If the Current Clique Algorithms Are Optimal, so Is Valiant's Parser

SIAM J. COMPUT.
Vol. 47, No. 6, pp. 2527–2555

IF THE CURRENT CLIQUE ALGORITHMS ARE OPTIMAL, SO IS VALIANT'S PARSER*

AMIR ABBOUD†, ARTURS BACKURS‡, AND VIRGINIA VASSILEVSKA WILLIAMS‡

Abstract. The CFG recognition problem is as follows: given a context-free grammar $\mathcal{G}$ and a string w of length n, decide whether w can be obtained from $\mathcal{G}$. This is the most basic parsing question and is a core computer science problem. Valiant's parser from 1975 solves the problem in $O(n^\omega)$ time, where $\omega < 2.373$ is the matrix multiplication exponent. Dozens of parsing algorithms have been proposed over the years, yet Valiant's upper bound remains unbeaten. The best *combinatorial* algorithms have mildly subcubic $O(n^3/\log^3 n)$ time complexity. Evidence for the nonexistence of more efficient algorithms was given in a result of Lee (JACM'01), who showed that any algorithm with running time $O(|\mathcal{G}| \cdot n^{3-\varepsilon})$, which can solve a more general parsing problem in the unusual case that $|\mathcal{G}| = \Omega(n^6)$, can be converted into a surprising subcubic time algorithm for Boolean Matrix Multiplication. However, nothing was known for the more relevant case of constant size grammars. In this work, we make the first progress on the constant size grammar case in 40 years, and prove that *any* improvement on Valiant's algorithm, either in terms of runtime or by avoiding the inefficiencies of the known fast matrix multiplication algorithms, would imply a breakthrough algorithm for the k-Clique problem: given a graph of n nodes, decide whether there are k nodes that form a clique. Using similar techniques, we derive similar lower bounds for more modern and well-studied cubic time problems for which faster algorithms are highly desirable in practice: *RNA Folding*, a central problem in computational biology, and *Dyck Language Edit Distance*, answering an open question of Saha (FOCS'14).

Key words. CFG parsing, Dyck Edit Distance, RNA Folding, conditional lower bounds, context-free grammars, Valiant's algorithm, clique algorithms, hardness in P, combinatorial algorithms

AMS subject classification. 68Q25

DOI. 10.1137/16M1061771

1. Introduction.

Context-free grammars (CFGs) and languages (CFLs), introduced by Chomsky in 1959 [22], play a fundamental role in computability theory [63], formal language theory [40], programming languages [4], natural language processing [42], and computer science in general with applications in diverse areas such as computational biology [27] and databases [46]. They are essentially a sweet spot between very expressive languages (such as natural languages) that computers cannot parse well and the more restrictive languages (such as regular languages) that even a deterministic finite automata (DFA) can parse.

In this paper, we will be concerned with the following very basic definitions. A CFG $\mathcal{G}$ in *Chomsky Normal Form* over a set of terminals (i.e., alphabet) Σ consists of a set of nonterminals $\mathcal{T}$, including a specified starting symbol $\mathbf{S} \in \mathcal{T}$, and a set of productions (or derivation rules) of the form $\mathbf{A} \to \mathbf{B}\ \mathbf{C}$ or $\mathbf{A} \to \sigma$ for some $\mathbf{A}, \mathbf{B}, \mathbf{C} \in$

*Received by the editors February 16, 2016; accepted for publication (in revised form) August 1, 2018; published electronically December 18, 2018. A preliminary version of this paper appeared in *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science*, 2015, pp. 98–117.

http://www.siam.org/journals/sicomp/47-6/M106177.html

Funding: This research was completed while the first and third authors were at Stanford University, where they were supported by NSF grant CCF-1417238, BSF grant 2012338, and a Stanford SOE Hoover Fellowship. The research of the second author was supported by the NSF, an IBM Ph.D. Fellowship, and the Simons Foundation.

†IBM Almaden, San Jose, CA 95120 (amir.abboud@ibm.com).

‡Computer Science & Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 (backurs@mit.edu, virgi@mit.edu).

$\mathcal{T}$ and $\sigma \in \Sigma$. Each CFG $\mathcal{G}$ defines a CFL $\mathcal{L}(\mathcal{G})$ of strings in Σ^* that can be obtained by starting with $\mathbf{S}$ and recursively applying arbitrary derivation rules from the grammar. The *CFG recognition* problem is as follows: given a CFG $\mathcal{G}$ and a string $w \in \Sigma^*$, determine whether w can be obtained from $\mathcal{G}$ (i.e., whether $w \in \mathcal{L}(\mathcal{G})$). The problem is of most fundamental and practical interest when we restrict $\mathcal{G}$ to be of fixed size and let the length of the string $n = |w|$ be arbitrary.

The main question we will address in this work is the following: What is the time complexity of the CFG recognition problem?

Besides the clear theoretical importance of this question, the practical motivation is overwhelming. CFG recognition is closely related to the *parsing* problem, in which we also want to output a possible derivation sequence of the string from the grammar (if $w \in \mathcal{L}(\mathcal{G})$). Parsing is essential: this is how computers understand our programs, scripts, and algorithms. Any algorithm for parsing solves the recognition problem as well, and Ruzzo [55] showed that CFG recognition is at least as hard as parsing, at least up to logarithmic factors, making the two problems roughly equivalent.

Not surprisingly, the critical nature of CFG recognition has led to the development of a long list of clever algorithms for it, including classical works [67, 28, 23, 77, 44, 45, 26, 14, 47], and the search for practical parsing algorithms, which work well for varied applications, is far from over [51, 54, 64, 24]. For example, the canonical CYK algorithm from the 1960's [23, 44, 77] constructs a dynamic programming table D of size $n \times n$ such that cell $D(i, j)$ contains the list of all nonterminals that can produce the substring of w from position i to position j. The table can be computed with linear time per entry, by enumerating all derivation rules $\mathbf{A} \to \mathbf{B}\ \mathbf{C}$ and checking whether for some $i \leq k \leq j$, $D(i, k)$ contains $\mathbf{B}$ and $D(k + 1, j)$ contains $\mathbf{C}$ (and, if so, add $\mathbf{A}$ to $D(i, j)$). This gives an upper bound of $O(n^3)$ for the problem. Another famous algorithm is Earley's from 1970 [28], which proceeds by a top-down dynamic programming approach and could perform much faster when the grammar has certain properties. Variants of Earley's algorithm were shown to run in mildly subcubic $O(n^3/\log^2 n)$ time [34, 56].[1]

In 1975, a big theoretical breakthrough was achieved by Valiant [67], who designed a sophisticated recursive algorithm that is able to utilize many fast boolean matrix multiplications (BMMs) to speed up the computation of the dynamic programming table from the CYK algorithm. The time complexity of the CFG problem decreased to $O(g^2 n^\omega)$, where $\omega < 2.373$ is the matrix multiplication exponent [73, 32] and g is the size of the grammar; because in most applications $g = O(1)$, Valiant's runtime is often cited as $O(n^\omega)$. In 1995, Rytter [57] described this algorithm as "probably the most interesting algorithm related to formal languages," and it is hard to argue with this quote even today, 40 years after Valiant's result. Follow-up works proposed simplifications of the algorithm [57], generalized it to stochastic CFG parsing [15], and applied it to other problems [5, 79].

Despite its vast academic impact, Valiant's algorithm has enjoyed little success in practice. The theoretically fastest matrix multiplication algorithms are not currently practical, and Valiant's algorithm can often be outperformed by *combinatorial* methods in practice, even if the most practical truly subcubic fast matrix multiplication algorithm (Strassen's in [66]) is used. Theoretically, the fastest combinatorial algorithms for BMM run in time $O(n^3/\log^4 n)$ [78]. To date, no combinatorial algorithm for BMM or CFG recognition with truly subcubic running time is known.

[1] As is typical, we distinguish between "truly subcubic" runtimes, $O(n^{3-\varepsilon})$ for constant $\varepsilon > 0$, and "mildly subcubic" for all other subcubic runtimes.

In the absence of efficient algorithms and the lack of techniques for proving superlinear unconditional lower bounds for any natural problem, researchers have turned to conditional lower bounds for CFG recognition and parsing. Since the late 1970's, Havel and Harrison [38] observed that any algorithm for the problem would imply an algorithm that can verify a BMM of two $\sqrt{n} \times \sqrt{n}$ matrices. This reduction shows that a combinatorial $O(n^{1.5-\varepsilon})$ recognition algorithm for $\varepsilon > 0$ would imply a breakthrough subcubic algorithm for BMM. Ruzzo [55] showed that a parsing algorithm that says whether each prefix of the input string is in the language could even compute the BMM of two $\sqrt{n} \times \sqrt{n}$ matrices. Even when only considering combinatorial algorithms, these $\Omega(n^{1.5})$ lower bounds left a large gap compared to the cubic upper bound. A big step towards *tight* lower bounds was in the work of Satta [61] on parsing Tree Adjoining Grammars, which was later adapted by Lee [48] to prove her famous conditional lower bound for CFG parsing. Lee proved that BMM of two $n \times n$ matrices can be reduced to "parsing" a string of length $O(n^{1/3})$ with respect to a CFG of size $\Theta(n^2)$, where the parser is required to say for each nonterminal T and substring $w[i : j]$ whether T can derive $w[i : j]$ in a valid derivation of w from the grammar. This reduction proves that such parsers cannot be combinatorial and run in $O(gn^{3-\varepsilon})$ time for $\varepsilon > 0$ without implying a breakthrough in BMM algorithms.

Lee's result is important; however, it suffers from significant limitations which have been pointed out by many researchers (see, e.g. [55, 48, 58, 59]). We describe a few of these below. Despite the limitations, however, the only progress after Lee's result is a recent result by Saha [60] that one can replace BMM in Lee's proof with all-pairs shortest paths (APSP) by augmenting the production rules with probabilities, thus showing an APSP-based lower bound for *Stochastic CFG* parsing. Because Saha uses Lee's construction, her lower bound suffers from exactly the same limitations.

The first (and most major) limitation of Lee's lower bound is that it is irrelevant unless the size of the grammar is much larger than the string; in particular, it is cubic only when $g = \Omega(n^6)$. A CFG whose description needs to grow with the input string does not really define a CFL, and as Lee points out, this case can be unrealistic in many applications. In programming languages, for instance, the grammar size is much smaller than the programs in which one is interested, and in fact the grammar can be hardcoded into the parser. A parsing algorithm that runs in time $O(g^3 n)$, which is *not* ruled out by Lee's result, could be much more appealing than one that runs in $O(gn^{2.5})$ time.

The second limitation of both Lee's and Ruzzo's lower bounds is the quite demanding requirement from the parser to provide extra information besides returning some parse tree. These lower bounds do not hold for recognizers or any parser with minimal but meaningful output.

Theoretically, it is arguably more fundamental to ask the following: What is the time complexity of CFG recognition and parsing that can be obtained by *any* algorithm, not necessarily combinatorial? Lee's result cannot give a meaningful answer to this question. To get a new upper bound for BMM via Lee's reduction, one needs a parser that runs in near-linear time. Lee's result does not rule out, say, an $O(n^{1.11})$ time parser that uses fast matrix multiplication; such a parser would be an amazing result.

Our first observation is that to answer these questions and understand the complexity of CFG recognition, we may need to find a problem other than BMM from which to reduce. Despite the apparent similarity in complexities of both problems—we expect both to be cubic for combinatorial algorithms and $O(n^\omega)$ for unrestricted ones—there is a big gap in complexities because of the input size. When the grammar

is fixed, a reduction cannot encode any information in the grammar and can only use the n letters of the string, i.e., $O(n)$ bits of information, while an instance of BMM requires $\Theta(n^2)$ bits to specify. Thus, at least with respect to many-one reductions that produce a single instance of CFG recognition, we do not expect BMM to imply a higher than $\Omega(n^{1.5})$ lower bound for parsing a fixed size grammar.

Main result. In this paper, we present a tight reduction from the k-Clique problem to the recognition of a fixed CFG and prove a new lower bound for CFG recognition that overcomes all the above limitations of the previously known lower bounds. Unless a breakthrough k-Clique algorithm exists, our lower bound completely matches Valiant's upper bound for unrestricted algorithms and completely matches those of CYK and Earley for combinatorial algorithms, thus (conditionally) resolving the complexity of CFG recognition even on fixed size grammars.

Before formally stating our results, let us give some background on k-Clique. This fundamental graph problem asks whether a given undirected unweighted graph on n nodes and $O(n^2)$ edges contains a clique on k nodes. This is the parameterized version of the famously NP-hard Max-Clique (or, equivalently, Max Independent Set) [43]. k-Clique is amongst the most well-studied problems in theoretical computer science, and it is the canonical intractable (W[1]-complete) problem in parameterized complexity.

A naive algorithm solves k-Clique in $O(n^k)$ time. By a reduction from 1985 to BMM on matrices of size $n^{k/3} \times n^{k/3}$, it can be solved with fast matrix multiplication in $O(n^{\omega k/3})$ time [50] whenever k is divisible by 3 (otherwise, the runtime is only slightly worse [29]). No better algorithms are known, and researchers have wondered whether improvements are possible [75, 11]. As is the case for BMM, obtaining faster than trivial *combinatorial* algorithms, by more than polylogarithmic factors, for k-Clique is a longstanding open question. The fastest combinatorial algorithm runs in $O(n^k/\log^k n)$ time [68].

Let $0 \leq F \leq \omega$ and $0 \leq C \leq 3$ be the smallest numbers such that $3k$-Clique can be solved combinatorially in $n^{Ck+o(1)}$ time and in $n^{Fk+o(1)}$ time by any algorithm for any (large enough) constant $k \geq 1$. A conjecture in graph algorithms and parameterized complexity is that $C = 3$ and $F = \omega$. It is known that an algorithm refuting this conjecture immediately implies a faster exact algorithm for MAX-CUT [71, 76]. Note that even a linear time algorithm for BMM ($\omega = 2$) would not prove that $F < 2$. A well-known result by Chen et al. [20, 21] shows that $F > 0$ under the Exponential Time Hypothesis. A plausible conjecture about the parameterized complexity of Subset-Sum implies that $F \geq 1.5$ [2]. There are many other negative results that intuitively support this conjecture: Williams and Williams proved that a truly subcubic combinatorial algorithm for 3-Clique implies such an algorithm for BMM as well [74]. Unconditional lower bounds for k-Clique are known for various computational models, such as $\Omega(n^k)$ for monotone circuits [7]. The planted Clique problem has also proven to be very challenging (see, e.g. [6, 8, 39, 41]). Max-Clique is also known to be hard to efficiently approximate within nontrivial factors [37].

Formally, our reduction from k-Clique to CFG recognition proves the following theorem.

THEOREM 1.1. *There is CFG $\mathcal{G}_C$ of constant size such that if we can determine whether a string of length n can be obtained from $\mathcal{G}_C$ in $T(n)$ time, then k-Clique on n node graphs can be solved in $O\left(T\left(n^{k/3+1}\right)\right)$ time for any $k \geq 3$. Moreover, the reduction is combinatorial.*

To see the tightness of our reduction, let $1 \leq f \leq \omega$ and $1 \leq c \leq 3$ denote

the smallest numbers such that CFG recognition can be solved in $n^{f+o(1)}$ time and combinatorially in $n^{c+o(1)}$ time. An immediate corollary of Theorem 1.1 is that $f \geq F$ and $c \geq C$. Under the plausible assumption that current k-Clique algorithms are optimal, up to $n^{o(1)}$ improvements, our theorem implies that $f \geq \omega$ and $c \geq 3$. Combined with Valiant's algorithm we get that $f = \omega$, and with standard CFG parsers we get that $c = 3$. Because our grammar size g is fixed, we also rule out $O(h(g) \cdot n^{3-\varepsilon})$ time (for $\varepsilon > 0$) combinatorial CFG parsers for any computable function $h(g)$.

In other words, we construct a single fixed CFG $\mathcal{G}_C$ for which the recognition problem (and therefore any parsing problem) cannot be solved any faster than by Valiant's algorithm and for which any combinatorial recognizer will not run in truly subcubic time, without implying a breakthrough algorithm for the Clique problem. This (conditionally) proves that these algorithms are optimal general purpose CFG parsers, and more efficient parsers will only work for CFL with special restricting properties. On the positive side, our reduction might hint at what a CFG should look like to allow for efficient parsing.

The *online* version of CFG recognition is as follows: preprocess a CFG such that given a string w that is revealed one letter at a time, so that at stage i we get $w[1 \cdots i]$, we can say as quickly as possible whether $w[1 \cdots i]$ can be derived from the grammar (before seeing the next letters). One usually tries to minimize the total time it takes to provide all the $|w| = n$ answers. This problem has a long history of algorithms [70, 34, 56] and lower bounds [36, 33, 62]. The current best upper bound is $O(n^3/\log^2 n)$ total running time, and the best lower bound is $\Omega(n^2/\log n)$. Since this is a harder problem, our lower bound for CFG recognition also holds for it.

1.1. More results. The main ingredient in the proof of Theorem 1.1 is a lossless encoding of a graph into a string that belongs to a simple CFL if and only if the graph contains a k-clique. Besides classifying the complexity of a fundamental problem, this construction has led us to new lower bounds for two more modern and well-studied cubic time problems for which faster algorithms are highly desirable in practice.

RNA Folding. The RNA folding problem is a version of maximum matching and is one of the most relevant problems in computational biology. Its most basic version can be neatly defined as follows. Let Σ be a set of letters, and let $\Sigma' = \{\sigma' \mid \sigma \in \Sigma\}$ be the set of "matching" letters, such that for every letter $\sigma \in \Sigma$ the pair σ, σ' match. Given a sequence of n letters over $\Sigma \cup \Sigma'$, the RNA folding problem asks for the maximum number of *noncrossing* pairs $\{i, j\}$ such that the ith and jth letters in the sequence match.

The problem can be viewed as a *Stochastic CFG* parsing problem, in which the CFG is very restricted. This intuition has led to many mildly subcubic algorithms for the problem [65, 5, 12, 52, 69, 31, 79]. The main idea is to adapt Valiant's algorithm by replacing his BMM with a $(\min, +)$-matrix product computation (i.e., distance or tropical product). Using the fastest known algorithm for $(\min, +)$-matrix multiplication, it is possible to solve RNA Folding in $n^3/2^{\Theta(\sqrt{\log n})}$ time [72]. The fastest *combinatorial* algorithms, however, run in roughly $O(n^3/\log^2 n)$ time [65, 12], and we show that a truly subcubic such algorithm would imply a breakthrough clique algorithm. Our result gives a negative partial answer to an open question raised by Andoni [9].

THEOREM 1.2. *If RNA Folding on a sequence of length n can be solved in $T(n)$ time, then k-Clique on n node graphs can be solved in $O\left(T\left(n^{k/3+O(1)}\right)\right)$ time for any $k \geq 3$. Moreover, the reduction is combinatorial.*

Besides a tight lower bound for combinatorial algorithms, our result also shows that a faster than $O(n^\omega)$ algorithm for RNA Folding is unlikely. Such an upper bound is not known for the problem, leaving a small gap in the complexity of the problem after this work. However, we observe that the known reductions from RNA Folding to $(\min,+)$-matrix multiplication produce matrices with very special "bounded monotone" structure: the entries are bounded by n, and all rows and columns are monotonically increasing. This structure was recently exploited by Bringmann et al. [16] to develop the first truly subcubic algorithm for RNA Folding and other problems.

Dyck Edit Distance. The lower bound in Theorem 1.1 immediately implies a similar lower bound for the *Language Edit Distance* problem on CFLs, in which we want to be able to return the minimal edit distance between a given string w and a string in the language, i.e., zero, if w is in the language and the length of the shortest sequence of insertions, deletions, substitutions that is needed to convert w to a string that is in the language otherwise. This is a classical problem introduced by Aho and Peterson in the early 1970s with cubic time algorithms [3, 49] and many diverse applications [46, 35, 30]. Very recently, Rajasekaran and Nicolae [53] and Saha [59] obtained truly subcubic time *approximation* algorithms for the problem for arbitrary CFGs.

However, in many applications the CFG is very restricted and therefore easy to parse in linear time. One of the simplest CFGs with big practical importance is the Dyck grammar, which produces all strings of well-balanced parentheses. The Dyck recognition problem can be easily solved in linear time with a single pass on the input. Despite the grammar's very special structure, the Dyck Edit Distance problem is not known to have a subcubic time algorithm. In a recent breakthrough, Saha [58] presented a near-linear time algorithm that achieves a logarithmic approximation for the problem. Dyck Edit Distance can be viewed as a generalization of the classical string edit distance problem, whose complexity is essentially quadratic [25, 13], and Saha's approximation algorithm nearly matches the best known approximation algorithms for string edit distance of Andoni, Krauthgamer, and Onak [10], both in terms of running time and approximation factor. This naturally leads one to wonder whether the complexity of (exact) Dyck Edit Distance might also be quadratic. We prove that this is unlikely unless $\omega = 2$ or there are faster clique-finding algorithms.

THEOREM 1.3. *If Dyck Edit Distance on a sequence of length n can be solved in $T(n)$ time, then $3k$-Clique on n node graphs can be solved in $O\left(T\left(n^{k+O(1)}\right)\right)$ time for any $k \geq 1$. Moreover, the reduction is combinatorial.*

Our result gives an answer to an open question of Saha [58], who asked whether Lee's lower bound holds for the Dyck Edit Distance problem, and shows that the search for good approximation algorithms for the problem is justified since efficient exact algorithms are unlikely.

Remark. A simple observation shows that the *longest common subsequence* (LCS) problem on two sequences x, y of length n over an alphabet Σ can be reduced to RNA Folding on a sequence of length $2n$: if $y = y_1 \dots y_n$, then let $\hat{y} := y'_n \dots y'_1$, and then $RNA(x \circ \hat{y}) = LCS(x, y)$. A quadratic lower bound for LCS was recently shown under the Strong Exponential Time Hypothesis (SETH) [1, 17], which implies such a lower bound for RNA Folding as well (and, with other ideas from this work, for CFG recognition and Dyck Edit Distance). However, we are interested in higher lower bounds, ones that match Valiant's algorithm, and basing such lower bounds on SETH would imply that faster matrix multiplication algorithms refute SETH—

a highly unexpected breakthrough. Instead, we base our hardness on k-Clique and devise more delicate constructions that use the cubic-time nature of our problems.

Proof outlines. In our three proofs, the main approach is the following. We will first preprocess a graph G in $O(n^{k+O(1)})$ time in order to construct an encoding of it into a string of length $O(n^{k+O(1)})$. This will be done by enumerating all k-cliques and representing them with carefully designed gadgets such that a triple of clique gadgets will "match well" if and only if the triple forms a $3k$-clique together, that is, if all the edges between them exist. We will use a fast (subcubic time) CFG recognizer, an RNA folder, or a Dyck Edit Distance algorithm to speed up the search for such a "good" triple and solve $3k$-clique in faster than $O(n^{3k})$ time. These clique gadgets will be constructed in similar ways in all of our proofs. The main differences will be in the combination of these clique gadgets into one sequence. The challenge will be to find a way to combine $O(n^k)$ gadgets into a string in such a way that a "good" triple will affect the overall score or parseability of the string.

Subsequent work. In a follow-up work, Chang [19] greatly simplified our proof for Dyck Edit Distance by a simple direct reduction from RNA Folding to it. Moreover, the author showed that our reduction to RNA Folding can be implemented with an alphabet of size 4, as opposed to 36. The high-level structure of his reduction to RNA Folding is the same as ours, except that the atomic gadgets are implemented with a smaller alphabet, using the constructions of Bringmann and Künnemann [17].

As mentioned earlier, Bringmann et al. [16] obtained the first truly subcubic time algorithm for Language Edit Distance and RNA Folding using fast matrix multiplication and exploiting the bounded difference property of matrices. Afterwards, Bringmann and Wellnitz [18] showed conditional lower bounds for parsing tree-adjoining grammars as well.

Notation and preliminaries. All graphs in this paper will be on n nodes and $O(n^2)$ undirected and unweighted edges. We associate each node with an integer in $[n]$ and let $\bar{v}$ denote the encoding of v in binary, and we will assume that it has length exactly $2 \log n$ for all nodes. When a graph G is clear from the context, we will denote the set of all k-cliques of G by $\mathcal{C}_k$. We will denote the concatenation of sequences by $x \circ y$ and the reverse of a sequence x by x^R. Problem definitions and additional problem-specific preliminaries will be given in the corresponding section.

2. Clique to CFG recognition. In this section, we reduce Clique to CFG recognition and prove Theorem 1.1.

Given a graph $G = (V, E)$, we will construct a string w of length $O(k^2 \cdot n^{k+1})$ that encodes G. The string will be constructed in $O(k^2 \cdot n^{k+1})$ time, which is linear in its length. Then we will define our CFG $\mathcal{G}_C$, which will be independent of G or k, and it will be of constant size, such that our string w will be in the language defined by $\mathcal{G}_C$ if and only if G contains a $3k$-clique. This will prove Theorem 1.1.

Let $\Sigma = \{\mathtt{0}, \mathtt{1}, \$, \#, \mathtt{a}_{\text{start}}, \mathtt{a}_{\text{mid}}, \mathtt{a}_{\text{end}}, \mathtt{b}_{\text{start}}, \mathtt{b}_{\text{mid}}, \mathtt{b}_{\text{end}}, \mathtt{c}_{\text{start}}, \mathtt{c}_{\text{mid}}, \mathtt{c}_{\text{end}}\}$ be our set of 13 terminals (alphabet). As usual, ε will denote the empty string. We will denote the derivation rules of a CFG by $\rightarrow$ and the derivation (by applying one or multiple rules) by $\Longrightarrow$.

The string. First, we will define *node* and *list* gadgets:

$$NG(v) = \#\ \bar{v}\ \# \qquad \text{and} \qquad LG(v) = \#\ \bigcirc_{u \in N(v)} (\$\ \bar{u}^R\ \$)\ \#.$$

The CFG $\mathcal{G}_C$ (defined below) will use these gadgets to detect whether a particular vertex v (encoded as $NG(v)$) appears in the neighborhood of another vertex v' (encoded as $LG(v')$).

Consider some $t = \{v_1, \ldots, v_k\} \in \mathcal{C}_k$. We now define "clique node" and "clique list" gadgets:

$$CNG(t) = \bigcirc_{v \in t}(NG(v))^k \quad \text{and} \quad CLG(t) = (\bigcirc_{v \in t} LG(v))^k.$$

The CFG $\mathcal{G}_C$ will use these two gadgets to detect whether a k-clique t (encoded as $CNG(t)$) and a k-clique t' (encoded as $CLG(t')$) form a $2k$-clique in the graph.

Our main clique gadgets will be

$$CG_\alpha(t) = \mathtt{a}_{\text{start}}\ CNG(t)\ \mathtt{a}_{\text{mid}}\ CNG(t)\ \mathtt{a}_{\text{end}},$$

$$CG_\beta(t) = \mathtt{b}_{\text{start}}\ CLG(t)\ \mathtt{b}_{\text{mid}}\ CNG(t)\ \mathtt{b}_{\text{end}},$$

$$CG_\gamma(t) = \mathtt{c}_{\text{start}}\ CLG(t)\ \mathtt{c}_{\text{mid}}\ CLG(t)\ \mathtt{c}_{\text{end}}.$$

$\mathcal{G}_C$ will use these gadgets to detect whether three k-cliques t, t', t'' together form a $3k$-clique. It proceeds by detecting whether $t \cup t'$, $t \cup t''$, and $t' \cup t''$ are $2k$-cliques.

Finally, our encoding of a graph into a sequence is the following:

$$w = (\bigcirc_{t \in \mathcal{C}_k} CG_\alpha(t))\,(\bigcirc_{t \in \mathcal{C}_k} CG_\beta(t))\,(\bigcirc_{t \in \mathcal{C}_k} CG_\gamma(t))\,.$$

The Clique-Detecting CFG. The set of nonterminals in our grammar $\mathcal{G}_C$ is

$$\mathcal{T} = \{\mathbf{S}, \mathbf{W}, \mathbf{W}', \mathbf{V}, \mathbf{S}_{\alpha\gamma}, \mathbf{S}_{\alpha\beta}, \mathbf{S}_{\beta\gamma}, \mathbf{S}^\star_{\alpha\gamma}, \mathbf{S}^\star_{\alpha\beta}, \mathbf{S}^\star_{\beta\gamma}, \mathbf{N}_{\alpha\gamma}, \mathbf{N}_{\alpha\beta}, \mathbf{N}_{\beta\gamma}\}.$$

The "main" rules are

$$\begin{aligned}
\mathbf{S} &\to\ \mathbf{W}\ \mathtt{a}_{\text{start}}\ \mathbf{S}_{\alpha\gamma}\ \mathtt{c}_{\text{end}}\ \mathbf{W},\\
\mathbf{S}^\star_{\alpha\gamma} &\to\ \mathtt{a}_{\text{mid}}\ \mathbf{S}_{\alpha\beta}\ \mathtt{b}_{\text{mid}}\mathbf{S}_{\beta\gamma}\ \mathtt{c}_{\text{mid}},\\
\mathbf{S}^\star_{\alpha\beta} &\to\ \mathtt{a}_{\text{end}}\ \mathbf{W}\ \mathtt{b}_{\text{start}},\\
\mathbf{S}^\star_{\beta\gamma} &\to\ \mathtt{b}_{\text{end}}\ \mathbf{W}\ \mathtt{c}_{\text{start}}.
\end{aligned}$$

For every $xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}$, we will have the following rules in our grammar. These rules will be referred to as "listing" rules:

$$\begin{aligned}
\mathbf{S}_{xy} &\to\ \mathbf{S}^\star xy,\\
\mathbf{S}_{xy} &\to\ \#\ \mathbf{N}_{xy}\ \$\ \mathbf{V}\ \#,\\
\mathbf{N}_{xy} &\to\ \#\ \mathbf{S}_{xy}\ \#\ \mathbf{V}\ \$,\\
\mathbf{N}_{xy} &\to\ \sigma\ \mathbf{N}_{xy}\ \sigma \qquad\qquad \forall \sigma \in \{\mathtt{0}, \mathtt{1}\}.
\end{aligned}$$

Then we also add "assisting" rules:

$$\begin{aligned}
\mathbf{W} &\to\ \varepsilon\ \mid \sigma\ \mathbf{W} \qquad\qquad \forall \sigma \in \Sigma,\\
\mathbf{W}' &\to\ \varepsilon\ \mid \sigma\ \mathbf{W}' \qquad\qquad \forall \sigma \in \{\mathtt{0}, \mathtt{1}\},\\
\mathbf{V} &\to\ \varepsilon\ \mid \$\ \mathbf{W}'\ \$\ \mathbf{V}.
\end{aligned}$$

Our Clique-Detecting Grammar $\mathcal{G}_C$ has 13 nonterminals $\mathcal{T}$, 13 terminals Σ, and 38 derivation rules. The *size* of $\mathcal{G}_C$, i.e., the sum of the lengths of the derivation rules, is 132.

The proof. This proof is essentially by following the derivations of the CFG, starting from the starting symbol $\mathbf{S}$ and ending at some string of terminals, and showing that the resulting string must have certain properties. Any encoding of a graph into a string as we describe will have these properties if and only if the graph contains a $3k$-clique. In particular, we show the following property. If the string (corresponding to a graph) can be derived from the Clique-Detecting Grammar, then the graph has three distinct cliques of size $2k$. One clique is supported on vertices $t \cup t'$, another on vertices $t' \cup t''$, and the third on vertices $t \cup t''$. The sets t, t', t'' are disjoint, and $|t| = |t'| = |t''| = k$. There are three such sets of vertices if and only if the graph contains a $3k$-clique. The following lemma is useful for detecting whether a particular pair of sets of k vertices form a $2k$-clique.

LEMMA 2.1. *If for some $t, t' \in \mathcal{C}_k$ and $xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}$ we can get the derivation $\mathbf{S}_{xy} \implies CNG(t)\ \mathbf{S}^{\star}{}_{xy}\ CLG(t')$, only using the "listing" rules, then $t \cup t'$ forms a $2k$-clique in G.*

Proof. Any sequence of derivations starting at $\mathbf{S}_{xy}$ and ending at $\mathbf{S}^{\star}{}_{xy}$ will have the following form. From $\mathbf{S}_{xy}$ we can only proceed to nonterminals $\mathbf{V}$ and $\mathbf{N}_{xy}$. The nonterminal $\mathbf{V}$ does not produce any other nonterminals. A single instantiation of $\mathbf{S}_{xy}$ can only produce (via the second "listing" rule) a single instantiation of $\mathbf{N}_{xy}$. In turn, from $\mathbf{N}_{xy}$ we can only proceed to nonterminals $\mathbf{V}$ and $\mathbf{S}_{xy}$, and again a single instantiation of $\mathbf{N}_{xy}$ can produce a single instantiation of $\mathbf{S}_{xy}$. Thus, we produce some terminals (on the left and on the right) from the set $\{\#, \$, \mathtt{0}, \mathtt{1}\}$, and then we arrive at $\mathbf{S}_{xy}$ again. This can repeat an arbitrary number of times, until we apply the rule $\mathbf{S}_{xy} \to \mathbf{S}^{\star}{}_{xy}$.

Thus, the derivations must look like this:

$$\mathbf{S}_{xy} \implies \ell\ \mathbf{S}^{\star}_{xy}\ r$$

for some strings ℓ, r of the form $\{\#, \$, \mathtt{0}, \mathtt{1}\}^*$, and our goal is to prove that ℓ, r satisfy certain properties.

It is easy to check that $\mathbf{V}$ can derive strings of the form $p = (\$\ \{\mathtt{0}, \mathtt{1}\}^*\ \$)^*$, that is, it produces a list (possibly of length 0) of binary sequences (possibly of length 0) surrounded by \$ symbols (between every two neighboring binary sequences, there are two \$). A key observation is that repeated application of the fourth "listing" rule gives derivations $\mathbf{N}_{xy} \implies s\ \mathbf{N}_{xy}\ s^R$ for any $s \in \{\mathtt{0}, \mathtt{1}\}^*$. Combining these last two observations, we see that when starting with $\mathbf{S}_{xy}$ we can only derive strings of the following form, or terminate via the rule $\mathbf{S}_{xy} \to \mathbf{S}^{\star}{}_{xy}$:

(2.1)
$$\mathbf{S}_{xy} \implies \#\ \mathbf{N}_{xy}\ \$\ p_1\ \# \implies \#\ s\ \mathbf{N}_{xy}\ s^R\ \$\ p_1\ \# \implies \#\ s\ \#\ \mathbf{S}_{xy}\ \#\ p_2\ \$\ s^R\ \$\ p_1\ \#$$

for some p_1, p_2 of the form $(\$\ \{\mathtt{0}, \mathtt{1}\}^*\ \$)^*$.

Now consider the assumption in the statement of the lemma, and recall our constructions of "clique node gadget" and "clique list gadget." By construction, $CNG(t)$ is composed of k^2 node gadgets (NG) separated by # symbols, and CLG is composed of k^2 list gadgets (LG) separated by # symbols. Note also that the list gadgets contain $O(n)$ node gadgets within them and are separated by \$ symbols, and there are no # symbols within the list gadgets.

For every $i \in [k^2]$, let ℓ_i be the ith NG in $CNG(t)$ and let r_i be the ith LG in $CLG(t')$. Then, for every $2 \le i \le k$, in the derivation $\mathbf{S}_{xy} \implies CNG(t)\ \mathbf{S}^{\star}_{xy}\ CLG(t')$,

we must have had the derivation

$$\begin{aligned}\mathbf{S}_{xy} &\implies \ell_1 \ \cdots \ \ell_{i-1} \ (\ \mathbf{S}_{xy}\ )\ r_{k^2-i+2} \ \cdots \ r_{k^2} \\ &\implies \ \ell_1 \ \cdots \ \ell_{i-1} \ (\ \ell_i \ \mathbf{S}_{xy}\ r_{k^2-i+1}\ )\ r_{k^2-i+2} \ \cdots \ r_{k^2},\end{aligned}$$

and by (2.1) this implies that the binary encoding of the node $v \in t$ that appears in the ith NG in $CNG(t)$ must appear in one of the NGs that appears in the (k^2-i+1) LG in $CLG(t')$ which corresponds to a node $u \in t'$. Since $LG(u)$ contains a list of neighbors of the node u, this implies that $v \in N(u)$ and $\{u, v\} \in E$. Also note that $u \notin N(u)$; therefore u does not appear in $LG(u)$, and hence v cannot be equal to u if this derivation occurs.

Now consider any pair of nodes $v \in t, u \in t'$. By the construction of CNG and CLG, we must have an index $i \in [k^2]$ such that the ith NG in $CNG(t)$ is $NG(v)$ and the $(k^2 - i + 1)$ LG in $CLG(t')$ is $CLG(u)$. By the previous argument, we must have that $u \neq v$ and $\{u, v\} \in E$ is an edge. Given that t, t' are k-cliques themselves, and any pair of nodes $v \in t, u \in t'$ must be neighbors (and therefore different), we conclude that $t \cup t'$ is a $2k$-clique. ∎

Now the correctness of the reduction follows from the two claims below.

CLAIM 2.2. *If $\mathcal{G}_C \implies w$, then G contains a $3k$-clique.*

Proof. The derivation of w must look as follows. First, we must apply the only starting rule,

$$\mathbf{S} \implies w_1\ \mathtt{a}_{\text{start}}\ \mathbf{S}_{\alpha\gamma}\ \mathtt{c}_{\text{end}}\ w_2,$$

where $\mathtt{a}_{\text{start}}$ appears in $CG_\alpha(t_\alpha)$ for some $t_\alpha \in \mathcal{C}_k$ and $\mathtt{c}_{\text{start}}$ appears in $CG_\gamma(t_\gamma)$ for some $t_\gamma \in \mathcal{C}_k$, and w_1 is the prefix of w before $CG(t_\alpha)$ and w_2 is the suffix of w after $CG(t_\gamma)$. Then we can get

$$\mathbf{S}_{\alpha\gamma} \implies CNG(t_\alpha)\ \mathbf{S}^\star_{\alpha\gamma}\ CLG(t_\gamma)$$

by repeatedly applying the xy-"listing" rules, where $xy = \alpha\gamma$, and finally terminating with the rule $\mathbf{S}_{\alpha\gamma} \to \mathbf{S}^\star_{\alpha\gamma}$. By Lemma 2.1 above, this derivation is only possible if the nodes of $t_\alpha \cup t_\gamma$ make a $2k$-clique (call this observation (*)). Then we have to apply the derivation

$$\mathbf{S}^\star_{\alpha\gamma} \implies \mathtt{a}_{\text{mid}}\ \mathbf{S}_{\alpha\beta}\ \mathtt{b}_{\text{mid}}\mathbf{S}_{\beta\gamma}\ \mathtt{c}_{\text{mid}},$$

and for some $t_\beta \in \mathcal{C}_k$ we will have

$$\mathbf{S}_{\alpha\beta} \implies CNG(t_\alpha)\ \mathbf{S}^\star_{\alpha\beta}\ CLG(t_\beta) \quad \text{and} \quad \mathbf{S}_{\beta\gamma} \implies CNG(t_\beta)\ \mathbf{S}^\star_{\beta\gamma}\ CLG(t_\gamma),$$

where in both derivations we repeatedly use "listing" rules before exiting with the $\mathbf{S}_{xy} \to \mathbf{S}^\star{}_{xy}$ rule. Again, by Lemma 2.1 we get that the nodes of $t_\alpha \cup t_\beta$ are a $2k$-clique in G and that the nodes of $t_\beta \cup t_\gamma$ are a $2k$-clique as well (call this observation (**)). Finally, we will get the rest of w using the derivations

$$\mathbf{S}^\star{}_{\alpha\beta} \implies \mathtt{a}_{\text{end}}\ w_3\ \mathtt{b}_{\text{start}} \quad \text{and} \quad \mathbf{S}^\star{}_{\beta\gamma} \implies \mathtt{b}_{\text{end}}\ w_4\ \mathtt{c}_{\text{start}},$$

where w_3 is the substring of w between $CG(t_\alpha)$ and $CG(t_\beta)$, and similarly w_4 is the substring of w between $CG(t_\beta)$ and $CG(t_\gamma)$.

Combining observations (*) and (**), which we got from the above derivation scheme and Lemma 2.1, we conclude that the nodes of $t_\alpha \cup t_\beta \cup t_\gamma$ form a $3k$-clique in G, and we are done. ∎

CLAIM 2.3. *If G contains a $3k$-clique, then $\mathcal{G}_C \implies w$.*

Proof. This claim follows by following the derivations in the proof of Claim 2.2 with any triple $t_\alpha, t_\beta, t_\gamma \in \mathcal{C}_k$ of k-cliques that together form a $3k$-clique. ∎

We are now ready to prove Theorem 1.1.

Reminder of Theorem 1.1. *There is CFG $\mathcal{G}_C$ of constant size such that if we can determine whether a string of length n can be obtained from $\mathcal{G}_C$ in $T(n)$ time, then k-Clique on n node graphs can be solved in $O\left(T\left(n^{k/3+1}\right)\right)$ time for any $k \geq 3$. Moreover, the reduction is combinatorial.*

Proof. Given an instance of $3k$-Clique and a graph $G = (V, E)$, we construct the string w as described above, which will have length $O(k^2 \cdot n^{k+1})$, in $O(k^2 \cdot n^{k+1})$ time. Given a recognizer for $\mathcal{G}_C$ as in the statement, we can check whether $\mathcal{G}_C \implies w$ in $O\left(T\left(n^{k/3+1}\right)\right)$ time (treating k as a constant). By Claims 2.2 and 2.3, $\mathcal{G}_C \implies w$ if and only if the graph G contains a $3k$-clique. ∎

3. Clique to RNA Folding. In this section, we prove Theorem 1.2 by reducing k-Clique to RNA Folding, defined below.

Let Σ be an alphabet of letters of constant size. For any letter $\sigma \in \Sigma$, there will be exactly one "matching" letter, which will be denoted by σ'. Let $\Sigma' = \{\sigma' \mid \sigma \in \Sigma\}$ be the set of matching letters to the letters in Σ. Throughout this section, we will say that a pair of letters $\{x, y\}$ match if and only if $y = x'$ or $x = y'$.

Two pairs of indices $(i_1, j_1), (i_2, j_2)$ such that $i_1 < j_1$ and $i_2 < j_2$ are said to "cross" if and only if at least one of the following three conditions holds: (i) $i_1 = i_2$ or $i_1 = j_2$, or $j_1 = i_2$, or $j_1 = j_2$; (ii) $i_1 < i_2 < j_1 < j_2$; (iii) $i_2 < i_1 < j_2 < j_1$. Note that by our definition, noncrossing pairs cannot share any indices.

DEFINITION 3.1 (RNA Folding). *Given a sequence S of n letters from $\Sigma \cup \Sigma'$, what is the maximum number of pairs $A = \{(i, j) \mid i < j$ and $i, j \in [n]\}$ such that for every pair $(i, j) \in A$ the letters $S[i]$ and $S[j]$ match and there are no crossing pairs in A? We will denote this maximum value by $RNA(S)$.*

It is interesting to note that RNA can be seen as a language distance problem with respect to some easy-to-parse grammar. Because of the specific structure of this grammar, our reduction from section 2 does not apply. However, the ideas we introduced allow us to replace our clique-detecting grammar with an easier grammar if we ask the parser to return more information, like the distance to a string in the grammar. At a high level, this is how we get the reduction to RNA Folding presented in this section.

To significantly simplify our proofs, we will reduce k-Clique to a more general *weighted* version of RNA Folding. Below we show that this version can be reduced to the standard RNA folding problem with a certain overhead.

DEFINITION 3.2 (Weighted RNA Folding). *Given a sequence S of n letters from $\Sigma \cup \Sigma'$ and a weight function $w : \Sigma \to [M]$, what is the* maximum weight *of a set of pairs $A = \{(i, j) \mid i < j$ and $i, j \in [n]\}$ such that for every pair $(i, j) \in A$ the letters $S[i]$ and $S[j]$ match and there are no crossing pairs in A? The weight of A is defined as $\sum_{(i,j)\in A} w(S[i])$. We will denote this maximum value by $WRNA(S)$.*

LEMMA 3.3. *An instance S of Weighted RNA Folding on a sequence of length n, alphabet $\Sigma \cup \Sigma'$, and weight function $w : \Sigma \to [M]$ can be reduced to an instance $\hat{S}$ of RNA Folding on a sequence of length $O(Mn)$ over the same alphabet.*

Proof. Let $S = S_1 \cdots S_n$, and set $\hat{S} := S_1^{w(S_1)} \cdots S_n^{w(S_n)}$; that is, each symbol S_i is repeated $w(S_i)$ times. First, we can check that $WRNA(S) \leq RNA(\hat{S})$. This holds because we can replace each matching pair $\{a, a'\}$ in the folding achieving weighted RNA score of $WRNA(S)$ with $w(a)$ such pairs in the (unweighted) RNA folding instance $\hat{S}$ giving the same contribution to $RNA(\hat{S})$.

Now we will show that $RNA(\hat{S}) \leq WRNA(S)$. Suppose that there are symbols a and a' in $\hat{S}$ that are paired. The symbol a comes from a sequence s_1 of a symbols of length $w(a)$. The sequence s_1 was produced from a single symbol a when transforming S into $\hat{S}$. Similarly, the symbol a' comes from a sequence s_2 of a' symbols of length $w(a)$. Also, assume that there exists a symbol in s_1 that is paired to a symbol that is outside of s_2 or there exists a symbol in s_2 that is paired to a symbol outside of s_1. While we can find such symbols a and a', we repeat the following procedure. Choose a and a' that satisfy the above properties. And choose them so that the number of other symbols between a and a' is as small as possible. Break ties arbitrarily. We match all symbols in s_1 to their counterparts in s_2. Also, we rematch all symbols that were previously matched to s_1 or s_2 among themselves. We can check that we can rematch these symbols so that the number of matched pairs does not decrease.

Therefore, we can assume that in some optimal folding of $\hat{S}$, for any pair $a \in \Sigma, a' \in \Sigma'$ that is matched the corresponding substrings s_1 and s_2 are completely paired up. Thus, to get a folding of S that achieves $WRNA(S)$ at least $RNA(\hat{S})$ we can now simply fold the corresponding symbols to s_1 and s_2 for any such pair a, a'. ∎

3.1. The reduction. Given a graph $G = (V, E)$ on n nodes and $O(n^2)$ unweighted undirected edges, we will describe how to efficiently construct a sequence S_G over an alphabet Σ of constant size, such that the RNA score of S_G will depend on whether G contains a $3k$-clique. The length of S_G will be $O(k^d n^{k+c})$ for some small fixed constants $c, d > 0$ independent of n and k, and the time to construct it from G will be linear in its length. This will prove that a fast (e.g., subcubic time) RNA folder can be used as a fast $3k$-clique detector (one that runs much faster than in $O(n^{3k})$ time).

Our main strategy will be to enumerate all k-cliques in the graph and then search for a triple of k-cliques that have all the edges between them. We will be able to find such a triple if and only if the graph contains a $3k$-clique. An RNA folder will be utilized to speed up the search for such a "good" triple. Our reduction will encode every k-clique of G using a "short" sequence of length $O(n^c)$ such that the RNA folding score of a sequence composed of the encodings of a triple of sequences will be large if and only if the triple is "good." Then we will show how to combine the short encodings into our long sequence S_G such that the existence of a "good" triple affects the overall score of an optimal folding.

The RNA sequence. Our sequence S_G will be composed of many smaller gadgets which will be combined in certain ways by other padding gadgets. We construct these gadgets now and explain their useful properties. The proofs of these properties are postponed until after we present the whole construction of S_G.

For a sequence $s \in \Sigma^*$, let $p(s) \in (\Sigma')^*$ be the sequence obtained from s by replacing every letter $\sigma \in \Sigma$ with the matching letter $\sigma' \in \Sigma'$. That is, if $s = s_1 \cdots s_n$, then $p(s) = s'_1 \cdots s'_n$.

Our alphabet Σ will contain the letters `1` and `$` and some additional symbols which we will add as needed in our gadgets. We set the weights $w(\texttt{1}) = w(\$) = 1$,

and the extra symbols we add will be more "expensive." We define *node* gadgets as

$$NG(v) = \bigcirc_{u \in V} (\$\ P(u = v)\ \$),$$

where $P(A) = \mathtt{1}$ if the event A holds and $P(A)$ is an empty sequence if the event A does not hold. We define *list* or *neighborhood* gadgets as

$$LG(v) = \bigcirc_{u \in V} (\$\ P(u \in N(v))\ \$).$$

These gadgets are constructed so that for any two nodes $u, v \in V(G)$, the RNA folding score of the sequence $NG(v) \circ p\,(LG(u))^R$ is large (equal to some fixed value E_1) if v is in the neighborhood of u, that is, $(u, v) \in E(G)$, and smaller otherwise (at most $E_1 - 1$). This is because the more expensive \$ symbols must be matched in pairs (contributing a fixed score) and $P(v = v) \circ p(P(v \in N(u)))^R$ contributes score 1 if $v \in N(u)$ and otherwise contributes score 0. We formalize this in Claim 3.4.

Let $\ell_1 = 10k^2 \cdot n$, and note that ℓ_1 is an upper bound on the total weight of all the symbols in the gadgets $NG(v)$ and $LG(v)$ for any node $v \in V(G)$. Let $\mathcal{C}_k$ be the set of k-cliques in G, and consider some $t = \{v_1, \ldots, v_k\} \in \mathcal{C}_k$. We will now combine the node and list gadgets into larger gadgets that will be encoding k-cliques.

We will add the # symbol to the alphabet and set $w(\#) = \ell_1$; i.e., a single # letter is more expensive than an entire k^2 node or list gadget. We will encode a clique in two ways. The first one is

$$CNG(t) = \bigcirc_{v \in t} (\#NG(v)\#)^k,$$

and the second one is

$$CLG(t) = (\bigcirc_{v \in t} (\#LG(v)\#))^k.$$

These clique gadgets are very useful because of the following property. For any two k-cliques $t_1, t_2 \in \mathcal{C}_k$, the RNA folding score of the sequence $CNG(t_1) \circ p\,(CLG(t_2))^R$ is large (equal to some fixed value E_2) if t_1 and t_2 together form a $2k$-clique, and is smaller otherwise (at most $E_2 - 1$). That is, the RNA folding score of the sequence tells us whether any pair of nodes $u \in t_1, v \in t_2$ are connected by $(u, v) \in E(G)$. There are two ideas in the construction of these gadgets. First, we copy the gadgets corresponding to the k nodes of the cliques k times, resulting in k^2 gadgets, and we order them in such a way so that for any pair of nodes $u \in t_1, v \in t_2$ there will be a position i such that the gadget of u in $CNG(t_1)$ and the gadget of v in $p(CLG(t_2))^R$ are both at position i. Then we use the expensive # separators to make sure that in an optimal RNA folding of $CNG(t_1)$ and $p(CLG(t_2))^R$ the gadgets at positions i are folded together, and not to other gadgets; otherwise, some # symbol will not be paired. This is formally proved in Claim 3.5.

Let $\ell_2 = 10 \cdot k^2 \cdot \ell_1 = O(n)$, and note that it is an upper bound on the total weight of all the symbols in the $CNG(t)$ and $CLG(t)$ gadgets. Finally, we introduce a new letter to the alphabet $\mathtt{g}$ and set its weight to $w(\mathtt{g}) = \ell_2$, which is much more expensive than the entire gadgets we constructed before, and then define our final clique gadgets. Moreover, we will now duplicate our alphabet three times to force only "meaningful" foldings between our gadgets. It will be convenient to think of α, β, γ as three *types* such that we will be looking for three k-cliques: one from type α, one from β, and one from γ. For any pair of types $xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}$, we will construct a new alphabet $\Sigma_{xy} = \{\sigma_{xy} \mid \sigma \in \Sigma\}$, in which we mark each letter with the pair of types in which it should be participating. For a sequence $s \in (\Sigma \cup \Sigma)^*$, we

use the notation $[s]_{xy}$ to represent the sequence in $(\Sigma_{xy} \cup \Sigma'_{xy})^*$ in which we replace every letter σ with the letter σ_{xy}.

We will need three types of these clique gadgets in order to force the desired interaction between them:

$$CG_\alpha(t) = [\mathtt{g}\ CNG(t)\ \mathtt{g}]_{\alpha\gamma}\ \circ\ [\mathtt{g}'\ p(CLG(t))^R\ \mathtt{g}']_{\alpha\beta},$$
$$CG_\beta(t) = [\mathtt{g}\ CNG(t)\ \mathtt{g}]_{\alpha\beta}\ \circ\ [\mathtt{g}'\ p(CLG(t))^R\ \mathtt{g}']_{\beta\gamma},$$
$$CG_\gamma(t) = [\mathtt{g}\ CNG(t)\ \mathtt{g}\ ]_{\beta\gamma}\ \circ\ [\mathtt{g}'\ p(CLG(t))^R\ \mathtt{g}']_{\alpha\gamma}.$$

These clique gadgets achieve exactly what we want: for any three k-cliques $t_\alpha, t_\beta, t_\gamma \in \mathcal{C}_k$, the RNA folding score of the sequence $CG_\alpha(t_\alpha) \circ CG_\beta(t_\beta) \circ CG_\gamma(t_\gamma)$ is large (equal to some value E_3) if $t_\alpha \cup t_\beta \cup t_\gamma$ is a $3k$-clique and smaller otherwise (at most $E_3 - 1$). In other words, an RNA folder can use these gadgets to determine whether three separate k-cliques can form a $3k$-clique. This is achieved by noticing that the highest priority for an optimal folding would be to match the $\mathtt{g}_{xy}$ letters with their counterparts $\mathtt{g}'_{xy}$, which leaves us with three sequences to fold: $S_{\alpha\gamma} = [CNG(t_\alpha) \circ p(CLG(t_\gamma))^R]_{\alpha\gamma}$, $S_{\alpha\beta} = [p(CLG(t_\alpha))^R \circ CNG(t_\beta)]_{\alpha\beta}$, and $S_{\beta\gamma} = [p(CLG(t_\beta))^R \circ CNG(t_\gamma)]_{\beta\gamma}$. The maximal score ($E_2$) in each one of these three sequences can be achieved if and only if every pair of our k-cliques form a $2k$-clique, which happens if and only if they form a $3k$-clique. This is formally proved in Claim 3.6.

The only remaining challenge is to combine all the sequences corresponding to all the $O(n^k)$ k-cliques in the graph into one sequence in such a way that the existence of a "good" triple, one that makes a $3k$-clique, affects the RNA folding score of the entire sequence. Note that if we naively concatenate all the clique gadgets into one sequence, the optimal sequence will choose to fold clique gadgets in pairs instead of triples since folding a triple makes other gadgets unable to fold without crossings. Instead, we will use the structure of the RNA folding problem again to implement a "selection" gadget that forces exactly three clique gadgets to fold together in any optimal folding. We remark that the implementation of such "selection" gadgets is very different in the three proofs in this paper: In section 2 we use the derivation rules, in this section we use the fact that even when folding all expensive separators in a sequence to the left or right we are left with an interval that is free to fold with other parts of the sequence, and in section 4 we rely on the restriction of Dyck that an opening bracket can match only with closing brackets to its right.

Towards this end, we introduce some extremely expensive symbols α, β, γ. Let $\ell_3 = 10\ell_2$ be an upper bound on the total weight of the $CG_x(t)$ gadgets, and set $w(\alpha) = w(\beta) = w(\gamma) = \ell_3$. Our clique-detecting RNA sequence is defined as follows:

$$\begin{aligned} S_G\ =\ & \alpha^{2n^k}\ \bigcirc_{t\in\mathcal{C}_k} (\alpha'\ CG_\alpha(t)\ \alpha')\ \alpha^{2n^k}, \\ & \circ\ \beta^{2n^k}\ \bigcirc_{t\in\mathcal{C}_k} (\beta'\ CG_\beta(t)\ \beta')\ \beta^{2n^k}, \\ & \circ\ \gamma^{2n^k}\ \bigcirc_{t\in\mathcal{C}_k} (\gamma'\ CG_\gamma(t)\ \gamma')\ \gamma^{2n^k}. \end{aligned}$$

The added padding makes sure that all but one CG_α gadget are impossible to fold without giving up an extremely valuable α, α' pair, and similarly all but one CG_β and one CG_γ cannot be folded. To see this, assume all the α' are paired (left or right) and note that if both α' symbols surrounding a clique gadget $CG_\alpha(t)$ are paired to one side (say, left), then the only noncrossing pairs in which the gadget

could participate are either with α symbols (but those cannot be matches) or within itself. Our marking of symbols with pairs of types xy make it so that a clique gadget cannot have any matches with itself. Therefore, if all α' symbols are matched, then all but one $CG_\alpha(t)$ gadgets do not participate in any foldings. The argument for β, γ is symmetric. We are left with a folding of a sequence of three clique gadgets $CG_\alpha(t_\alpha), CG_\beta(t_\beta), CG_\gamma(t_\gamma)$ which can achieve maximal score if and only if $t_\alpha \cup t_\beta \cup t_\gamma$ is a $3k$-clique.

This proves our main claim that the (weighted) RNA folding score of our clique-detecting sequence S_G is large (equal to some fixed value E_C) if the graph contains a $3k$-clique and smaller (at most $E_C - 1$) otherwise. See Claim 3.7 for the formal proof.

Our final alphabet Σ has size 17 (together with Σ' this makes 34 symbols):

$$\Sigma = \{\alpha, \beta, \gamma\} \cup \bigcup_{xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}} \{\mathtt{1}, \$, \#, g\}_{xy}.$$

Observe that S_G can be constructed from G in $O(n^{k+1})$ time, by enumerating all subsets of k nodes, and that it has length $O(n^{k+1})$. The construction of S_G should be seen as a heavy preprocessing and encoding of the graph, after which we only have to work with k-cliques. The largest weight we use in our construction is $\ell_3 = O(k^{O(1)}n)$, and therefore using Lemma 3.3 we can reduce the computation of the weighted RNA of S_G to an instance of (Unweighted) RNA Folding on a sequence of length $O(|S_G|k^{O(1)}n) = \tilde{O}(k^{O(1)}n^{k+2})$, which proves Theorem 1.2.

Formal proofs. We will start with the proof that the list and node gadgets have the desired functionality. Let $E_1 = 1 + 2n$.

CLAIM 3.4. *For any $xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}$ and two nodes $u, v \in V(G)$, we have that the weighted RNA folding score $WRNA([NG(u) \circ p(LG(v))^R]_{xy})$ is E_1 if $u \in N(v)$ and at most $E_1 - 1$ otherwise.*

Proof. Since all letters in the sequence we are concerned with have the same mark xy, we will omit the subscripts. If $u \in N(v)$, then we can match all \$ in pairs (giving total score $2n$) and we can match $P(u = u)$ with $p(P(u \in N(v)))^R$ since $P(u = u) = P(u \in N(v)) = \mathtt{1}$ (giving score 1). Therefore, in this case, the weighted RNA score is $E_1 = 1 + 2n$. We note that in this case we were able to match all symbols from $NG(u)$. It remains to observe that in the case of $u \notin N(v)$ we cannot match $P(u = u) = \mathtt{1}$ with $P(u \in N(v))$, which is the empty sequence. Therefore, the total score in this case is no more than $E_1 - 1$. ∎

Next, we prove that the "clique node gadgets" and "clique list gadgets" check that two k-cliques form one bigger $2k$-clique. Let $E_2 = 2k^2 \cdot \ell_1 + k^2 \cdot E_1$.

CLAIM 3.5. *For any $xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}$ and two k-cliques $t_1, t_2 \in \mathcal{C}_k$, we have that the weighted RNA folding score $WRNA([CNG(t_1) \circ p(CLG(t_2))^R]_{xy})$ is E_2 if $t_1 \cup t_2$ is a $2k$-clique and at most $E_2 - 1$ otherwise.*

Proof. We will omit the irrelevant xy subscripts. First, note that the sequences $CNG(t_1)$ and $p(CLG(t_2))^R$ have the same number of # and #′ symbols, respectively. By not pairing a single one of them with its counterpart, we lose a contribution of $w(\#)$ to the WRNA score, which is much more than we could gain by pairing all the symbols in all the node and list gadgets (that is, the rest of the sequence). Therefore, we assume that all the # and #′ symbols are paired. Let $t_1 = \{u_1, \ldots, u_k\}$ and

$t_2 = \{v_1, \ldots, v_k\}$. We can now say that

$$WRNA(CNG(t_1) \circ p(CLG(t_2))^R) = (2k^2)w(\#) + \sum_{i\in[k]} \sum_{j\in[k]} WRNA(NG(u_i) \circ p(LG(v_j))^R),$$

and by Claim 3.4 we know that $WRNA(NG(u_i) \circ p(LG(v_j))^R) = E_1$ if u_i and v_j are connected and less otherwise. Therefore, we can only get the maximal $E_2 = 2k^2 \cdot \ell_1 + k^2 \cdot E_1$ if and only if every pair of nodes in $t_1 \times t_2$ are connected by an edge. Since $u \notin N(u)$ for all $u \in V(G)$ and since t_1, t_2 are k-cliques, we conclude that $t_1 \cup t_2$ is a $2k$-clique. ∎

We are now ready to prove the main property of our clique gadgets: a sequence of three clique gadgets (one from each type) achieves maximal score if and only if they form a $3k$-clique together. Let $E_3 = 6\ell_2 + 3E_2$.

Claim 3.6. *For any $xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}$ and three k-cliques $t_\alpha, t_\beta, t_\gamma \in \mathcal{C}_k$, we have that the weighted RNA folding score $WRNA(CG_\alpha(t_\alpha) \circ CG_\beta(t_\beta) \circ CG_\gamma(t_\gamma))$ is E_3 if $t_1 \cup t_2 \cup t_3$ is a $3k$-clique and at most $E_3 - 1$ otherwise.*

Proof. If for some $xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}$ there is a symbol $\mathtt{g}_{xy}$ which is not paired up with its counterpart, we lose a contribution to the $WRNA$ score that is more than we could get by pairing up all symbols that are not $\mathtt{g}_{xy}$. Therefore, we have the equality

$$\begin{aligned} WRNA(CG_\alpha(t_\alpha) \circ CG_\beta(t_\beta) \circ CG_\gamma(t_\gamma)) = {} & 3 \cdot 2\ell_2 \\ & + WRNA([CNG(t_\alpha) \circ p(CLG(t_\gamma))^R]_{\alpha\gamma}) \\ & + WRNA([CNG(t_\beta) \circ p(CLG(t_\gamma))^R]_{\beta\gamma}) \\ & + WRNA([p(CLG(t_\alpha))^R \circ CNG(t_\beta)]_{\alpha\beta}). \end{aligned}$$

By Claim 3.5, the last three summands are equal to E_2 if all our three k-cliques are pairwise $2k$-cliques and otherwise at least one of the summands is less than E_2. The claim follows by noticing that $t_\alpha \cup t_\beta \cup t_\gamma$ is a $3k$-clique if and only if the three k-cliques are pairwise $2k$-cliques. ∎

We are now ready to prove our main claim about S_G. This proof shows that our "selection" gadgets achieve the desired property of having exactly one clique from each type of fold in an optimal matching. Let $N = O(n^k)$ be the size of $\mathcal{C}_k$ which is the number of k-cliques in our graph and therefore the number of clique gadgets we will have from each type. We will set $E_C = 6N + E_3$.

Claim 3.7. *The weighted RNA score of S_G is E_C if G contains a $3k$-clique and at most $E_C - 1$ otherwise.*

Proof. Let $x \in \{\alpha, \beta, \gamma\}$, and let $t_x \geq 0$ denote the number of x' symbols in S_G that are not paired. Because any clique gadget CG_x can only have matches with letters from clique gadgets CG_y for some $y \in \{\alpha, \beta, \gamma\}$ such that $y \neq x$, we can say that at most $t_x/2 + 1$ clique gadget sequences CG_x can have letters that participate in the folding.

Recall that by the definition of our weights, the total weight of any clique gadget is much less than $\ell_3/10$, where ℓ_3 is the weight of a letter α, β, or γ, and recall the definition of $N = |\mathcal{C}_k|$. We will use the inequalities

$$\begin{aligned} WRNA(S_G) \leq {} & ((t_\alpha/2 + 1) + (t_\beta/2 + 1) \\ & + (t_\beta/2 + 1)) \cdot \ell_3/10 + ((2N - t_a) + (2N - t_b) + (2N - t_c))\ell_3 \end{aligned}$$

and

$$WRNA(S_G) \geq ((2N - t_a) + (2N - t_b) + (2N - t_c))\ell_3.$$

Since $\ell_3 \gg \ell_3/10$, we must have that $t_\alpha = t_\beta = t_\gamma = 0$ in any optimal folding of S_G. Now we get that

$$WRNA(S_G) = 6N\ell_3 + WRNA(CG_\alpha(t_\alpha) \circ CG_\beta(t_\beta) \circ CG_\gamma(t_\gamma))$$

for some k-cliques $t_\alpha, t_\beta, t_\gamma \in \mathcal{C}_k$. By Claim 3.6, the last summand can be equal to E_3 if and only if the graph has a $3k$-clique and must be at most $E_3 - 1$ otherwise. This and the fact that $E_C = 6N\ell_3 + E_3$ completes the proof. ∎

We are now ready to show that the construction of S_G from graph G proves Theorem 1.2.

Reminder of Theorem 1.2. *If RNA Folding on a sequence of length n can be solved in $T(n)$ time, then k-Clique on n node graphs can be solved in $O\left(T\left(n^{k/3+O(1)}\right)\right)$ time for any $k \geq 3$. Moreover, the reduction is combinatorial.*

Proof. Given a graph G on n nodes, we construct the sequence S_G as described above. The sequence can be constructed in $O(k^{O(1)} \cdot n^{k+1})$ time by enumerating all subsets of k nodes and the fact that it has length $O(k^{O(1)} \cdot n^{k+1})$. The largest weight we use in our construction is $\ell_3 = O(k^{O(1)}n)$, and therefore using Lemma 3.3 we can reduce the computation of the weighted RNA of S_G to an instance of (Unweighted) RNA Folding on a sequence of length $O(|S_G|k^{O(1)}n) = \tilde{O}(k^{O(1)}n^{k+2})$. Thus, an RNA folder as in the statement returns the weighted RNA folding score of S_G in $T(n^{k/3+2})$ time (treating k as a constant), and by Claim 3.7 this score determines whether G contains a $3k$-clique. All the steps in our reduction are combinatorial. ∎

4. Clique to Dyck Edit Distance. In this section, we prove Theorem 1.3 by reducing k-Clique to the Dyck Edit Distance problem, defined below.

The Dyck grammar is defined over a fixed size alphabet of opening brackets Σ and of closing brackets $\Sigma' = \{\sigma' \mid \sigma \in \Sigma\}$, such that σ can only be closed by σ'. A string S belongs to the Dyck grammar if the brackets in it are well-formed. More formally, the Dyck grammar is defined by the rules $\mathbf{S} \to \mathbf{SS}$ and $\mathbf{S} \to \sigma\ \mathbf{S}\ \sigma'$ for all $\sigma \in \Sigma$ and $\mathbf{S} \to \varepsilon$. This grammar defines the Dyck CFL (which can be parsed in linear time).

The Dyck Edit Distance problem is as follows: given a string S over $\Sigma \cup \Sigma'$, find the minimum edit distance from S to a string in the Dyck CFL. In other words, find the shortest sequence of substitutions and deletions that is needed to convert S into a string that belongs to Dyck. We will refer to this distance as the Dyck *score* or *cost* of S.

Let us introduce alternative ways to look at the Dyck Edit Distance problem that will be useful for our proofs. Two pairs of indices $(i_1, j_1), (i_2, j_2)$ such that $i_1 < j_1$ and $i_2 < j_2$ are said to "cross" if and only if at least one of the following three conditions holds:

- $i_1 = i_2$ or $i_1 = j_2$, or $j_1 = i_2$, or $j_1 = j_2$;
- $i_1 < i_2 < j_1 < j_2$;
- $i_2 < i_1 < j_2 < j_1$.

Note that by our definition, noncrossing pairs cannot share any indices. We define an *alignment* A of a sequence S of length n to be a set of noncrossing pairs (i, j), $i < j, i, j \in [n]$. If (i, j) is in our alignment, we say that letters i and j are *aligned*. We say that an aligned pair is a *match* if $S[i] = \sigma$ for some $\sigma \in \Sigma$ and $S[j] = \sigma'$, i.e., an

opening bracket and the corresponding closing bracket. Otherwise, we say that the aligned pair is a *mismatch*. Mismatches will correspond to substitutions in an edit distance transcript. A letter at an index i that does not appear in any of the pairs in the alignment is said to be *deleted*. We define the cost of an alignment to be the number of mismatches plus the number of deleted letters. One can verify that any alignment of cost E corresponds to an edit distance transcript from S to a string in Dyck of cost E, and vice versa.

4.1. The reduction. Given a graph $G = (V, E)$ on n nodes and $O(n^2)$ unweighted undirected edges, we will describe how to efficiently construct a sequence S_G over an alphabet Σ of constant size, such that the Dyck score of S_G will depend on whether G contains a $3k$-clique. The length of S_G will be $O(k^d n^{k+c})$ for some small fixed constants $c, d > 0$ independent of n and k, and the time to construct it from G will be linear in its length. This will prove that a fast (e.g., subcubic) algorithm for Dyck Edit Distance can be used as a fast $3k$-clique detector (one that runs much faster than in $O(n^{3k})$ time).

As in the other sections, our main strategy will be to enumerate all k-cliques in the graph and then search for a triple of k-cliques that have all the edges between them. We will be able to find such a triple if and only if the graph contains a $3k$-clique. A Dyck Edit Distance algorithm will be utilized to speed up the search for such a "good" triple. Our reduction will encode every k-clique of G using a "short" sequence of length $O(n^c)$ such that the Dyck score of a sequence composed of the encodings of a triple of sequences will be large if and only if the triple is "good." Then we will show how to combine the short encodings into our long sequence S_G such that the existence of a "good" triple affects the overall score of an optimal alignment.

The sequence. Our sequence S_G will be composed of many smaller gadgets which will be combined in certain ways by other padding gadgets. We construct these gadgets now and explain their useful properties. The proofs of these properties are postponed until after we present the whole construction of S_G.

For a sequence $s \in \Sigma^*$, let $p(s) \in (\Sigma')^*$ be the sequence obtained from s by replacing every letter $\sigma \in \Sigma$ with the closing bracket $\sigma' \in \Sigma'$. That is, if $s = s_1 \cdots s_n$, then $p(s) = s'_1 \cdots s'_n$.

Our alphabet Σ will contain the letters $\mathtt{0}, \mathtt{1}$ and some additional symbols which we will add as needed in our gadgets, like $\$, \#$. We will use the numbers $\ell_2, \ldots, \ell_5$ such that $\ell_i = (1000 \cdot n^2)^{i+1}$, which can be bounded by $n^{O(1)}$. We define *node* gadgets as

$$NG(v) = \bigcirc_{u \in V} (\$\ P(u = v)\ \$),$$

where $P(A) = \mathtt{1}$ if the event A holds and $P(A)$ is an empty sequence if the event A does not hold. We define *list* or *neighborhood* gadgets as

$$LG(v) = \bigcirc_{u \in V} (\$\ P'(u \in N(v))\ \$),$$

where $P'(A) = \mathtt{1}$ if the event A holds and $P'(A) = \mathtt{0}$ if the event A does not hold.

These gadget are constructed so that for any two nodes $u, v \in V(G)$, the Dyck score of the sequence $NG(v) \circ p\left(LG(u)\right)^R$ is small (equal to some fixed value E_1) if v is in the neighborhood of u, that is, $(u, v) \in E(G)$, and larger otherwise (at least $E_1 + 1$). This is proved formally in Claim 4.1 by arguments similar to those in section 3.

Note that ℓ_2 is an upper bound on the total length of all the symbols in the gadgets $NG(v)$ and $LG(v)$ for any node $v \in V(G)$. Let $\mathcal{C}_k$ be the set of k-cliques in

G, and consider some $t = \{v_1, \ldots, v_k\} \in \mathcal{C}_k$. We will now combine the node and list gadgets into larger gadgets that will be encoding k-cliques.

We will encode a clique in two ways. The first one is

$$CNG(t) = \bigcirc_{v \in t} \ (\#^{\ell_2} \ NG(v) \ \#^{\ell_2})^k,$$

and the second one is

$$CLG(t) = (\bigcirc_{v \in t} \ (\#^{\ell_2} \ LG(v) \ \#^{\ell_2}))^k.$$

Note that ℓ_3 is an upper bound on the total length of all the symbols in the $CNG(t)$ and $CLG(t)$ gadgets. We will add the symbol $\mathtt{g}$ to the alphabet. Moreover, we will now duplicate our alphabet three times to force only "meaningful" alignments between our gadgets. It will be convenient to think of α, β, γ as three *types* such that we will be looking for three k-cliques: one from type α, one from β, and one from γ. For any pair of types $xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}$, we will construct a new alphabet $\Sigma_{xy} = \{\sigma_{xy} \mid \sigma \in \Sigma\}$, in which we mark each letter with the pair of types in which it should be participating. For a sequence $s \in (\Sigma \cup \Sigma')^*$, we use the notation $[s]_{xy}$ to represent the sequence in $(\Sigma_{xy} \cup \Sigma'_{xy})^*$ in which we replace every letter σ with the letter σ_{xy}.

We will need three types of these clique gadgets in order to force the desired interaction between them:

$$\begin{aligned}
CG_\alpha(t) &= \mathtt{a}^{\ell_4} \ (\mathtt{x}'_\alpha)^{\ell_5} \ [\mathtt{g}^{\ell_3} \ CNG(t) \ \mathtt{g}^{\ell_3}]_{\alpha\gamma} \quad \circ \quad [\mathtt{g}^{\ell_3} \ p(CNG(t))^R \ \mathtt{g}^{\ell_3}]_{\alpha\beta} \ \ \mathtt{y}_\alpha^{\ell_5} \ (\mathtt{a}')^{\ell_4}, \\
CG_\beta(t) &= \mathtt{b}^{\ell_4} \ (\mathtt{x}'_\beta)^{\ell_5} \ [(\mathtt{g}')^{\ell_3} \ CLG(t) \ (\mathtt{g}')^{\ell_3}]_{\alpha\beta} \quad \circ \quad [\mathtt{g}^{\ell_3} \ p(CNG(t))^R \ \mathtt{g}^{\ell_3}]_{\beta\gamma} \ \ \mathtt{y}_\beta^{\ell_5} \ (\mathtt{b}')^{\ell_4}, \\
CG_\gamma(t) &= \mathtt{c}^{\ell_4} \ (\mathtt{x}'_\gamma)^{\ell_5} \ [(\mathtt{g}')^{\ell_3} \ CLG(t) \ (\mathtt{g}')^{\ell_3} \ ]_{\beta\gamma} \quad \circ \quad [(\mathtt{g}')^{\ell_3} \ p(CLG(t))^R \ (\mathtt{g}')^{\ell_3}]_{\alpha\gamma} \ \ \mathtt{y}_\gamma^{\ell_5} \ (\mathtt{c}')^{\ell_4}.
\end{aligned}$$

These clique gadgets achieve exactly what we want: for any three k-cliques $t_\alpha, t_\beta, t_\gamma \in \mathcal{C}_k$, the Dyck score of the sequence $CG_\alpha(t_\alpha) \circ CG_\beta(t_\beta) \circ CG_\gamma(t_\gamma)$ is small (equal to some value E_3) if $t_\alpha \cup t_\beta \cup t_\gamma$ is a $3k$-clique and larger otherwise (at least $E_3 + 1$). This is formally proved in Claim 4.4, again by arguments similar to those in section 3 (but more complicated because of the possible mismatches).

The main difference over the proof of section 3 is the way we implement the "selection" gadgets. We want to combine all the clique gadgets into one sequence in such a way that the existence of a "good" triple, one that makes a $3k$-clique, affects the Dyck score of the entire sequence. The ideas we used in the RNA proof do not immediately work here because of the "beneficial mismatches" of the separators we add with themselves and because in Dyck (σ, σ') match but (σ', σ) do not (while in RNA we do not care about the order). We will use some new ideas.

Our clique-detecting sequence is defined as follows:

$$\begin{aligned}
S_G \ = \quad & \mathtt{x}_\alpha{}^{\ell_5} \ (\bigcirc_{t \in \mathcal{C}_k} CG_\alpha(t)) \ \mathtt{y}'_\alpha{}^{\ell_5} \\
& \circ \ \mathtt{x}_\beta{}^{\ell_5} \ (\bigcirc_{t \in \mathcal{C}_k} CG_\beta(t)) \ \mathtt{y}'_\beta{}^{\ell_5} \\
& \circ \ \mathtt{x}_\gamma{}^{\ell_5} \ (\bigcirc_{t \in \mathcal{C}_k} CG_\gamma(t)) \ \mathtt{y}'_\gamma{}^{\ell_5}.
\end{aligned}$$

As the $\mathtt{x}_\alpha, \mathtt{y}'_\alpha$ symbols are very rare and "expensive," an optimal alignment will match them to some of their counterparts within the α part of the sequence. However, when the $\mathtt{x}'_\alpha, \mathtt{y}_\alpha$ letters are matched, we cannot match the adjacent $\mathtt{a}, \mathtt{a}'$ symbols, which are also quite expensive. Therefore, the optimal behavior is to match the $\mathtt{x}_\alpha, \mathtt{y}'_\alpha$ from *exactly one* "interval" from the α part. A similar argument holds for the β, γ parts. This behavior leaves exactly one clique gadget from each type to be

aligned freely with each other as a triple. By the construction of these gadgets, an optimal score can be achieved if and only if there is a $3k$-clique.

This proves our main claim that the Dyck score of our clique-detecting sequence S_G is small (equal to some fixed value E_C) if the graph contains a $3k$-clique and larger (at least $E_C + 1$) otherwise. See Claim 4.5 for the formal proof.

When k is fixed, S_G can be constructed from G in $O(n^{k+O(1)})$ time by enumerating all subsets of k nodes and the fact that it has length $O(n^{k+O(1)})$. This proves Theorem 1.3.

Our final alphabet Σ has size 24 (together with Σ' this makes 48 symbols):

$$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\} \cup \bigcup_{xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}} \{\mathtt{0}, \mathtt{1}, \$, \#, \mathtt{g}, \mathtt{x}, \mathtt{y}\}_{xy}.$$

Formal proofs. Let $E_1 = n - 1$.

CLAIM 4.1. *For any $xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}$, if $v \in N(u)$, then*

$$Dyck([NG(v) \circ p(LG(u))]_{xy}) = E_1$$

and $> E_1$ otherwise.

Proof. We will omit the subscripts xy since they do not matter for the proof.

Case $v \in N(u)$. In this case, $NG(v)$ is a subsequence of $LG(u)^R$ and the Dyck score is $|LG(u)| - |NG(v)| = n - 1$.

Case $v \notin N(u)$. In this case, $NG(v)$ is *not* a subsequence of $LG(u)^R$ and the Dyck score is $> |LG(u)| - |NG(v)| = n - 1$. ∎

For the next proofs, we will use the following definition.

DEFINITION 4.2. *Given two sequences P and T, we define*

$$pattern(P, T) := \min_{\substack{Q \text{ is a contiguous} \\ \text{subsequence of } T}} Dyck(P \circ Q).$$

Let $E_2 = k^2 \cdot E_1$.

CLAIM 4.3. *For any $xy \in \{\alpha\beta, \alpha\gamma, \beta\gamma\}$ and two k-cliques $t_1, t_2 \in \mathcal{C}_k$, if $t_1 \cup t_2$ is a $2k$-clique, then*

$$Dyck([CNG(t_1) \circ CLG(t_2)]_{xy}) = E_2$$

and $Dyck([CNG(t_1) \circ CLG(t_2)]_{xy}) > E_2$ otherwise.

Proof. We will omit the subscripts xy since they do not matter for the proof. We have that

$$Dyck(CNG(t_1) \circ CLG(t_2)) \geq \sum_{v \in t_1} k \cdot pattern(NG(v), CLG(t_2)).$$

Suppose that for some $v \in t_1$, $NG(v)$ is aligned with more than one gadget $p(LG(u))$. Then the # symbols between these gadgets $p(LG(u))$ will be substituted or deleted. The cost of these operations is $\geq \ell_2 > E_2$. Therefore, we have that any one of the k^2 gadgets $NG(v)$ is aligned with at most one gadget $p(LG(u))$ for some $u \in t_2$. By the construction of CNG and CLG, we have that

$$\begin{aligned} &Dyck(CNG(t_1) \circ CLG(t_2)) \\ &\geq \sum_{v \in t_1} \sum_{u \in t_2} pattern(NG(v), (\#')^{2\ell_2} p(LG(u)) (\#')^{2\ell_2}) \\ &= \sum_{v \in t_1} \sum_{u \in t_2} Dyck(NG(v) \circ p(LG(u))), \end{aligned}$$

where the last equality follows because # does not appear among the symbols of $NG(v)$. Now we have that

$$Dyck(CNG(t_1) \circ CLG(t_2)) \geq \sum_{v \in t_1} \sum_{u \in t_2} E_1 = k^2 \cdot E_1 = E_2,$$

where we use Claim 4.1. The equality above implies that we have equality in all k^2 invocations of Claim 4.1, which implies that $v \in N(u)$ for all $v \in t_1$, $u \in t_2$. This gives that there is a biclique between the vertices of t_1 and t_2. Also, one can verify that equality occurs if there is a biclique. ∎

Let $E_3 = 3(\ell_4 + E_2)$.

CLAIM 4.4. *Suppose that for some triple of k-cliques $t_\alpha, t_\beta, t_\gamma \in \mathcal{C}_k$, the union $t_\alpha \cup t_\beta \cup t_\gamma$ is a $3k$-clique. Then*

$$\begin{aligned} Dyck(\quad & \mathtt{x}_\alpha{}^{\ell_5} CG_\alpha(t_\alpha)(\mathtt{y}_\alpha{}')^{\ell_5} \\ & \circ\ \mathtt{x}_\beta{}^{\ell_5} CG_\beta(t_\beta)(\mathtt{y}_\beta{}')^{\ell_5} \\ & \circ\ \mathtt{x}_\gamma{}^{\ell_5} CG_\gamma(t_\gamma)(\mathtt{y}_\gamma{}')^{\ell_5} \quad) = E_3 \end{aligned}$$

and $> E_3$ otherwise.

Proof. We need to lower bound

$$\begin{array}{lll} Dyck(\ \mathtt{x}_\alpha{}^{\ell_5}\mathtt{a}^{\ell_4}(\mathtt{x}_\alpha{}')^{\ell_5}\mathtt{g}_{\alpha\gamma}^{\ell_3} & CNG_{\alpha\gamma}(t_\alpha)\mathtt{g}_{\alpha\gamma}^{\ell_3}\mathtt{g}_{\alpha\beta}^{\ell_3} & CNG_{\alpha\beta}(t_\alpha)\mathtt{g}_{\alpha\beta}^{\ell_3}\mathtt{y}_\alpha{}^{\ell_5}(\mathtt{a}')^{\ell_4}(\mathtt{y}_\alpha{}')^{\ell_5} \\ \mathtt{x}_\beta{}^{\ell_5}\mathtt{b}^{\ell_4}(\mathtt{x}_\beta{}')^{\ell_5}(\mathtt{g}'_{\alpha\beta})^{\ell_3} & CLG_{\alpha\beta}(t_\beta)(\mathtt{g}'_{\alpha\beta})^{\ell_3}\mathtt{g}_{\beta\gamma}^{\ell_3} & CNG_{\beta\gamma}(t_\beta)\mathtt{g}_{\beta\gamma}^{\ell_3}\mathtt{y}_\beta{}^{\ell_5}(\mathtt{b}')^{\ell_4}(\mathtt{y}_\beta{}')^{\ell_5} \\ \mathtt{x}_\gamma{}^{\ell_5}\mathtt{c}^{\ell_4}(\mathtt{x}_\gamma{}')^{\ell_5}(\mathtt{g}'_{\beta\gamma})^{\ell_3} & CLG_{\beta\gamma}(t_\gamma)(\mathtt{g}'_{\beta\gamma})^{\ell_3}(\mathtt{g}'_{\alpha\gamma})^{\ell_3} & CLG_{\alpha\gamma}(t_\gamma)(\mathtt{g}'_{\alpha\gamma})^{\ell_3}\mathtt{y}_\gamma{}^{\ell_5}(\mathtt{c}')^{\ell_4}(\mathtt{y}_\gamma{}')^{\ell_5}\ ). \end{array}$$

Assume that some symbol $\mathtt{a}$ is aligned with a symbol to the right of $\mathtt{x}_\alpha{}'$. Then sequence $(\mathtt{x}_\alpha{}')^{\ell_5}$ will contribute $\geq \ell_5/2 > E_3$ to the Dyck score and we are done. (We prove later that we can achieve Dyck score E_3 if there is a clique.) Now let r denote the number of symbols from the sequence $j = \mathtt{x}_\alpha{}^{\ell_5}\mathtt{a}^{\ell_4}(\mathtt{x}_\alpha{}')^{\ell_5}$ that are aligned with a symbol that does not belong to sequence j. Let s denote the set of these r symbols. Let l denote the symbols $\mathtt{x}_\alpha$ that are aligned with a symbol $\mathtt{x}_\alpha{}'$. There are $2\ell_5 + \ell_4 - r - 2l$ symbols from j that are not considered yet. These symbols are not matched to their counterparts and, therefore, contribute at least $\lceil (2\ell_5 + \ell_4 - r - 2l)/2 \rceil$ to the Dyck score (we divide by 2 because the symbols can be mismatched among themselves in pairs). We have that $\lceil (2\ell_5 + \ell_4 - r - 2l)/2 \rceil + r \geq \ell_4/2 + \lceil r/2 \rceil$ by the definition of l (hence $l \leq \ell_5$). We note that $\ell_4/2 = Dyck(j)$. $Dyck(j) \leq \ell_4/2$ can be obtained by aligning the symbols $\mathtt{x}_\alpha$ with the symbols $\mathtt{x}_\alpha{}'$ and mismatching the symbols $\mathtt{a}$ in pairs. The reverse inequality follows by observing that all symbols $\mathtt{a}$ will be mismatched. Also, we note that if we mismatch the symbols from s among themselves, this costs $\lceil r/2 \rceil$.

The above observations imply that when we want to bound the Dyck score, we can assume that the symbols in j do not interact with any symbols that are not in j. Similarly, we can argue when

$$j = \mathtt{y}_\alpha{}^{\ell_5}(\mathtt{a}')^{\ell_4}(\mathtt{y}_\alpha{}')^{\ell_5}, \mathtt{x}_\gamma{}^{\ell_5}\mathtt{b}^{\ell_4}(\mathtt{x}_\gamma{}')^{\ell_5}, \mathtt{y}_\beta{}^{T}(\mathtt{b}')^{\ell_4}(\mathtt{y}_\beta{}')^{\ell_5}, \mathtt{x}_\gamma{}^{\ell_5}\mathtt{c}^{\ell_4}(\mathtt{x}_\gamma{}')^{\ell_5}, \mathtt{y}_\gamma{}^{\ell_5}(\mathtt{c}')^{\ell_4}(\mathtt{y}_\gamma{}')^{\ell_5}.$$

Thus, we need to show that

$$\begin{array}{llllll} Dyck(\ \mathtt{g}_{\alpha\gamma}^{\ell_3} & CNG_{\alpha\gamma}(t_\alpha) & \mathtt{g}_{\alpha\gamma}^{\ell_3} & \mathtt{g}_{\alpha\beta}^{\ell_3} & CNG_{\alpha\beta}(t_\alpha) & \mathtt{g}_{\alpha\beta}^{\ell_3} \\ (\mathtt{g}'_{\alpha\beta})^{\ell_3} & CLG_{\alpha\beta}(t_\beta) & (\mathtt{g}'_{\alpha\beta})^{\ell_3} & \mathtt{g}_{\beta\gamma}^{\ell_3} & CNG_{\beta\gamma}(t_\beta) & \mathtt{g}_{\beta\gamma}^{\ell_3} \\ (\mathtt{g}'_{\beta\gamma})^{\ell_3} & CLG_{\beta\gamma}(t_\gamma) & (\mathtt{g}'_{\beta\gamma})^{\ell_3} & (\mathtt{g}'_{\alpha\gamma})^{\ell_3} & CLG_{\alpha\gamma}(t_\gamma) & (\mathtt{g}'_{\alpha\gamma})^{\ell_3} \quad) \geq 3E_2 \end{array}$$

with equality if and only if there is a biclique between the vertices of t_α and t_β, between the vertices of t_α and t_γ, and between the vertices of t_β and t_γ. Let h be the argument to the Dyck function. We want to show that $Dyck(h) \geq 3E_2$ with the stated condition for the equality.

Consider three gadgets CNG and three gadgets CLG as above. We can assume that no symbol of any of these six gadgets is aligned with any symbol g_{xy} or g'_{xy}. Assume that it is not the case. Then we can delete all symbols from the gadgets that are aligned with symbols g_{xy} or g'_{xy}. After this, we rematch g_{xy} or g'_{xy} among themselves. We can check that we can always make this rematching of symbols g_{xy} or g'_{xy} so that the cost does not increase. Furthermore, if some CNG_{xy} gadget is aligned with a $CNG_{x'y'}$ (or $CLG_{x'y'}$) for $(x,y) \neq (x',y')$, then there are two substrings of type g_{ab} or g'_{ab} that do not have their counterpart between CNG_{xy} and $CNG_{x'y'}$ (or $CLG_{x'y'}$). Hence, their contribution to the Dyck score is at least $2\ell_3/2 = \ell_3 \gg 3E_2$. Thus, for all (x,y) the only gadget that CNG_{xy} can be aligned with is CLG_{xy}, and vice versa. This means that we can assume that all the g and g' symbols are completely aligned.

We have shown that the Dyck cost of the string is exactly

$$\begin{aligned} &Dyck(CNG_{\alpha\gamma}(t_\alpha) \circ CLG_{\alpha\gamma}(t_\alpha)) \\ +\,&Dyck(CNG_{\alpha\beta}(t_\beta) \circ CLG_{\alpha\beta}(t_\beta)) \\ +\,&Dyck(CNG_{\beta\gamma}(t_\gamma) \circ CLG_{\beta\gamma}(t_\gamma)). \end{aligned}$$

We want to show that this is $\geq 3E_2$ with equality if and only if $t_\alpha \cup t_\beta \cup t_\gamma$ is a $3k$-clique. This was shown in Claim 4.3. ∎

We now turn to the behavior of our "selection gadgets." Let E_C be a fixed integer to be defined later that depends on n, N, and k.

CLAIM 4.5. *If the G contains a $3k$-clique, then $Dyck(S_G) = E_C$ and $Dyck(S_G) > E_C$ otherwise.*

The proof of this claim will require several claims and lemmas. We start with some lemmas about general properties of Dyck edit distance.

LEMMA 4.6. *Let Z_1 be a substring of sequence Z. Assume that Z_1 is of even length. If all symbols from Z_1 participate in mismatches and deletions only, then we can modify the alignment so that there is no symbol in Z_1 that is aligned with a symbol not in Z_1.*

Proof. Let l denote the number of symbols from Z_1 that are aligned with symbols outside of Z_1. Let s denote the set of all these symbols outside of Z_1 that are aligned with Z_1. There are two cases to consider:

- l is even. Then the *Dyck* cost induced by symbols in s and in Z_1 is at least $S_1 := l + (|Z_1| - l)/2$ by the properties from the statement of the lemma. We modify the alignment as follows. We align all symbols in s among themselves in pairs. This induces cost $S_2 := l/2$. We align all symbols in Z_1 among themselves. This induces cost $S_3 = |Z_1|/2$. The total induced cost after the modification is $S_2 + S_3 \leq S_1$, and we satisfy the requirement in the lemma.
- l is odd. Then the *Dyck* cost induced by symbols in s and in Z_1 is at least $S_1 := l + (|Z_1| - l + 1)/2$. We modify the alignment as follows. We align all symbols in s among themselves in pairs, except one symbol, which we delete (remember that l is odd). This induces cost $S_2 := (l+1)/2$. We align all symbols in Z_1 among themselves. This induces cost $S_3 = |Z_1|/2$. The

total induced cost after the modification is $S_2 + S_3 \leq S_1$, and we satisfy the requirement in the lemma. ∎

Consider an optimal alignment of some string $w \in (\Sigma \cup \Sigma')^*$.

LEMMA 4.7. *Let Z be a maximal substring of w consisting entirely of z symbols (for some symbol z appearing in w). Let Z_0 be a maximal substring of w consisting entirely of z_0 symbols. Let $|Z| = |Z_0|$, and let there be matches between Z and Z_0. Also, there are no other maximal substrings of w containing z other than Z. Then we can increase the Dyck score by at most 2 by modifying the alignment and get that the symbols of Z_0, which are matched to z, form a substring of Z_0 and that the substring is a suffix or prefix of Z_0 (we can choose whether it is a suffix of a prefix). We can also assume that the rest of the symbols in Z_0 are deleted or mismatched among themselves.*

Proof. Without loss of generality, we will show that we can make the substring be the suffix. We write $Z_0 = Z_1 Z_2 Z_3$ so that the first symbol of Z_2 is the first symbol of Z_0 that is aligned to some symbol z and the last symbol of Z_2 is the last symbol of Z_0 that is aligned with some symbol z. First, we modify the alignment as follows. If the length of Z_1 is even, we do not do anything. Otherwise, consider the first symbol of Z_1. If it is aligned with some symbol, delete the first symbol of Z_1 and the symbol aligned with it by increasing the *Dyck* cost by 1. Similarly, delete the last symbol of Z_3 and the symbol aligned with it if Z_3 is of odd length. Now, by Lemma 4.6, we can assume that all symbols in Z_1 and Z_3 are mismatched among themselves (except, possibly, the first symbol of Z_1 and the last symbol of Z_3). Now we are at the state when all symbols from Z_0 are mismatched among themselves, except a few that are matched with symbols z from Z. Now we can rematch these symbols from Z with symbols z_0 from Z_0 so that the z_0 come from the suffix of Z_0. We see that this does not increase the *Dyck* score, except for two possible deletions. ∎

Now we prove some claims about the properties of the optimal alignment of S_G. These claims essentially show that any "bad behavior," in which we do not align exactly one clique gadget of each type, is suboptimal.

CLAIM 4.8. *For any gadget $CG_\beta(t)$ from the sequence, if none of the $\mathtt{x}'_\beta, \mathtt{y}_\beta$ symbols is matched with its counterparts, then all the $\mathtt{b}$ symbols from $CG_\beta(t)$ must be matched to their counterparts $\mathtt{b}'$ from $CG_\beta(t)$. When this happens, the gadgets contribute E_l to the Dyck score, where E_l is some value that depends only on n and k. We specify this in the proof. Analogous claims hold for α, γ.*

Proof. By Lemma 4.6, we can assume that all the symbols $\mathtt{x}'_\beta$ are mismatched among themselves. We can say the same about the symbols $\mathtt{y}_\beta$. Let Z be the substring of the gadget between symbols $\mathtt{x}'_\beta$ and $\mathtt{y}_\beta$. If Z has matches with some symbols, then, by the construction of w, no symbol $\mathtt{b}$ or $\mathtt{b}'$ is matched with its counterpart and by Lemma 4.6 we get that $\mathtt{b}$ and $\mathtt{b}'$ are mismatched among themselves. Now we can decrease the *Dyck* score by deleting all symbols from Z and all symbols that Z is aligned with outside the gadget. This increases the *Dyck* score, but then we can decrease it substantially by matching all symbols $\mathtt{b}$ with their counterparts $\mathtt{b}'$ from the gadget. In the end, we get a smaller *Dyck* score because $\ell_4 \geq 100|Z|$. Now it remains to consider the case when the symbols in Z do not participate in matches. Then, by Lemma 4.6, we again conclude that the symbols in Z participate only in mismatches and only among themselves.

Let s be the union of all symbols $\mathtt{b}$ and $\mathtt{b}'$ of the gadget and all symbols that these symbols are aligned with outside the gadget. Let l denote the number of symbols from

s that are not from the gadget. Consider three cases:

- $l = 0$. We satisfy the requirements of the claim by matching all symbols $\mathtt{b}$ with their counterparts $\mathtt{b}'$.
- There is no symbol among the $l \geq 1$ symbols that participates in a match. We have that all symbols in s contribute at least l to the *Dyck* score. We modify the alignment as follows. We match all symbols $\mathtt{b}$ with their counterparts $\mathtt{b}'$. We match the rest of the l symbols from s among themselves. If there is an odd number of them, we delete one. This contributes at most $S_1 := (l+1)/2$ to the Dyck score after the modification. We have that $S_1 \leq l$ for $l \geq 1$.
- The complement of the previous two cases: there is a symbol among the $l \geq 1$ symbols that participates in a match. Without loss of generality, the $\mathtt{b}$ symbols from the gadget participate in at least one matching. Then none of the symbols $\mathtt{b}'$ from the gadget participates in any matching and by Lemma 4.6 we have that all symbols $\mathtt{b}'$ from the gadget are mismatched among themselves. Therefore, we can assume that $l \leq \ell_4$. We need to consider two subcases:
  - l is even. The symbols in s contribute at least $S_1 = (2\ell_4 - l)/2$ to the Dyck score. We modify the algorithm as follows. We match all symbols $\mathtt{b}$ with their counterparts $\mathtt{b}'$. We mismatch l symbols in pairs among themselves. After the modification, the *Dyck* contribution of symbols from s is $S_2 := l/2$. We see that $S_2 \leq S_1$.
  - l is odd. The symbols in s contribute at least $S_1 = (2\ell_4 - l + 1)/2$ to the Dyck score (at least one symbol is deleted because $2\ell_4 - l$ is odd). We modify the algorithm as follows. We match all symbols $\mathtt{b}$ with their counterparts $\mathtt{b}'$. We mismatch l symbols in pairs among themselves, except that we delete one symbol. After the modification, the *Dyck* contribution of symbols from s is $S_2 := (l+1)/2$. We see that $S_2 \leq S_1$.

Because the symbols in between $\mathtt{b}$ and $\mathtt{b}'$ do not have their counterparts among themselves, the gadget contributes $E_l := (|CG_\beta(t)| - 2\ell_4)/2$ to the *Dyck* cost. This quantity only depends on n and k, and this can be verified from the construction of S_G. ∎

CLAIM 4.9. *In all optimal alignments, there is some symbol* $\mathtt{x}_\beta$ *that is aligned with its counterpart* $\mathtt{x}'_\beta$. *Analogous statements can be proved about symbols* $\mathtt{y}_\beta$, $\mathtt{y}'_\beta$, $\mathtt{x}_\alpha$, $\mathtt{x}'_\alpha$, $\mathtt{y}_\alpha$, $\mathtt{y}'_\alpha$, $\mathtt{x}_\gamma$, $\mathtt{x}'_\gamma$, $\mathtt{y}_\gamma$, $\mathtt{y}'_\gamma$.

Proof. Suppose that there is no pair of symbols $\mathtt{x}_\beta$ and $\mathtt{x}'_\beta$ that are aligned. We will modify the alignment so that all symbols $\mathtt{x}_\beta$ are aligned with $\mathtt{x}'_\beta$ coming from the first substring f of S_G consisting entirely of $\mathtt{x}'_\beta$. This will decrease the *Dyck* cost. From the statement we have that all symbols $\mathtt{x}_\beta$ and all symbols from f are mismatched or deleted. Therefore, by Lemma 4.6, we get that all symbols $\mathtt{x}_\beta$ are mismatched among themselves and all symbols in f are mismatched among themselves. Now we modify the alignment as follows to achieve our goal. We delete all symbols b between $\mathtt{x}_\beta$ and f. Also, we delete all the symbols that the deleted b's were aligned with before the deletion. This increases the *Dyck* cost by at most $2\ell_4$. Then we match all symbols $\mathtt{x}_\beta$ to $\mathtt{x}'_\beta$ in pairs. This decreases the *Dyck* cost by ℓ_5. As a result, we decreased the *Dyck* cost because $\ell_5 \gg 2\ell_4$. ∎

CLAIM 4.10. *In all optimal alignments there is a symbol* $\mathtt{x}'_\beta$ *that is matched to a symbol* $\mathtt{x}_\beta$ *and there is a symbol* $\mathtt{y}_\beta$ *that is matched to a symbol* $\mathtt{y}'_\beta$ *so that the symbols* $\mathtt{x}'_\beta$ *and* $\mathtt{y}_\beta$ *come from the same gadget* $CG_\beta(t)$. *Analogous statements can be proved about symbols* $\mathtt{x}_\alpha$, $\mathtt{x}'_\alpha$, $\mathtt{y}_\alpha$, $\mathtt{y}'_\alpha$, $\mathtt{x}_\gamma$, $\mathtt{x}'_\gamma$, $\mathtt{y}_\gamma$, $\mathtt{y}'_\gamma$.

Proof. By Claim 4.9, $\mathtt{x}_\beta$ is aligned with some sequence consisting of $\mathtt{x}'_\beta$. Suppose that the sequence comes from gadget $CG_\beta(t_1)$ for some t_1. Also, by Claim 4.9, $\mathtt{y}'_\beta$ is aligned with some sequence consisting of $\mathtt{y}_\beta$. Suppose that the sequence comes from gadget $CG_\beta(t_2)$ for some t_2. We want to prove that $t_1 = t_2$. Suppose that this is not the case and $CG_\beta(t_1)$ comes to the left of $CG_\beta(t_2)$ (the order cannot be the reverse because of the construction of S_G and because the alignments cannot cross). Suppose that there is some other gadget $CG_\beta(t_3)$ in between $CG_\beta(t_1)$ and $CG_\beta(t_2)$. Then we can verify that $CG_\beta(t_3)$ satisfy the conditions of Claim 4.8 and we can assume that all symbols in $CG_\beta(t_3)$ are aligned with symbols in $CG_\beta(t_3)$. Therefore, we can temporarily remove the gadget $CG_\beta(t_3)$ from S_G. We can do that because it does not interact with symbols outside of it. We do this just for the sake of the proof, and we restore this gadget at the end of the following rematching of symbols. We do that until $CG_\beta(t_1)$ is to the left of $CG_\beta(t_2)$ and they are neighboring. Now we will change the alignment so that $\mathtt{x}_\beta$ is aligned with a symbol $\mathtt{x}'_\beta$ from $CG_\beta(t_2)$ and as a result we will decrease *Dyck* cost.

We can verify from the construction of S_G that the symbols $\mathtt{b}$, $\mathtt{y}_\beta$, $\mathtt{b}'$ from $CG_\beta(t_1)$ do not participate in matches. Also, the symbols $\mathtt{b}$ and $\mathtt{x}'_\beta$ from $CG_\beta(t_2)$ do not participate in matches. By Lemma 4.6, we conclude that all these symbols have mismatches among themselves. Let g be the sequence between symbols $\mathtt{x}'_\beta$ and $\mathtt{y}_\beta$ in $CG_\beta(t_1)$. Now we modify the alignment so that the symbols in the sequence g have mismatches only among themselves and the symbols that were aligned with symbols in g are deleted. This increases the *Dyck* cost by at most $10|g| \leq 100\ell_3 =: S_1$. Some symbols $\mathtt{x}'_\beta$ are aligned with symbols outside $CG_\beta(t_1)$. We transfer these alignments of symbols $\mathtt{x}'_\beta$ from $CG_\beta(t_1)$ to symbols $\mathtt{x}'_\beta$ in $CG_\beta(t_2)$ so that we only have mismatches among $\mathtt{x}'_\beta$ in $CG_\beta(t_1)$ and we do not change the *Dyck* cost. Now we can align all symbols $\mathtt{b}$ in $CG_\beta(t_1)$ with $\mathtt{b}'$ in $CG_\beta(t_1)$ in pairs. This decreases the *Dyck* cost by ℓ_4. In the end, we decreased the *Dyck* cost by $\ell_4 - S_1 > 0$ and we proved what we wanted. Finally, we observe that after the above rematching of symbols (so that $t_1 = t_2$), we can reintroduce the removed gadgets $CG_\beta(t_3)$ in their original place. ∎

Claim 4.11. *In all optimal alignments, the only symbols $\mathtt{x}'_\beta$ to which the symbols $\mathtt{x}_\beta$ are matched come from the same gadget $CG_\beta(t)$, and the only symbols $\mathtt{y}_\beta$ to which the symbols $\mathtt{y}'_\beta$ are matched come from the same gadget $CG_\beta(t)$. In both cases, this is the same gadget $CG_\beta(t)$. Analogous statements can be proved about symbols $\mathtt{x}_\alpha$, $\mathtt{x}'_\alpha$, $\mathtt{y}_\alpha$, $\mathtt{y}'_\alpha$, $\mathtt{x}_\gamma$, $\mathtt{x}'_\gamma$, $\mathtt{y}_\gamma$, $\mathtt{y}'_\gamma$.*

Proof. Suppose that $\mathtt{x}_\beta$ is matched to symbols $\mathtt{x}'_\beta$ coming from two different gadgets $CG_\beta(t_1)$ and $CG_\beta(t_2)$. $CG_\beta(t_1)$ comes earlier in S_G than $CG_\beta(t_2)$. Assume that there is no gadget $CG_\beta(t_3)$ in between $CG_\beta(t_1)$ and $CG_\beta(t_2)$ in the sequence S_G. We can make this assumption because otherwise we can remove $CG_\beta(t_3)$ as in Claim 4.8. We can check that none of the symbols x'_β and y_β from $CG_\beta(t_3)$ participates in matches, and thus we satisfy the requirements of Claim 4.8. Now we can check that the symbols $\mathtt{b}$, $\mathtt{y}_\beta$, $\mathtt{b}'$ in $CG_\beta(t_1)$ do not participate in matches. Also, the symbols $\mathtt{b}'$ and those between $\mathtt{x}'_\beta$ and $\mathtt{y}_\beta$ in $CG_\beta(t_1)$ do not participate in matches. Therefore, by Lemma 4.6 we conclude that all these symbols participate in mismatches only among themselves.

Let X_1 denote the sequence of $\mathtt{x}'_\beta$ from $CG_\beta(t_1)$ and X_2 denote the sequence of $\mathtt{x}'_\beta$ from $CG_\beta(t_2)$. By Lemma 4.7, we can assume that the symbols $\mathtt{x}'_\beta$ from X_1 and X_2 that are matched to $\mathtt{x}_\beta$ form the suffix in both sequences and the rest of the symbols in both sequences are mismatched among themselves or deleted. The corresponding modification increases the *Dyck* cost by at most $\leq 4 =: S_1$. Let Z_1 be the suffix of

X_1 and Z_2 be the suffix of X_2. $|Z_1|+|Z_2|$ is less than or equal to the total number of symbols $\mathtt{x}_\beta$ in S_G by the construction of S_G. Suppose that $|Z_1|$ is even. We mismatch all symbols in Z_1 among themselves and match the resulting unmatched $|Z_1|$ symbols $\mathtt{x}_\beta$ to $\mathtt{x}'_\beta$ from X_2. This does not change the *Dyck* cost. Suppose, on the other hand, that $|Z_1|$ is odd. Then there is a deletion among the symbols in X_1 that are not in Z_1. We do mismatches among the symbols Z_1 and the one deleted. We match the resulting unmatched $|Z_1|$ symbols $\mathtt{x}_\beta$ to $\mathtt{x}'_\beta$ from X_2. We can check that we can do this matching so that the *Dyck* cost does not increase. Now we can match all symbols $\mathtt{b}$ from $CG_\beta(t_1)$ to their counterparts $\mathtt{b}'$ in $CG_\beta(t_1)$. This decreases the *Dyck* cost by $S_2 := \ell_4$. In total, we decrease the *Dyck* cost by $\geq S_2 - S_1 > 0$. ∎

Let N be the number of k-cliques in the graph. In the proof of Claim 4.11, we remove $3N-3$ cliques from the graph, each removal costing E_l. After all the removals, we arrive at a sequence of the form required in Claim 4.4. Thus, we set $E_C := (3N-3)E_l + E_c$ and our proof for Claim 4.5 is finished.

We are now ready to show that the construction of S_G from the graph G proves Theorem 1.3.

Reminder of Theorem 1.3. *If Dyck Edit Distance on a sequence of length n can be solved in $T(n)$ time, then $3k$-Clique on n node graphs can be solved in $O\left(T\left(n^{k+O(1)}\right)\right)$ time for any $k \geq 1$. Moreover, the reduction is combinatorial.*

Proof. Given a graph G on n nodes, we construct the sequence S_G as described above. The sequence can be constructed in $O(k^{O(1)} \cdot n^{k+O(1)})$ time by enumerating all subsets of k nodes. The sequence has length $O(k^{O(1)} \cdot n^{k+O(1)})$. Thus, an algorithm for Dyck Edit Distance as in the statement returns a *Dyck* score of S_G in $T(n^{k/3+O(1)})$ time (treating k as a constant) and by Claim 4.5 this score determines whether G contains a $3k$-clique. All the steps in our reduction are combinatorial. ∎

Acknowledgments. We would like to thank Piotr Indyk for a discussion that led to this work, and Roy Frostig for introducing us to many modern works on CFG parsing. We also thank the anonymous reviewers for providing helpful comments.

REFERENCES

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

[2] A. Abboud, K. Lewi, and R. Williams, *Losing weight by gaining edges*, in Algorithms—ESA 2014, Springer, Heidelberg, 2014, pp. 1–12.

[3] A. V. Aho and T. G. Peterson, *A minimum distance error-correcting parser for context-free languages*, SIAM J. Comput., 1 (1972), pp. 305–312, https://doi.org/10.1137/0201022.

[4] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison–Wesley Longman Publishing, Boston, MA, 1986.

[5] T. Akutsu, *Approximation and exact algorithms for RNA secondary structure prediction and recognition of stochastic context-free languages*, in Algorithms and Computation, Springer, Berlin, 1998, pp. 337–346.

[6] N. Alon, A. Andoni, T. Kaufman, K. Matulef, R. Rubinfeld, and N. Xie, *Testing k-wise and almost k-wise independence*, in Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, CA, 2007, pp. 496–505.

[7] N. Alon and R. B. Boppana, *The monotone circuit complexity of boolean functions*, Combinatorica, 7 (1987), pp. 1–22.

[8] N. Alon, M. Krivelevich, and B. Sudakov, *Finding a large hidden clique in a random graph*, Random Structures Algorithms, 13 (1998), pp. 457–466.

[9] A. Andoni, *Question on RNA Folding*, http://sublinear.info/index.php?title=Open_Problems:61, 2014 (accessed March 31, 2015).

[10] A. Andoni, R. Krauthgamer, and K. Onak, *Polylogarithmic approximation for edit distance and the asymmetric query complexity*, in Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, 2010, pp. 377–386.
[11] M. Authors, *Queries and problems*, SIGACT News, 16 (1984), pp. 38–47.
[12] R. Backofen, D. Tsur, S. Zakov, and M. Ziv-Ukelson, *Sparse RNA folding: Time and space efficient algorithms*, J. Discrete Algorithms, 9 (2011), pp. 12–31.
[13] A. Backurs and P. Indyk, *Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)*, in Proceedings of the 47th Annual Symposium on the Theory of Computing, 2015, pp. 51–58.
[14] J. K. Baker, *Trainable grammars for speech recognition*, J. Acoust. Soc. Amer., 65 (1979), pp. S132–S132.
[15] J.-M. Benedí and J.-A. Sánchez, *Fast stochastic context-free parsing: A stochastic version of the Valiant algorithm*, in Pattern Recognition and Image Analysis, Springer, Berlin, Heidelberg, 2007, pp. 80–88.
[16] K. Bringmann, F. Grandoni, B. Saha, and V. V. Williams, *Truly sub-cubic algorithms for language edit distance and RNA-folding via fast bounded-difference min-plus product*, in Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science, 2016, pp. 375–384.
[17] K. Bringmann and M. Künnemann, *Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping*, preprint, https://arxiv.org/abs/1502.01063, 2015.
[18] K. Bringmann and P. Wellnitz, *Clique-Based Lower Bounds for Parsing Tree-Adjoining Grammars*, LIPIcs. Leibniz Int. Proc. Inform. 78, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Wadern, Germany, 2017.
[19] Y. Chang, *Hardness of RNA folding problem with four symbols*, in Proceedings of the 27th Annual Symposium on Combinatorial Pattern Matching, Tel Aviv, Israel, pp. 13:1–13:12.
[20] J. Chen, B. Chor, M. Fellows, X. Huang, D. W. Juedes, I. A. Kanj, and G. Xia, *Tight lower bounds for certain parameterized NP-hard problems*, Inform. and Comput., 201 (2005), pp. 216–231.
[21] J. Chen, X. Huang, I. A. Kanj, and G. Xia, *Strong computational lower bounds via parameterized complexity*, J. Comput. System Sci., 72 (2006), pp. 1346–1367.
[22] N. Chomsky, *On certain formal properties of grammars*, Information and Control, 2 (1959), pp. 137–167.
[23] J. Cocke and J. T. Schwartz, *Programming Languages and Their Compilers: Preliminary Notes*, Tech. Report, 2nd revised ed., Courant Institute of Mathematical Sciences, New York University, New York, NY, 1970.
[24] S. B. Cohen, G. Satta, and M. Collins, *Approximate PCFG parsing using tensor decomposition*, in Proceedings of HLT-NAACL, 2013, pp. 487–496.
[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., The MIT Press, Cambridge, MA, 2009.
[26] F. L. DeRemer, *Practical Translators for LR (k) Languages*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1969.
[27] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis*, Cambridge University Press, Cambridge, UK, 1998.
[28] J. Earley, *An efficient context-free parsing algorithm*, Comm. ACM, 13 (1970), pp. 94–102.
[29] F. Eisenbrand and F. Grandoni, *On the complexity of fixed parameter clique and dominating set*, Theoret. Comput. Sci., 326 (2004), pp. 57–67.
[30] C. N. Fischer and J. Mauney, *On the role of error productions in syntactic error correction*, Comput. Lang., 5 (1980), pp. 131–139.
[31] Y. Frid and D. Gusfield, *A simple, practical and complete $O(\frac{n^3}{\log n})$-time algorithm for RNA folding using the four-Russians speedup*, in Algorithms in Bioinformatics, Springer, Berlin, Heidelberg, 2009, pp. 97–107.
[32] F. L. Gall, *Powers of tensors and fast matrix multiplication*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, Kobe, Japan, 2014, pp. 296–303.
[33] H. Gallaire, *Recognition time of context-free languages by on-line Turing machines*, Information and Control, 15 (1969), pp. 288–295.
[34] S. Graham, W. L. Ruzzo, and M. Harrison, *An improved context-free recognizer*, ACM Trans. Program. Lang. Syst., 2 (1980), pp. 415–462.
[35] R. R. Gutell, J. Cannone, Z. Shang, Y. Du, and M. Serra, *A story: Unpaired adenosine bases in ribosomal RNAs*, J. Mol. Biol., 304 (2000), pp. 335–354.
[36] J. Hartmanis and R. E. Stearns, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc., 117 (1965), pp. 285–306.

[37] J. Hastad, *Clique is hard to approximate within $n^{1-\varepsilon}$*, Acta Math., 182 (1999), pp. 105–142.
[38] I. M. Havel and M. A. Harrison, *On the parsing of deterministic languages*, J. Assoc. Comput. Mach., 21 (1974), pp. 525–548.
[39] E. Hazan and R. Krauthgamer, *How hard is it to approximate the best Nash equilibrium?*, SIAM J. Comput., 40 (2011), pp. 79–91, https://doi.org/10.1137/090766991.
[40] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed., Addison–Wesley Longman Publishing, Boston, MA, 2006.
[41] M. Jerrum, *Large cliques elude the Metropolis process*, Random Structures Algorithms, 3 (1992), pp. 347–360.
[42] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice–Hall, Upper Saddle River, NJ, 2000.
[43] R. M. Karp, *Reducibility among combinatorial problems*, in Proceedings of the Symposium on Complexity of Computer Computations, Yorktown Heights, NY, 1972, pp. 85–103.
[44] T. Kasami, *An Efficient Recognition and Syntax Algorithm for Context-Free Languages*, Tech. Report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA, 1965.
[45] D. E. Knuth, *On the translation of languages from left to right*, Information and Control, 8 (1965), pp. 607–639.
[46] F. Korn, B. Saha, D. Srivastava, and S. Ying, *On repairing structural problems in semi-structured data*, Proc. VLDB Endow., 6 (2013), pp. 601–612.
[47] B. Lang, *Deterministic techniques for efficient non-deterministic parsers*, in Automata, languages and programming, Springer, Berlin, 1974, pp. 255–269.
[48] L. Lee, *Fast context-free grammar parsing requires fast boolean matrix multiplication*, J. ACM, 49 (2002), pp. 1–15.
[49] G. Myers, *Approximately matching context-free languages*, Inform. Process. Lett., 54 (1995), pp. 85–92.
[50] J. Nešetřil and S. Poljak, *On the complexity of the subgraph problem*, Comment. Math. Univ. Carolin., 26 (1985), pp. 415–419.
[51] A. Pauls and D. Klein, *K-best a* parsing*, in Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, Vol. 2, Association for Computational Linguistics, 2009, pp. 958–966.
[52] T. Pinhas, D. Tsur, S. Zakov, and M. Ziv-Ukelson, *Edit distance with duplications and contractions revisited*, in Combinatorial Pattern Matching, Springer, Heidelberg, 2011, pp. 441–454.
[53] S. Rajasekaran and M. Nicolae, *An error correcting parser for context free grammars that takes less than cubic time*, in Language and Automata Theory and Applications, Springer, Cham, 2016, pp. 533–546.
[54] A. M. Rush, D. Sontag, M. Collins, and T. Jaakkola, *On dual decomposition and linear programming relaxations for natural language processing*, in Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2010, pp. 1–11.
[55] W. Ruzzo, *On the complexity of general context-free language parsing and recognition (extended abstract)*, in Proceedings of the 6th Colloquium on Automata, Languages and Programming, 1979, pp. 489–497.
[56] W. Rytter, *Fast recognition of pushdown automaton and context-free languages*, Inform. and Control, 67 (1985), pp. 12–22.
[57] W. Rytter, *Context-free recognition via shortest paths computation: A version of Valiant's algorithm*, Theoret. Comput. Sci., 143 (1995), pp. 343–352.
[58] B. Saha, *The Dyck language edit distance problem in near-linear time*, in Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science, Philadelphia, PA, 2014, pp. 611–620.
[59] B. Saha, *Faster language Edit Distance, Connection to All-Pairs Shortest Paths and Related Problems*, CoRR, abs/1411.7315, 2014.
[60] B. Saha, *Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems*, in Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science, 2015, pp. 118–135.
[61] G. Satta, *Tree-adjoining grammar parsing and boolean matrix multiplication*, Comput. Linguist., 20 (1994), pp. 173–191.
[62] J. I. Seiferas, *A simplified lower bound for context-free-language recognition*, Inform. and Control, 69 (1986), pp. 255–260.

[63] M. Sipser, *Introduction to the Theory of Computation*, Vol. 2, Thomson Course Technology, Boston, MA, 2006.
[64] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng, *Parsing with compositional vector grammars*, in Proceedings of the ACL Conference, Citeseer, 2013.
[65] Y. Song, *Time and Space Efficient Algorithms for RNA Folding with the Four-Russians Technique*, preprint, https://arxiv.org/abs/1503.05670, 2015.
[66] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356, https://doi.org/10.1007/BF02165411.
[67] L. G. Valiant, *General context-free recognition in less than cubic time*, J. Comput. System Sci., 10 (1975), pp. 308–315.
[68] V. Vassilevska, *Efficient algorithms for clique problems*, Inform. Process. Lett., 109 (2009), pp. 254–257.
[69] B. Venkatachalam, D. Gusfield, and Y. Frid, *Faster algorithms for RNA-folding using the four-Russians method*, in Algorithms in Bioinformatics, Springer, Berlin, Heidelberg, 2013, pp. 126–140.
[70] R. Weicker, *General Context-Free Language Recognition by a RAM with Uniform Cost Criterion in Time $n^2 \log n$*, Tech. Report 182, Pennsylvania State University, State College, PA, 1976.
[71] R. Williams, *A new algorithm for optimal 2-constraint satisfaction and its implications*, Theoret. Comput. Sci., 348 (2005), pp. 357–365.
[72] R. Williams, *Faster all-pairs shortest paths via circuit complexity*, in Proceedings of the 46th Annual ACM Symposium on Theory of Computing, 2014, pp. 664–673.
[73] V. V. Williams, *Multiplying matrices faster than Coppersmith-Winograd*, in Proceedings of the 44th Annual Symposium on Theory of Computing, 2012, pp. 887–898.
[74] V. V. Williams and R. Williams, *Subcubic equivalences between path, matrix and triangle problems*, in Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, 2010, pp. 645–654.
[75] G. J. Woeginger, *Space and time complexity of exact algorithms: Some open problems*, in Parameterized and Exact Computation, Lecture Notes in Comput. Sci. 3162, Springer, Berlin, Heidelberg, 2004, pp. 281–290.
[76] G. J. Woeginger, *Open problems around exact algorithms*, Discrete Appl. Math., 156 (2008), pp. 397–405.
[77] D. H. Younger, *Recognition and parsing of context-free languages in time n^3*, Information and Control, 10 (1967), pp. 189–208.
[78] H. Yu, *An improved combinatorial algorithm for boolean matrix multiplication*, in Automata, Languages, and Programming, Springer, Berlin, Heidelberg, 2015, pp. 1094–1105.
[79] S. Zakov, D. Tsur, and M. Ziv-Ukelson, *Reducing the worst case running times of a family of RNA and CFG problems, using Valiant's approach*, in Algorithms in Bioinformatics, Springer, Berlin, Heidelberg, 2010, pp. 65–77.