The Fine-Grained Complexity of CFL Reachability

PARASCHOS KOUTRIS, University of Wisconsin-Madison, USA
SHALEEN DEEP, Microsoft Gray Systems Lab, USA

Many problems in static program analysis can be modeled as the context-free language (CFL) reachability problem on directed labeled graphs. The CFL reachability problem can be generally solved in time $O(n^3)$, where n is the number of vertices in the graph, with some specific cases that can be solved faster. In this work, we ask the following question: given a specific CFL, what is the exact exponent in the monomial of the running time? In other words, for which cases do we have linear, quadratic or cubic algorithms, and are there problems with intermediate runtimes? This question is inspired by recent efforts to classify classic problems in terms of their exact polynomial complexity, known as *ne-grained complexity*. Although recent efforts have shown some conditional lower bounds (mostly for the class of combinatorial algorithms), a general picture of the fine-grained complexity landscape for CFL reachability is missing.

Our main contribution is lower bound results that pinpoint the exact running time of several classes of CFLs or specific CFLs under widely believed lower bound conjectures (e.g., Boolean Matrix Multiplication, k-Clique, APSP, 3SUM). We particularly focus on the family of Dyck-k languages (which are strings with well-matched parentheses), a fundamental class of CFL reachability problems. Remarkably, we are able to show a $\Omega(n^{2.5})$ lower bound for Dyck-2 reachability, which to the best of our knowledge is the first super-quadratic lower bound that applies to all algorithms, and shows that CFL reachability is strictly harder that Boolean Matrix Multiplication. We also present new lower bounds for the case of sparse input graphs where the number of edges m is the input parameter, a common setting in the database literature. For this setting, we show a cubic lower bound for Andersen's Pointer Analysis which significantly strengthens prior known results.

CCS Concepts: • **Software and its engineering** → **Software verification and validation**; • **Theory of computation** → *Theory and algorithms for application domains*; *Program analysis*.

Additional Key Words and Phrases: fine-grained complexity, Dyck reachability, static pointer analysis, Datalog, sparse graphs

ACM Reference Format:
Paraschos Koutris and Shaleen Deep. 2023. The Fine-Grained Complexity of CFL Reachability. *Proc. ACM Program. Lang.* 7, POPL, Article 59 (January 2023), 27 pages. https://doi.org/10.1145/3571252

1 INTRODUCTION

Static analysis is the problem of approximating the run-time behaviors that a program may exhibit. It is of paramount importance in detecting bugs [Bessey et al. 2010; Olivo et al. 2015], detecting security violations and malware [Christodorescu and Jha 2003; Livshits and Lam 2005], and enabling compiler transformations and optimizations. Techniques for static analysis do not run programs on specific inputs, but instead analyze the program behavior by considering all possible inputs and executions. Since for most programs it is impossible to go through all possible executions, it is common to use instead various approximation methods.

Authors' addresses: Paraschos Koutris, Department of Computer Sciences, University of Wisconsin-Madison, USA, paris@cs.wisc.edu; Shaleen Deep, Microsoft Gray Systems Lab, USA, shaleen.deep@microsoft.com.

2475-1421/2023/1-ART59
https://doi.org/10.1145/3571252

A standard way to express many static analysis problems is via a generalization of graph reachability called language reachability. In this setting, a directed graph $G = (V, E)$ with labelled edges from a fixed alphabet is constructed from the program code. Then, given a language $\mathcal{L}$ over the same alphabet, we seek to find pairs of nodes s, t for which there is a directed path from s to t in G such that the word formed by concatenating the labels along the path belongs to $\mathcal{L}$. An important case of language reachability, which is going to be the main subject in this work, is *CFL reachability*, where $\mathcal{L}$ is a context-free language [Reps 1998; Yannakakis 1990]. The CFL reachability problem has applications to a wide range of static analysis problems, including interprocedural data-flow analysis [Reps et al. 1995], shape analysis [Reps 1995], type-based flow analysis [Rehof and Fähndrich 2001], and points-to analysis [Shang et al. 2012; Zheng and Rugina 2008]. CFL reachability is also an important problem in database theory, since it is equivalent to a class of Datalog programs called *chain Datalog programs* [Reps 1998; Smaragdakis and Balatsouras 2015], where the bodies of the recursive rules form a chain of binary predicates.

The Complexity of CFL reachability. It was shown by Yannakakis [Yannakakis 1990] that the general CFL reachability problem can be solved in $O(n^3)$ time on general graphs for a fixed language, where n is the number of vertices in the input graph. This runtime has only been slightly improved by a logarithmic factor to $O(n^3/\log n)$ [Chaudhuri 2008] for the general case. Some improvements exist for more restricted languages: for example, regular languages admit an $O(n^\omega)$ algorithm [Fischer and Meyer 1971], where ω is the matrix multiplication exponent (the current best known value is $\omega \approx 2.37$). In fact, a simple algorithm (with a slightly worse running time) can be obtained as follows. A graph can be encoded as a square matrix $A(i, j)$ and using the observation that A^2 encodes the number of walks from vertex i to j of length 2, one can perform a logarithmic number of iterated multiplications to double the walk path length in each step and obtain an $O(n^\omega \log n)$ algorithm. However, algorithms based on fast matrix multiplication are not desirable practically since they hide large constants in the big-O running time. Currently all known truly sub-cubic algorithms use matrix multiplication. This lack of progress has lead to a conjecture that no better algorithm exists for CFL reachability, that is, a $O(n^{3-\epsilon})$ runtime is not possible for any constant $\epsilon > 0$. This conjecture has been (conditionally) proven but only for the class of combinatorial algorithms [Abboud et al. 2018; Chatterjee et al. 2018]. Combinatorial algorithms are algorithms with a small constant in the big-O that can be implemented efficiently in practice. In contrast, non-combinatorial algorithms can use algebraic methods such as fast matrix multiplication, which potentially obtain a faster theoretical runtime but are not practical.

However, these results do not tell us how efficiently we can evaluate CFL reachability for a specific language $\mathcal{L}$. This motivates us to take a different approach. Given a CFL (or a context-free grammar – CFG), we ask to *identify the exact expression of the running time as a function of the input size*. For example, which languages run in linear time, and for which programs do we need quadratic or cubic time? Answering such a question is important since CFL reachability can naturally capture several fundamental computational problems that are in P. We consider two variants of the CFL reachability problem: in the All-Pairs problem, we produce all pairs that are reachable, while in the On-Demand problem we check reachability for a given pair of vertices.

Fine-grained Complexity. The research direction of pinpointing the exact running time of the problems as a function of their input size is related to the area of *ne-grained complexity* [Williams and Williams 2018]. Since obtaining unconditional lower bounds for polynomial running times is not within our reach, the goal of fine-grained complexity is to reduce a given problem in P to one of a small set of problems that are widely believed to have an optimal algorithm (e.g., 3-SUM, Boolean Matrix Multiplication (BMM, for short), k-Clique). Our goal in this paper follows the same general direction: we seek to show that the complexity for a given language is optimal conditional to one of these conjectures.

The Case of Sparse Graphs. In the CFL reachability problem, the most common parameter used as input size is the number of vertices $n = |V|$. In this case, the lower bounds usually construct instances that are dense (in the sense that the number of edges is super-linear or even quadratic with respect to n). However, in many practical instances the graph G constructed is sparse, and thus it is meaningful to use as input size the number of edges $m = |E|$. This is also the case when we view CFL reachability from a database lens, since m translates to the size of the database (i.e., the number of tuples across all relations). In this work, we will state our fine-grained complexity results using both parameters.

1.1 Our Contributions

Dyck Reachability and a New Lower Bound for CFL reachability. We first study the fine-grained complexity of a fundamental class of CFL reachability problems, called Dyck reachability (Section 3). The Dyck-k grammar produces words of well-matched parentheses of k different types. When we restrict to combinatorial algorithms, it is known that a conditional cubic lower bound exists for the On-Demand problem for Dyck-k for any $k \geq 1$ [Chatterjee et al. 2018; Hansen et al. 2021; Zhang 2020]. We show that there is no algorithm that solves All-Pairs Dyck-2 reachability that requires $O(n^{2.5-\epsilon})$ time for any $\epsilon > 0$ (Theorem 3.2). This lower bound uses a reduction from the APSP or 3SUM hypothesis and thus applies to *all algorithms*, even non-combinatorial ones. Since matrix multiplication is done in time $O(n^{\omega})$ and $\omega < 2.5$, this shows that the general CFL reachability problem is likely strictly harder than Boolean Matrix Multiplication. To the best of our knowledge, this is the first result that shows a super-quadratic lower bound for CFL reachability that applies to all algorithms.

All-Pairs CFL Reachability. Our next set of results looks at general context-free grammars (Section 4). We identify a syntactic condition that is checkable in polynomial time w.r.t. the size of the grammar, such that any CFG that satisfies this condition is as hard as BMM; otherwise, it can be solved in time $O(m)$. Since the combinatorial BMM hypothesis says that BMM cannot be done faster than $O(n^{3-\epsilon})$, this implies a surprising classification result in the combinatorial setting (Theorem 4.5): we can say exactly for which CFGs the all-pairs CFL Reachability problem can be solved in optimal time $O(n^3)$, and for which in optimal time $O(n^2)$. In other words, there exists a sharp dichotomy in the runtime, with no in-between exponents in the polynomial. In the non-combinatorial setting this dichotomy disappears, and we can identify problems with intermediate running times of $O(n^{\omega})$ (when CFG is regular), and $O(n^{(3+\omega)/2})$. We also show that identifying for a given language its exact exponent is actually an undecidable problem (although this does not exclude a possible dichotomy result with an undecidable syntactic condition).

On-Demand CFL Reachability. Next we turn our attention to the easier problem of on-demand CFL Reachability (Section 5). We sketch the fine-grained complexity landscape for both dense and sparse graphs, and provide new conditional lower bounds for several interesting CFGs. A summary of our results can be shown in Table 2. Interestingly, all runtimes we have identified for combinatorial algorithms are either linear, quadratic, or cubic to the input size, so it is an intriguing question whether other intermediate exponents are possible.

Andersen's Pointer Analysis. Finally, we look at the fine-grained complexity for the Andersen's Pointer Analysis (APA), a fundamental type of points-to analysis (Section 6). Although APA is not captured directly as a CFL reachability problem, we can slightly rewrite the program so that it behaves as one. In this way, we can use our techniques to show a lower bound of $O(m^{3-\epsilon})$ for any $\epsilon > 0$ under the combinatorial k-Clique hypothesis, even applying to the on-demand setting. So far a cubic lower bound was only known with respect to n [Mathiasen and Pavlogiannis 2021], so this is a significant strengthening of the lower bound to sparse inputs.

2 PRELIMINARIES

Context-Free Grammars. A *context-free grammar (CFG)* $\mathcal{G}$ can be described by a tuple (V, Σ, R, S), where: V is a finite set of variables (which are non-terminal), Σ is a finite set (disjoint from V) of terminal symbols, R is a set of production rules where each production rule maps a variable to a string $\in (V \cup \Sigma)^*$, and S is a start symbol from V. For example, the grammar $\{S \leftarrow \epsilon, S \leftarrow aSb\}$ is a CFG that describes all strings of the form $a^i b^i$ for some $i \geq 0$. A *context-free language (CFL)* is a language that is produced by some CFG. We will denote by $L(\mathcal{G})$ the language produced by $\mathcal{G}$.

A CFG is *right-regular* if all productions rules are of the form $S \leftarrow \epsilon, S \leftarrow a$ or $S \leftarrow aB$. We can similarly define a left-regular CFG. Right-regular (or left-regular) grammars generate exactly all regular languages. A CFG is *linear* if every production rule contains at most one non-terminal symbol in its body. For example, the grammar $\{S \leftarrow \epsilon, S \leftarrow aSb\}$ is linear.

Dyck-k Grammars. Of particular interest to us will be the family of Dyck-k grammars, which are not regular and linear. The Dyck-k grammar $\mathcal{D}_k$ captures the language of strings with well-matched parentheses of k different types.

$$S \leftarrow \epsilon \mid a_1 S \bar{a}_1 \mid a_2 S \bar{a}_2 \mid \cdots \mid a_k S \bar{a}_k$$

CFL Reachability. The CFL reachability problem takes as an input a directed graph $G = (V, E)$ whose edges are labelled by an alphabet Σ, and a CFG $\mathcal{G}$ defined over the same alphabet Σ. We say that a vertex $v \in V$ is L-reachable from a vertex $u \in V$ if there is a path from u to v in G such that the labels of the edges form a string that belongs in the language L. We consider two variants of the CFL reachability problem:

- All-Pairs: output all pairs of vertices u, v such that v is $L(\mathcal{G})$-reachable from u in G.
- On-Demand: given a pair of vertices u, v, check whether v is $L(\mathcal{G})$-reachable from u in G.

Complexity Problems. In this paper, we will consider the grammar $\mathcal{G}$ as being fixed (i.e., of constant size), and we will be interested in the complexity of CFL reachability with input the graph G. Following this, we define as $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ the All-Pairs problem for a fixed grammar $\mathcal{G}$, and as $\mathsf{CFL}^{\mathsf{od}}(\mathcal{G})$ the On-Demand problem for a fixed grammar $\mathcal{G}$. We then ask the following question: how does the grammar $\mathcal{G}$ effect the computational complexity of CFL reachability problem? This deviates from most previous approaches, which were interested in the computational complexity across all possible CFGs.

Parameters. The input to both problems is a graph $G = (V, E)$. To measure the complexity of CFL reachability, we will use as parameters both the number of vertices $n = |V|$ and the number of edges $m = |E|$. As we will see over the next sections, our results differ depending on the parameter we focus on. To simplify our presentation, we will assume w.l.o.g. that G does not have any isolated vertices (i.e., vertices without adjacent edges). Isolated vertices can only help to satisfy the empty string (if the CFG accepts it), and hence can be handled in time $O(n)$ and then removed from G. This will only add a linear term w.r.t. n in the running time of any algorithm, which we will thus ignore when we measure complexity w.r.t. m. Hence, we will use throughout the paper the following inequality: $n/2 \leq m \leq n^2$. We will also use the notation V_G and E_G to denote the vertex set V and edge set E corresponding to the graph G.

Computational Model. We will consider the word-RAM model with $O(\log n)$ bit words. This is a RAM machine that can read from memory, write to memory and perform operations on $O(\log n)$ bit blocks of data in constant time.

Combinatorial Algorithms. In this work, we will often restrict our attention to *combinatorial* algorithms [Williams and Williams 2018]. This notion is not precisely defined, but informally, it means that the algorithm is discrete, graph-theoretic, and with a runtime has a small constant in the big-O. This requirement disallows the use of fast matrix multiplication, including Strassen's algorithm [Strassen et al. 1969]. The notion of combinatorial algorithms is used to distinguish them from *algebraic* algorithms, the most common example of these being the subcubic algorithms that multiply two boolean $n \times n$ matrices in time $O(n^{\omega})$ with $\omega < 3$.

2.1 Fine-Grained Complexity

Fine-grained complexity is a powerful tool to reason about lower bounds for problems solvable in polynomial time. Consider a problem A with input size n. If A can be solved in polynomial time, our goal is to find the smallest constant $c > 0$ such that A can be solved in time $O(n^c)$

Let us consider one of the simplest problems that have widespread use in fine-grained complexity. The 3SUM problem asks whether, given n integers, three integers exist that sum to 0. There exists a straightforward algorithm that solves 3SUM in quadratic time. However, despite decades of research, it remains unknown if there exists a sub-quadratic time algorithm, i.e., is there an algorithm that takes time $O(n^{2-\epsilon})$ for some constant $\epsilon > 0$. The first step in fine-grained complexity is establishing reasonable conjectures about the running times for well-studied computational problems. In this paper, we will use the following well-established conjectures to prove our conditional lower bounds:

3SUM hypothesis [Gajentaan and Overmars 1995]: There is no $O(n^{2-\epsilon})$ time algorithm for 3SUM, for any constant $\epsilon > 0$. The 3SUM problem takes as input n integers in $\{-n^c, \dots, n^c\}$ for a constant c, and asks whether any three of the integers sum to 0.

APSP hypothesis [Williams and Williams 2018]: There is no $O(n^{3-\epsilon})$ time algorithm for the All-Pairs Shortest Path problem, for any constant $\epsilon > 0$.

Combinatorial BMM hypothesis: There is no combinatorial algorithm that can solve Boolean Matrix Multiplication on boolean matrices of dimensions $n \times n$ with running time $O(n^{3-\epsilon})$ for any constant $\epsilon > 0$.

Combinatorial k-Clique hypothesis: For any $k \geq 3$, there is no combinatorial algorithm that detects a k-Clique in a graph with n nodes in time $O(n^{k-\epsilon})$ for any constant $\epsilon > 0$.

The Combinatorial k-Clique hypothesis is a generalization of the Combinatorial BMM hypothesis, since combinatorial BMM is equivalent to combinatorial triangle (clique with $k = 3$) detection [Williams and Williams 2018].

The second step in fine-grained complexity is to reason about *ne-grained reductions*. Suppose we have a problem A with running time $a(n)$ and a problem B with running time $b(n)$. Given an oracle that can solve problem B in time $O(b(n)^{1-\epsilon})$ for some $\epsilon > 0$, we would like to somehow use this oracle to obtain an algorithm for problem A with running time $O(a(n)^{1-\epsilon'})$ for some $\epsilon' > 0$. The transformation of instances of A to instances of B (aka a reduction) requires some special properties. It is not enough to have a polynomial time reduction from A to B. Instead, we want to ensure that the reduction runs in time faster than $a(n)$. Further, we also want the ability to make multiple calls to the oracle for B (aka a Turing-style reduction).

Fine-grained complexity based lower bounds have been a useful yardstick to understand the hardness for many problems solvable in polynomial time where progress has been stalled for several years (and in some cases, decades). This is not limited to static problems. Seminal work [Henzinger et al. 2015] has also proposed fundamental conjectures for dynamic problems that have been used to reason about the optimality of algorithms in the dynamic setting.

3 DYCK REACHABILITY

In this section, we study the running time of the All-Pairs and On-Demand CFL reachability problems for the family of Dyck-k grammars. We focus on this family of grammars for two reasons. First, Dyck reachability is a fundamental problem at the heart of static analysis. Second, Dyck-2 is in some sense the "hardest" CFG [Greibach 1973], so its complexity will be informative of the behavior of other CFGs. Here, we should note here that any Dyck-k problem for $k \geq 2$ is equivalent (w.r.t. running time) to Dyck-2.[1]

We begin by recalling a result for the On-Demand problem on $\mathcal{D}_k$. This result was proven in a non peer-reviewed publication [Zhang 2020] and [Hansen et al. 2021].

Theorem 3.1. *[Hansen et al. 2021; Zhang 2020] Under the combinatorial BMM hypothesis, there is no combinatorial algorithm that evaluates* $\mathsf{CFL}^{\mathrm{od}}(\mathcal{D}_k)$ *for* $k \geq 1$ *in* $O(n^{3-\epsilon})$ *for any constant* $\epsilon > 0$.

Using fast matrix multiplication, [Mathiasen and Pavlogiannis 2021] showed that $\mathsf{CFL}^{\mathrm{ap}}(\mathcal{D}_1)$ can actually be evaluated in $O(n^{\omega} \log^2 n)$. This result may lead the reader to ask whether truly sub-cubic evaluation is possible for all $\mathcal{D}_k$ for $k \geq 2$. We answer this question negatively. In particular, when we consider any algorithm, including non-combinatorial algorithms that can use fast matrix multiplication, it is still possible to show an $O(n^{2.5})$ conditional lower bound. This implies that it is unlikely that we can construct a fast (or at least quadratic) algorithm for Dyck-k grammars.

Theorem 3.2. *Under the APSP or 3SUM hypothesis,* $\mathsf{CFL}^{\mathrm{ap}}(\mathcal{D}_k)$ *for* $k \geq 2$ *does not admit an* $O(n^{2.5-\epsilon})$ *algorithm for any constant* $\epsilon > 0$.

Proof. We show a reduction from the All-Edges Monochromatic Triangle problem (AE-MonoΔ). In this problem, we are given an n-node graph $H = (V, E)$, where each edge $e \in E$ has a color $c(e)$. We ask to determine for every edge e, whether it appears in a monochromatic triangle in H, i.e., all edges of the triangle have the same color. We will use the fact that AE-MonoΔ does not admit an $O(n^{2.5-\epsilon})$ algorithm unless both the APSP and 3SUM hypotheses fail [Williams and Xu 2020].

The key idea in the reduction is that Dyck-2 (and hence Dyck-k for $k \geq 2$) can encode numbers. We associate with each edge in the graph an integer in $\{1, \dots, m\}$, and with each color an integer $m+1, \dots, m+C$, where C is the number of distinct colors. Let $\bar{e}$ denote the binary encoding of an edge e, and $\bar{e}^R$ denote the reverse sequence of $\bar{e}$. Similarly, we use $\bar{c}$ and $\bar{c}^R$ for the binary encoding of colors. We will assume that the length of the binary encoding is exactly $N = \log(m + C)$ (we can always pad with 0's). Since we will construct a Dyck-2 instance, bits 0, 1 in $\bar{e}, \bar{c}$ will be encoded using symbols [and (respectively and the bits in $\bar{e}^R, \bar{c}^R$ will be encoded using] and) instead of 0 and 1 respectively. The numbers will be encoded as directed line graphs with N edges[2]. The label of the i^{th} edge in the line graph corresponds to the i^{th} bit in the encoding. For example, if $\bar{e} = 0011$, then we encode $\bar{e}$ as:

$$L(e) = \blacksquare \xrightarrow{[} \square \xrightarrow{[} \square \xrightarrow{(} \square \xrightarrow{(} \blacksquare$$

and $\bar{e}^R = 1100$ as:

$$L^R(e) = \blacksquare \xrightarrow{)} \square \xrightarrow{)} \square \xrightarrow{]} \square \xrightarrow{]} \blacksquare$$

We denote by $L_1 \circ L_2$ the stitching of the two line graphs, where the end node of L_1 becomes the start node of L_2. For example, $L(e) \circ L^R(e) = \blacksquare \xrightarrow{[} \square \xrightarrow{[} \square \xrightarrow{(} \square \xrightarrow{(} \blacksquare \xrightarrow{)} \square \xrightarrow{)} \square \xrightarrow{]} \square \xrightarrow{]} \blacksquare$ for edge e with $\bar{e} = 0011$.

[1]One can encode k types of parentheses with 2 types using a simple binary encoding.
[2]Recall that a graph $G = (V, E)$ is a line graph if the vertices V can be arranged into a sequence $v_1, \dots, v_{|V|}$ such that all edges $e \in E$ are of the form (v_i, v_{i+1}).

We now construct the input graph G as follows. We start by creating five copies of the vertex set of V_H: A, B, C, D, E. We use v_A, v_B, v_C, v_D, v_E to denote the copy of vertex v in A, B, C, D, E respectively. For every edge $e = (u, v) \in E_H$, we then do the following:

- connect u_A to v_B with $L(e) \circ L(c(e))$;
- connect u_B to v_C with $L^R(c(e)) \circ L(c(e))$;
- connect u_C to v_D with $L^R(c(e))$; and
- connect u_D to v_E with $L^R(e)$.

We now execute $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{D}_2)$ on G, which has $O(n \log n)$ vertices. To obtain a solution to AE-MonoΔ, we simply filter the pairs (u, v) such that $u \in A$ and $v \in E$: this can be done in time $O(n^2)$. Let P be the resulting set. It remains to argue that the reduction is correct.

Suppose that $(u_A, v_E) \in P$. Then, there is a path from u_A to v_E in G of the following form:

$$u_A \xrightarrow{\bar{e} \circ \bar{c_1}} w_B \xrightarrow{\bar{c_2}^R \circ \bar{c_2}} t_C \xrightarrow{\bar{c_3}^R} z_D \xrightarrow{\bar{e'}^R} v_E$$

Since the labels of this path are recognized by Dyck-2, we must have that $c_1 = c_2$, $c_2 = c_3$, and $e = e'$. But $e = (u, w)$ and $e' = (z, v)$. Thus, $u = z$ and $v = w$. This implies that we have a triangle in H that is formed by the nodes u, v, t. Moreover, the edges all have the same color, and hence the triangle is monochromatic.

For the other direction, suppose that we have an edge (u, v) that forms a monochromatic triangle u, v, t. It is easy to see by the above argument that (u_A, v_E) will then appear in P. □

Finally, we consider the case of sparse graphs, where we are only interested in the number of edges m as the input parameter. We can show the following result.

Theorem 3.3. *Under the combinatorial k-Clique hypothesis,* $\mathsf{CFL}^{\mathsf{od}}(\mathcal{D}_k)$ *(and thus* $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{D}_k)$*) for $k \geq 2$ cannot be solved by a combinatorial algorithm in time $O(m^{3-\epsilon})$ for any constant $\epsilon > 0$.*

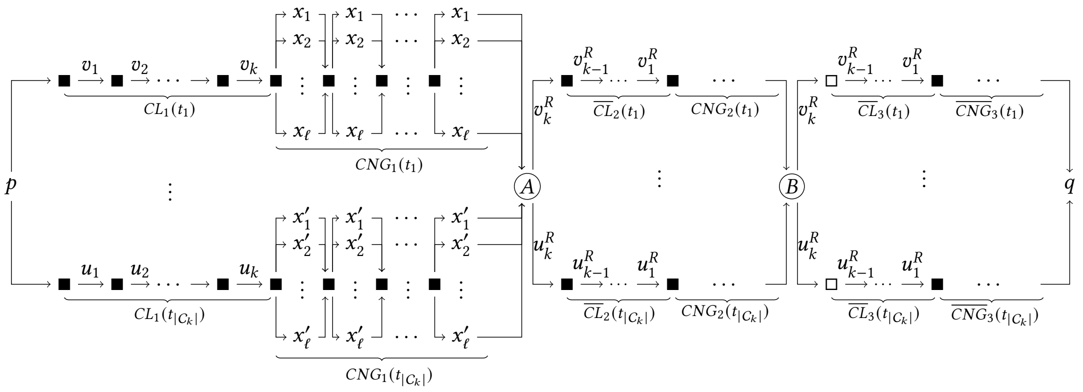
Compared to Theorem 3.1, the above lower bound is stronger, but it is based on a weaker hypothesis, since combinatorial BMM is equivalent to combinatorial 3-Clique. Moreover, Theorem 3.3 does not apply to Dyck-1; we do not know whether Dyck-1 admits a faster algorithm on sparse inputs. We should note that this lower bound was also shown in [Schepper 2018], but indirectly via a reduction from a problem in pushdown automata. Our simplified construction allows us to reuse the same gadgets for proving the lower bounds for Andersen's Pointer Analysis in Section 6. We present the proof next.

3.1 Proof of Theorem 3.3

The reduction uses the same core idea as the one used in [Abboud et al. 2018]. The construction is based on the following idea: If there is a $3k$-clique, then there are 3 disjoint k-cliques. Moreover, if every pair of these 3 cliques forms a $2k$-clique, then there is a 3k-clique in the graph. To reduce the $3k$-Clique problem, we will take as input a graph G with n nodes and transform it to an instance for Dyck-2 with $O(n^{k+1} \log n)$ edges. This transformation will be done in time $O(n^{k+1} \log n)$ time. We obtain the desired bound by letting k grow depending on the constant ϵ.

Notation. We associate with each node in the graph an integer in $\{1, \ldots, n\}$. Let $\bar{v}$ denote the binary encoding of a node v and let $\bar{v}^R$ denote the reverse sequence of $\bar{v}$. We will assume that the length of the binary encoding $\bar{v}$ is exactly $N = \log n$ (we can always pad with 0's). Since we will construct a Dyck-2 instance, bits 0, 1 in $\bar{v}$ will be encoded using symbols [and (respectively and the bits in $\bar{v}^R$ will be encoded using] and) instead of 0 and 1 respectively.

The numbers $\bar{v}, \bar{v}^R$ will be encoded as directed line graphs with N edges. The label of the ith edge in the line graph corresponds to the ith bit in $\bar{v}$ (resp. $\bar{v}^R$). We call this process *vertex expansion (VE)*.



p
q
v_1 v_2 v_k
$CL_1(t_1)$
x_1 x_2 x_ℓ
$CNG_1(t_1)$
A
B
v_k^R v_{k-1}^R v_1^R
$\overline{CL}_2(t_1)$ $CNG_2(t_1)$
$\overline{CL}_3(t_1)$ $\overline{CNG}_3(t_1)$
u_1 u_2 u_k
$CL_1(t_{|C_k|})$
x'_1 x'_2 x'_ℓ
$CNG_1(t_{|C_k|})$
u_k^R u_{k-1}^R u_1^R
$\overline{CL}_2(t_{|C_k|})$ $CNG_2(t_{|C_k|})$
$\overline{CL}_3(t_{|C_k|})$ $\overline{CNG}_3(t_{|C_k|})$

Fig. 1. The input graph constructed for the On-Demand Dyck-2. We are checking whether (p, q) is in the output. For the clique t_1, $N_{t_1} = \{x_1, \dots, x_\ell\}$ and $CNG_1(t_1)$ uses k copies of N_{t_1}.

Gadgets. Similar to the reduction in [Abboud et al. 2018], we begin by constructing the set C_k of all cliques of size k in G. This takes time $O(n^k)$. For each clique $t \in C_k$, we assume that the vertices forming $t = \{v_1, \dots, v_k\}$ are sorted in lexicographic order. We define two types of gadgets.

The first gadget is the *clique list* (CL). Consider a k-clique $t = \{v_1, v_2, \dots, v_k\}$. To create the gadget $CL(t)$, we take the line graphs $L(v_1), L(v_2), \dots, L(v_k)$ and stitch them together to form a line graph with $k \cdot N$ edges. In particular, the last node of $L(v_i)$ is the first node of $L(v_{i+1})$ for every $i = 1, \dots, k-1$. For simplicity of presentation, we will directly use the vertex instead of its expansion as shown in Figure 1. We also define the reverse of a clique list $\overline{CL}(t)$. Here, we take the line graphs $L^R(v_1), L^R(v_2), \dots, L^R(v_k)$ and stitch them in reverse order, i.e. the last node of $L^R(v_i)$ is the first node of $L^R(v_{i-1})$ for every $i = 2, \dots, k$. The clique list construction does not repeat any vertex, unlike the reduction in [Abboud et al. 2018] that repeats each vertex k times.

The second gadget is the *clique neighbor gadget* (CNG). For a given clique t, consider the set of all vertices $N_t = \{w_1, \dots, w_\ell\}$ (sorted in lexicographic order), such that for every vertex $u \in N_t$ forms an edge with every vertex in t. Note that $t \cap N_t = \emptyset$. These sets can be computed in time $O(n^{k+1})$ as follows. For a given cliquet of size k, we iterate over all the vertices $v \in V$ in the graph and check in constant time whether v connects to all k vertices of t. Therefore, each clique requires $O(n)$ time to be processed. As there are $\binom{n}{k}$ cliques of size exactly k (these can be generated in time $O(n^k)$ straightforwardly), the total processing time is $O(n^k \cdot n) = O(n^{k+1})$.

Then, $CNG(t)$ is a directed graph that is constructed as follows. First, for every w_i we create a line graph with $k \cdot N$ edges by stitching together k copies of $L(w_i)$. Then, we stitch these line graphs together by making the first node of the first copy be the same for all w_i, the first node of the second copy be the same, etc. The last node of the last copy is also the same (see Figure 1). Similar to clique list, we also define $\overline{CNG}(t)$ where the gadget uses $L^R(w_i)$ for every vertex $w_i \in N_t$.

Graph Construction. The instance for the On-Demand Dyck-2 language is constructed as follows. For each clique t, we stitch $CL_1(t)$ with $CNG_1(t)$ such that the last node of $CL_1(t)$ is the same as the first node of $CNG_1(t)$. All $CNG_1(t)$ flow into a common connector vertex A (shown in black in Figure 1 right after $CNG_1(t_i)$). Then, we construct $\overline{CL}_2(t)$ for each clique, make A to be the start vertex of all $\overline{CL}_2(t)$, and connect it to $CNG_2(t)$, which flows into another common connector vertex. Finally, we perform the same process but for $\overline{CL}_3(t)$ and $\overline{CNG}_3(t)$. The first vertex of every $CL_1(t)$ and last vertex of every $\overline{CNG}_3(t)$ connect to vertices p and q respectively. We label the outgoing edges from p with [and the incoming edges to q with]. We now use this instance as the extensional input for the On-Demand problem over Dyck-2 and ask whether $T(p, q)$ is true or not.

Before proving the result, we state a simple but useful observation that follows from the construction.

Observation 3.4. *Consider the gadget $CL(t)$ (resp. $\overline{CL}(t)$) that is immediately followed by $CNG(t)$ (resp. $\overline{CNG}(t)$). Then, the set of vertices traversed by any path in $CNG(t)$ (resp. $\overline{CNG}(t)$) has no vertex in common with t.*

Claim 3.5. *If the On-Demand problem on Dyck-2 returns true, then there exists a $3k$-clique in the input graph.*

Proof. Let t_1 be the clique chosen by the vertices in $CL_1(t)$ and let V be the set of vertices traversed in $CNG_1(t)$ by the grammar. Since $CNG_1(t)$ is followed by $\overline{CL}_2(t)$, a valid Dyck-2 can be formed only if V corresponds to some clique $t_2 \in C_k$. Indeed, if this was not the case, then the word will not be well-formed. For instance, if $\overline{CL}_2(t)$ corresponds to the reverse of t_1, then we will have a set of open brackets between a set of balanced brackets which is not a valid word in Dyck-2. Further, Observation 3.4 guarantees that there is no common vertex between $t_1$and t_2. Thus, it holds that $t_1 \cup t_2$ is a $2k$-clique. Next, suppose V' is the set of vertices traversed in $CNG_2(t)$. We need to argue that V' corresponds to a clique t_3 with no common vertices with $t_1 \cup t_2$. This claim follows from the observation that since the remaining two gadgets are $\overline{CL}_3(t)$ and $\overline{CNG}_3(t)$, the Dyck-2 word can be valid only if the last $\overline{CNG}_3(t)$ uses the vertex set of t_1. Similar to our previous argument, if this was not the case, and $\overline{CL}_3(t)$ uses vertices from (say) t_1, the word is not balanced because V' contains open brackets within the word where t_1 is balanced. Thus, $\overline{CNG}_3(t)$ must correspond to t_1. Observation 3.4 tells us that $\overline{CL}_3(t)$ cannot have any common vertices with $\overline{CNG}_3(t_1)$. Applying Observation 3.4 again, we see that V' also cannot have any common vertex with $\overline{CL}_2(t_2)$. Thus, $t_3 \cup t_2$ and $t_3 \cup t_1$ are both $2k$-cliques. This completes the proof. □

Claim 3.6. *If there exists a $3k$-clique in the input graph, then the On-Demand problem on Dyck-2 returns true.*

Proof. Let $t_1, t_2, t_3 \in C_k$ be three disjoint k-cliques. We will show that there exists a path from u to v that forms a valid Dyck-2 word. Consider the path formed by the vertices

$$CL(t_1), CNG_2(t_2), \overline{CL}_2(t_2), CNG_2(t_3), \overline{CL}_3(t_3), \overline{CNG}_3(t_1).$$

The brackets on the outgoing and incoming edges from p, q and all connector vertices are also balanced. It is also straightforward to see that t_2 and t_3 are balanced within the Dyck word formed by the balanced brackets of t_1. □

Given an instance of 3k-Clique graph $G = (V, E)$, we construct the instance as described above and solve the On-Demand problem over the Dyck-2 language. By Claims 3.5 and 3.6, the On-Demand problem returns true iff the graph G contains a $3k$-clique.

4 THE ALL-PAIRS PROBLEM

In this section, we will study the all-pairs problem for CFL reachability. We recall here the following known facts about this problem:

- For every grammar $\mathcal{G}$, $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be solved by a combinatorial algorithm in time $O(n^3)$.
- If $L(\mathcal{G})$ is regular, then $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be solved in $O(n^{\omega})$ time via fast matrix multiplication.
- If $\mathcal{G}$ is a linear grammar, then $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be solved by a combinatorial algorithm in $O(m \cdot n)$ time.

4.1 A Reduction to BMM

It is known that for some grammar $\mathcal{G}$, there exists a fine-grained reduction to BMM. The first question we answer is: for which CFGs can we reduce CFL reachability to BMM, in the sense that a faster running time for $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ implies a faster running time for BMM?

To answer this question, we first need the following definition:

De nition 4.1 (Join-Inducing CFG). Let $\mathcal{G}$ be a context-free grammar. We say that $\mathcal{G}$ is *join-inducing* if it produces at least one string of length ≥ 2. Otherwise, we say that $\mathcal{G}$ is *join-free*.

It turns out that we can check whether this property is satisfied efficiently.

Lemma 4.2. *Let $\mathcal{G}$ be a CFG. Then, in polynomial time (w.r.t. the size of $\mathcal{G}$) we can check whether $\mathcal{G}$ is join-inducing, and if so, output a string of length ≥ 2 produced by $\mathcal{G}$ in polynomial time (w.r.t. the size of $\mathcal{G}$).*

Proof. We say that a grammar is *proper* if: (i) it has no rules of the form $X \longleftarrow \epsilon$ (with the exception of one rule of the form $S \longleftarrow \epsilon$ if $\mathcal{G}$ produces the empty string), (ii) it has no cycles (meaning that a non-terminal symbol cannot derive itself), (iii) all non-terminal symbols are productive (i.e. they can derive a word with terminal symbols), and (iv) all non-terminal symbols are reachable from the start symbol S. We can always transform $\mathcal{G}$ into a weakly equivalent[3] grammar $\mathcal{G}'$ that is proper, and we can do this in polynomial time in the size of the grammar. We can then transform $\mathcal{G}'$ into a weakly equivalent grammar $\mathcal{G}''$ that is in Chomsky Normal Form: this means that every production rule is of the form $A \longleftarrow BC$ or $A \longleftarrow \alpha$, where A, B, C are non-terminal symbols and α is a terminal symbol. This can also be done in polynomial time. Note that $\mathcal{G}''$ is also proper after this transformation.

Claim: $\mathcal{G}$ is join-inducing if and only if $\mathcal{G}''$ has a rule of the form $A \longleftarrow BC$.

Indeed, if every rule in $\mathcal{G}''$ has one terminal symbol on the right hand side, then $\mathcal{G}$ can generate only strings of length one, hence it is join-free. Otherwise, since $\mathcal{G}''$ is proper and in Chomsky Normal Form, it must have a rule of the form $S \longleftarrow AB$. Since A, B are non-terminal symbols, they must each derive a string with at least one terminal symbol. Hence, the grammar $\mathcal{G}''$ (and thus $\mathcal{G}$) can produce a string of at least length 2. Note that the string can be computed in polynomial time. In fact, once $\mathcal{G}''$ has been produced, generating the output takes $O(|\mathcal{G}''|)$ time: a linear pass beginning from the rule with the start symbol is sufficient to find a string of length ≥ 2. □

We can now prove the following conditional lower bound.

Lemma 4.3. *Let $\mathcal{G}$ be a join-inducing CFG. Suppose that $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be computed in time $T(n) = \Omega(n^2)$. Then, BMM can be solved in time $O(T(n))$.*

Proof. By Lemma 4.2, we can find a string of length at least 2 that is produced by $\mathcal{G}$ (note that $|\mathcal{G}|$ is a constant). Let this string be $r_1 r_2 \dots r_k$ where $k \geq 2$.

[3]Weakly equivalent means that G and G' produce the same language.

Now, suppose we want to multiply two $n \times n$ Boolean matrices A, B. We encode the matrices as a directed $(k+1)$-partite graph H with vertex sets $V_0, \dots, V_k$ of size n. Let $V_\ell = \{v_1^{(\ell)}, \dots, v_n^{(\ell)}\}$. We only add edges between two consecutive vertex sets $V_\ell, V_{\ell+1}$, where $\ell = 0, \dots, k-1$ as follows:

$$E_1 = \{(v_i^{(0)}, v_j^{(1)}) \mid A[i][j] = 1, i \in [n], j \in [n]\}$$
$$E_i = \{(v_j^{(i-1)}, v_j^{(i)}) \mid j \in \{1, \dots, n\}\} \quad \text{for } i \in \{2, \dots, k-1\}$$
$$E_k = \{(v_i^{(k-1)}, v_j^{(k)}) \mid B[i][j] = 1, i \in [n], j \in [n]\}$$

Finally, we label the edges of H such that if $(u, v) \in E_i$, then we assign label r_i.

Now, consider the set the pairs P produced if we run $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ on H. We take the result and filter it such that the first column has values from V_0 and the second from V_k; more specifically, we compute $P' = P \cap (V_0 \times V_k)$. Since $|V_0| \cdot |V_k| = n^2$, this computation can run in time $O(n^2)$. Observe that the input graph has $\Theta(n)$ vertices, hence the total running time is $O(T(n) + n^2) = O(T(n))$.

We now claim that P' computes $C = A \times B$, i.e., $C[i][j] = 1$ if and only if $(v_i^{(0)}, v_j^{(k)}) \in P'$.

$\Rightarrow$ For the one direction, suppose that $C[i][j] = 1$. Then, there exists some $k \in \{1, \dots, n\}$ such that $A[i][k] = B[k][j] = 1$. Now, consider the following directed path in H:

$$(v_i^{(0)}, v_k^{(1)}), (v_k^{(1)}, v_k^{(2)}), \dots, (v_k^{(k-1)}, v_j^{(k)})$$

First, notice that $v_i^{(0)} \in V_0$ and $v_j^{(k)} \in V_k$. Second, the word along the path is labeled $r_1 \dots r_k$, hence it is accepted by $\mathcal{G}$. From these two facts, we obtain that $(v_i^{(0)}, v_j^{(k)}) \in P'$.

$\Leftarrow$ For the other direction, consider some $(v_i^{(0)}, v_j^{(k)}) \in P'$. Since H is a directed $(k+1)$-partite graph, any string that produces a result in P' will be a substring of $r_1 r_2 \dots r_k$. However, the intersection with the cartesian product $V_0 \times V_k$ keeps only the strings that start from V_0 and end at V_k, so these will be exactly $r_1 r_2 \dots r_k$. Hence, there is a path $(v_i^{(0)}, v_k^{(1)}), (v_k^{(1)}, v_k^{(2)}), \dots, (v_k^{(k-1)}, v_j^{(k)})$ for some $k \in \{1, \dots, n\}$, which means that $A[i][k] = B[k][j] = 1$ and consequently $C[i][j] = 1$. □

The above lemma tells us that solving $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ is at least as hard as BMM if $\mathcal{G}$ is join-inducing. On the other hand, the problem becomes trivial for join-free CFGs.

Lemma 4.4. *Let $\mathcal{G}$ be a join-free CFG. Then, $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be evaluated in time $O(m+n)$.*

Proof. Since $\mathcal{G}$ is not join-inducing, the language $L(\mathcal{G})$ can be described as a set A of strings of length one (plus possibly the empty string). Hence, we can simply return the edges in the graph with labels from A, a task that can be done in time linear to the number of edges and nodes in the graph. □

4.2 The Landscape for Combinatorial Algorithms

Combining the results of the previous section, we can obtain the following dichotomy theorem that characterizes the complexity of the problem when we restrict to combinatorial algorithms.

Theorem 4.5. *Let $\mathcal{G}$ be a context-free grammar.*

- *If $\mathcal{G}$ is join-inducing, then $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be evaluated in time $O(n^3)$ by a combinatorial algorithm. Moreover, under the combinatorial BMM hypothesis, there is no combinatorial algorithm that evaluates $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ in time $O(n^{3-\epsilon})$ for any constant $\epsilon > 0$.*
- *If $\mathcal{G}$ is join-free, $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be evaluated in time $\Theta(m+n) = \Theta(n^2)$.*

Moreover, we can decide which of the two cases holds in time polynomial to the size of $\mathcal{G}$.

The above dichotomy theorem shows a sharp behavior of the running time with respect to n. Indeed, the exponent of n can be either 2 or 3, with nothing in between. Surprisingly, it is even decidable (in polynomial time) in which of two classes each grammar belongs.

4.3 The Landscape for All Algorithms

If we allow for any type of algorithm (including ones that use fast matrix multiplication), then the complexity landscape changes considerably. Indeed, we already know that regular grammars admit a subcubic algorithm. Using Lemma 4.3 we can characterize the complexity within the class of CFGs that describe regular languages.

Theorem 4.6. *Let $\mathcal{G}$ be a CFG such that $L(\mathcal{G})$ is regular and join-inducing. Then, $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be solved in time $O(n^{\omega})$. Moreover, there is no algorithm that evaluates $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ in time $O(n^{\omega-\epsilon})$ for any constant $\epsilon > 0$.*

We next consider CFGs that describe non-regular languages. Is it the case that all such CFGs require cubic time even with non-combinatorial tools? [Mathiasen and Pavlogiannis 2021] already showed that $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{D}_1)$ can be solved in time $O(n^{\omega} \log^2 n)$. We present next another example of a natural non-regular CFG that can be solved in subcubic time. The running time for this CFG has a different exponent, which means that it possibly captures a different class of problems.

Consider the following non-regular language: $\mathcal{L}_{\geq} = \{a^i b^j \mid i \geq j\}$. This language can be expressed by the following CFG $\mathcal{G}_{\geq}$:

$$S \leftarrow T_1 T_2 \qquad T_1 \leftarrow \epsilon \mid aT_1 \qquad T_2 \leftarrow \epsilon \mid aT_2 b.$$

Lemma 4.7. *$\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G}_{\geq})$ can be solved in time $O(n^{(3+\omega)/2})$.*

Proof. The algorithm works in three steps. In the first step, we compute the all-pairs shortest paths in the graph G_a where we keep only edges with label a with weight -1, and remove any edges with label b. This can be done in time $O(n^{(3+\omega)/2})$ [Alon et al. 1997]. A shortest path between i and j in G_a means a longest path of a-edges in G. Let M_a be the matrix such that $M_a[i][j]$ is the length of the longest a-path between i and j.

In the second step, we compute the all-pairs shortest paths in the graph G_b where we keep only edges with label b with weight +1, and remove any edges with label a. A shortest path here means a shortest path of b-edges in G. This can also be done in time $O(n^{(3+\omega)/2})$ [Alon et al. 1997]. Let M_b be the matrix such that $M_b[i][j]$ is the length of the shortest b-path between i and j.

In the final step, we compute the existence-dominance product of the two matrices M_a, M_b. The existence-dominance product of two integer matrices A and B is the Boolean matrix C such that $C[i][j] = 0$ iff there exists a k such that $A[i][k] \geq B[k][j]$. We can solve the existence-dominance product problem also in time $O(n^{(3+\omega)/2})$ [Matousek 1991].

We finally claim that the algorithm is correct. Indeed, if the resulting existence-dominance product matrix C has $C[i][j] = 1$, this means that there exists an a-path from i to some k that is at least as long as another b-path from k to j. The concatenation of these two paths forms a path with labels thats satisfies $\mathcal{G}_{\geq}$. On the other hand, if $C[i][j] = 0$, for any intermediate node k, the longest a-path from i to k is strictly shorter that the shortest b-path from k to j, hence no path from i to j satisfies the CFG. □

What is the best possible lower bound we can get for a non-regular CFG? Recall that in the previous section we showed that CFL reachability for Dyck-2 admits a $O(n^{2.5})$ conditional lower bound. This implies that solving CFL reachability for Dyck-2 (and thus in general) is likely strictly harder than BMM. However, it is an open problem if the lower bound can be improved, or if there exists a faster non-combinatorial algorithm.

An Undecidability Result. We will show next that it is not possible to determine whether a given CFG can be evaluated in time $O(n^{\omega})$ or not. In other words, even if a characterization of the complexity exists for different CFGs, this characterization will not be decidable. The undecidability result is based on Greibach's theorem. First, we need the following definition.

De nition 4.8 (Right Quotient). If $\mathcal{L}$ is a language and α is a single symbol, we define the language $\mathcal{L}/\alpha = \{w \mid w\alpha \in \mathcal{L}\}$ to be the *right quotient* of $\mathcal{L}$ with respect to α.

Theorem 4.9 (Greibach [Greibach 1968]). *Let C be any non-trivial property for the class of CFLs that is true for all regular languages and that is preserved under the right quotient with a single symbol. Then C is undecidable for the class of CFLs.*

We now state the theorem formally.

Theorem 4.10. *Suppose the APSP or 3SUM hypothesis holds. Then, for any constant $c \in [\omega, 2.5)$, it is undecidable whether for a given CFG $\mathcal{G}$, $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be evaluated in time $O(n^c)$.*

Proof. Fix a constant $c \in [\omega, 2.5)$. To prove undecidability, we apply Greibach's theorem. Consider the following property C for a CFL: the All-Pairs CFL reachability problem for any CFG that produces the language can be evaluated in time $O(n^c)$. As we have seen, C is satisfied by all regular languages since $c \geq \omega$. It is also non-trivial, since under the APSP or 3SUM hypothesis, Dyck-2 cannot be evaluated in time $O(n^{2.5-\epsilon})$ for any constant $\epsilon > 0$, hence it does not admit an $O(n^c)$ algorithm. It remains to show that C is closed under the right quotient by a single symbol.

Indeed, take a CFL $\mathcal{L}$ and a CFG $\mathcal{G}$ that produces it. Consider the language $\mathcal{L}/\alpha$ for a single symbol α. We now want to evaluate the CFG $\mathcal{G}_\alpha$ that corresponds to the language $\mathcal{L}/\alpha$. To do this, we extend the input graph G as follows: for each vertex $v \in V$, we add an edge (v, t_v) with label α, where t_v is a fresh distinct vertex. Let G' be the resulting graph. Note that $|V(G')| = 2|V(G)| = O(n)$. Then, we run the algorithm for $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ on the new graph G', which runs in time $O(n^c)$. Finally, we can see that by construction, (u, v) is an output pair for $\mathcal{G}_\alpha$ if and only if (u, t_v) is an output pair for $\mathcal{G}$. Hence, to obtain the output for $\mathcal{G}_\alpha$ it remains to do the following: for every pair of the form (u, t_v) for $\mathcal{G}$, output (u, v). This is doable in time $O(n^2)$ by iterating over all output pairs for $\mathcal{G}$. □

4.4 All-Pairs on Sparse Graphs

Finally, we turn our attention to the case where we interested in running time as a function of the number of edges in the graph m (instead of the number of nodes). This is particularly helpful when the input graph to the CFL reachability is sparse. We summarize our results in Table 1.

As we proved in the previous subsection, a CFG that is join-free can be evaluated in time $O(m)$, which is optimal. On the other hand, we can show the following *unconditional lower bound* for join-inducing grammars. This result uses the fact that for any join-inducing grammar, we can construct a worst-case input instance that produces an output of size $\Omega(m^2)$.

Lemma 4.11. *Let $\mathcal{G}$ be a join-inducing CFG. Then, any algorithm that computes $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ needs time $\Omega(m^2)$.*

Proof. We follow the same construction as in the proof of Lemma 4.3. In particular, since $\mathcal{G}$ is join-inducing, it can produce a string $r_1 r_2 \dots r_k$ with $k \geq 2$.

Consider the following family of $(k+1)$-partite graphs with vertex sets $V_0, \dots, V_k$ of size n. Let $V_i = \{v_1^{(i)}, \dots, v_n^{(i)}\}$. We only add edges between two consecutive vertex sets V_i, V_{i+1} as follows (fix

CFG	**upper bound**	**lower bound**	
join-free	$O(m)$	$\Omega(m)$	unconditional
join-inducing	$O(m^3)$	$\Omega(m^2)$	unconditional [Thm 4.11]
join-inducing + linear	$O(mn) = O(m^2)$	$\Omega(m^2)$	unconditional [Thm 4.11]
Dyck-1	$O(m^3)$	$\Omega(m^2)$	unconditional [Thm 4.11]
Dyck-k, $k \geq 2$	$O(m^3)$	$\Omega(m^{3-\epsilon})$	under comb. k-Clique [Thm 3.3]

Table 1. Upper and lower bounds for the **all-pairs** CFL reachability problem.

some $j^\star \in \{1, \dots, n\}$):

$$E_1 = \{(v_i^{(0)}, v_{j^\star}^{(1)}) \mid i \in \{1, \dots, n\}\}$$
$$E_i = \{(v_j^{(i-1)}, v_j^{(i)}) \mid j \in \{1, \dots, n\}\} \quad i \in \{2, \dots, k-1\}$$
$$E_k = \{(v_{j^\star}^{(k-1)}, v_i^{(k)}) \mid i \in \{1, \dots, n\}\}$$

Finally, we assign label r_i to any edge in E_i. It is easy to see that the input size is $m = k \cdot n$, while the output size is $n^2 = \Omega(m^2)$. Since any algorithm must produce this output, the desired lower bound is obtained. □

In terms of upper bounds, the problem $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can always be evaluated in time $O(n^3) = O(m^3)$. Hence, every join-inducing CFG can be evaluated in time $O(m^c)$ for some exponent $c \in [2, 3]$. Additionally, for linear CFGs Yannakakis [Yannakakis 1990] showed that if $\mathcal{G}$ is *linear*, then $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be evaluated in time $O(mn)$. Thus, we have proved the following dichotomy theorem.

Theorem 4.12. *Let $\mathcal{G}$ be a linear CFG. Then:*

- *If $\mathcal{G}$ is join-inducing, then $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be evaluated in time $O(m^2)$ by a combinatorial algorithm. Moreover, every algorithm that evaluates $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ needs time $\Omega(m^2)$.*
- *If $\mathcal{G}$ is join-free, $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be evaluated in (optimal) linear time $O(m)$.*

Moreover, we can decide which of the two cases holds in time polynomial to the size of the grammar.

Unfortunately, the landscape becomes murkier for non-linear CFGs. Indeed, the $O(mn)$ algorithm is not applicable in this case, hence we do not know whether the $O(m^2)$ upper bound holds. As we showed in Theorem 3.3, Dyck-k for $k \geq 2$ has an $\Omega(m^{3-\epsilon})$ lower bound under the combinatorial k-Clique hypothesis. It is an open problem whether there exists any CFGs with intermediate complexity that can be evaluated in time $\Theta(m^c)$ for $c \in (2, 3)$.

Can we determine whether a given CFG can be evaluated in $O(m^2)$ time, or in general in time $O(m^c)$ for some constant c strictly smaller than 3? We answer this question negatively.

Theorem 4.13. *Suppose the combinatorial k-Clique hypothesis holds. Then, for any constant $c \in [2, 3)$, it is undecidable whether $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be evaluated by a combinatorial algorithm that runs in time $O(m^c)$.*

5 THE ON-DEMAND PROBLEM

For every CFG that corresponds to a regular grammar, we can solve the On-Demand problem in time $O(m) = O(n^2)$, which is optimal (we can always construct an input with size n^2). Hence, the CFGs of interest are the non-regular grammars. In Section 3, we saw that $\mathsf{CFL}^{\mathsf{od}}(\mathcal{D}_k)$ for $k \geq 1$ is BMM-hard. We can use the same reduction to show BMM-hardness for other non-regular CFGs:

	CFG	upper bound	lower bound	
n	any	$O(n^3)$	$\Omega(n^2)$	unconditional
	regular	$O(n^2)$	$\Omega(n^2)$	unconditional
	$\{a^ib^i \mid i \geq 0\}$	$O(n^3)$	$\Omega(n^{3-\epsilon})$	under comb. BMM [Thm 5.1]
	palindromes with ≥ 2 symbols	$O(n^3)$	$\Omega(n^{3-\epsilon})$	under comb. BMM [Thm 5.1]
	Dyck-k, $k \geq 1$	$O(n^3)$	$\Omega(n^{3-\epsilon})$	under comb. BMM [Thm 3.1]
m	any	$O(m^3)$	$\Omega(m)$	unconditional
	regular	$O(m)$	$\Omega(m)$	unconditional
	linear CFG	$O(m^2)$	?	
	$\{a^ib^i \mid i \geq 0\}$	$O(m^2)$	$\Omega(m^{2-\epsilon})$	under comb. k-Clique [Thm 5.4]
	palindromes with ≥ 2 symbols	$O(m^2)$	$\Omega(m^{2-\epsilon})$	under comb. k-Clique [Thm 5.4]
	Dyck-1	?	$\Omega(m^{2-\epsilon})$	under comb. k-Clique [Thm 5.4]
	Dyck-k, $k \geq 2$	$O(m^3)$	$\Omega(m^{3-\epsilon})$	under comb. k-Clique [Thm 3.3]

Table 2. Upper and lower bounds of combinatorial algorithms for the **on-demand** problem.

Theorem 5.1. *The On-Demand CFL reachability problem is BMM-hard for the (non-regular) CFGs that produce the following CFLs:*

(1) The language $\{a^isb^i \mid i \geq 0\}$ where s can be any string, including the empty one;
(2) Strings over $\{a, b\}$ where the number of a's is equal to the b's;
(3) Palindrome strings of even (odd) length over an alphabet with at least 2 symbols.

The above theorem implies that the $O(n^3)$ algorithm we used for the All-Pairs problem is optimal for the On-Demand problem of all the above grammars if we restrict to combinatorial algorithms (under the combinatorial BMM hypothesis).

The reader may now ask: is it true that every non-regular CFG has a cubic lower bound conditional to the combinatorial BMM hypothesis? We answer this question in the negative. Indeed, consider the following non-regular grammar $\mathcal{G}_{\geq}$ we defined in the previous section, that encodes the language $\{a^ib^j \mid i \geq j\}$. As we show with the next lemma, the On-Demand problem can be answered in time $O(m) = O(n^2)$.

Lemma 5.2. $\mathsf{CFL}^{\mathrm{od}}(\mathcal{G}_{\geq})$ *can be solved by a combinatorial algorithm in time $O(m)$.*

Proof. Let (s, t) be the input pair of constants and G be the input labeled graph (with labels in $\{a, b\}$). We will do the following: for each vertex v, we will first compute the length $\ell_a[v]$ of the longest path from s to v that uses only a's (we allow this length to be $+\infty$ in the case that we can make this path infinitely long, and $-\infty$ if v is unreachable from s). We claim that we can do this in time $O(m)$. To do this, we first construct the graph G_a that contains only the edges labeled with a, which can be done in linear time. Then, we perform a depth-first search starting from the vertex u that produces as an output the strongly connected components (SCCs) of G_a, along with a topological sort of the SCCs. This step can also be performed in linear time. Let $C_1, C_2, \ldots, C_t$ be the SCCs in the topological order. W.l.o.g., assume that $s \in C_1$. We let $\ell_a[C_1] = 0$ if $|C_1| = 1$, otherwise $\ell_a[C_1] = +\infty$. Then, we iterate over all SCCs following the topological order: for C_i, if $|C_i| > 1$ we assign $\ell_a[C_i] = +\infty$; otherwise, $\ell_a[C_i] = \max_j \ell_a[C_j] + 1$, where j iterates over all SCCs C_j such that there is an edge from C_j to C_i (if there is no such edge, we let $\ell_a[C_i] = -\infty$). Finally, we let $\ell_a[v] = \ell_a[C_i]$, where $v \in C_i$.

Similarly, for each vertex v we compute the length $\ell_b[v]$ of the shortest path from v to t that uses only $b's$ (the distance can be $+\infty$ if there is no such path). We can also do this in time $O(m)$.

Indeed, we first construct the graph G_b that contains only the edges labeled with b. Then, we solve a single-target shortest path problem in an unweighted graph, where the target is t; this can be done in linear time using breadth-first search.

Finally, we iterate over every node $v \in V(G) \setminus \{s, t\}$. If there exists such a node where $\ell_a[v] \neq -\infty$, $\ell_b[v] \neq +\infty$ and $\ell_a[v] > \ell_b[v]$ then we claim that the desired path between s, t exists; otherwise not. The final step can be done in time $O(n)$. □

Finally, we can show an analogous undecidability result to the one in Theorem 4.13.

Theorem 5.3. *Suppose the combinatorial BMM hypothesis holds. Then, for any constant $c \in [2, 3)$, it is undecidable whether the On-Demand CFL reachability problem for a given CFG can be evaluated by an $O(n^c)$ combinatorial algorithm.*

5.1 On-demand on Sparse Graphs

Finally, we study the On-Demand problem with respect to the input size m. We identify CFGs that can be evaluated in linear, quadratic, and cubic time (see Table 2).

The On-Demand problem for a program that corresponds to a regular CFG is in time $O(m)$ [Yannakakis 1990], while there are non-regular programs that are also in linear time. On the other hand, there are several non-regular (and even linear) programs that have a quadratic lower bound. To show this lower bound, we use the fact that, under the combinatorial k-Clique hypothesis, for any constant $\epsilon > 0$, for any $k > 2/\epsilon$, finding whether a graph contains a k-cycle can be detected in time $\Omega(m^{2-\epsilon})$ [Lincoln and Vyas 2020].

Theorem 5.4. *Under the combinatorial k-Clique hypothesis, the On-Demand problem for following cases cannot be solved by a combinatorial algorithm in time $O(m^{2-\epsilon})$ for any constant $\epsilon > 0$.*

(1) Dyck-k, for any $k \geq 1$;
(2) The language $\{a^i s b^i \mid i \geq 0\}$ where s can be any string, including the empty one;
(3) Strings over $\{a, b\}$ where the number of a's is equal to the b's;
(4) Palindrome strings of even (odd) length over an alphabet with at least 2 symbols.

We can prove lower bounds for other CFGs using *inverse homomorphisms*. A homomorphism h on an alphabet Σ is a function that gives a word (in a possibly different alphabet) for every symbol in Σ. We can extend h naturally to map a word to another word. If h is a homomorphism and L a language whose alphabet is the output language of h, then we define the inverse homomorphism as $h^{-1}(L) = \{w \mid h(w) \in L\}$. CFLs are known to be closed under inverse homomorphisms.

Lemma 5.5. *Suppose the CFG that corresponds to a CFL L admits an $O(m^c)$ algorithm for the All-Pairs (resp. On-Demand) problem for some constant $c \geq 1$. Then, the CFG that corresponds to $h^{-1}(L)$ also admits an $O(m^c)$ algorithm for the All-Pairs (resp. On-Demand) problem.*

Example 5.6. To show how to apply Lemma 5.5 to obtain further lower bounds, consider the CFL $L_1 = \{(ad)^i cb^i \mid i \geq 0\}$. Now, take the CFL $L_0 = \{a^i cb^i \mid i \geq 0\}$. Consider the homomorphism h with $h(a) = ad, h(b) = b$, and $h(c) = c$. It is easy to see that $L_0 = h^{-1}(L_1)$. Lemma 5.5 now tells us that the $\Omega(m^{2-\epsilon})$ lower bound for the On-Demand version of L_1 holds for L_0 as well.

Some CFGs, even under sparse inputs, do not admit a truly subcubic combinatorial algorithm. Abboud et. al [Abboud et al. 2018] constructed a (fairly complex) CFG for which parsing is not truly subcubic. Since parsing corresponds to running CFL reachability over a graph that is a path (and hence $m = n - 1$), this construction already finds a grammar with the desired lower bound. But as we showed in Section 3, Dyck-2 already achieves this lower bound.

We end this section with another undecidability result, which shows that we cannot even hope to determine for which programs the On-Demand problem can be evaluated in linear time.

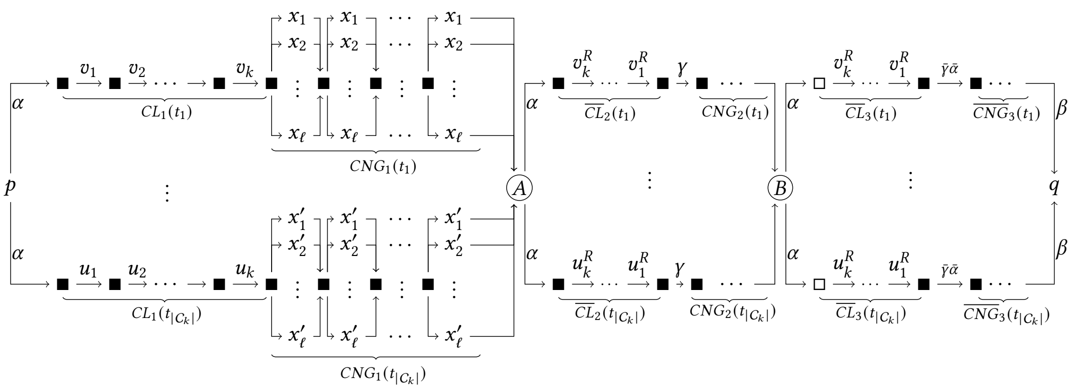


Fig. 2. Input graph constructed for Andersen's analysis.

Theorem 5.7. *Suppose the combinatorial k-Clique hypothesis holds. Then, for any constant $c \in [1,3)$, it is undecidable whether the On-Demand problem for a CFG can be evaluated by an $O(m^c)$ combinatorial algorithm.*

6 A LOWER BOUND FOR ANDERSEN'S POINTER ANALYSIS

In this section, we show a conditional lower bound for the On-Demand problem for Andersen's Pointer analysis (APA). Andersen's analysis is a fundamental points-to analysis that produces an over-approximation of the memory locations that each pointer may point-to at runtime. It will be convenient for our purposes to define APA as a Datalog program that computes the inverse points-to relation $T(x,y)$: this means that variable y points to variable x.

$$T(x,y) \text{ :- } \alpha(x,y). \tag{1}$$
$$T(x,y) \text{ :- } T(x,z), e(z,y). \tag{2}$$
$$T(w,y) \text{ :- } T(w,z), T(z,x), \beta(x,y). \tag{3}$$
$$T(w,z) \text{ :- } T(w,x), \gamma(x,y), T(z,y). \tag{4}$$

This is not a chain Datalog program, and hence APA cannot be expressed as a CFL reachability problem directly. However, following a technique from [Reps 1998], we can define $\bar{T}$ as the inverse of T, i.e. $\bar{T}(x,y) = T(y,x)$. With this, we can think of the program above as the following grammar:

$$T \leftarrow \alpha \mid Te \mid TT\beta \mid T\gamma\bar{T}.$$

In recent work [Mathiasen and Pavlogiannis 2021], it was shown that the On-Demand problem for APA has a $O(n^3)$ combinatorial lower bound under the combinatorial k-Clique hypothesis. We strengthen this result by showing that the same cubic lower bound holds even on sparse graphs.

Theorem 6.1. *Under the combinatorial k-Clique hypothesis, the On-Demand problem for APA cannot be solved by a combinatorial algorithm in time $O(m^{3-\epsilon})$ for any constant $\epsilon > 0$.*

In the remaining section, we provide a proof of the above theorem. The proof uses the same construction as the lower bound for Dyck-2. In particular, we will start with a graph G and attempt to find a $3k$-clique. The only difference is how the numbers are encoded. For the lower bound we will need only rules 1,3 and 4; hence we can ignore the label e in the constructed instance and we will ignore rule 2 completely. In other words, it suffices to consider the grammar $T \leftarrow \alpha \mid TT\beta \mid T\gamma\bar{T}$. It will be convenient to think that the labels of a path from u to v that is recognized by the program form a word with forward and backward edges. We will write a backward edge with label ℓ as a forward edge labeled $\bar{\ell}$. For example, if we have a path of the form $\alpha(u,w), \gamma(w,z), \alpha(v,z)$, we will think of it as $\alpha\gamma\bar{\alpha}$. We will also use the notation ℓ^i to denote i repetitions of the label ℓ.

Notation. We associate with each vertex an integer in $\{1,\dots,n\}$. As with the construction for Dyck-2, we will create two line graphs for a vertex v, $L(v)$ and $L^R(v)$. The line graph $L(v)$ has v edges labeled α followed by two edges with labels $\alpha\gamma$ (so it forms the string $\alpha^v\alpha\gamma$). The line graph $L^R(v)$ has two edges with labels $\bar{\gamma}\bar{\alpha}$ followed by v edges labeled β (so it forms the string $\bar{\gamma}\bar{\alpha}\beta^v$).

Graph Construction. The construction follows the one for Dyck-2, with the only difference that we need a few additional edges as shown in Figure 2. We now ask whether $T(p,q)$ is true or not.

Correctness. We show in Appendix C that the On-Demand problem on APA returns true if and only if there exists a $3k$-Clique in the constructed graph. The resulting graph has $O(n^{k+2})$ edges and can be constructed in the same amount of time. To obtain the desired bound, we then let k grow depending on the constant ϵ.

7 RELATED WORK

Static Program Analysis. The connection between CFL reachability and program analysis has been observed since a long time [Melski and Reps 2000; Reps 1998; Smaragdakis and Bravenboer 2010; Whaley et al. 2005]. Cubic time complexity is a common feature of algorithms proposed in several of these works. Prior work [Heintze and McAllester 1997] has also explained the sub-cubic barrier by showing that several data-flow reachability problems are 2NPDA-hard, a complexity class that does not admit sub-cubic algorithm for problems lying in this class. The related problem of *certifying* whether an instance of CFL reachability has a small and efficiently checkable certificate was studied in [Chistikov et al. 2022]. Their main result shows that succint certificates of size $O(n^2)$ can be checked in sub-cubic time using matrix multiplication.

Several variants of Andersen's analysis [Andersen 1994] have been developed over the years that incorporate different features such as flow and field-sensitivity [Hirzel et al. 2004; Lyde et al. 2015; Pearce et al. 2004; Whaley and Lam 2002]. However, the study of the precise complexity is a relatively recent effort. In this direction, the authors in [Mathiasen and Pavlogiannis 2021] explored the fine-grained complexity of Andersen's analysis. The complexity of the Dyck Reachability problem, which can be captured as a Context-Free grammar, has also been studied previously [Chatterjee et al. 2018]. Many program analysis tasks can also be expressed by Interleaved Dyck, which is the intersection of multiple Dyck languages based on an interleaving operator. Several recent works have also established the precise combined complexity [Li et al. 2020, 2021] and fine-grained complexity [Kjelstrøm and Pavlogiannis 2022] of this problem. Fine-grained complexity and parameterized complexity based lower bounds have also found great success for other related problems such as finding violations in concurrent programs and checking program consistency [Chini et al. 2017; Chini and Saivasan 2020] and safety verification [Chini et al. 2018, 2020]. We refer the reader to [Chini 2022] for more details on application of fine-grained complexity for program verification.

Datalog. CFL reachability is essentially equivalent to a class of Datalog programs called *chain Datalog programs*. The seminal work of Yannakakis [Yannakakis 1990] established a tight upper bound for evaluation of chain Datalog programs for regular and linear languages. This raises the question of when we can rewrite a non-linear Datalog program into a linear program, which has been studied extensively [Afrati et al. 1996, 2003; Afrati and Toni 1997; Dong 1992; Ullman and Van Gelder 1988]. When we restrict CFL reachability to regular languages, then the problem is equivalent to the evaluation of Regular Path Queries (RPQs) [Baeza 2013]. In particular, [Martens and Trautner 2018] studied the parameterized complexity of RPQ evaluation over graphs. Bagan et al. [Bagan et al. 2013] characterized the class of regular languages that are tractable for RPQs. More recently, [Casel and Schmid 2021] studied the fine-grained static and dynamic complexity of RPQ evaluation, enumeration, and counting problems. This is similar to our effort in this paper with the difference that we study the data complexity of a fixed RPQ.

Fine-grained Complexity. The area of fine-grained complexity attempts to prove the optimality of several well-known algorithms by constructing reductions to problems with widely believed lower bound conjectures. Such problems are BMM, the 3-SUM problem (and k-SUM more generally), all-pairs shortest paths, cycle detection, and finding orthogonal vectors [Williams 2018]. This line of work has conditionally proved cubic or quadratic lower bounds for sparse and dense variants of problems such as CFG parsing [Abboud et al. 2018], finding subgraphs in graphs [Williams and Xu 2020], variations of path-related problems [Lincoln et al. 2018], and dynamic problems [Abboud and Williams 2014]. The development of these conjectures has led to widespread activity in establishing lower bounds for problems such as join query processing [Berkholz et al. 2017, 2018; Carmeli and Kröll 2021; Keppeler 2020], concurrency analysis [Kulkarni et al. 2021; Mathur et al. 2020], and cryptography [Golovnev et al. 2020; LaVigne et al. 2019] to name a few.

8 CONCLUSION

In this work, we take the first step towards studying the fine-grained complexity landscape for the CFL reachability problem. We identify the precise polynomial running time (under widely believed lower bound conjectures) for several fundamental grammars. Despite the significant progress we made, there are many exciting questions that we have left open.

The running time for Dyck-1. Prior work has established that Dyck-1 has a combinatorial cubic lower bound w.r.t. n, but its running time w.r.t. m remains open. We were not able to find whether the running time is cubic, quadratic, or somewhere in between. In general, we do not even know whether there exists a grammar with running time $O(m^c)$ for some constant $c \in (2, 3)$.

The lower bound for Dyck-2. Even though we have established the complexity of Dyck-2 in the combinatorial setting, the general running time remains open. It is still possible that a cubic lower bound exists, but it is also an intriguing possibility that a truly sub-cubic algorithm that uses fast matrix multiplication exists.

Faster Algorithms for Restricted Inputs. Our algorithmic techniques are designed to work for worst-case inputs. However, restricting the instances we consider (e.g., instances with bounded treewidth) can potentially lead to faster algorithms. It may also possible to obtain better bounds if we take the output size into account.

ACKNOWLEDGMENTS

This research was supported by National Science Foundation under Grant No. IIS-1910014. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. 2018. If the Current Clique Algorithms Are Optimal, so Is Valiant's Parser. *SIAM J. Comput.* 47, 6 (2018), 2527–2555.

Amir Abboud and Virginia Vassilevska Williams. 2014. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *FOCS*. IEEE Computer Society, 434–443.

Foto Afrati, Manolis Gergatsoulis, and Maria Katzouraki. 1996. On transformations into linear database logic programs. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*. Springer, 433–444.

Foto Afrati, Manolis Gergatsoulis, and Francesca Toni. 2003. Linearisability on datalog programs. *Theoretical Computer Science* 308, 1-3 (2003), 199–226.

Foto Afrati and Francesca Toni. 1997. Chain queries expressible by linear datalog programs. In *Proc. of the 5th International Workshop on Deductive Databases and Logic Programming (DDLP'97)*. 49–58.

Noga Alon, Zvi Galil, and Oded Margalit. 1997. On the Exponent of the All Pairs Shortest Path Problem. *J. Comput. Syst. Sci.* 54, 2 (1997), 255–262.

Lars Ole Andersen. 1994. *Program analysis and specialization for the C programming language*. Ph. D. Dissertation. Citeseer.

Pablo Barceló Baeza. 2013. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, Richard Hull and Wenfei Fan (Eds.). ACM, 175–188. https://doi.org/10.1145/2463664.2465216

Guillaume Bagan, Angela Bonifati, and Benoît Groz. 2013. A trichotomy for regular simple path queries on graphs. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*. 261–272.

Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering conjunctive queries under updates. In *proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*. 303–318.

Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2018. Answering UCQs under Updates and in the Presence of Integrity Constraints. In *21st International Conference on Database Theory (ICDT 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Al Bessey, Ken Block, Benjamin Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles-Henri Gros, Asya Kamsky, Scott McPeak, and Dawson R. Engler. 2010. A few billion lines of code later: using static analysis to find bugs in the real world. *CACM* 53, 2 (2010), 66–75. https://doi.org/10.1145/1646353.1646374

Nofar Carmeli and Markus Kröll. 2021. On the Enumeration Complexity of Unions of Conjunctive Queries. *ACM Transactions on Database Systems (TODS)* 46, 2 (2021), 1–41.

Katrin Casel and Markus L. Schmid. 2021. Fine-Grained Complexity of Regular Path Queries. In *ICDT (LIPIcs, Vol. 186)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 19:1–19:20.

Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. 2018. Optimal Dyck reachability for data-dependence and alias analysis. *Proc. ACM Program. Lang.* 2, POPL (2018), 30:1–30:30.

Swarat Chaudhuri. 2008. Subcubic algorithms for recursive state machines. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, George C. Necula and Philip Wadler (Eds.). ACM, 159–169. https://doi.org/10.1145/1328438.1328460

Peter Chini. 2022. *Fine-Grained Complexity of Program Veri cation*. Ph. D. Dissertation.

Peter Chini, Jonathan Kolberg, Andreas Krebs, Roland Meyer, and Prakash Saivasan. 2017. On the Complexity of Bounded Context Switching. In *25th Annual European Symposium on Algorithms (ESA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Peter Chini, Roland Meyer, and Prakash Saivasan. 2018. Fine-Grained Complexity of Safety Verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 20–37.

Peter Chini, Roland Meyer, and Prakash Saivasan. 2020. Fine-grained complexity of safety verification. *Journal of Automated Reasoning* 64, 7 (2020), 1419–1444.

Peter Chini and Prakash Saivasan. 2020. A Framework for Consistency Algorithms. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Dmitry Chistikov, Rupak Majumdar, and Philipp Schepper. 2022. Subcubic certificates for CFL reachability. *Proceedings of the ACM on Programming Languages* 6, POPL (2022), 1–29.

Mihai Christodorescu and Somesh Jha. 2003. Static Analysis of Executables to Detect Malicious Patterns. In *USENIX Security*. USENIX Association.

Guozhu Dong. 1992. On datalog linearization of chain queries. In *Theoretical Studies in Computer Science*. Elsevier, 181–206.

Michael J Fischer and Albert R Meyer. 1971. Boolean matrix multiplication and transitive closure. In *12th annual symposium on switching and automata theory (swat 1971)*. IEEE, 129–131.

Anka Gajentaan and Mark H Overmars. 1995. On a class of $O(n^2)$ problems in computational geometry. *Computational geometry* 5, 3 (1995), 165–185.

Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan. 2020. Data structures meet cryptography: 3SUM with preprocessing. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 294–307.

Sheila A. Greibach. 1968. A Note on Undecidable Properties of Formal Languages. *Math. Syst. Theory* 2, 1 (1968), 1–6. https://doi.org/10.1007/BF01691341

Sheila A. Greibach. 1973. The Hardest Context-Free Language. *SIAM J. Comput.* 2, 4 (1973), 304–310.

Jakob Cetti Hansen, Adam Husted Kjelstrøm, and Andreas Pavlogiannis. 2021. Tight bounds for reachability problems on one-counter and pushdown systems. *Inform. Process. Lett.* 171 (2021), 106135.

Nevin Heintze and David McAllester. 1997. On the cubic bottleneck in subtyping and flow analysis. In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*. IEEE, 342–351.

Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. 2015. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 21–30.

Martin Hirzel, Amer Diwan, and Michael Hind. 2004. Pointer analysis in the presence of dynamic class loading. In *European Conference on Object-Oriented Programming*. Springer, 96–122.

Jens Keppeler. 2020. Answering Conjunctive Queries and FO+ MOD Queries under Updates. (2020).

Adam Husted Kjelstrøm and Andreas Pavlogiannis. 2022. The decidability and complexity of interleaved bidirected Dyck reachability. *Proceedings of the ACM on Programming Languages* 6, POPL (2022), 1–26.

Rucha Kulkarni, Umang Mathur, and Andreas Pavlogiannis. 2021. Dynamic Data-Race Detection Through the Fine-Grained Lens. In *32nd International Conference on Concurrency Theory*.

Rio LaVigne, Andrea Lincoln, and Virginia Vassilevska Williams. 2019. Public-key cryptography in the fine-grained setting. In *Annual International Cryptology Conference*. Springer, 605–635.

Yuanbo Li, Qirun Zhang, and Thomas Reps. 2020. Fast graph simplification for interleaved Dyck-reachability. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 780–793.

Yuanbo Li, Qirun Zhang, and Thomas Reps. 2021. On the complexity of bidirected interleaved Dyck-reachability. *Proceedings of the ACM on Programming Languages* 5, POPL (2021), 1–28.

Andrea Lincoln and Nikhil Vyas. 2020. Algorithms and Lower Bounds for Cycles and Walks: Small Space and Sparse Graphs. In *ITCS (LIPIcs, Vol. 151)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 11:1–11:17.

Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. 2018. Tight Hardness for Shortest Cycles and Paths in Sparse Graphs. In *SODA*. SIAM, 1236–1252.

V. Benjamin Livshits and Monica S. Lam. 2005. Finding Security Vulnerabilities in Java Applications with Static Analysis. In *USENIX Security*, Patrick McDaniel (Ed.). USENIX Association.

Steven Lyde, William E Byrd, and Matthew Might. 2015. Control-flow analysis of dynamic languages via pointer analysis. *ACM SIGPLAN Notices* 51, 2 (2015), 54–62.

Wim Martens and Tina Trautner. 2018. Evaluation and enumeration problems for regular path queries. In *21st International Conference on Database Theory (ICDT 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Anders Alnor Mathiasen and Andreas Pavlogiannis. 2021. The fine-grained and parallel complexity of andersen's pointer analysis. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–29.

Umang Mathur, Andreas Pavlogiannis, and Mahesh Viswanathan. 2020. The complexity of dynamic data race prediction. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. 713–727.

Jirí Matousek. 1991. Computing Dominances in E^n. *Inf. Process. Lett.* 38, 5 (1991), 277–278.

David Melski and Thomas Reps. 2000. Interconvertibility of a class of set constraints and context-free-language reachability. *Theoretical Computer Science* 248, 1-2 (2000), 29–98.

Oswaldo Olivo, Isil Dillig, and Calvin Lin. 2015. Static detection of asymptotic performance bugs in collection traversals. In *PLDI*, David Grove and Steve Blackburn (Eds.). ACM, 369–378. https://doi.org/10.1145/2737924.2737966

David J Pearce, Paul HJ Kelly, and Chris Hankin. 2004. Online cycle detection and difference propagation: Applications to pointer analysis. *Software Quality Journal* 12, 4 (2004), 311–337.

Jakob Rehof and Manuel Fähndrich. 2001. Type-base flow analysis: from polymorphic subtyping to CFL-reachability. In *POPL*. ACM, 54–66.

Thomas W. Reps. 1995. Shape Analysis as a Generalized Path Problem. In *PEPM*. ACM Press, 1–11.

Thomas W. Reps. 1998. Program analysis via graph reachability. *Inf. Softw. Technol.* 40, 11-12 (1998), 701–726.

Thomas W. Reps, Susan Horwitz, and Shmuel Sagiv. 1995. Precise Interprocedural Dataflow Analysis via Graph Reachability. In *POPL*. ACM Press, 49–61.

Philipp Johann Schepper. 2018. The Complexity of Formal Language Decision Problems. (2018).

Lei Shang, Xinwei Xie, and Jingling Xue. 2012. On-demand dynamic summary-based points-to analysis. In *CGO*. ACM, 264–274.

Yannis Smaragdakis and George Balatsouras. 2015. Pointer Analysis. *Found. Trends Program. Lang.* 2, 1 (2015), 1–69.

Yannis Smaragdakis and Martin Bravenboer. 2010. Using Datalog for Fast and Easy Program Analysis. In *Datalog (Lecture Notes in Computer Science, Vol. 6702)*. Springer, 245–251.

Volker Strassen et al. 1969. Gaussian elimination is not optimal. *Numerische mathematik* 13, 4 (1969), 354–356.

Jeffrey D Ullman and Allen Van Gelder. 1988. Parallel complexity of logical query programs. *Algorithmica* 3, 1 (1988), 5–42.

John Whaley, Dzintars Avots, Michael Carbin, and Monica S Lam. 2005. Using Datalog with binary decision diagrams for program analysis. In *Asian Symposium on Programming Languages and Systems*. Springer, 97–118.

John Whaley and Monica S Lam. 2002. An efficient inclusion-based points-to analysis for strictly-typed languages. In *International Static Analysis Symposium*. Springer, 180–195.

Virginia Vassilevska Williams. 2018. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*. World Scientific, 3447–3487.

Virginia Vassilevska Williams and R Ryan Williams. 2018. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)* 65, 5 (2018), 1–38.

Virginia Vassilevska Williams and Yinzhan Xu. 2020. Monochromatic Triangles, Triangle Listing and APSP. In *FOCS*. IEEE, 786–797.

Mihalis Yannakakis. 1990. Graph-Theoretic Methods in Database Theory. In *PODS*. ACM Press, 230–242.

Qirun Zhang. 2020. Conditional Lower Bound for Inclusion-Based Points-to Analysis. *arXiv preprint arXiv:2007.05569* (2020).

Xin Zheng and Radu Rugina. 2008. Demand-driven alias analysis for C. In *POPL*. ACM, 197–208.

A MISSING PROOFS

Theorem 5.1. *The On-Demand CFL reachability problem is BMM-hard for the (non-regular) CFGs that produce the following CFLs:*

(1) The language $\{a^i s b^i \mid i \geq 0\}$ where s can be any string, including the empty one;
(2) Strings over $\{a, b\}$ where the number of a's is equal to the b's;
(3) Palindrome strings of even (odd) length over an alphabet with at least 2 symbols.

Proof. For the language $\{a^i s b^i \mid i \geq 0\}$, we follow the same construction as Theorem 3.1, with the only difference that (i) each edge of the form (b_i, c_j) is replaced by a (fresh and unique) path of length $|s|$ with labels from s, and (ii) the on-demand pair is (a_1, a'_1). If $s = \emptyset$, we simply use $s = ab$.

The language with equal number of a's and b's is captured by the following CFG:

$$S \leftarrow aSbS \mid bSaS \mid ab \mid ba \mid aabb \mid abab \mid abba \mid baab \mid baba \mid bbaa$$

It is easy to see that the construction we used for Dyck-1 can be used for this CFG.

Finally, palindrome words of odd length are captured by the following CFG:

$$S \leftarrow \alpha_1 \mid \cdots \mid \alpha_k \mid \alpha_1 S \alpha_1 \mid \cdots \mid \alpha_k S \alpha_k$$

To do this construction, we keep the vertex set as the one in Theorem 3.1 plus one more vertex v, but the edge set becomes:

$$\begin{array}{ll}
\{(u, a_1), (a_1, a_2), (a_2, a_3), \ldots, (a_{n-1}, a_n)\} & \text{with label } \alpha_1 \\
\{(a_i, b_j) \mid (a_i, b_j) \in E\} & \text{with label } \alpha_2 \\
\{(b_i, c_j) \mid (b_i, c_j) \in E\} & \text{with label } \alpha_2 \\
\{(c_i, a'_j) \mid (c_i, a_j) \in E\} & \text{with label } \alpha_2 \\
\{(a'_n, a'_{n-1}), (a'_{n-1}, a'_{n-2}), \ldots, (a'_2, a'_1), (a'_1, v)\} & \text{with label } \alpha_1
\end{array}$$

Hence, every path from u to v will have labels that form a word of the form $\alpha_1 \ldots \alpha_1 \alpha_2 \alpha_2 \alpha_2 \alpha_1 \ldots \alpha_1$. It is easy to see that a triangle exists in G if and only if the word is a palindrome (in particular, the number of α_1 in the beginning and the end must be the same). For palindromes of even length the construction is similar. □

Theorem 5.4. *Under the combinatorial k-Clique hypothesis, the On-Demand problem for following cases cannot be solved by a combinatorial algorithm in time $O(m^{2-\epsilon})$ for any constant $\epsilon > 0$.*

(1) Dyck-k, for any $k \geq 1$;
(2) The language $\{a^i s b^i \mid i \geq 0\}$ where s can be any string, including the empty one;
(3) Strings over $\{a, b\}$ where the number of a's is equal to the b's;
(4) Palindrome strings of even (odd) length over an alphabet with at least 2 symbols.

Proof. We will use the fact that, under the combinatorial k-Clique hypothesis, for any constant $\epsilon > 0$, for any $k > 2/\epsilon$, finding whether a (k-partite) directed graph contains a k-cycle can be detected in time $\Omega(m^{2-\epsilon})$ [Lincoln and Vyas 2020].

We will prove the statement only for Dyck-1, since the construction is similar for the other cases. We apply the same construction as in the proof of Theorem 3.1, with the only difference that what connects the vertices in A and their copies in A' is now a path of length k (instead of a path of length 3, as in the case of triangles). Without any loss of generality, we can pick k to be an odd number. Then, the first $\lfloor k/2 \rfloor$ edges in the path will have label (, followed by $k - \lfloor k/2 \rfloor$ edges with label). It is easy to see that, following the same argument as before, (u, a'_1) is a correct pair if and only if the graph G has a k-cycle. Since the number of edges we added are $O(n) = O(m)$, the desired result follows. □

Lemma 5.5. *Suppose the CFG that corresponds to a CFL L admits an $O(m^c)$ algorithm for the All-Pairs (resp. On-Demand) problem for some constant $c \geq 1$. Then, the CFG that corresponds to $h^{-1}(L)$ also admits an $O(m^c)$ algorithm for the All-Pairs (resp. On-Demand) problem.*

Proof. Let $\mathcal{G}$ be the CFG for L, and $\mathcal{G}'$ be the CFG for $L' = h^{-1}(L) = \{w \mid h(w) \in L\}$. Consider an input graph G for $\mathcal{G}$. We construct an input G' for $\mathcal{G}'$ as follows.

For a symbol α in the alphabet of L, let $h(a) = \beta_1 \dots \beta_\ell$ where $\ell \geq 1$. Then, for any edge $(u, v) \in G$ with label α, we introduce in G the following path:

$$u \xrightarrow{\beta_1} w_1 \xrightarrow{\beta_2} w_2 \dots w_{\ell-1} \xrightarrow{\beta_\ell} v.$$

Here, $w_1, \dots, w_{\ell-1}$ are fresh distinct vertices. If $h(\alpha)$ is the empty word, we simply merge the nodes u, v in G'. The new instance G' has input size $O(m)$. We claim the following two statements:

Claim 1: if (u, v) is an output pair for $\mathcal{G}, G$ then (u, v) is an output pair for $\mathcal{G}', G'$. Indeed, assume that (u, v) is in the output of G. Then, there is a path from u to v in G with labels that form some word $w \in L$. From our construction, G' contains a path from u to v with labels that form the word $h(w)$. Hence, (u, v) is in the output of G'.

Claim 2: if (u, v) is an output pair for $\mathcal{G}', G'$ and u, v occur in G, then (u, v) is an output pair for $\mathcal{G}, G$. Indeed, assume that (u, v) is in the output of G'. Then, there is a path from u to v in G' with labels that form a word $w' \in h^{-1}(L)$. Since u, v are vertices in G, by our construction there must be a path in G from u to v with labels that form a word w such that $w' = h(w)$. But then, for w we have the property that $h(w) \in h^{-1}(L)$, and thus $w \in L$.

Now, the algorithm for the On-Demand problem with input u, v creates the instance G' and then runs $\mathcal{G}'$ on G' to check whether (u, v) is true. This has running time $O(m^c)$.

For the All-Pairs problem, we also create the instance G' and then run $\mathcal{G}'$ on G'. This takes time $O(m^c)$. Then, we need to filter out the outputs that have constants not in G, which takes time linear in the size of the output. Since the output is of size at most $O(m^c)$, the claim follows. □

B UNDECIDABILITY PROOFS

In this section, we prove the results on the undecidability of classifying CFGs in terms of their data complexity using Greibach's theorem. All proofs use the same technique with small variations.

Theorem 4.13. *Suppose the combinatorial k-Clique hypothesis holds. Then, for any constant $c \in [2, 3)$, it is undecidable whether $\mathsf{CFL}^{\mathsf{ap}}(\mathcal{G})$ can be evaluated by a combinatorial algorithm that runs in time $O(m^c)$.*

Proof. Fix a constant $c \in [2, 3)$. To prove undecidability, we apply Greibach's theorem. Consider the following property C for a CFL: any CFG that produces the language can be evaluated in time $O(m^c)$ by a combinatorial algorithm. As we have seen, C is satisfied by all regular languages (since every regular language can be produced by a linear CFG, and a linear CFG can be evaluated in time $O(m^2) = O(m^c)$). It is also non-trivial, since under the combinatorial k-Clique hypothesis, Dyck-k cannot be evaluated in time $O(m^{3-\epsilon})$ for any constant $\epsilon > 0$ by a combinatorial algorithm, hence it does not admit an $O(m^c)$ combinatorial algorithm. It remains to show that C is closed under the right quotient by a single symbol.

Indeed, take a CFL $\mathcal{L}$ and a corresponding CFG $\mathcal{G}$. Consider the language $\mathcal{L}/\alpha$ for a single symbol α. We now want to evaluate the CFG $\mathcal{G}_\alpha$ that corresponds to the language $\mathcal{L}/\alpha$. To do this, we extend the input graph G ($m = |E|$) as follows: for each possible vertex v, we add an edge (v, t_v) with label α, where t_v is a fresh distinct vertex. Let G' be the resulting graph. Note that $|E'| = |E| + n = O(m)$. Then, we run the algorithm for $\mathcal{G}$ on the new instance G', which runs in time $O(m^c)$. Finally, we can see that by construction, (u, v) is an output pair for $\mathcal{G}_\alpha$ if and only if (u, t_v) is an output tuple for $\mathcal{G}$. Hence, to obtain the output for $\mathcal{G}_\alpha$ it remains to do the following: for every pair of the form (u, t_v), output (u, v). This can be done in time $O(n^2) = O(m^2)$ by iterating over all output pairs. □

Theorem 5.3. *Suppose the combinatorial BMM hypothesis holds. Then, for any constant $c \in [2, 3)$, it is undecidable whether the On-Demand CFL reachability problem for a given CFG can be evaluated by an $O(n^c)$ combinatorial algorithm.*

Proof. Fix a constant $c \in [2, 3)$. To prove undecidability, we apply again Greibach's theorem. Consider the following property C for a CFL: the On-Demand problem for any CFG that produces the language can be evaluated in time $O(n^c)$ by a combinatorial algorithm. As we have seen, C is satisfied by all regular languages. It is also non-trivial, since under the combinatorial BMM hypothesis, $\{a^i b^i \mid i \geq 0\}$ cannot be evaluated in time $O(n^{3-\epsilon})$ for any constant $\epsilon > 0$ by a combinatorial algorithm, hence it does not admit an $O(n^c)$ combinatorial algorithm. It remains to show that C is closed under the right quotient by a single symbol.

Indeed, take a CFL $\mathcal{L}$ and a corresponding a corresponding CFG $\mathcal{G}$. Consider the language $\mathcal{L}/\alpha$ for a single symbol α. We now want to evaluate the On-Demand problem for a CFG $\mathcal{G}_\alpha$ that produces the language $\mathcal{L}/\alpha$. Let (s, t) be the input pair. To do this, we extend the input graph G as follows: we add an edge $\alpha(t, t')$ to the instance, where t' is a fresh distinct vertex. Let G' be the resulting instance. Then, we run the algorithm for $\mathcal{G}$ on the new instance G', which runs in time $O((n + 1)^c) = O(n^c)$. Finally, we can see that by construction, (s, t) is an output tuple for $\mathcal{G}_\alpha$ if and only if (s, t') is an output tuple for $\mathcal{G}$. □

Theorem 5.7. *Suppose the combinatorial k-Clique hypothesis holds. Then, for any constant $c \in [1, 3)$, it is undecidable whether the On-Demand problem for a CFG can be evaluated by an $O(m^c)$ combinatorial algorithm.*

Proof. Fix a constant $c \in [1, 3)$. To prove undecidability, we apply again Greibach's theorem. Consider the following property C for a CFL: the On-Demand problem for any CFG that produces the language can be evaluated in time $O(m^c)$ by a combinatorial algorithm. As we have seen, C is satisfied by all regular languages, since the on-demand problem can be evaluated in linear time. It is also non-trivial, since under the k-Clique hypothesis, Dyck-2 cannot be evaluated in time

$O(m^{3-\epsilon})$ for any constant $\epsilon > 0$ by a combinatorial algorithm, hence it does not admit an $O(m^c)$ combinatorial algorithm. It remains to show that C is closed under the right quotient by a single symbol.

Indeed, take a CFL $\mathcal{L}$ and a corresponding CFG $\mathcal{G}$. Consider the language $\mathcal{L}/\alpha$ for a single symbol α. We now want to evaluate the On-Demand problem for the CFG $\mathcal{G}_\alpha$ that produces the language $\mathcal{L}/\alpha$. Let (s,t) be the input pair. To do this, we extend the input instance I as follows: we add an edge $\alpha(t,t')$ to the instance, where t' is a fresh distinct value. Let G' be the resulting instance. Then, we run the algorithm for $\mathcal{G}$ on the new instance G', which runs in time $O((m+1)^c) = O(m^c)$. Finally, we can see that by construction, (s,t) is an output tuple for $\mathcal{G}_\alpha$ if and only if (s,t') is an output tuple for $\mathcal{G}$. □

C REMAINING PROOF FOR APA LOWER BOUND

We start with the following simple observations.

Proposition C.1. *Every valid word starts with α. Moreover, every word with length at least 2 ends with either β or $\bar{\alpha}$.*

Proposition C.2. *The following productions are valid:*

- $T \leftarrow \alpha\gamma T\bar{\gamma}\bar{\alpha}$
- $T \leftarrow T\gamma T\bar{\gamma}\bar{\alpha}$

Note that α is the only word (of length 1) that ends with a symbol that is not $\beta, \bar{\alpha}$.

Claim C.3. *If the On-Demand problem on Andersen's analysis returns true, then there exists a $3k$-clique in the input graph.*

Proof. Let w be the word that forms on the path from p to q. In particular, w is of the form:

$$\begin{aligned} w &= \alpha\{L(v_1)\dots L(v_k)\}\{L(w_1)\dots L(w_k)\}\alpha\{L^R(w'_k)\dots L^R(w'_1)\} \\ &\qquad \gamma\{L(z_1)\dots L(z_k)\}\alpha\{L^R(z'_k)\dots L^R(z'_1)\}\bar{\gamma}\bar{\alpha}\{L^R(v'_k)\dots L^R(v'_1)\}\beta \\ &= \alpha\{\alpha^{v_1}\alpha\gamma\dots\alpha^{v_k}\alpha\gamma\}\{\alpha^{w_1}\alpha\gamma\dots\alpha^{w_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{w'_k}\dots\bar{\gamma}\bar{\alpha}\beta^{w'_1}\} \\ &\qquad \gamma\{\alpha^{z_1}\alpha\gamma\dots\alpha^{z_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{z'_k}\dots\bar{\gamma}\bar{\alpha}\beta^{z'_1}\}\bar{\gamma}\bar{\alpha}\{\bar{\gamma}\bar{\alpha}\beta^{v'_k}\dots\bar{\gamma}\bar{\alpha}\beta^{v'_1}\}\beta \end{aligned}$$

First, note that w ends with β. This means that w was generated by the rule $T \leftarrow TT\beta$. From Proposition C.1 and our construction, the first T can only match the first α of this word. Thus, the following word is also valid:

$$\begin{aligned} &\{\alpha^{v_1}\alpha\gamma\dots\alpha^{v_k}\alpha\gamma\}\{\alpha^{w_1}\alpha\gamma\dots\alpha^{w_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{w'_k}\dots\bar{\gamma}\bar{\alpha}\beta^{w'_1}\} \\ &\qquad \gamma\{\alpha^{z_1}\alpha\gamma\dots\alpha^{z_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{z'_k}\dots\bar{\gamma}\bar{\alpha}\beta^{z'_1}\}\bar{\gamma}\bar{\alpha}\{\bar{\gamma}\bar{\alpha}\beta^{v'_k}\dots\bar{\gamma}\bar{\alpha}\beta^{v'_1}\} \end{aligned}$$

We repeat this process v'_1 more times. Observe that if $v'_1 > v_1$, the first T would not be able to match, since the word would start with γ, which is not valid. Hence, $v'_1 \leq v_1$. We are now left with the following word:

$$\begin{aligned} &\{\alpha^{v_1-v'_1}\alpha\gamma\dots\alpha^{v_k}\alpha\gamma\}\{\alpha^{w_1}\alpha\gamma\dots\alpha^{w_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{w'_k}\dots\bar{\gamma}\bar{\alpha}\beta^{w'_1}\} \\ &\qquad \gamma\{\alpha^{z_1}\alpha\gamma\dots\alpha^{z_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{z'_k}\dots\bar{\gamma}\bar{\alpha}\beta^{z'_1}\}\bar{\gamma}\bar{\alpha}\{\bar{\gamma}\bar{\alpha}\beta^{v'_k}\dots\bar{\gamma}\bar{\alpha}\} \qquad (5) \end{aligned}$$

Since the above word ends with $\bar{\alpha}$, it must have been generated by the rule $T \leftarrow T\gamma\bar{T}$. We will refer to the central γ in Equation 5 as $\gamma^\star$. First, note that no γ to the left of $\gamma^\star$ (other than the very first γ) can act as the separator for another application of the rule $T \leftarrow T\gamma\bar{T}$. This is because every γ is preceded by an α, which would lead T to end with an α and thus violates Proposition C.1. Similarly, no γ to the right of $\gamma^\star$ can also lead to valid parsing. We now argue that $\gamma^\star$ is also an invalid choice. Consider the following $\bar{T}$:

$$\bar{T} = \{\alpha^{z_1}\alpha\gamma \dots \alpha^{z_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{z'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{z'_1}\}\bar{\gamma}\bar{\alpha}\{\bar{\gamma}\bar{\alpha}\beta^{v'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{v'_1}\}$$

The inverse of this string can only be parsed by $T \leftarrow TT\beta$ (since there is no valid choice of γ as every γ is preceded by an α), and the parsing forces $z_1 = v'_1, z_2 = v'_2$ and so on. Eventually, we will need to parse T that starts with $\bar{\gamma}\bar{\alpha}$ which violates Proposition C.1. We have now established that the only choice of γ is the very first one in Equation 5 which implies that T is equal to $\alpha^{v_1 - v'_1}\alpha$. But this word is valid only if it is of length one, hence it must be that $v'_1 = v_1$. Now, we are left with the following word that is in $\bar{T}$:

$$\{\alpha^{v_2}\alpha\gamma \dots \alpha^{v_k}\alpha\gamma\}\{\alpha^{w_1}\alpha\gamma \dots \alpha^{w_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{w'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{w'_1}\}$$
$$\gamma\{\alpha^{z_1}\alpha\gamma \dots \alpha^{z_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{z'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{z'_1}\}\bar{\gamma}\bar{\alpha}\{\bar{\gamma}\bar{\alpha}\beta^{v'_k} \dots \bar{\gamma}\bar{\alpha}\}$$

Since the inverse of this word ends in $\bar{\gamma}\bar{\alpha}$, the only way to generate it is by $T \longrightarrow T\gamma\bar{T} \longrightarrow T\bar{\gamma}\bar{\alpha}$. This leaves us with the following word in T:

$$\{\alpha^{v_2}\alpha\gamma \dots \alpha^{v_k}\alpha\gamma\}\{\alpha^{w_1}\alpha\gamma \dots \alpha^{w_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{w'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{w'_1}\}$$
$$\gamma\{\alpha^{z_1}\alpha\gamma \dots \alpha^{z_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{z'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{z'_1}\}\bar{\gamma}\bar{\alpha}\{\bar{\gamma}\bar{\alpha}\beta^{v'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{v'_2}\}$$

We now repeat the same logic $(k-1)$ more times and obtain that $v_2 = v'_2, \dots, v_k = v'_k$. At this point, we are left with the following word that is in T:

$$\{\alpha^{w_1}\alpha\gamma \dots \alpha^{w_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{w'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{w'_1}\}\gamma\{\alpha^{z_1}\alpha\gamma \dots \alpha^{z_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{z'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{z'_1}\}\bar{\gamma}\bar{\alpha}$$

Since this word ends with $\bar{\alpha}$, it must be generated by the rule $T \leftarrow T\gamma\bar{T}$. The only valid way to do this production is that γ corresponds to $\gamma^\star$ (the central γ) by following the derivation shown above. This means that the following word is recognized by the grammar:

$$\{\alpha^{w_1}\alpha\gamma \dots \alpha^{w_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{w'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{w'_1}\}$$

For this case, we can use the same logic as above to show that $w_1 = w'_1, \dots, w_k = w'_k$. Also, we have the inverse of the following word is recognized by the grammar:

$$\{\alpha^{z_1}\alpha\gamma \dots \alpha^{z_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{z'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{z'_1}\}\bar{\gamma}\bar{\alpha}$$

For this to happen, the following word must be in T:

$$\{\alpha^{z_1}\alpha\gamma \dots \alpha^{z_k}\alpha\gamma\}\alpha\{\bar{\gamma}\bar{\alpha}\beta^{z'_k} \dots \bar{\gamma}\bar{\alpha}\beta^{z'_1}\}$$

This implies that $z_1 = z'_1, \dots, z_k = z'_k$. We have now established that there exist $v_1, \dots, v_k, w_1, \dots w_k, z_1, \dots z_k$ that satisfy the grammar. From the gadget construction, these vertices correspond to three k-cliques. Using the same argument from the hardness proof of Dyck-2 in conjunction with Observation 3.4, it is now straightforward to establish that the three k-cliques are also disjoint. □

Claim C.4. *If there exists a 3k-clique in the input graph, then the On-Demand problem on Andersen's analysis returns true.*

Proof. Let $t_1, t_2, t_3 \in C_k$ be three disjoint k-cliques. We will show that there exists a path from u to v that forms a valid word. Consider the path formed by the vertices $CL(t_1), CNG_2(t_2), \overline{CL}_2(t_2), CNG_2(t_3), \overline{CL}_3(t_3), \overline{CNG}_3(t_1)$. Let $t_1 = \{v_1, \dots, v_k\}$, $t_2 = \{w_1, \dots, w_k\}$ and $t_3 = \{z_1, \dots, z_k\}$. By applying the rule $T \leftarrow TT\beta$ $(v_1 + 1)$ times, we obtain $T^{v_1+1}T\beta^{v_1+1}$. Now, we apply $T \leftarrow \alpha$ to the first $(v_1 + 1)$ occurrences of T to obtain $\alpha\alpha^{v_1}T\beta^{v_1}\beta$. By Proposition C.2, we obtain $\alpha\alpha^{v_1}\alpha\gamma T\bar{\gamma}\bar{\alpha}\beta^{v_1}\beta$.

Since $L(v_1) = \alpha^{v_1}\alpha\gamma$ and $L^R(v_1) = \bar{\gamma}\bar{\alpha}\beta^{v}$, we have so far generated $\alpha L(v_1) T L^R(v_1)\beta$. We now repeat this process $k-1$ more times for T to generate the following word:

$$\alpha L(v_1)L(v_2)\dots L(v_k)TL^R(v_k)\dots L^R(v_2)L^R(v_1)\beta$$

Now, from Proposition C.2 we obtain:

$$\alpha L(v_1)L(v_2)\dots L(v_k)T\gamma T\bar{\gamma}\bar{\alpha}L^R(v_k)\dots L^R(v_2)L^R(v_1)\beta$$

Finally, we use the same construction as above for each of the two T's to generate the following final word:

$$\alpha\{L(v_1)\dots L(v_k)\}\{L(w_1)\dots L(w_k)\}\alpha\{L^R(w_k)\dots L^R(w_1)\}$$
$$\gamma\{L(z_1)\dots L(z_k)\}\alpha\{L^R(z_k)\dots L^R(z_1)\}\bar{\gamma}\bar{\alpha}\{L^R(v_k)\dots L^R(v_1)\}\beta$$

One can observe that this word matches the labels of the path we considered in the beginning. □

Received 2022-07-07; accepted 2022-11-07