing the algorithm, we need to compute transitive closure using matrix products, where the underlying structure does not even satisfy associativity.

We are able to overcome this significant difficulty by introducing several new ideas. Randomization

[2] In fact, the authors do not specify the dependency on grammar size at all, and dependency of $O(|G|^4)$ is the best possible. Their method might require even higher dependency because of more complex operation that they employ to use Valiant's approach.

plays a critical role. We restrict the number of possible edit distances to few distinct values, and use randomization carefully to maintain the language edit distance for each substring on expectation and show that variance does not blow up. A 1941 theorem by Erdös and Turán [14], and a construction by Ruzsa [43] 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. Details of the algorithm and analysis are given in Section 3. This also leads to the first sub-cubic algorithm for stochastic context free grammar parsing near-optimally (Section 4).

Lower Bound. *We show that a sub-cubic algorithm for computing language edit distance will result in a breakthrough result for several fundamental graph problems.* This is the first paper to lay out the connection between context free language edit distance problem and many basic graph and matrix problems.

Theorem 2. *Given a context-free grammar $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$, and a string $s \in \Sigma^*$, $|s| = n$, if the language edit distance problem can be solved in $O(|G|n^{3-\delta})$ time then that implies an algorithm with running time $O(m^{3-\delta/3})$ for all-pairs shortest path problem on directed graphs with m vertices.*

Corollary 1. *Given a context-free grammar $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$, and a string $s \in \Sigma^*$, $|s| = n$, if the language edit distance problem can be solved in $O(|G|n^{3-\delta})$ time then that implies an algorithm with running time $O(m^{3-\gamma})$, $\gamma, \delta > 0$ for all of the following problems.*

1. *Minimum weight triangle: Given an n-node graph with real edge weights, compute u, v, w such that $(u, v), (v, w), (w, u)$ are edges and the sum of edge weights is minimized.*

2. *Negative weight triangle: Given an n-node graph with real edge weights, compute u, v, w such that $(u, v), (v, w), (w, u)$ are edges and the sum of edge weights is negative.*

3. *Metricity: Determine whether an $n \times n$ matrix over $\mathbb{R}$ defines a metric space on n points.*

4. *Minimum cycle: Given an n-node graph with real positive edge weights, find a cycle of minimum total edge weight.*

5. *Second shortest paths: Given an n-node directed graph with real positive edge weights and two nodes s and t, determine the second shortest simple path from s to t.*

6. *Replacement paths: Given an n-node directed graph with real positive edge weights and a shortest path P from node s to node t, determine for each edge $e \in P$ the shortest path from s to t in the graph with e removed.*

7. *Radius problem: Given an n-node weighted graph with real positive edge weights, determine the minimum distance r such that there is a vertex v with all other vertices within distance r from v.*

We establish similar connection with SCFG parsing and distance computation over graphs and matrices.

Theorem 3. *Given a stochastic context-free grammar $\{G = (\mathcal{N}, \Sigma, \mathcal{P}, S), \mathbf{p}\}$, and a string $s \in \Sigma^*$, $|s| = n$, if the stochastic context free grammar parsing problem can be solved in $O(|G|n^{3-\delta})$ time then that implies an algorithm with running time $O(m^{3-\delta/3})$ for all-pairs shortest path problem on directed graphs with m vertices.*

3 A Near-Optimal Algorithm for Computing Language Edit Distance in $\tilde{O}(|G|^2 n^\omega)$ Time

In this section we provide our main upper bound result and prove Theorem 1. Let $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ be an arbitrary context-free grammar in CNF form. We give an $(1+\epsilon)$-approximation algorithm for language edit distance problem on G that runs in $\tilde{O}(\frac{|G|^2 n^\omega}{\epsilon^4})$ time. The algorithm also returns an edit script. In addition, we show $\tilde{O}(\frac{|G|^2 n^\omega}{\epsilon^4})$ time is enough to solve local alignment problem within $(1 \pm \epsilon)$-approximation factor.

3.1 Error Producing Productions & Assigning Scores

The first step of our algorithm is to introduce new error-producing rules in $\mathcal{P}(G)$, each of which adds one type of errors: insertion, deletion and substitution. Error-producing rules are assigned boolean scores, and our goal is to use the fewest possible error-producing rules to parse a given string s. If parsing s must use error-producing rules with a total score of k, then $d(G, s) = k$. The rules introduced are extremely simple and intuitive.

Substitution Rule. If there exists rule $A \to x_1|x_2|...|x_l,\ x_i \in \Sigma, i = 1, 2, ...l$ for some $l \geqslant 1$, then cnsider adding an error-producing rule $A \to y_1|y_2|...|y_{l'}$, such that $\{y_1, y_2, ..., y_{l'}\} = \Sigma \setminus \{x_1, x_2, ..., x_l\}$. This rule corresponds to a single substitution error where a x_i is substituted by a y_j. Each of these new rules referred as *elementary substitution rule* gets a score of 1. However, instead of maintaining these rules explicitly, we maintain a single rule $A \to \$x_1x_2...x_l$, where \$ is a special symbol not occurring in Σ. This implicit representation has the same effect as maintaining $A \to y_1|y_2|...|y_{l'}$ explicitly only the size of the rule is $l + 3$ (counts the number of terminals and nonterminals on LHS and RHS). If we want to use $A \to y_i$ as an elementary substitution rule, we check if there exists a rule $A \to \$\alpha$ where $\alpha \in Sigma^*$ and y_i is not in α. If so, we use $A \to y_i$ with score 1 for parsing.

Insertion Rule. For each $x \in \Sigma$, we add an error-producing rule $I \to x$ which corresponds to a single insertion error. Again the score for each of these rules referred as *elementary insertion rule* is 1. We also introduce for each nonterminal $A \in \mathcal{N}$, a new rule $A \to IA$ to allow a single insertion right before a substring derivable from A. Further, we add a new rule $A \to AI$ for each nonterminal $A \in \mathcal{N}$ to let single insertion happen at the end of a substring derivable from A. Finally, we add a rule $I \to II$ to combine two insertion errors. These later rules that combine elementary insertion rule with other nonterminals 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. We call them *elementary deletion rules*. Inclusion of these rules violate the CNF property, as in a CNF grammar, we are not allowed to have ε on the RHS of any production except for $S \to \varepsilon$.

Observation 1. $|G_e| = O(|G|)$.

Scoring. Each of the elementary error-producing rules gets a score of 1 as described above. The existing rules and remaining new rules get a score of 0. If a parsing of a string s requires applying productions $\pi = P_1, P_2, ..., P_n$ respectively then the parse has a score

$$\mathrm{score}(\pi) = \sum_{i=1}^{n} \mathrm{score}(P_i).$$

Let G_e denote the new grammar after adding the error-producing rules. We prove in Lemma 1 and Lemma 2 that a string s has a parse of minimum score l if and only if $d_G(G, s) = l$. Lemma 1 serves as a base case for Lemma 2.

Lemma 1. *For each string* s*,* $d_G(G, s) = 1$ *if and only if there is a derivation sequence in* G_e *to parse* s *with a score of* 1*, and none with score of* 0*. And* $d_G(G, s) = 0$ *if and only if there exists a derivation sequence in* G_e *to parse* s *with score of* 0*.*

Proof. $d_G(G, s) = 0$. Recall that all original productions of G have score 0 in G_e. For a string s with $d_G(G, s) = 0$, parsing in G_e can only use these original productions of G with a total score of 0. To prove the other direction, if we use any of the error-producing productions, then to produce a terminal, one must use one of the elementary error-producing rules with score 1. Hence, if G_e parses a string s with score 0, it must use only original productions of G. Hence $d_G(G, s) = 0$.

Claim 1. $d_G(G, s) = 1$ *if and only if the minimum score of any parsing of* s *in* G_e *is* 1.

Let us first prove the "only if" part. So, $d_G(G, s) = 1$. This single error has caused by either substitution or insertion or deletion.

First consider a single substitution error at the lth position. Let $s_l = b$. Suppose, if $s_l = a$ then $s \in \mathcal{L}(G)$. Considering $s_l = a$, obtain the left-most derivation of s in G. Stop at the step in the left-most derivation when a production of the form $A \to a$ is applied to derive $s_l = a$. By definition of left-most derivation, we have a parsing step $\alpha A \beta \to \alpha a \beta$, where $\alpha \in \Sigma^*$, $\beta \in (\Sigma \cup \mathcal{N})^*$. Use $A \to b$ with a score of 1 instead and keep all the remaining parsing steps identical to obtain a parse in G_e with score 1.

Now consider a single insertion error at the lth position, $l < n$, i.e., the string $s' = s_1 s_2 .. s_{l-1} s_{l+1} ... s_n$ is in $\mathcal{L}(\mathcal{G})$. Consider the left-most derivation of s' in G. Stop at the step when a production of the form $A \to s_{l+1}$ is used to produce the terminal s_{l+1}. To parse s in G_e, follow the parsing steps of s' in G till $A \to s_{l+1}$ is applied. Instead use $A \to IA$ with score 0, followed by $I \to s_l$ with score 1 and $A \to s_{l+1}$ with score 0. Now, continue the rest of the derivation steps of s' in G. Clearly, the score to parse s in G_e is 1. If $l = n$, then use $S \to SI$ as the first production, derive $s_1 s_2 ... s_{n-1}$ from S as in G and then use $I \to s_n$ with a score of 1 to complete the parsing of s in G_e.

Finally, consider a single deletion error. Suppose, the deletion happens right before the lth position, $l = 1, .., n+1$. Then for some $y \in \Sigma$, the string $s' = s_1 s_2 .. s_{l-1} y s_l .. s_n$ is in $\mathcal{L}(\mathcal{G})$. Consider a parse of s' in G. Consider the step in which a production of the form $B \to AC$ (or $B \to CA$) is used with subsequent application of $A \to y$ to produce the terminal y. Apply all the parsing steps of s', except $A \to y$, apply $A \to \varepsilon$ with score of 1.

This completes the "only if" part.

We now prove the "if" part. We have a parse $\pi(s) = P_1 P_2 ..... P_r$ of s of score 1 in G_e. Therefore, there exists exactly one P_i, $1 \leqslant i \leqslant r$ which is an elementary error-producing rule. If P_i is an elementary substitution rule $A \to y$ which is generated from the original rule $A \to x$, then replace in s the symbol y which P_i produces by x to map s to $\mathcal{L}(G)$. Hence, $d_G(G, s) = 1$. Similarly, if P_i is an elementary insertion rule $I \to y$, then delete y from s and remove P_i as well all productions (there can be at most one such production) of the form $A \to AI$ or $A \to IA$. The remaining productions in $\pi(s)$ uses only original rules and parse s after removal of symbol y in G. Hence $d_G(G, s) = 1$. If P_i is of the form $A \to \varepsilon$, then similarly modify P_i to $A \to x \in \mathcal{P}(G)$ to obtain a modified string $s' \in \mathcal{L}(G)$ with $\mathrm{dist}(s, s') = 1$. Thus, $d_G(G, s) = 1$. □

This completes the proof. □

The following corollary follows directly from the above lemma.

Corollary 2. *Given any string* $s \in \Sigma^*$, *there exists a nonterminal* $A \in \mathcal{N}(G)$ *such that* $A \overset{*}{\Rightarrow} s$ *with a parse score of* 1 *in* G_e *if and only if there exists a* $s' \in \mathcal{L}(G)$ *such that* $A \overset{*}{\Rightarrow} s'$ *with score of* 0 *in* G, *and* $\mathtt{dist}(s, s') = 1$.

Lemma 2. *Given any string* $s \in \Sigma^*$, *there exists a nonterminal* $A \in \mathcal{N}(G)$ *such that* $A \overset{*}{\Rightarrow} s$ *with a parse of score* l *in* G_e *if and only if there exists a string* $s' \in \Sigma^*$ *such that* $A \overset{*}{\Rightarrow} s'$ *in* G *(and* G_e*) with a parse of score* 0 *such that* $d(s, s') = l$. *If* $A = S$, *then* $d(G, s) = l$.

Proof. Corollary 3 serves as a base case when $l = 0$ and $l = 1$. Suppose the result is true up to $l - 1$.

Let us first assume that there exists nonterminal $A \in \mathcal{N}(G)$ such that $A \overset{*}{\Rightarrow} s'$ in G with a parse of score 0 and $d(s, s') = l$. Then to match s with s', it requires exactly l substitution, insertion and deletions in s. Consider the left-most edit position, correct it to obtain a string s'' such that $d(s'', s') = l - 1$. By induction hypothesis, there exists a parsing $\pi(s'')$ of s'' of score $l - 1$ starting from A. Now consider s, s'' and $\pi(s'')$ and depending on the type of error, follow exactly the same argument as in Lemma 1 to obtain a parse of s in G_e with a score just one more than $\pi(s'')$, that is, l starting from A.

We now prove the other direction. Let the parse for s in G_e, $\pi(s) = P_1P_2...P_r$, $P_1 = A$, has a score of l, then it uses exactly l elementary error-producing rules. Let P_i be the left-most elementary error-producing rule. Depending on the type of the rule, follow exactly the same argument as in Lemma 1 to modify P_i to obtain a string s'' such that $d(s, s'') = 1$, and s'' has a parsing in G_e with score exactly $l - 1$ starting from A. Therefore, by induction hypothesis, there exists $A \overset{*}{\Rightarrow} s'$ in G with a parse score of 0 such that $\mathtt{dist}(s', s'') = l - 1$. Hence, $d(s', s) \leqslant l$.

This establishes the lemma. $\square$

3.2 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 $R \geqslant 1$ elementary error-producing rules. We often use $\mathcal{N}, \mathcal{P}$ when in fact we mean $\mathcal{N}(G_e), \mathcal{P}(G_e)$ respectively. We start with a few definitions.

3.2.1 Definitions

Definition 3 (Operation-r)**.** *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 follows*

$$\begin{aligned} (A, u) *_r (B, v) &= (C, x) \text{ If } C \to AB \in \mathcal{P} \text{ and } x = u + v < r \\ &= \phi \text{ otherwise} \end{aligned} \tag{1}$$

Note the peculiarity of the above operation. $(A, u) *_r ((B, v) *_r (D, y))$ may well be different from $((A, u) *_r (B, v)) *_r (D, y)$. Therefore, the new binary operation is not associative. **This is important to note, since non-associativity of the above operation is the main source of difficulty for designing efficient algorithms.**

We omit r to keep the representation of the operator concise whenever its value is clear from the context.

Definition 4 (Elem-Mult)**.** *Given* $T_1 = \{(A_1, u_1), (A_2, u_2), ..., (A_k, u_k)\}$ *and* $T_2 = \{(B_1, v_1), (B_2, v_2), ..., (B_l, v_l)\}$ *for some* $k, l \in \mathbb{N} \cup \{0\}$, *define*

$$T = T_1.T_2 = \bigcup_{\forall i=1,2,..,k, \text{ and } j=1,2,...,l}^{\min} (A_i, u_i) * (B_j, v_j).$$

where $\bigcup^{\min}$ *implies if we have* $(C, x_1), (C, x_2), ..., (C, x_i)$ *involving the same nonterminal, we only keep* $(C, \min\{x_1, x_2, .., x_i\})$.

Clearly, $\mathtt{Elem\text{-}Mult}$ is also non-associative.

We can define some operation on matrices in terms of the above where matrix elements are collection of tuples $(A \in \mathcal{N}, u \in \{0, 1, 2, ..., r\})$. We define matrix multiplication $a.b = c$ where a and b have suitable sizes as follows

$$c(i, k) = \bigcup_{j=1}^{n} a(i, j).b(j, k) \tag{Matrix-Mult}$$

The *transitive closure* of a square matrix is defined as

$$a^+ = a_1 \cup a_2 \cup ...$$

where $a_1 = a$ and $a_i = \bigcup_{j=1}^{i-1} a_j.a_{i-j}$.

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, again because the "." operation is non-associative.

We will soon need to obtain a transitive closure of a matrix, where after each multiplication (possibly of two square submatrices (say) of dimension $m \times m$, $m \leqslant n$, we would need to perform some auxiliary tasks taking $O(m^2|G|^2)$ time. The underlying matrix multiplication will anyway need $\Omega(m^2|G|^2)$ time, thus the overall time for computing transitive closure will not change.

3.2.2 Algorithm

We are now ready to describe our algorithm.

Generating Deletion-r sets :D_r

We let $r = n$ and first generate several sets $D_1, D_2, D_3, D_4, ..., D_n$ where

$$D_1 = \begin{array}{ll} \{(A, 1) \mid A \to \varepsilon \in \mathcal{P}(G_e)\} \cup (S, 0) & \text{if } S \to \varepsilon \in \mathcal{P}(G) \\ \{(A, 1) \mid A \to \varepsilon \in \mathcal{P}(G_e)\} & \text{otherwise} \end{array}$$

We define $D_2 = D_1 \cup D_1.D_1$, and $D_i = D_{i-1} \cup D_{i-1}.D_{i-1}$. The following lemma establishes the desired properties of these sets.

Lemma 3. *For all* $A \in \mathcal{N}(G_e)$ *if* $A \overset{*}{\Rightarrow} \varepsilon$ *with a parse of score* u *then* $(A, u') \in D_u$ *where* $u' \leqslant u$ *and if* $(A, i) \in D_u$ *then* $A \overset{*}{\Rightarrow} \varepsilon$ *with a parse of score* i *(*i *may be larger than* u*). All the sets* $D_1, D_2, ..., D_n$ *can be computed in* $O(n|\mathcal{P}(G_e)|)$ *time.*

Proof. The claim is true for D_1 by construction. Suppose the claim is true till $i-1$ by induction hypothesis.

Now suppose $(A, u) \in D_i$ then either $(A, u) \in D_{i-1}$ or there exists $(B, v) \in D_{i-1}, (C, x) \in D_{i-1}$ such that $A \to BC \in \mathcal{P}(G)$, and $u = v + x$ which serves as an evidence why $(A, u) \in D_i$. By induction hypothesis $A \overset{*}{\Rightarrow} \varepsilon$ with a score of u in the first case; in the second case induction hypothesis applies to B and C, and by definition of "." operator, we have the desired claim for A.

On the other hand, if $A \overset{*}{\Rightarrow} \varepsilon$ with a score of $u \leqslant n$, then either $A \to \varepsilon$–in that case $A \in D_1 \subseteq D_s$, or $A \to BC \in \mathcal{P}(G)$ and each B and C derives ε with a score of u_1 and v_1 respectively where $u_1 \leqslant u-1, v_1 \leqslant$

$u-1$. Therefore, both B and C are in D_{u-1} and hence (A, u) is considered for inclusion in D_u. If (A, u) is not in D_u that only means there is some $u' < u$ and $(A, u') \in D_u$.

D_i can be calculated from D_{i-1} in $O(h)$ time where $h = |\mathcal{P}(G_e)|$. We go through the list of productions $A \to BC$ in G_e and check if both B and C are in D_{i-1} to generate A. For each nonterminal A thus generated we maintain its minimum score in $O(1)$ time. Taking union with D_{i-1} also taken only $O(h)$ time. In fact h only includes the productions of G and the elementary deletion rules of G_e. Hence, all the $D_1, D_2, ..., D_n$ can be computed in time $O(nh)$. □

Computing $\mathsf{TransitiveClosure}^+$

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.

Let the input string be $s = s_1 s_2 ... s_n \in \Sigma^*$.

1. First compute the $(n+1) \times (n+1)$ upper triangular matrix defined by

$$b(i, i+1) = \{(A_k, \mathrm{score}(A_k \to x_i)) \mid A_k \to x_i\}$$
$$b(i, j) = \phi \ \text{ for } j \neq i+1$$

2. Next, compute b^R, where for every entry $b^R(i, j)$ we follow the following subroutine.

   - $b^R(i, j) = b(i, j)$
   - FOR $\mathrm{count} = 1$ to $|G_e|$
     - $b^R(i, j) = b^R(i, j) \cup D_n.b^R(i, j) \cup b^R(i, j).D_n$
     - If there are multiple tuples involving same nonterminal $(A, u_1), (A, u_2), ..., (A, u_l), l \geqslant 1$ only keep $(A, \min\{u_1, u_2, .., u_l\})$
   - END FOR
   - Discard any (A, u) that appears with $u > R$.

   We could have used D_R instead of D_n here. But, since this subroutine will be repeatedly used in the final algorithm with D_n, we use D_n here to avoid later confusion.

3. Finally, compute the transitive closure of b^R but after each multiplication (possibly of two submatrices of dimension $m \times m$, $m < n$), multiply every entry $M(i, j)$ of the resultant $m \times m$ matrix by D_n $|G_e|$ times.

   - FOR $\mathrm{count} = 1$ to $|G_e|$
     - $M(i, j) = M(i, j) \cup D_n.M(i, j) \cup M(i, j).D_n$
     - If there are multiple tuples involving same nonterminal $(A, u_1), (A, u_2), ..., (A, u_l), l \geqslant 1$ only keep $(A, \min\{u_1, u_2, .., u_l\})$
   - END FOR
   - Discard any (A, u) that appears with $u > R$.

   Each iteration of the for loop takes time $O(|G_e|)$. Hence overall time for updating each matrix entry is $O(|G_e|^2)$. Therefore, total time taken for this auxiliary operation is $O(m^2|G_e|^2)$. Since, each entry of M may contain $|G_e|$ nonterminals, assuming each product takes $\Omega(m^2|G_e|^2)$ (which indeed will be the case) time, adding this auxiliary operation does not change the time to compute the overall transitive closure. To distinguish it from normal transitive closure, we call it $\mathsf{TransitiveClosure}^{+}$. Therefore, this step computes $M = \mathsf{TransitiveClosure}^{+}(b^R)$.

The following lemma proves the correctness.

Lemma 4. $(A_k, u) \in M(i, j)$, *if and only if* $A_k \overset{*}{\Rightarrow} s_i^{j-1}$ *in* G_e *with a parse score of* $u \leqslant R$, *and there does not exist any* $u' < u$ *such that* $A_k \overset{*}{\Rightarrow} s_i^{j-1}$ *with parse score of* u'.

Proof. The proof is by an induction on length of s_i^{j-1}.

Base case. Note that b is upper triangular. This implies $M(i, i+1) = b^R(i, i+1)$. Now by construction of b, $M(i, i+1)$ contains all nonterminals that derive x_i either with score 0 or by a single insertion error, or by a single substitution error. The only case remains when x_i may be derived by deletion of symbols either on the right, or on the left, or both.

Consider a derivation of x_i which involves deletion error. Then the first production used to derive x_i must be of the form $C_0 \to B_1C_1$ for $C_0, B_1, C_1 \in \mathcal{N}(G_e)$ and among B_1 and C_1 one must derive ε. If $B_1 \Rightarrow \varepsilon$ and $C_1 \to x$ or $C_1 \Rightarrow \varepsilon$ and $B_1 \to x$, then on the first multiplication by D_n, C_0 will be included in b^R if the total score of it is less than R. Otherwise, w.l.o.g say $B_1 \Rightarrow \varepsilon$, and $C_1 \to B_2C_2$. Note that $C_1 \neq C_0$ because otherwise, it is sufficient to consider derivation from C_1. One of B_2 or C_2 must derive ε, if the other directly derives x_1, then both C_0 and C_1 will be included in b^R after two successive multiplications by D_n. If none of B_2 or C_2 derives ε, then again w.l.o.g we can assume $B_2 \Rightarrow \varepsilon$ and $C_2 \to B_3C_3$ where again $C_2 \neq C_1 \neq C_0$–this process can continue at most $|G_e|$ steps. Thereby, after $|G_e|$ successive multiplications with D_n, $M(i, i+1)$ contains all the nonterminals that derive s_i with a score of at most R.

Induction Hypothesis. Suppose, the claim is true for all substrings of length up to l.

Induction. Consider a substring of length $l+1$, say, s_i^{i+l+1}. If $M(i, i+l+2)$ contains A_k, then either

(1) there must be some j, $i < j < i+l+2$ (because M is upper triangular) such that $M(i, j)$ contains a (B_k, u_1) and $M(j, i+l+2)$ contains a (C_k, u_2) such that $A_k \to B_kC_k$ and $u_1 + u_2 < R$, or

(2) A_k is included in $M(i, i+l+2)$ due to multiplications by D_n.

For (1) by induction hypothesis B_k derives s_i^{j-1} with a parse of score u_1 and C_k derives s_j^{i+l+1} using a parse of score u_2. Since $A_k \to B_kC_k$ has a score of 0, A_k derives s_1^{i+l+1} using a parse of score $u_1+u_2 < R$. For (2) there must exist a $(B, u_1) \in M(i, i+l+2)$, $B \overset{*}{\Rightarrow} s_i^{i+l+1}$ with score of u_1 and a derivation sequence starting from A_k where each production contains two nonterminals on RHS with one of them producing ε and the derivation sequence terminates when B is generated. In that case again, $A_k \overset{*}{\Rightarrow} s_i^{i+l+1}$ and is included if and only if the total score of the derivation sequence including the score for B is at most R.

For the other direction, note that for every index j, $i < j < i+l+2$, $M(i, j)$ and $M(j, i+l+2)$ contain all the nonterminals that derive s_1^{j-1} and s_j^{i+l+2} respectively within a parse score of at most R. Hence, by the definition of $\mathsf{TransitiveClosure}^{+}$, if $(B_k, s_1) \in M_{i,j}, B_k \overset{*}{\Rightarrow} s_i^{j-1}$ with a parse score of u_1 and $(C_k, s_2) \in M(j, i+l+2), C_k \overset{*}{\Rightarrow} s_j^{i+l+1}$ with a parse score of u_2 and we have $u = u_1 + u_2 < R$ and $A_k \to B_kC_k \in \mathcal{P}$, then $(A_k, u) \in M(j, i+l+2)$. Therefore, $M(j, i+l+2)$ contains all the nonterminals that derive s_i^{i+l+1} by decomposition (that is the first production applied to derive s_i^{i+l+1} has two nonterminals on the RHS, none of which produces ε) with a score of at most R (even before multiplication with D_n).

The only case remains when there is a derivation tree starting from A to derive s_i^{i+l+1} by application of a rule of the form $A \to BC$, where either B or C produces ε. We grow the derivation tree for each nonterminal that does not produce ϵ, and stop as soon as the current production (say) $X \to YZ$ considered generates two nonterminals Y, Z none of which produces ϵ. We know (X, u') must be in $M(i, i+l+2)$ if $u' \leqslant R$. Then we argue as in the base case for incorporating deletion errors that (A, u) must be included in $M(j, i+l+2)$, if $u \leqslant R$. Note that $u > u'$, hence if $(X, u') \notin M(j, i+l+2)$, (A, u) cannot be in $M(j, i+l+2)$. □

Corollary 3. *For all* $A_k \in \mathcal{P}(G_e)$, $A_k \in M(i, j)$, *if and only if* $A_k \overset{*}{\Rightarrow} s_i^{j-1}$ *in* G_e *with score of at most* R *such that* $\exists t' \in \mathcal{L}(G)$ *with* $d(s_i^{j-1}, t') \leqslant R$ *and* $A_k \overset{*}{\Rightarrow} t'$ *with a score of* 0.

Proof. The proof follows from Lemma 4 and Lemma 2. □

3.2.3 Running Time Analysis

Therefore, the complete information of all the substrings of s that can be derived from a nonterminal in G_e within a parse of score of R is available in M. We now calculate the time needed to compute M. The analysis goes in two steps.

• Step 1. Show that time needed to compute M ($\mathsf{TransitiveClosure}^+(b^R)$) is asymptotically same as the time needed to compute a single $\mathsf{Matrix\text{-}Mult}$.

• Step 2. Show that time needed to compute a single $\mathsf{Matrix\text{-}Mult}$ 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.

Step 1. Reduction from $\mathsf{TransitiveClosure}^+$ **to** $\mathsf{Matrix\text{-}Mult}$ This reduction falls off directly from the proof of Valiant [48]. We briefly explain it here and detail the necessary modifications.

First of all note that computing $\mathsf{TransitiveClosure}^+$ has same asymptotic time as computing $\mathsf{TransitiveClosure}$. Therefore, we can just focus on the time complexity of $\mathsf{TransitiveClosure}$ computation of b^R.

We start with b^R. Recall our definition of $\mathsf{Elem\text{-}Mult}$. Suppose, we do not consider the second component, that is score when performing $\mathsf{Elem\text{-}Mult}$. So given (A, s) and (B, t), if $\exists C \to AB$, when multiplying (A, s) and (B, t), we generate C irrespective of score $s+t$. Let us refer it as $\mathsf{Elem'\text{-}Mult}$. This is exactly the situation handled by Valiant. In Valiant's own words, "*several analogous procedures for the special case of Boolean matrix multiplication are known .... However, these all assume associativity, and are therefore not applicable here. Instead of the customary method of recursively splitting into disjoint parts, we now require a more complex procedure based on "splitting with overlaps". Fortunately, and perhaps surprisingly, the extra cost involved in such a strategy can be made almost negligible.*"

Valiant's Proof Sketch.

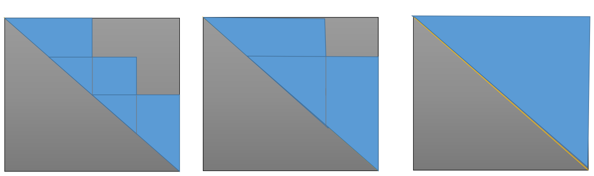


Figure 1: Decomposition into overlapping parts. While Valiant's Method computes transitive closure in asymptotically same time as single matrix multiplication, there could be cells in the matrix that takes part in $O(n)$ multiplication steps.

Let c be an upper triangular matrix. Define $c^{+(r:s)}$ to be the result of the following operations (i) collapse c by removing all elements $c(i, j)$ where $r < i \leqslant s$ and $r < j \leqslant s$, (ii) compute the transitive closure of the $(n+r-s) \times (n+r-s)$ matrix, and (iii) expand the matrix back to its original size by restoring the elements that were removed to their original position.

Valiant's result is based on the following observation. If the submatrices of c specified by by

$[1 \leqslant i, j \leqslant s]$, and $[r < i, j \leqslant n]$ are both already transitively closed, and $s \geqslant r$ then

$$c^{+} = (c \cup (c.c))^{+(r:s)} \tag{Valiant-Eq}$$

This expresses the facts that to obtain c^{+} we just need to multiply c once and then consider the submatrix $[1 \leqslant i \leqslant r, s < j \leqslant n]$. This gives a divide-and-conquer approach and leads to the following theorem. Recall again, we are working with a modified version $\mathsf{Elem'}$-Mult of Elem-Mult.

Theorem (Theorem 2 [48].). *Let $m(n)$ denote the time required to compute a single matrix multiplication of n-dimensional square matrices and $t(n)$ the time required to compute the transitive closure. If $m(n) \geqslant n^{\gamma}, \gamma \geqslant 2$, then*

$$t(n) = m(n) + O(n^2) \quad \text{if } \gamma > 2$$

$$t(n) = m(n) \log n + O(n^2) \quad \text{if } \gamma = 2.$$

To incorporate Elem-Mult is trivial. The above theorem and the relation $\mathsf{Valiant}$-Eq do not change at all, only $m(n)$ is replaced by $m'(n)$ which now represents the time needed for our Matrix-Mult operation, and not the one used by Valiant. Since $\mathsf{TransitiveClosure}^{+}$ has same asymptotic time as $\mathsf{TransitiveClosure}$, we get the following lemma.

Lemma 5. *Let $m'(n)$ and $t'(n)$ denote the time required to compute a single matrix multiplication of n dimensional square matrices according to Matr-Mult and $\mathsf{TransitiveClosure}^{+}$ respectively. If $m'(n) \geqslant n^{\gamma}, \gamma \geqslant 2$ then*

$$t'(n) = m'(n) + O(n^2) \quad \text{if } \gamma > 2$$

$$t'(n) = m'(n) \log n + O(n^2) \quad \text{if } \gamma = 2.$$

Reduction from Matrix-Mult to Distance Production Computation. This reduction is simple.

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

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

To do so, we compute two real matrices a' of dimension (hn, n) and b' of dimension (n, 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 h consecutive rows for each row index p of a. For each $(A_i, x) \in a(p, q), \forall p, q \in \{1, 2, .., n+1\}$ we set $a'((p-1)h + i, q) = x$.

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

We compute the distance product c' of a' and b' in time $O(h^2 T(n))$.

To construct c from c', consider all the rows $(p-1)h+1 \leqslant i \leqslant ph$, and all the columns $(q-1)h+1 \leqslant j \leqslant qh$, if $c'(i, j) = z$ then if $i - (p-1)h = x$, $j - (q-1)h = y$, $B \to A_x A_y \in \mathcal{P}(G_e)$, generate (B, z) 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 $c(i, j)$, we go through this list and for each production $B \to A_x A_y$ we check appropriate cells for A_x and A_y to obtain the score for B. Therefore, $c(i, j)$ can be constructed in time $O(h)$. Therefore, total time to compute c' is $O(h^2 T(n))$ and time to compute c from

c' is $O(hn^2)$. That the reduction is correct follows directly from the correctness proof of reducing distance product to ordinary matrix product computation [4, 47], and the way the matrices are computed. The details are simple and left to the reader.

Overall this takes $O(|G_e|^2 T(n))$ time.

If we set $R = n$ and each $\mathsf{Matrix\text{-}Mult}$ takes time $O(|G_e|^2 T(n))$ and $\mathsf{TransitiveClosure}^+$ can also be computed in $O(|G_e|^2 T(n))$ time (assuming $T(n) = o(n^2)$, if $T(n) = O(n^2)$, an additional $\log n$ term will be added in the running time. Therefore, we get the following proposition.

Proposition 1. *Given a grammar* $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$*, and a string* $s \in \Sigma^*$*, language edit distance* $d_G(G, s)$ *can be computed exactly in* $O(|G|^2 T(n))$ *time.*

Proof. Set $R = n$, and if $(S, x) \in M(1, n+1)$ return x. Note that (S, n) is a candidate for inclusion in $M(1, n+1)$ where S generates s by all substitutions. Hence, there is an entry with nonterminal S in $M(1, n+1)$. By Lemma 4 $d_G(G, s) = x$. The total time taken is $O(|G_e|^2 T(n))$. Since $|G_e| = O(|G|)$, we get the desired bound in the running time. □

With little bookkeeping, the entire parsing information for s can also be stored. We elaborate on that in the appendix.

Reduction from $\mathsf{Matrix\text{-}Mult}$ to Ordinary Matrix Multiplication. $\mathsf{Matrix\text{-}Mult}$ 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.

We use the reduction of distance product computation to ordinary matrix multiplication by Alon, Galil and Margalit [4] and Takaoka [47]. 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 $O(h^2 n^{\omega})$. 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 $B \to A_x A_y$ 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 [4, 47], and the way the matrices are computed. The details are simple and left to the reader.

Therefore a single $\mathsf{Matrix\text{-}Mult}$ operation can be done in time $O(R\omega(nh, n, nh))$. Together with Lemma 5 this proves that parsing strings with score at most R (and the complete parsing information of all its substrings) can be obtained in $O(R\omega(nh, n, nh) + h^2 n^2)$ time.

Lemma 6. *Given* G_e *and string* $s \in \Sigma^*$*, one can compute a* $(n+1) \times (n+1)$ *matrix* M *in* $O(R\omega(n|G_e|, n, n|G_e|) + |G_e|^2 n^2)$ *time such that its* (i, j)*th entry contains all nonterminals that can derive the substring* s_i^j *with a parse of score at most* R.

3.3 $\mathsf{Matrix\text{-}Mult}$ 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, $\mathsf{Matrix\text{-}Mult}$ can be solved in $O(|G|^2 R n^{\omega} + |G|^2 R n^2)$ time. We extend this to handle the case when there could be R distinct values in the matrix, but with arbitrary values.

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. Much better constructions matching $O(k^2)$ bound are known due to Ruzsa, Bose, Singer (for comprehensive literature survey see [38]).

Our Construction. Given R distinct values $v_1 < v_2 < v_3 < ... < v_R$ that may appear during $\mathsf{Matrix\text{-}Mult}$, 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. Then using the map $f^-(g_i)$ and $f^-(g_j)$, we can identify the original sum $f^-(g_i)+f^-(g_j)$. Using the known construction of Sidon sequences, $g_R = O(R^2)$.

By Lemma 6 $\mathsf{Matrix\text{-}Mult}$ can be performed in $O(|G|^2R^2n^\omega + |G|^2R^2n^2)$ time if the values are bounded by R^2. We first use the map f to bound all the values within $O(R^2)$, then compute $\mathsf{Matrix\text{-}Mult}$ in $O(|G|^2R^2n^\omega + |G|^2R^2n^2)$ time and finally use the look-up table and f^- mapping to obtain the original values. This last step takes at most $O(n^2 \log R)$ time.

Therefore, overall the time complexity of $\mathsf{Matrix\text{-}Mult}$ when there are at most R distinct parsing scores is $O(|G|^2R^2n^\omega + |G|^2R^2n^2)$.

Remark.

- The construction by Yuster [55] cannot be used here. Even for $R = O(1)$ their construction yields an $O(n^{2.5})$ algorithm (see Lemma 3.3 [55]).

- It is possible to reduce parsing with R distinct scores to boolean matrix multiplication, rather than ordinary matrix multiplication with nearly same bound.

3.4 Final Algorithm

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

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

After every $\mathsf{Matrix\text{-}Mult}$ (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

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

Clearly, the number of distinct parsing scores that can appear on any tuple (A, y) during the computation of $\mathsf{TransitiveClosure}^{++}$ is bounded by $\lceil \log_{(1+\delta)} n \rceil$. Multiplying two submatrices of dimension $m \times m$ requires time $\Omega(|G_e|^2m^2)$, whereas this auxiliary Round operation can be performed in $O(m^2)$ time. Hence overall the asymptotic time taken is not affected due to Round.

Before multiplying any two submatrices of (say) dimension $m \times m$, map $R = \lceil \log_{1+\delta} n \rceil$ possible parsing scores to $O(R^2)$ Sidon sequences as discussed in Section 3.3. And, after the $\mathsf{Matrix\text{-}Mult}$ is completed infer the actual sum from the inverse mapping from Sidon sequence to original (possibly previously rounded) values.

Hence, overall the time to compute $\mathsf{TransitiveClosure}^{++}(M^C, R)$ is same as $\mathsf{TransitiveClosure}^{+}(M^C, R^2)$. The blow-up from R to R^2 comes from mapping to Sidon sequences and inverse mapping to original sequences before and after each $\mathsf{Matrix\text{-}Mult}$.

Starting from M^C we repeat the process of computing $\mathsf{TransitiveClosure}^{++}(M^C, R)$ $\eta = 6\log n$ 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 $S \rightarrow SI$, $I \rightarrow II$ and $I \rightarrow x, x \in \Sigma$, 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. We elaborate on it in the appendix.

3.4.1 Analysis

While the running time bound has already been established, we now analyze the performance of the above algorithm in terms of approximating language edit distance.

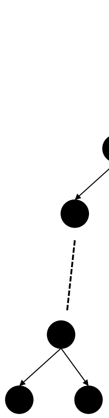
Consider any $\mathcal{M}_k$ and let us abuse notation and use $\mathcal{M}$ to denote it. Consider any substring $s' = s_i^{j-1}$. Given a parse tree $\mathcal{P}$, we say $\mathcal{M}$ retains $\mathcal{T}$ if for every intermediate nonterminal A of $\mathcal{T}$ deriving some substring $s_{i'}^{j'-1}$ the corresponding entry $(A, x) \in \mathcal{M}(i,j)$ contains the actual estimate 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 $\hat{e_O} \geqslant \hat{e_P}$ [3].

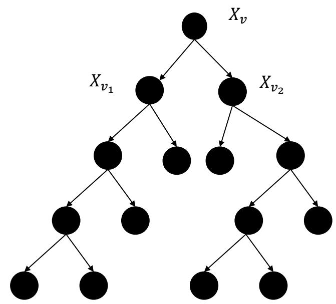
We now show that $\hat{e_P} \in [(1-\epsilon)e_P, (1+\epsilon)e_P]$ and similarly $\hat{e_O} \in [(1-\epsilon)e_O, (1+\epsilon)e_O]$ with high probability. Therefore,

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

[3]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 $\hat{\hat{e_O}}$ 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 $\hat{e_P} \leqslant \hat{\hat{e_O}} \leqslant \hat{e_O}$



(a) An extreme parse tree showing the depth of parsing can be as high as O(n)



(b) A parse tree and associated random variables. A parent node is formed from two children either by the result of *Matrix-Mult* or by multiplying an intermediate matrix with D_n. The random variables indicate the scores after *Round* operation.

We now prove $\mathsf{ApproxEstimate}$.

Lemma 7. $\mathsf{E}[\hat{e_P}] = e_P$.

Proof. Consider the parse tree P and let P' denote the truncated parse tree after the algorithm completes execution of $\mathsf{TransitiveClosure}^+(b^C, C)$. The leaves of P' denote the exact parsing scores corresponding to the associated nonterminals by Lemma 4 and due to exact computation of D_n. Hence, if $l_1, l_2, ..., l_m$ are the leaves of P' and $\mathsf{score}(l_i)$ is the score computed by $\mathsf{TransitiveClosure}^+(b^C, C)$ or D_n, then $e_P = \sum_i \mathsf{score}(l_i)$.

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 $\mathsf{score}(v) = (1+\delta)^r + k$ for some $r \in \{0, 1, 2, ..., \lceil \log_{(1+\delta)} n \rceil\}$ and $k \in [0, \delta(1+\delta)^r)$. Then

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

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.

Claim 2. $\mathsf{E}[\mathcal{X}_v] = \mathsf{E}[\mathcal{X}_{v_1}] + \mathsf{E}[\mathcal{X}_{v_2}]$.

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

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

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

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

$$\mathsf{E}[\mathcal{X}_v] = \sum_x x \mathsf{Pr}[X_v = x]$$

$$
\begin{aligned}
&= \sum_y \sum_x x \mathsf{Pr}\left[\mathcal{X}_v = x \mid \mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \\
&= \sum_{y = \lfloor y \rfloor_\delta} y \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \\
&+ \sum_{y \neq \lfloor y \rfloor_\delta} \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \Big(\lfloor y \rfloor_\epsilon \mathsf{Pr}\left[\mathcal{X}_v = \lfloor y \rfloor_\epsilon \mid \mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] + \lceil y \rceil_\epsilon \mathsf{Pr}\left[\mathcal{X}_v = \lceil y \rceil_\epsilon \mid \mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \Big) \\
&= \sum_y y \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] \quad \text{By same calculation as Eqn. 2} \\
&= \mathsf{E}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2}\right] = \mathsf{E}\left[\mathcal{X}_{v_1}\right] + \mathsf{E}\left[\mathcal{X}_{v_2}\right] \quad \text{By linearity of expectation} \\
&= \sum_{l \in L(v)} \mathtt{score}(l) \quad \text{since } L(v_1) \text{ and } L(v_2) \text{ are obviously disjoint} \quad \square
\end{aligned}
$$

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

Corollary 4. $\mathsf{E}\left[\hat{e_O}\right] = e_O$.

Proof. Follows using the same argument as in Lemma 7. $\square$

We use second moment method to bound the deviation of our estimate from expectation. Variance calculation is complicated and needs to be done with care. We use the same notation of $\mathcal{X}_v, \mathcal{T}_v, L(v), \lfloor y \rfloor_\delta, \lceil y \rceil_\delta$ etc.

Lemma 8. $\mathsf{Var}\left[\mathcal{X}_v\right] \leqslant \mathsf{Var}\left[\mathcal{X}_{v_1}\right] + \mathsf{Var}\left[\mathcal{X}_{v_2}\right] + \delta\Big(\sum_{g \in L(v_1)} \mathtt{score}(g)\Big)\Big(\sum_{h \in L(v_2)} \mathtt{score}(h)\Big)$ *if v is an intermediate node, and if v is a leaf node then* $\mathsf{Var}\left[\mathcal{X}_v\right] = (\mathtt{score}(v) - \lfloor \mathtt{score}(v) \rfloor_\delta)(\lceil \mathtt{score}(v) \rceil_\delta - \mathtt{score}(v))$.

Proof. First consider the case when v is a leaf node. 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

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

Hence,

$$
\begin{aligned}
\mathsf{Var}\left[\mathcal{X}_v\right] &= \mathsf{E}\left[\mathcal{X}_v^2\right] - (\mathsf{E}\left[\mathcal{X}_v\right])^2 \\
&= (1+\delta)^r((1+\delta)^r + k(2+\delta)) - (1+\delta)^{2r} - k^2 - 2k(1+\delta)^r \\
&= k(\delta(1+\delta)^r - k) \\
&= (\mathtt{score}(v) - \lfloor \mathtt{score}(v) \rfloor_\delta)(\lceil \mathtt{score}(v) \rceil_\delta - \mathtt{score}(v)) \qquad (3)
\end{aligned}
$$

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

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

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

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

Therefore,

$$
\begin{aligned}
&\mathsf{Var}\left[\mathcal{X}_v\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) + \sum_y y^2 \mathsf{Pr}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2} = y\right] - \left(\mathsf{E}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2}\right]\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] - \left(\mathsf{E}\left[\mathcal{X}_{v_1} + \mathcal{X}_{v_2}\right]\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\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] \left((1+\delta) \lfloor y \rfloor_\delta - y \right)\left(y - \lfloor y \rfloor_\delta \right) + \mathsf{Var}\left[\mathcal{X}_{v_1}\right] + \mathsf{Var}\left[\mathcal{X}_{v_2}\right]
\end{aligned}
$$

$$=\Big(\sum_{y\neq\lfloor y\rfloor_\delta}(\lceil y\rceil_\delta - y)(y-\lfloor y\rfloor_\delta)\mathsf{Pr}\left[\mathcal{X}_{v_1}+\mathcal{X}_{v_2}=y\right]\Big)+\mathsf{Var}\left[\mathcal{X}_{v_1}\right]+\mathsf{Var}\left[\mathcal{X}_{v_2}\right] \tag{a}$$

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

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

Now using Lemma 7 we get

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

This establishes the lemma.

□

Lemma 9. $\mathsf{Var}\left[\mathcal{X}_v\right]\leqslant\sum_{l\in L(v)}\mathsf{Var}\left[\mathcal{X}_l\right]+\delta\Big(\sum_{g,h\in L(v),g<h}\mathtt{score}(g)\mathtt{score}(h)\Big)$.

Proof. We use the tree $\mathcal{T}(v)$ rooted at v. We start from v and Lemma 8

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

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,

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

We now bound the second term of the above equation.

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

$$
\begin{aligned}
\mathsf{E}\left[\Big(\sum_{l\in L(v)} \mathcal{X}_l\Big)^2\right] &= \mathsf{E}\left[\bigg(\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big) + \Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)\bigg)^2\right] \\
&= \mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)^2 + \Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)^2 + 2\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)\right] \\
&= \mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)^2\right] + \mathsf{E}\left[\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)^2\right] + 2\mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)\right] \\
&\qquad \text{by linearity of expectation} \\
&= \mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)^2\right] + \mathsf{E}\left[\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)^2\right] + 2\mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)\right]\mathsf{E}\left[\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)\right] \\
&\qquad \text{since } \Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big) \text{ and } \Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big) \text{ are independent} \\
&= \mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)^2\right] + \mathsf{E}\left[\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)^2\right] + 2\Big(\sum_{g\in L(v_1)} \mathsf{E}\left[\mathcal{X}_g\right]\Big)\Big(\sum_{h\in L(v_2)} \mathsf{E}\left[\mathcal{X}_h\right]\Big) \\
&\qquad \text{again by linearity of expectation} \\
&= \mathsf{E}\left[\Big(\sum_{g\in L(v_1)} \mathcal{X}_g\Big)^2\right] + \mathsf{E}\left[\Big(\sum_{h\in L(v_2)} \mathcal{X}_h\Big)^2\right] + 2\Big(\sum_{g\in L(v_1)} \mathsf{score}(g)\Big)\Big(\sum_{h\in L(v_2)} \mathsf{score}(h)\Big) \text{ by Lemma 7}
\end{aligned}
$$

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

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

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

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

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 7 we get

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

□

Lemma 10. *Let* $\delta' \geqslant 2\sqrt{\delta}$, $\mathsf{Pr}\left[|e_P - \hat{e_P}| > \delta'\right] \leqslant \frac{1}{4}$.

Proof. From Lemma 8 for any leaf node l

$$\begin{aligned}\mathsf{Var}\left[\mathcal{X}_l\right] &= (\mathrm{score}(v) - \lfloor \mathrm{score}(v) \rfloor_\delta)(\lceil \mathrm{score}(v) \rceil_\delta - \mathrm{score}(v)) \text{ (see Eqn. 3)} \\ &\leqslant \mathrm{score}(v)(\lceil \mathrm{score}(v) \rceil_\delta - \lfloor \mathrm{score}(v) \rfloor_\delta) = \delta \mathrm{score}(v) \lfloor \mathrm{score}(v) \rfloor_\delta \leqslant \delta \mathrm{score}(v)^2\end{aligned}$$

Therefore, by Lemma 9

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

Hence, by Chebyshev's inequality

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

□

Corollary 5. *Let* $\delta' \geqslant 2\sqrt{\delta}$, $\mathsf{Pr}\left[|e_O - \hat{e_O}| > \delta'\right] \leqslant \frac{1}{4}$.

Proof. By argument same as Lemma 10. □

Lemma 11. $\mathsf{Pr}\left[\frac{e_O}{(1+\epsilon/2)} \leqslant \hat{e_P} \leqslant (1+\epsilon)e_O\right] > \frac{1}{2}$.

Proof. We have $\delta = \epsilon^2/16$. Hence $\delta' = \epsilon/2$, or $\epsilon = 2\delta'$. Now the lemma follows from Eqn. $(\mathrm{ApproxEstimate})$, Lemma 10 and Corollary 5. □

Lemma 12. *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. □

Theorem (1)**.** *Given any arbitrary context-free grammar* $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$*, 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.*

- *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 $i, j \in \{1, 2, ..., n\}$ *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.*

4 Stochastic Context Free Grammar Parsing

Definition 5. *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*
  - $p(\alpha \to \beta) > 0$ *for all* $\alpha \to \beta \in \mathcal{P}$
  - $\sum_{\alpha \to \beta \in \mathcal{P} : \alpha = X} p(\alpha \to \beta) = 1$ *for all* $X \in \mathcal{N}$

  $p(\alpha \to \beta)$ *can be seen 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*

$$\mathsf{Pr}\left[\pi(s)\right] \prod_{i=1}^{l} p(P_i)$$

Stochastic context free grammars lie the foundation of statistical natural language processing, they generalize hidden Markov models, and are ubiquitous in computer science. A basic question regarding SCFG is parsing, where given a string $s \in \Sigma^*$, we want to find the most likely parse of s.

$$\arg\max_{\pi(s)} \mathsf{Pr}\left[\pi(s) \mid \; s, (G, p)\right]$$

The CYK algorithm for context free grammar parsing also provides an $O(|G|n^3)$ algorithm for the above problem. As noted in [3, 56], Valiant's framework for fast context free grammar parsing can be employed to shed off a polylogarithmic factor in the running time. Indeed, SCFG parsing can be handled by Valiant's framework when each $\mathtt{Matrix\text{-}Mult}$ is again equivalent to a distance product computation– thus a $O(|G|^2 T(n))$ algorithm follows. Instead of computing the costly distance product, if we follow our algorithm for language edit distance computation, that directly gives an $\tilde{O}(|G|^2 n^\omega \log\log \frac{1}{p_{max}})$ algorithm to compute a parse $\pi'(s)$ such that

$$|\log \mathsf{Pr}\left[\pi'(s)\right]| \geqslant (1 - \epsilon)|\log \mathsf{Pr}\left[\pi(s)\right]| \qquad \text{(Approx-SCFG)}$$

where $\pi(s)$ is the most likely parse of s.

To the best of our knowledge, this is the first algorithm for parsing SCFG near-optimally in sub-cubic time. [3] claims an $\tilde{O}(n^{2.976} + \frac{1}{\epsilon^{O(1)}})$ algorithm for SCFG parsing with the above $(1-\epsilon)$ approximation in the context of RNA secondary structure prediction, but it only works for very restrictive class of probability distributions[4].

To obtain the desired bound of $\mathsf{Approx\text{-}SCFG}$ some modifications to our language edit distance algorithm and analysis (Section 3) are required.

1. For each production $P \in \mathcal{P}(G)$ we assign a score, $\mathsf{score}(P) = \log \frac{1}{p(P)}$. Then if $\pi(s)$ maximizes $\Pr[\pi(s)]$, it must minimize $\mathsf{score}(\pi(s))$ where recall $\mathsf{score}(\pi(s)) = \sum_{P \in \pi(s)} \mathsf{score}(P)$.

2. We modify Definition of Operation-r (Eq. 3) as follows

$$\begin{aligned}(A,u) *_r (B,v) \quad &= (C,x) \text{ If } C \to AB \in \mathcal{P} \text{ and } x = u + v + p(C \to AB) < r \\ &= \phi \text{ otherwise}\end{aligned} \tag{4}$$

   To compute $\mathsf{Matrix\text{-}Mult}$ of two $m \times m$ matrix under this new operation, we simply follow the $\mathsf{Matrix\text{-}Mult}$ algorithm from Section 3.2.3 and generate the tuples $A, B, (u+v)$ from $(A,u) *_r (B,v)$. Next, we go through the entire list of productions of the form $C \to AB$ as in Section 3.2.3, and generate $(C, u + v + \mathsf{score}(C \to AB))$.

3. We only use G, no error-producing rules, or generating D_n set and multiplying by it while computing transitive closure. Essentially, we do not require $\mathsf{TransitiveClosure}^+$, and $\mathsf{TransitiveClosure}$ computation of b matrix (see Section 3.2.2) suffices for exact computation of SCFG parsing problem. R needs to be set to $n \max_{P \in \mathcal{P}} \log \frac{1}{p(P)}$. Proof of Lemma 4 is way simpler, since there is no multiplication with D_n happening, in fact, it falls off directly as we are computing transitive closure of b matrix.

4. For approximate computation, we follow the final algorithm (Section 3.4). We set $R = \lceil \log_{(1+\delta)} (n \max_{P \in \mathcal{P}} \log \frac{1}{p(P)}) \rceil$, and compute $\mathsf{TransitiveClosure}^{++}$ (note again there is no multiplication by D_n, and no G_e, our $\mathsf{Elem-Mult}$ operation is slightly different due to modified Definition 3). The possible values of scores are $1, (1+\delta), (1+\delta)^2, \ldots., (1+\delta)^{\lceil \log_{(1+\delta)} (n \max_{P \in \mathcal{P}} \log \frac{1}{p(P)}) \rceil}$. We map these R scores to an $O(R^2)$ Sidon sequences. Now each $\mathsf{Matrix\text{-}Mult}$ can be computed cheaply in $O(R^2|G|^2 n^{\omega} + R^2|G|^2 n^2)$ time. We use the same Round operation, but after each $\mathsf{Matrix\text{-}Mult}$ operation when we generate $A, B, u+v$, we first apply Round on $(u+v)$, then Round on $\mathsf{score}(C \to AB)$ and finally Round on $\mathsf{Round}(u+v) + \mathsf{Round}(\mathsf{score}(C \to AB))$. While only applying Round on $u + v + \mathsf{score}(C \to AB)$ is sufficient, this keeps the analysis identical to Section 3. Note that now in the analysis of the final algorithm, not only leaves have scores, but also intermediate nodes. It is easy to get away with that.

   We add dummy nodes to the original parse tree as in Figure 2 so that in the modified tree, scores are only on leaves. Now, we are in identical situation as Section 3.4.1, and the same analysis applies.

[4]We could not verify the claim Theorem 8 of [3] of an $\tilde{O}(n^{2.976} + \frac{1}{\epsilon^{O(1)}})$ algorithm. For approximation guarantee, Theorem 8 refers to Lemma 4 which refers to Lemma 2 that highly restricts the probability assignment to productions. For example, if a parse tree applies the production $A \to BC$, followed by $C \to DE$ where each B, D, E produces some terminals, then $p(A \to BC)p(C \to DE) \leqslant \frac{1}{4}$. Not only, our running time is much better, we do not have any restriction on the probability distribution associated with a SCFG.

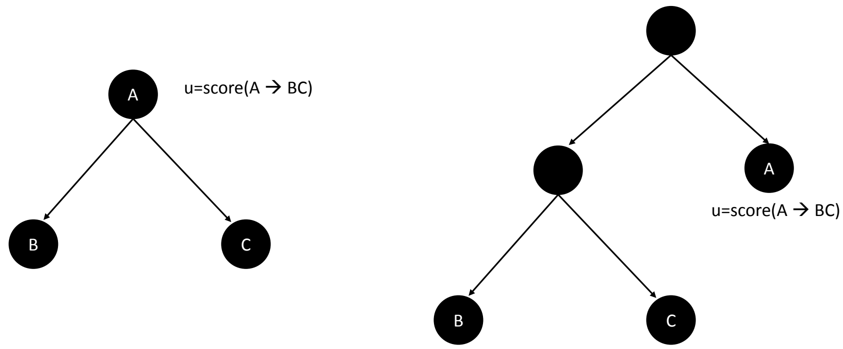


Figure 2: Converting parse tree so that only leaves have scores.

5. For a substring s_i^j if $M(i, j+1)$ does not contain any entry with nonterminal S, then we declare $s_i^j \notin \mathcal{L}(G)$ or equivalently its most likely parse has score 0. Otherwise, we compute the median estimate $d_{med}^{i:j}$ with respect to S, and return $\frac{1}{2^{d_{med}^{i:j}}}$ to convert back to probability of the parsing from its computed score.

The bound of APPROX-SCFG now follows from Theorem 1.

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 $|\log \Pr[\pi'(s)]| \geqslant (1-\epsilon)|\log \Pr[\pi(s)]|$ 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 $(1-\epsilon)|\log \Pr\left[\pi(s_i^j)\right]| \leqslant e(G, s_i^j) \leqslant (1+\epsilon)|\log \Pr\left[\pi(s_i^j)\right]|$ where $\pi(s_i^j)$ is the most likely parse of s_i^j.*

5 Lower Bound

5.1 Reducing APSP to Language Edit Distance

We now show improving on the cubic running time of exact computation of language edit distance problem will lead to breakthrough in several long standing problems.

Theorem (2)**.** *Given a context-free grammar $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$, and a string $s \in \Sigma^*$, $|s| = n$, if the language edit distance problem can be solved in $O(|G|n^{3-\delta})$ time then that implies an algorithm with running time $O(m^{3-\delta/3})$ for all-pairs shortest path (APSP) problem on directed graphs with m vertices.*

This leads to the following corollary by sub-cubic equivalence of all-pairs shortest path with many other fundamental problems on graphs and matrices [1, 54].

Corollary (1)**.** *Given a context-free grammar $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$, and a string $s \in \Sigma^*$, $|s| = n$, if the language edit distance problem can be solved in $O(|G|n^{3-\delta})$ time then that implies an algorithm with running time $O(m^{3-\gamma})$, $\gamma, \delta > 0$ for all of the following problems.*

1. *Minimum weight triangle: Given an n-node graph with real edge weights, compute u, v, w such that $(u, v), (v, w), (w, u)$ are edges and the sum of edge weights is minimized.*

2. *Negative weight triangle: Given an n-node graph with real edge weights, compute u, v, w such that $(u, v), (v, w), (w, u)$ are edges and the sum of edge weights is negative.*

3. *Metricity: Determine whether an $n \times n$ matrix over $\mathbb{R}$ defines a metric space on n points.*

4. *Minimum cycle: Given an n-node graph with real positive edge weights, find a cycle of minimum total edge weight.*

5. *Second shortest paths: Given an n-node directed graph with real positive edge weights and two nodes s and t, determine the second shortest simple path from s to t.*

6. *Replacement paths: Given an n-node directed graph with real positive edge weights and a shortest path P from node s to node t, determine for each edge $e \in P$ the shortest path from s to t in the graph with e removed.*

7. *Radius problem: Given an n-node weighted graph with real positive edge weights, determine the minimum distance r such that there is a vertex v with all other vertices within distance r from v.*

Note that Theorem 2 requires the language edit distance computation has linear dependency on grammar size. We will try to alleviate this constraint latter.

We next define the output of a language edit distance algorithm rigorously. We use a notion of minimum consistent derivation. This is similar to the notion of consistent derivation used by Lee [32] but needs to additionally handle distance.

Definition 6. *Given a context free grammar $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$, and a string $s \in \Sigma^*$. A nonterminal $A \in \mathcal{N}$ mc-derives (minimally and consistently derives) s_i^j if and only if the following condition holds:*

1) If A derives s_i^j with a minimum score l, implying if $A(s)$ is the set of all strings that A derives, $\min_{s' \in A(s)}\{\mathrm{dist}_{ed}(s', s_i^j)\} = l$, and

2. There is a derivation sequence $S \overset{*}{\Rightarrow} s_1^{i-1} A s_{j+1}^{n}$.

Definition 7. *A* Lan-Ed *is an algorithm that takes a CFG $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ and a string $s \in \Sigma^*$ as input and produces output $\mathcal{F}_{G,s}$ that acts as an oracle about distance information as follows: for any $A \in \mathcal{N}$*

- *If A minimally and consistently derives s_i^j with a minimum score l, then $\mathcal{F}_{G,s}(A, i, j) = l$*
- *$\mathcal{F}_{G,s}$ answers queries in constant time.*

The above definition is weaker than the local alignment problem, because we are maintaining only those distances for substrings from which the full string can be derived. All known algorithms for parsing and language edit distance computation maintain this information, because not computing these intermediate results may lead to failure in parsing the full string, or parsing it with minimum number of edits.

The choice of an oracle instead of a particular data structure keeps open the possibility that time required for Lan-Ed may be $o(n^2)$, which will not be the case if we keep a table like most known parsers. The third condition can be relaxed to take poly-logarithmic time in string and grammar size without much effect.

Distance Product Computation Recall the definition of distance product computation. Given two $n \times n$ matrix $A = (a_{i,j})$ and $B = (b_{i,j})$, their distance product $C = (c_{i,j}) = A \cdot B$ is defined as a $n \times n$ matrix $ci, j = \min_k(a_{i,k} + b_{k,j})$. That is, minimum plays the role of addition and plus plays the role of multiplication. This is also known as min-plus matrix product, tropical product etc. The min-plus algebra does not form a ring, as min does not have any additive inverse; hence fast matrix multiplication algorithms do not directly apply to it. It is known that a $T(n)$ time algorithm for distance product implies an $O(T(n))$ time algorithm for APSP. Floyd-Warshall algorithm gives an $O(n^3)$ time procedure to compute this product [18,51]. There has been a long series of works attempting to improve the running time, but those works only saved a poly-logarithmic factors over Floyd-Warshall (see [11,22], and references within). A recent result of Williams is the best known in this regime, where an $O(\frac{n^3}{2^{\Omega(\sqrt{\log n})}})$ algorithm has been devised by using tools from circuit complexity [52].

We will in fact reduce the grammar computation to distance product computation. Then, by their equivalence, this also gives a reduction to APSP.

Reduction We are given two weighted matrix A and B of dimension $m \times m$. We assume weights are all positive integers by scaling and shifting and the maximum such weight is M, i.e, $M = \max\left(\max_{i,j}(a(i,j)), \max_{i,j}(b(i,j))\right)$. We produce a grammar G and a string s such that from $\mathcal{F}_{G,s}$ one can deduce the matrix $C = A \cdot B$.

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 $s_1 s_2 .... s_{d+2} s_{d+3} ... s_{2d+4} s_{2d+5} .... s_{3d+6}$.

Now 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 $f_2(i) \in [2, d+1]$. From $f_1(i)$ and $f_2(i)$, we can obtain i uniquely. For notational simplicity we use i_1 to denote $f_1(i)$ and i_2 to denote $f_2(i)$. Note that 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 proceed to create the grammar $G = \{\mathcal{N}, \Sigma, \mathcal{P}, S\}$. Start from $\mathcal{N} = \{S\}$ and $\mathcal{P} = \phi$. We create $\lfloor \log M \rfloor$ nonterminals as follows. Let $2^k \leqslant M < 2^{k+1}$, then create $X_{2^k}, X_{2^{k-1}}, ..., X_2, X_1$. We add the productions

$$X \longrightarrow x \qquad \text{(X Rule)}$$
$$X_{2^i} \longrightarrow X_{2^{i-1}} X_{2^{i-1}}, 1 \leqslant i \leqslant k$$

Given a weight w, let $\hat{w} = X_{2^{j_1}} X_{2^{j_2}} ... X_{2^{j_l}}$ if $w = 2^{j_1} + 2^{j_2} + ... + 2^{j_l}$.

We now add nonterminal W and productions to generate arbitrary non-empty substrings from $\Sigma \setminus \{x\}$.

$$W \longrightarrow s_l W | s_l,\ l \in [1, 3d+6] \qquad \text{(W-Rule)}$$

Next, we encode the entries of input matrices A and B in our grammar as follows. We add nonterminals from the sets $A_{p,q} : 1 \leqslant p, q, \leqslant d^2$, and $B_{p,q} : 1 \leqslant p, q \leqslant d^2$. For each entry $a_{i,j}$ if $a_{i,j} = w$ we add the production

$$A_{i_1,j_1} \longrightarrow s_{i_2} W \hat{w} s_{j_2+\delta} \qquad \text{(A-Rule)}$$

Similarly, for each entry $b_{i,j}$ if $b_{i,j} = w'$, then we add the production

$$B_{i_1,j_1} \longrightarrow s_{i_2+\delta+1} W \hat{w}' s_{j_2+2\delta} \tag{B-Rule}$$

If A_{i_1,j_1} is used to parse $s_{i_2}^{j_2+\delta}$ then one must pay edit cost for each nonterminals in $\hat{w}$ which is exactly equal to the weight of $a(i,j)$. Similarly, if B_{j_1,k_1} is used to parse $s_{j_2+\delta+1}^{k_2+2\delta}$ then the total edit cost paid is exactly equal to the weight of $b(j,k)$. Also note that production for $a(i,j)$ ends at $s_{j_2+\delta}$, and the production for $b(j,k)$ exactly starts at $s_{j_2+\delta+1}$. We now add nonterminals to combine these consecutive substrings.

We consider non-terminals from the set $\{C_{p,q} : 1 \leqslant p,q \leqslant d^2\}$ and add productions for all r, $1 \leqslant r \leqslant d^2$

$$C_{p,q} \longrightarrow A_{p,r} B_{r,q} \tag{C-Rule}$$

Finally, we add the production for the start symbol S for all p,q, $1 \leqslant p,q \leqslant d^2$

$$S \longrightarrow W C_{p,q} W \tag{S-Rule}$$

We now prove the following lemma about computing language edit distance with the above grammar and string s.

Lemma 13. *For $1 \leqslant i,j \leqslant m$, the entry $c_{i,j} = l$, if and only if $C_{i_1 j_1}$ mc-derives $s_{i_2}^{j_2+2\delta}$ with score l.*

Proof. Fix i,j. We first prove the "only-if" part. So let $c_{i,j} = l$. Then there must exists a k such that $a_{i,k} = z$ and $b_{k,j} = l - z$.

We have the C-Rule $C_{i_1,j_1} = A_{i_1,k_1} B_{k_1,j_1}$. Since $a_{i,k} = z$, we have the $A - \mathrm{Rule}$ $A_{i_1,k_1} \longrightarrow s_{i_2} W \hat{z} s_{k_2+\delta}$ and since $b_{k,j} = l - z$, we have the $B - \mathrm{Rule}$ $B_{k_1,j_1} \longrightarrow s_{k_2+\delta+1} W l \hat{-} z s_{j_2+2\delta}$. Finally, since $i_2 + 1 < k_2 + \delta - 1$ and $k_2 + \delta + 2 \leqslant j_2 + 2\delta - 1$, $W \overset{*}{\Rightarrow} s_{i_2+1}^{k_2+\delta-1}$ and $W \overset{*}{\Rightarrow} s_{k_2+\delta+2}^{j_2+2\delta}$. All the xs generated from $\hat{z}$ and $l \hat{-} z$ act as deletion errors in string s. Hence A_{i_1,k_1} derives $s_{i_2}^{k_2+\delta}$ with score z and B_{k_1,j_1} derives $s_{k_2+\delta+1}^{j_2+2\delta}$ with score $l - z$. Therefore, C_{i_1,j_1} derives $s_{i_2}^{j_2+2\delta}$ with score l.

Finally, $S \overset{*}{\Rightarrow} s_1^{i_2-1} C_{i_1,j_1} s_{j_2+2\delta+1}^{3\delta+6}$ with score l, since $i_2 - 1 \geqslant 1$ and $j_2 + 2\delta + 1 < 3\delta + 6$, hence C_{i_1,j_1} minimally and consistently derives $s_{i_2}^{j_2+2\delta}$ with score l.

Now, let us look at the "if" part and assume C_{i_1,j_1} derives $s_{i_2}^{j_2+2\delta}$ minimally and consistently with a score l'. This can only arise through an application of C-Rule $C_{i_1,j_1} \longrightarrow A_{i_1,k_1'} B_{k_1',j_1}$ such that $A_{i_1,k_1'}$ derives $s_{i_2}^{k_2'+\delta}$ within edit distance (say) z' and B_{k_1',j_1} derives $s_{k_2'+\delta+1}^{j_2+2\delta}$ within edit distance $l' - z'$. Then, we must have the productions $A_{i_1,k_1'} \longrightarrow s_{i_2} W \hat{z'} s_{k_2'+\delta}$ and $B_{k_1',j_1} \longrightarrow s_{k_2'+\delta+1} W l' \hat{-} z' s_{j_2+2\delta}$. But this can only happen, if there is a number k' such that $f_1(k') = k_1'$ and $f_2(k') = k_2'$ and $a(i,k) = z'$ and $b(k',j) = l' - z'$, and therefore $c(i,j) \leqslant l'$.

The theorem now follows. $\square$

Grammar Size The total number of nonterminals used in this grammar is $O(d^4 + \log M) = O(m^{4/3} + \log M)$ and the number of productions is $O(m^2 + d^6 + \log M) = O(m^2 + \log M)$, where d^6 term comes from the $\mathrm{C\text{-}Rule}$ and m^2 comes from considering all the entries of A and B. If we consider the number of nonterminals involved in each production, then the total size of the grammar is $|G| = O(m^2 \log M)$.

Note. The grammar constructed here is not in CNF form, but can easily be transformed into a CNF representation G' where the number of productions in G' increases at most by a factor of $\log m$. This happens because in G there is no ϵ production or unit productions. For every terminal s_j $j = 1,2,..,3d+6$, we create a nonterminal, S_j and replace their occurrences in productions with the newly created nonterminals. We add the productions $S_j \rightarrow s_j$ for $j = 1, 2, .., 3d+6$. Finally, for every rule of the form $Q \rightarrow R_1 R_2 ... R_s$, we create $s-1$ rules $Q \rightarrow R_1 Q_1$, $Q_1 \rightarrow R_2 Q_2$,..., $Q_{s-2} \rightarrow R_{s-1} R_s$. Since in G, the size of RHS is any production can be at most $\lceil \log M \rceil + 3$, we get the desired bound. Therefore, the claims in this section equally holds when parsers are restricted to work with CNF grammars.

Time Bound

Lemma 14. *Any language edit distance problem* P *with mc-derivation having run time* $O(T(|G|)t(n))$ *on grammars of size* $|G|$ *and strings of length* n *can be converted into an algorithm MP to compute distance product on positive-integer weighted* $m \times m$ *matrix with highest weight* M *that runs in time* $O(\max(m^2 + T(m^2)t(m^{1/3}) \log M)$. *In particular, if P takes* $O(|G|n^{3-\epsilon})$ *time then that implies an* $O(m^{3-\epsilon/3} \log M)$ *running time for MP.*

Proof. Given the two matrices A and B of dimension $m \times m$ with maximum weight (after shifting and scaling) M, time to read the entries is $O(m^2)$ and to create grammar G is $O(|G|) = O(m^2 + m^{4/3} \log M)$ (note that $d = \lfloor m^{1/3} \rfloor$) and string s is $O(m^{1/3})$. Assume, the parser takes time $O(T_1(G) + T_2(G)t(n))$ to create $\mathcal{F}_{G,s}$. Then we query $\mathcal{F}_{\mathcal{G},\int}$ for each $c_{i,j}$ by creating the query $(C_{i_1,j_1}, s_{i_2}^{j_2+2\delta})$. If the answer is K, we set $c_{i,j} = K$. By Lemma 13, the computed value of $c_{i,j} = \min_k(a_{i,k} + b_{k,j})$ is correct. Hence once parsing has been done, creating C again takes $O(m^2)$ time, assuming each query needs $O(1)$ time.

Suppose $T_1(G) = T_2(G) = |G|$ and $t(n) = n^{3-\epsilon}$, $0 < \epsilon \leqslant 1$then we get an algorithm to compute distance product in time $O(m^{3-\frac{\epsilon}{3}} \log M)$. □

Now, due to sub-cubic equivalence of distance product computation with APSP, Theorem 2 follows.

Theorem (2)**.** *Given a context-free grammar* $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$*, and a string* $s \in \Sigma^*$, $|s| = n$*, if the language edit distance problem can be solved in* $O(|G|n^{3-\delta})$ *time then that implies an algorithm with running time* $O(m^{3-\delta/3})$ *for all-pairs shortest path problem on directed graphs with* m *vertices.*

Corollary 1 also follows directly from the sub-cubic equivalence between APSP and the mentioned problems [1, 54].

5.2 Reducing APSP to Stochastic Context Free Parsing

The reduction takes the following steps.

1. Reduce (min, $\times$)-matrix product where matrix entries are drawn from $\mathbb{R}^+$ to stochastic context free grammar parsing, that is show if there exists am $O(n^{3-\epsilon})$ algorithm for stochastic context free grammar parsing, then there exists one with running time $O(n^{3-\beta})$ for (min, $\times$)-matrix product over $\mathbb{R}^+$, $\epsilon, \beta > 0$.

2. Next we show if there exists an algorithm with running time $O(n^{3-\beta})$ for (min, $\times$)-matrix product with entries in $\mathbb{R}^+$, $\beta > 0$, then there exists one with running time $O(n^{3-\beta})$ for detecting negative weight triangle in a weighted graph.

3. Finally, due to sub-cubic equivalence between minimum weight triangle detection with non-negative weights and APSP [54], the result follows.

Reducing $(\min, \times)$**-matrix product with entries in** $\mathbb{R}^+$ **to stochastic context free grammar parsing** This reduction is similar to the previous one used for reducing language edit distance problem to distance product computation. Instead of encoding $a_{i,j} = w$, in the production rules $\mathsf{A\text{-}Rule}$ and $\mathsf{B\text{-}Rule}$, this is encoded in the probability of the corresponding productions. We sketch the changes required.

Definition 8. *Given a stochastic context free grammar* $\{G = (\mathcal{N}, \Sigma, \mathcal{P}, S), \mathbf{p}\}$*, and a string* $s \in \Sigma^*$*. A nonterminal* $A \in \mathcal{N}$ *c-derives (consistently derives)* s_i^j *if and only if the following condition holds:*

1. A *derives* s_i^j

2. There is a derivation sequence $S \overset{*}{\Rightarrow} s_1^{i-1} A s_{j+1}^n$.

Definition 9. *A* $\mathsf{Stochastic\text{-}Parsing}$ *is an algorithm that takes a SCFG* $\{G = (\mathcal{N}, \Sigma, \mathcal{P}, S), \mathbf{p}\}$ *and a string* $s \in \Sigma^*$ *as input and produces output* $\mathcal{F}_{\{G,\mathbf{p}\},s}$ *that acts as an oracle about distance information as follows: for any* $A \in \mathcal{N}$

- *If* A *consistently derives* s_i^j *with maximum probability* q*, then* $\mathcal{F}_{\{G,\mathbf{p}\},s}(A, i, j) = q$
- $\mathcal{F}_{\{G,\mathbf{p}\},s}$ *answers queries in constant time.*

Creating the Grammar. We are given two matrices A and B with entries from $\mathbb{R}^+$, and want to compute their (min, $\times$)-product $C = A._{\min,\times} B$.

$$c_{i,j} = \min_{1 \leqslant k \leqslant n} (a_{i,k}.b_{k,j})$$

We use identical parameters $d, f_1(i), f_2(i)$ etc. like the previous reduction.

We add nonterminal W and productions to generate arbitrary non-empty substrings from $\Sigma \setminus \{x\}$.

$$W \longrightarrow s_l W | s_l,\ l \in [1, 3d+6] \text{ each rule having probability } \frac{1}{2(3d+6)} \tag{W-Rule}$$

Hence, the total probability of $W\text{-}\mathsf{Rules}$ add up to 1.

Next, we encode the entries of input matrices A and B in our grammar as follows. We add nonterminals from the sets $A_{p,q} : 1 \leqslant p, q, \leqslant d^2$, and $B_{p,q} : 1 \leqslant p, q \leqslant d^2$.

First we compute the following sums

- For i=1 to m
  - For j=1 to m
    - $x = f_1(i), y = f_1(j)$
    - $\mathsf{Count}_A(x, y) = \mathsf{Count}_A(x, y) + \frac{1}{a_{i,j}}$
    - $\mathsf{Count}_B(x, y) = \mathsf{Count}_B(x, y) + \frac{1}{b_{i,j}}$
  - End For
- End For
- $\mathsf{MaxCount}_A = \max_{x,y} \mathsf{Count}_A(x, y)$
- $\mathsf{MaxCount}_B = \max_{x,y} \mathsf{Count}_B(x, y)$

For each entry $a_{i,j}$ if $a_{i,j} = w$ we add the production

$$A_{i_1,j_1} \longrightarrow s_{i_2} W s_{j_2+\delta} \text{ with probability } \frac{1}{a_{i,j} MaxCount_A} \text{ where } f_1(i) = i_1, f_2(i) = i_2, f_1(j) = j_1, f_2(j) = j_2 \tag{A-Rule}$$

If $MaxCount_A > Count_A(i_1, j_1)$, then add a dummy rule

$$A_{i_1,j_1} \longrightarrow x \text{ with probability } \frac{MaxCount_A - Count_A(i_1, j_1)}{MaxCount_A}$$

Clearly, all the A-$Rules$ have probability > 0, and the probabilities of the rules with A_{i_1,j_1} on the LHS $1 \leqslant i, j \leqslant d^2$ add up to 1.

Similarly, we create the B-$Rules$ and assign probability.

For each entry $b_{i,j}$ we add the production

$$B_{i_1,j_1} \longrightarrow s_{i_2+\delta+1} W s_{j_2+2\delta} \text{ with probability } \frac{1}{b_{i,j} MaxCount_B} \text{ where } f_1(i) = i_1, f_2(i) = i_2, f_1(j) = j_1, f_2(j) = j_2 \tag{B-Rule}$$

If $MaxCount_B > Count_B(i_1, j_1)$, then add a dummy rule

$$B_{i_1,j_1} \longrightarrow x \text{ with probability } \frac{MaxCount_B - Count_B(i_1, j_1)}{MaxCount_B}$$

Clearly, all the B-$Rules$ have probability > 0, and the probabilities of the rules with B_{i_1,j_1} on the LHS $1 \leqslant i, j \leqslant d^2$ add up to 1.

We consider non-terminals from the set $\{C_{p,q} : 1 \leqslant p, q \leqslant d^2\}$ and add productions for all r, $1 \leqslant r \leqslant d^2$

$$C_{p,q} \longrightarrow A_{p,r} B_{r,q} \text{ with probability } \frac{1}{d^2} \tag{C-Rule}$$

Hence, probabilities of the rules with $C_{p,q}$ on the LHS $1 \leqslant p, q \leqslant d^2$ add up to 1.

Finally, we add the production for the start symbol S for all p, q, $1 \leqslant p, q \leqslant d^2$

$$S \longrightarrow W C_{p,q} W \text{ with probability } \frac{1}{d^4} \tag{S-Rule}$$

Probabilities of the rules with S on the LHS add up to 1. Hence, the constructed grammar is a SCFG.

Lemma 15. *For $1 \leqslant i, j \leqslant m$, the entry $c_{i,j} = l$, if and only if $C_{i_1 j_1}$ c-derives $s_{i_2}^{j_2+2\delta}$ with probability* $\frac{1}{l d^2 . MaxCount_A . MaxCount_B}$

Proof. The proof is similar to Lemma 13 instead of computing the total edit distance, compute the total probability of the productions applied to parse $s_{i_2}^{j_2+2\delta}$. □

Lemma 16. *Any stochastic context free parsing problem P with c-derivation having run time $O(T(|G|)t(n))$ on grammars of size $|G|$ and strings of length n can be converted into an algorithm MP to compute* (min, $\times$)*-product of $m \times m$ matrices with entries in $\mathbb{R} \setminus \{0\}$ with highest weight M that runs in time $O(\max(m^2 + T(m^2)t(m^{1/3}) \log M)$. In particular, if P takes $O(|G|n^{3-\epsilon})$ time then that implies an $O(m^{3-\epsilon/3} \log M)$ running time for MP.*

Proof. The proof follows from Lemma 15 and following similar steps as Lemma 14. □

From $(\min, \times)$-matrix product with positive real entries to Negative Triangle Detection. We now show that if there exists an algorithm with running time $O(n^{3-\beta})$ for (min, $\times$)-matrix product over $\mathbb{R}^+$, $\beta > 0$, then there exists one with running time $O(n^{3-\beta})$ to detect if a weighted graph $G = (V, E)$ has a triangle of negative total edge weight.

We assume all weights are integers, and the maximum absolute weight is at least 3. Both of these can be achieves by appropriately scaling the edge weights.

Let W be the maximum absolute weight on any edge $e \in E$, $|W| \geqslant 3$. 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 $C = A \bigodot_{\min,\times} B$. 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 triangles, then $C'(i,j) \geqslant -W^2 + 2W^2 = W^2 \geqslant 9$ for all $1 \leqslant i,j \leqslant n$.

Therefore, there exists a negative triangle in G if and only if there is a negative entry in C'. While C can be computed in asymptotically same time as computing $(\min,\times)$-matrix product of two $n \times n$ dimensional matrices with real positive entries, C' can be computed from C in $O(n^2)$ time.

Hence, we get the following lemma.

Lemma 17. *Given two $n \times n$ matrices with positive real entries, if their* (min, $\times$)*-matrix product can be done in time $T(n)$ time, then $O(T(n)+n^2)$ time is sufficient to detect negative triangles on weighted graphs with n vertices.*

Now, due to sub-cubic equivalence between negative triangle detection and APSP, we get the following theorem.

Theorem (3)**.** *Given a stochastic context-free grammar $\{G = (\mathcal{N}, \Sigma, \mathcal{P}, S), \mathbf{p}\}$, and a string $s \in \Sigma^*$, $|s| = n$, if the SCFG parsing problem can be solved in $O(|G|n^{3-\delta})$ time then that implies an algorithm with running time $O(m^{3-\delta/3})$ for all-pairs shortest path problem on directed graphs with m vertices.*

6 Conclusion

In this paper, we make significant progress on the state-of-art of stochastic context free grammar parsing, and language edit distance problem. Context free grammars are the pillar of formal language theory. Grammar based distance computation and stochastic grammars have been proven to be very powerful tools with huge applications. Here, we give the the *first* sub-cubic algorithms with running time $\tilde{O}(\frac{n^\omega}{\epsilon})$ for both of these problems that return near-optimal results.

For the first time, we lay out their connections to fundamental problems on graphs and matrices, and show that improvement in the exact computation of either SCFG parsing or language edit distance computation will lead to major breakthroughs in a large variety of problems.

Many questions remain. Understanding the general relationship between parsing time and language edit distance computation is a big open problem. Allowing additive approximation may break the barrier of n^ω in running time. We have initial results that suggest in this direction.

References

[1] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. ICALP, pages 39–51, 2014.

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

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

[4] 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.

[5] 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.

[6] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. FOCS, 2010.

[7] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. STOC, 2009.

[8] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. FOCS, 2004.

[9] Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. SODA, 2006.

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

[11] Timothy M Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Transactions on Algorithms (TALG)*, 8(4):34, 2012.

[12] Richard Cole and Ramesh Hariharan. Approximate string matching: A simpler faster algorithm. *SIAM J. Comput.*, 31(6), June 2002.

[13] Richard Durbin. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[14] 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).

[15] C. N. Fischer and J. Mauney. On the role of error productions in syntactic error correction. *Comput. Lang.*, 5(3-4), January 1980.

[16] Charles N. Fischer and Jon Mauney. A simple, fast, and effective LL(1) error repair algorithm. *Acta Inf.*, 29(2), April 1992.

[17] Philippe Flajolet. Analytic models and ambiguity of context-free languages. *Theoretical Computer Science*, 49(2), 1987.

[18] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

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

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

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

[22] Yijie Han and Tadao Takaoka. An $o(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. SWAT. 2012.

[23] H. Hermes. *Enumerability, Decidability, Computability: An Introduction to the Theory of Recursive Functions*. Springer.

[24] 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).

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

[26] Ik-Soon Kim and Kwang-Moo Choe. Error repair with validation in LR-based parsing. *ACM Trans. Program. Lang. Syst.*, 23(4), July 2001.

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

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

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

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

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

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

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

[34] Bruce McKenzie, Corey Yeatman, and Lorraine De Vere. Error repair in shift-reduce parsers. *ACM Transactions on Programming Languages and Systems*, 17, 1995.

[35] 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).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[51] Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962.

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

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

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

[55] Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In *SODA*.

[56] 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.

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

[58] Uri Zwick. All pairs shortest paths in weighted directed graphs–exact and almost exact algorithms. In *FOCS*.

7 Appendix

Keeping Parsing Information. Parsing information can be kept easily. Suppose $(A_x, u) \in M(i, j)$ and $(A_y, v) \in M(j, k)$. Now maintain with (A_x, u), the start and end point of substring that it generates. So we maintain $(A_x, u, i, j-1)$. Similarly for (A_y, v) we instead maintain $(A_y, v, j, k-1)$. Now if $B \to A_x A_y$ is a production, and we include $(B, u+v, i, k-1)$ in $M(i, k)$ we also maintain the production that gave rise to this entry. Hence, overall we maintain at $M(i, k)$ the entry $\{(B, u+v, i, k-1), (B, u+v, i, k-1) \to (A_x, u, i, j-1)(A_y, v, j, k-1)$.

To obtain parsing information for any string s_i^k, we consider $M(i, k+1)$ and look for the entry that involves the start symbol S. If the retrieved entry is $\{(S, z, i, k), (S, z, i, k) \to (A_x, u, i, j-1)(A_y, v, j, k)$. Then, we include $S \to A_x A_y$ as the first derivation applied for parsing. We then look for the entry with A_x in $M(i, j)$ and the entry for A_y in $M(j, k+1)$, and proceed recursively to obtain the full parsing information. The total number of cells of M that we need to look up is $O(k-i)$, and hence the full parsing information for s_i^k can be obtained in $O(k-i)$ time.