Simple Reductions from Formula-SAT to Pattern Matching on Labeled Graphs and Subtree Isomorphism

Daniel Gibney∗  Gary Hoppenworth†  Sharma V. Thankachan‡

Abstract

The CNF formula satisfiability problem (CNF-SAT) has been reduced to many fundamental problems in \( \mathbb{P} \) to prove tight lower bounds under the Strong Exponential Time Hypothesis (SETH). Recently, the works of Abboud, Hansen, Vassilevska W. and Williams (STOC16), and later, Abboud and Bringmann (ICALP18) have proposed basing lower bounds on the hardness of general boolean formula satisfiability (Formula-SAT). Reductions from Formula-SAT have two advantages over the usual reductions from CNF-SAT: (1) conjectures on the hardness of Formula-SAT are arguably much more plausible than those of CNF-SAT, and (2) these reductions give consequences even for logarithmic improvements in a problem’s upper bounds.

Here we give tight reductions from Formula-SAT to two more problems: pattern matching on labeled graphs (PMLG) and subtree isomorphism. Previous reductions from Formula-SAT were to sequence alignment problems such as Edit Distance, LCS, and Frechet Distance and required some technical work. This paper uses ideas similar to those used previously, but in a decidedly simpler setting, helping to illustrate the most salient features of the underlying techniques.

∗Dept. of CS, University of Central Florida, Orlando, USA. e-mail: daniel.j.gibney@gmail.com
†Dept. of CS, University of Central Florida, Orlando, USA. e-mail: gary.hoppenworth@gmail.com
‡Dept. of CS, University of Central Florida, Orlando, USA. e-mail: sharma.thankachan@ucf.edu
1 Introduction and Related Work

The Strong Exponential Time Hypothesis (SETH) has proven to be a powerful tool in establishing conditional lower bounds for many problems with known polynomial-time solutions. However, recent work by Abboud, Hansen, Vassilevska W., and Williams [3], as well as Abboud andBringmann [2] has sought to use the hardness of general Formula-SAT problems as the basis for fine-grained conditional lower bounds, rather than CNF-SAT and SETH. Since general Formula-SAT contains within it all CNF-SAT instances, Formula-SAT is at least as hard as CNF-SAT. Additionally, when basing conditional lower bounds on Formula-SAT rather than CNF-SAT, the same algorithmic breakthroughs that previously would have violated SETH, now have far more remarkable consequences (see Section 1.2 for examples). This makes it plausible that conjectures based on the hardness of Formula-SAT are more likely to hold than those based on the hardness of CNF-SAT.

Aside from a plausible increase in the robustness of the conjectures, using Formula-SAT as a starting point has the advantage of allowing for tighter hardness results. Previous lower bounds based on SETH have been effective in establishing results of the form: an algorithm running in time $O(n^{c-\epsilon})$ for some $\epsilon > 0$, where the best-known solution has time complexity $\tilde{O}(n^c)$ would violate SETH. Despite this success, SETH has proven less effective at establishing tighter fine-grained hardness results regarding how many logarithmic-factors can be shaved. In fact, the impossibility of proving such a hardness result via fine-grained reductions from CNF-SAT was proven in [2]. Overcoming this by using Formula-SAT as a starting point, in [3] conditional lower bounds of this form were established for Edit Distance and Longest Common Subsequence (LCS). In [2], the results on LCS were further extended to show that an $O(n^2/\log^{1+\epsilon} n)$ time solution for LCS would imply major breakthroughs in circuit complexity. As a final example, work in [28] uses reductions from Formula-SAT to analyze which regular expression matching problems can have super-polylog factors shaved from their time complexity, and which cannot.

In this work, we will use Formula-SAT to establish hardness results similar to those listed above, but for two additional fundamental problems, Pattern Matching on Labeled Graphs (PMLG) and Subtree Isomorphism. We describe these problems next.

Pattern Matching On Labeled Graphs. (PMLG) Given an alphabet $\Sigma$, a labeled graph $G$ is a triplet $(V, E, L)$, where $(V, E)$ corresponds to the vertices and edges of a graph, and $L : V \rightarrow \Sigma^+$ is a function that defines a nonempty string (i.e., label) over $\Sigma$ to each vertex in $G$. For any string $S$, we use $S[..\ell]$ to denote its prefix ending at $\ell$ and $S[\ell..]$ to denote its suffix starting at $\ell$. We say that a pattern $P$ occurs in $G$ if there is a path $v_1, v_2, \ldots, v_m$ in $G$ such that $L(v_1)[\ell..] \circ L(v_2) \circ \cdots \circ L(v_m)[..\ell']$ equals $P$ for some $\ell, \ell'$. Given a labeled graph $G$ and a pattern $P$, the PMLG problem is to decide if there exists an occurrence of $P$ in $G$.

The PMLG problem began being intensely studied roughly thirty years ago in the context of alignment of strings (equivalent to approximate matching under edits, mismatches, etc.) in hypertext. This was initiated by Manber and Wu [19] and underwent several improvements [4, 5, 21, 22]. In the case where changes are allowed in the pattern, but not in the graph, the best-known algorithm runs in time $O(|V| + |E||P|)$, matching the time complexity of the dynamic programming solution of the exact problem, and is by Rautiainen and Marschall [24]. In the case where changes are allowed in the graph as well, the problem is NP-complete [5], even for binary alphabet [15]. The work by Equi et al. in [11] established the SETH based lower bounds for exact matching.

Subtree Isomorphism. Given two trees $T_1$ and $T_2$, is $T_1$ contained in $T_2$? This problem has
been the subject of extensive study [9, 17, 18, 25, 31, 33], much of this research dating back several decades. For general trees, both with at most \( n \) vertices, the currently best known solution has a time bound that is \( O(n^\omega) \), where \( \omega \) is the exponent on fast-matrix multiplication [31]; for rooted, constant maximum degree trees it is \( O(n^{2}/\log n) \) [17]; and, for ordered trees it is \( O(n \log n) \) [10]. Here we will be considering rooted trees with constant maximum degree. In terms of lower bounds, SETH based quadratic lower bounds for this version of the problem have been established in [1], even for binary rooted trees.

**Road Map.** We will first describe the Formula-SAT problem and deMorgan Formulas in more detail. Following this, we will state our results for PMLG and Subtree Isomorphism in terms of its implications for solving Formula-SAT, along with the resulting corollaries. Section 2 provides the reduction from Formula-SAT to PMLG. The reduction to Subtree Isomorphism is in Section 3. Finally, in Section 4 we discuss the similar themes and techniques that appear in both of these reductions.

### 1.1 Formula-SAT

**deMorgan Formulas.** For our purposes, we define a deMorgan formula over \( n \) Boolean input variables as a rooted binary tree where each leaf node represents an input variable or its negation, and every internal node represents a logical operator from the set \( \{\land, \lor\} \). Leaf nodes will be called input gates, and internal nodes will be called AND/OR gates. For a given bit assignment \( x \), we define \( F(x) \) as the binary value output at the root of \( F \) when the input bits are propagated from the leaves to the root of \( F \). The size of the formula, which we will denote as \( s \), is defined as the number of leaves in the tree.

**Problem 1 (Formula-SAT).** Given a deMorgan formula \( F \) of size \( s \) over \( n \) inputs, does there exist an input \( x \in \{0, 1\}^n \) such that \( F(x) = 1 \)?

The set of all Formula-SAT instances obviously contains within it all CNF-SAT instances. Unsurprisingly, due to its generality, it appears harder to derive efficient solutions for Formula-SAT. For CNF-SAT there exists ever-improving upper bounds [6, 12, 20, 23, 26, 29]. There also exists upper bounds for more general circuits such as ours, however, these work through restricting some parameter of the circuit, often some combination of the size, depth, and type of gates used within it (see for example [7, 13, 14, 27, 30, 32]).

### 1.2 Our Results

Our reduction will create an instance of PMLG (or Subtree Isomorphism) from a given instance of Formula-SAT. In doing so, we make explicit the roles that the size of the circuit \( s \) and the number of inputs \( n \) play in determining the size of the resulting instance.

**Theorem 1.** A Formula-SAT instance of size \( s \) on \( n \) inputs can be reduced to an instance of PMLG over a binary alphabet with a graph \( G = (V, E) \) and pattern \( P \) such that \( |P| \) is of size \( \mathcal{O}(2^{n/2} \cdot s) \) and \( |E| \) is of size \( \mathcal{O}(2^{n/2} \cdot s^2) \) in \( \mathcal{O}(|E|) \) time, where \( G \) is a DAG with maximum total degree\(^1\) three.

---

\(^1\)Total degree is in-degree plus out-degree.
Similarly, for Subtree Isomorphism we have the following theorem.

**Theorem 2.** A Formula-SAT instance of size $s$ on $n$ inputs can be reduced to an instance of Subtree Isomorphism on two binary trees $T_1$ and $T_2$, where the size of $T_1$ is $O(2^{n/2} \cdot s)$, and the size of $T_2$ is $O(2^{n/2} \cdot s^2)$ in $O(|T_2|)$ time.

Combining Theorems 1 and 2 with observations made by Abboud et al. in [3] (and restated in Appendix A), we obtain the following ‘breakthrough’ implications of a strongly subquadratic time algorithm for PMLG or Subtree Isomorphism. Proofs are deferred to Appendix A.

**Corollary 1.** The existence of a strongly subquadratic time algorithm for PMLG (or Subtree Isomorphism) would imply the class $\text{ENP}$ (1) does not have non-uniform $2^{o(n)}$-size Boolean formulas and (2) does not have non-uniform $o(n)$-depth circuits of bounded fan-in. It also implies that $\text{NTIME}[2^{O(n)}]$ is not in non-uniform $\text{NC}$.

The second corollary gives the consequences of being able to shave arbitrarily many logarithmic factors from the quadratic time complexity.

**Corollary 2.** If PMLG (or Subtree Isomorphism) can be solved in time $O\left(\frac{|E||P|}{\log |E|}\right)$ or $O\left(\frac{|E||P|}{\log |P|}\right)$ ($O\left(\frac{|T_1||T_2|}{\log |T_1|}\right)$ or $O\left(\frac{|T_1||T_2|}{\log |T_2|}\right)$ resp.) for all $c = \Theta(1)$, then $\text{NTIME}[2^{O(n)}]$ does not have non-uniform polynomial-size log-depth circuits.

In fact, we can give a particular constant $c$ for which shaving a $\log^c n$ factor would yield surprising new results in complexity theory. The following log-sensitive lower bounds leave a huge gap from the best known upper bounds; we present these corollaries purely for instructive purposes.

**Hardness of Shaving Log Factors.** We work under the Word-RAM model and limit the set of constant-time primitive operations to those operations which are robust to change in word size. Specifically, suppose we are given a word size of $w = \Theta(\log n)$ and an operation that can be performed in $O(1)$ time. We stipulate that we must be able to simulate this operation on words of size $W = \Theta(2^w)$ in $n^{1+o(1)}$. This is a reasonable assumption that is satisfied by many constant time operations such as addition, subtraction, multiplication, and division with remainder. See [2] for a detailed discussion.

The following hypothesis was suggested by Abboud and Bringmann in [2]. It reflects the fact that the best known algorithmic solutions to Formula-SAT fail to provide a time complexity better than the naïve solution on formulas of size $s = n^{3+\Omega(1)}$.

**Hypothesis 1 ([2]).** There is no algorithm that can solve SAT on deMorgan formulas of size $s = n^{3+\Omega(1)}$ in $O\left(\frac{n^c}{w^d}\right)$ time for some $c > 0$ in the Word-RAM model.

**Corollary 3.** Hypothesis 1 is false if PMLG (respectively Subtree Isomorphism) can be solved in time $O\left(\frac{|E||P|}{\log^{10^{a_1}+c}|E|}\right)$, (respectively $O\left(\frac{|T_1||T_2|}{\log^{10^{a_1}+c}|T_1|}\right)$ or $O\left(\frac{|T_1||T_2|}{\log^{10^{a_1}+c}|T_2|}\right)$) for any $\varepsilon > 0$.

**Proof.** We show the proof for PMLG; the proof for Subtree Isomorphism is identical. By Theorem 1 an $O\left(\frac{|E||P|}{\log^{10^{a_1}+c}|E|}\right)$ algorithm for PMLG can be converted to yield an algorithm running in $n^{1+o(1)}$.

---

2 As observed by Williams in [34], for deMorgan formulas of size $n^{3−o(1)}$ there exists a randomized $2^{n−n^{\Omega(1)}}$ time, zero error algorithm which can be obtained by applying results from [8] and [10].
\[
\frac{(2^{n/2}\cdot s^2)}{\log^{1+\epsilon}(2^{n/2}\cdot s^2)} = \mathcal{O}\left(\frac{2^{n/2}\cdot s^2}{n^{1+\epsilon}}\right) \text{time for Formula-SAT (note the } n^{1+o(1)} \text{ factor introduced when moving from a word size of } \Theta(n) \text{ to } \Theta(n)) \text{. If we choose } s = n^{3+\epsilon/6} \text{ then this yields an algorithm for Formula-SAT of time } \mathcal{O}\left(\frac{2^n}{n^{7/2}}\right) \text{, and Hypothesis } \text{[1]} \text{ is false.} \]

Again thanks to results highlighted by Abboud et al. in [3], we can also say the following about shaving a constant number of logarithmic factors from the quadratic time complexity. The proof is deferred to Appendix A.

**Corollary 4.** \( \mathsf{E}^{\mathsf{NP}} \) cannot be computed by non-uniform formulas of cubic size if \( \mathsf{PMLG} \) (respectively \( \mathsf{Subtree\ Isomorphism} \)) can be solved in time \( \mathcal{O}\left(\frac{|E||P|}{\log^{20+\epsilon} |E|}\right) \) or \( \mathcal{O}\left(\frac{|E||P|}{\log^{20+\epsilon} |P|}\right) \) (respectively \( \mathcal{O}\left(\frac{|T_1||T_2|}{\log^{20+\epsilon} |T_1|}\right) \) or \( \mathcal{O}\left(\frac{|T_1||T_2|}{\log^{20+\epsilon} |T_2|}\right) \)) for any \( \epsilon > 0 \).

The same hardness results for \( \mathsf{PMLG} \) apply for several more specific types of graphs (details will be presented in the full version of this paper). These include when the graph \( G \) is a deterministic DAG (at most one edge leaves a vertex with the same leading character on an edge label) of total degree at most 3, and the case when \( G \) is a directed or undirected planar graph of degree at most 3.

## 2. Reduction from Formula-SAT to PMLG

### 2.1 Technical Overview

Our reduction from Formula-SAT to PMLG uses an intermediate problem called Formula-Pair.

**Definition 1** (Formula-Pair). *Given a deMorgan Formula \( F = F(x_1, \ldots, x_m, y_1, \ldots, y_n) \) of size 2\( m \) where each input is used exactly once, and two sets \( A, B \subseteq \{0, 1\}^m \) each of size \( N \), does there exist \( a \in A \) and \( b \in B \) such that \( F(a, b) = F(a_1, \ldots, a_m, b_1, \ldots, b_n) = 1 \)?*

The role Formula-Pair plays in our reduction is analogous to the role of the Orthogonal Vectors Problem in many SETH reductions. It was proven in [2] that an instance of Formula-SAT on a formula of size \( s \) over \( n \) inputs can be reduced to an instance of Formula-Pair on two sets of size \( N = \mathcal{O}(2^{n/2}) \) and a formula of size \( \mathcal{O}(s) \) in linear time (in particular, they reduce from a harder problem they call \( \mathcal{F}_1 \text{-Formula-SAT} \)). Note that we may assume that \( F \) contains no input gates with negated binary variables, since if variable \( x_i \) is negated in \( F \), we can flip bit \( a_i \) for all \( a \in A \).

We begin our reduction from Formula-Pair to PMLG by considering a formula \( F \) and some input bit assignments \( a \in A \) and \( b \in B \). We then construct a pattern \( P \) and labeled graph \( G \) such that \( P \) occurs in \( G \) if and only if together \( a \) and \( b \) satisfy \( F \). In this step, we must ensure that our construction of \( P \) only relies on the input bit assignments of \( a \), and our construction of \( G \) only relies on the input bit assignments of \( b \). This allows us to create patterns \( P_1, P_2, \ldots, P_N \) corresponding to the \( N \) bit assignments in \( A \), and graphs \( G_1, G_2, \ldots, G_N \) corresponding to the \( N \) bit assignments in \( B \). Then we will have that \( P_i \) occurs in \( G_j \) if and only if \( F(a, b) = 1 \), where \( a \in A \) is the bit assignment corresponding to \( P_i \), and \( b \in B \) is the bit assignment corresponding to \( G_j \). Finally, we combine these patterns and graphs into a product pattern \( P \) and a product graph \( G \) such that \( P \) occurs in \( G \) if and only if some \( P_i \) occurs in some \( G_j \). This will complete the reduction.
2.2 Reduction

Given a deMorgan formula $F$ and a complete assignment of input bits $(a, b)$ where $a \in A$ and $b \in B$, we will construct a corresponding pattern $P$ and labeled DAG $G$ over alphabet $\{0, 1, \$\}$ such that $P$ occurs in $G$ if and only if the output of $F$ is 1 on input $(a, b)$. This pattern and graph will be built recursively, starting with the input gates as a base case. For a gate $g = (g_1 * g_2)$ where $* \in \{\lor, \land\}$, we will construct a corresponding pattern and graph for gate $g$ by merging the patterns and graphs of subgates $g_1$ and $g_2$. At each step in this process, the pattern corresponding to gate $g$ occurs in the graph corresponding to gate $g$ if and only if $g$ evaluates to 1 on input $(a, b)$.

**Invariants.** We will maintain the following invariants during this recursive procedure. Let $g$ be a gate of $F$ with height $h$, and let $P$ and $G$ be the pattern and graph corresponding to gate $g$ in our construction.

1. Graph $G$ will have a designated source vertex and sink vertex, both with label “1”. Every maximal path in $G$ will be of length $|P|$ and start and end at the source and sink vertices of $G$ respectively.

2. The construction of pattern $P$ is independent of the choice of bit assignment $b \in B$, and the construction of graph $G$ is independent of the choice of bit assignment $a \in A$.

3. Pattern $P$ occurs in $G$ if and only if $g$ has output 1 on input $(a, b)$.

Observe that by the first invariant, every occurrence of pattern $P$ in graph $G$ will start at the source vertex of $G$ and end at the sink vertex of $G$. If this is the case, we will say that $G$ matches $P$. We will also refer to the designated source and sink vertices of $G$ as the start and end vertices of $G$.

![Diagram](image.png)

Figure 1: From left to right: the graph constructed for gate $g = (g_1 \land g_2)$, the graph constructed for gate $g = (g_1 \lor g_2)$, and the Universal Subgraph $U(x)$. Note that Universal Subgraph $U(x)$ has a series of $x - 2$ vertex pairs labeled 0 and 1, so that its maximal path length is $x$. 

6
Input Gate. Each input gate $g$ in $F$ takes as input a binary variable $z$. We will design a graph $G$ and pattern $P$ such that $G$ matches $P$ if and only if $z$ had value 1 in bit assignment $(a, b)$, and hence $g$ evaluates to 1. Our construction depends on whether $z$ corresponds to an input bit in $a$ or $b$.

- **Case 1. $z$ corresponds to some $a_i \in a$.** We let $P := 1a_11$ and $G$ be a path of length three with all vertices labeled 1.

- **Case 2. $z$ corresponds to some $b_i \in b$.** We let $P := 111$ and $G$ be a path of length three with the first and last vertex labeled 1 and the middle vertex labeled $b_i$.

The start vertex of $G$ will be the first vertex in the path, and the end vertex of $G$ will be the third (last) vertex in the path. Then our graph $G$ matches pattern $P$ if and only if $z = 1$ and thus the input gate evaluates to true. Additionally, the construction of $P$ does not depend on $b$ and the construction of $G$ does not depend on $a$. All invariants are satisfied.

**AND Gate.** Given a gate $g = (g_1 \land g_2)$ and the graphs and patterns $(G_1, P_1)$ and $(G_2, P_2)$ corresponding to gates $g_1$ and $g_2$ respectively, we must construct a product graph $G$ and pattern $P$ such that $G$ matches $P$ if and only if $G_1$ matches $P_1$ and $G_2$ matches $P_2$. This is done rather easily. Let $P := 1P_1P_21$. Now let our product graph $G$ be defined as in Figure 1.a. Our start vertex is labeled 1 and has an outgoing edge to the start vertex of subgraph $G_1$. The end vertex of $G_1$ in turn has an outgoing edge to start vertex of subgraph $G_2$, whose own end vertex has an outgoing edge to the final vertex of $G$. We now verify all invariants are satisfied.

- **Invariant 1.** We assume that every maximal path in $G_1$ (respectively $G_2$) is of length $|P_1|$ (respectively $|P_2|$). Then by the construction of $P$ and $G$, every maximal path in $G$ is of length $|P|$. The invariant is maintained.

- **Invariant 2.** Assuming that the construction of $P_1$ and $P_2$ is independent of $b$, and the construct of $G_1$ and $G_2$ is independent of $a$, it follows that the construction of pattern $P$ is independent of bit assignment $b$, and the construction of graph $G$ is independent of bit assignment $a$.

- **Invariant 3.** Since every occurrence of $P$ in $G$ starts at the start vertex of $G$ and ends at the end vertex, we must conclude that $P$ occurs in $G$ if and only if $P_1$ occurs in $G_1$ and $P_2$ occurs in $G_2$. Then by our invariant $P$ occurs in $G$ if and only if $g$ evaluates to 1 on input $(a, b)$. The invariant is preserved.

**OR Gate.** Given a gate $g = (g_1 \lor g_2)$ and the graphs and patterns $(G_1, P_1)$ and $(G_2, P_2)$ corresponding to gates $g_1$ and $g_2$ respectively, we must construct a product graph $G$ and pattern $P$ such that $G$ matches $P$ if and only if $G_1$ matches $P_1$ or $G_2$ matches $P_2$. As with our AND gate, we let $P := 1P_1P_21$. Our product graph $G$ (see Figure 1b) splits into two branches. One branch checks if $G_1$ matches $P_1$ and ignores $P_2$, while the other branch checks if $G_2$ matches $P_2$ and ignores $P_1$. We are able to ignore $P_2$ (respectively $P_1$) by constructing a ‘universal’ subgraph that matches all binary strings that start and end with 1 and are of length $|P_2|$ (respectively $|P_1|$). We let $U(x)$ denote the universal subgraph for length $x$, and we depict our construction of $U(x)$ in Figure 1c. Observe that graphs $U(|P_1|)$ and $U(|P_2|)$ match $P_1$ and $P_2$ respectively. We now check that all invariants are satisfied.
• **Invariant 1.** A similar argument as in the AND gate shows that every maximal path in $G$ is of length $|P|$ and passes through the start and end vertices of $G$. The invariant is preserved.

• **Invariant 2.** Pattern $P$ is independent of bit assignment $b$ by a similar argument as with the AND gate construction. However, for our graph $G$, we must verify that subgraphs $U(|P_1|)$ and $U(|P_2|)$ of $G$ do not depend on bit assignment $a$. This will follow from proving that the lengths of patterns $P_1$ and $P_2$ do not depend on the bit assignment $a$. Note that in each of the input, AND, and OR gate constructions, the length of the constructed pattern is the same regardless of the bit assignment $a$. Thus we conclude that $U(|P_1|)$ and $U(|P_2|)$ are independent of the bit assignment $a$, and therefore the construction of graph $G$ is independent of the bit assignment $a$.

• **Invariant 3.** Since every occurrence of pattern $P$ starts at the start vertex of $G$ and ends at the end vertex, it is immediate that $G$ matches $P$ if and only if $G_1$ matches $P_1$ or $G_2$ matches $P_2$. It immediately follows from our invariant that $G$ matches $P$ if and only if gate $g = (g_1 \lor g_2)$ evaluates to 1 on input $(a, b)$.

### 2.3 Completing the Reduction

Now corresponding to our formula $F$ of size $s$ and a complete assignment of input bits $(a, b)$, we can build a pattern $P$ and a graph $G$ such that $G$ matches $P$ if and only if assignment $(a, b)$ satisfies $F$. Note that we only add a constant number of symbols to our pattern $P$ for each gate in $F$, and there are fewer than $2s$ gates in $F$, so $|P| = \mathcal{O}(s)$. On the other hand, each OR gate in $F$ can contribute $\mathcal{O}(|P|)$ vertices and edges to our final graph $G$. It follows that $G$ is of size $\mathcal{O}(s^2)$.

Using our construction, for every $a \in A$ we may construct a corresponding pattern $P_a$ and for every $b \in B$ we may construct a corresponding graph $G_b$. We will denote these patterns and graphs by $P_1, P_2, \ldots, P_N$ and $G_1, G_2, \ldots, G_N$ respectively. Note that each pattern $P_j$ makes no assumptions on the bit assignment $b$, and graph $G_i$ makes no assumptions on the bit assignment $a$. It follows that $G_i$ matches $P_j$ if and only if together the corresponding bit assignments $a \in A$ and $b \in B$ satisfy $F$.

Next, we construct a final graph $G$ and pattern $P$ such that $P$ occurs in $G$ if and only if some $G_i$ matches some $P_j$. This will complete our reduction. We define our final pattern $P$ as follows: $P := $$$P_1$$P_2$$\cdots$$P_N$$\$$$. The structure of our final graph $G$ is similar to the final graph presented in [11]. We present this graph in Figure 2 and briefly explain the intuition behind it. Let $\mu = |P_i|$ for any $i$. Then subgraph $U(\mu)$ will match any subpattern $P_i$ in $P$. The graph $G$ uses $U(\mu)$ to match the subpatterns $P_i$ in $P$ that do not match with any $G_j$. Note that since pattern $P$ has a prefix of two $\$$ symbols and a suffix of two $\$$ symbols, $P$ is forced to pass through the second row of $G$. More specifically, the first row of $G$ alone cannot match the $\$$ suffix of $P$, and the third row of $G$ alone cannot match the $\$$ prefix of $P$. Then it can be seen that $P$ occurs in $G$ only if $P$ passes through the second row of $G$, and hence some subgraph $G_i$ matches some subpattern $P_j$. Then by construction, $P$ occurs in $G$ if and only if there exists $a \in A$ and $b \in B$ such that $F(a, b) = 1$. Furthermore, our final graph is a DAG of size $\mathcal{O}(N \cdot s^2)$ and our final pattern $P$ is of length $\mathcal{O}(N \cdot s)$. This completes our reduction from Formula-SAT to PMLG on DAGs.
3 Reduction from Formula-SAT to Subtree Isomorphism

3.1 Technical Overview

We begin our reduction from Formula-Pair to Subtree Isomorphism by considering a formula $F$ and some input bit assignments $a \in A$ and $b \in B$. We then construct trees $T_a$ and $T_b$ such that $T_a$ is contained in $T_b$ if and only if together $a$ and $b$ satisfy $F$. In this step it is important that we ensure that our construction of $T_a$ only relies on the input bit assignments of $a$, and our construction of $T_b$ only relies on the input bit assignments of $b$. This allows us to create $N_T_a$ trees corresponding to the $N$ bit assignments $a$ in $A$, and $N_T_b$ trees corresponding to the $N$ bit assignments $b$ in $B$. Then we will have that some $T_a$ tree is contained in some $T_b$ tree if and only if the corresponding bit assignments $a \in A$ and $b \in B$ satisfy $F(a, b) = 1$. Finally, we combine these trees into two final trees $T_A$ and $T_B$ such that $T_A$ is contained in $T_B$ if and only if some $T_a$ is contained in some $T_b$. This will complete the reduction.

3.2 Reduction

Given a deMorgan formula $F$ and a complete assignment of input bits $(a, b)$ where $a \in A$ and $b \in B$, we will construct the corresponding rooted trees $T_a$ and $T_b$ such that $T_a$ is contained in $T_b$ if and only if the output of $F(a, b) = 1$. These trees will be constructed recursively, starting with the input gates of $F$ as a base case. For a gate $g = (g_1 \ast g_2)$ where $\ast \in \{\lor, \land\}$, we will construct the corresponding trees $T_{g_a}$ and $T_{g_b}$ for gate $g$ by merging the trees of subgates $g_1$ and $g_2$. At each step in this process, $T_{g_a}$ will be contained in $T_{g_b}$ if and only if gate $g$ has output 1 on input $(a, b)$.

**Invariants.** We will maintain the following invariants throughout our construction. Let $g$ be a gate of $F$ with height $h$.

1. The height of $T_{g_a}$ is equal to the height of $T_{g_b}$ and is at most $4h$. 

Figure 2: Our final graph $G$. Here $\mu = |P_i|$. 
2. The construction of $T_a^g$ is independent of the choice of bit assignment $b \in B$, and the construction of $T_b^g$ is independent of the choice of bit assignment $a \in A$.

3. Tree $T_a^g$ is contained in tree $T_b^g$ if and only if gate $g$ has output 1 on input $(a, b)$.

![Figure 3: The trees $T_a^g$ and $T_b^g$ corresponding to input gate $g = a_i$ or $g = b_j$.](image)

![Figure 4: The trees $T_a^g$ (top) and $T_b^g$ (bottom) corresponding to AND gate $g = (g_1 \land g_2)$.](image)

**Input Gate.** Given an input gate $g$ corresponding to a bit value $a_i \in a$ (respectively, a bit value $b_j \in b$), we will construct trees $T_a^g$ and $T_b^g$ so that $T_a^g$ is contained in $T_b^g$ if and only if $a_i = 1$ (respectively, $b_j = 1$). We construct $T_a^g$ and $T_b^g$ as in Figure 3. These trees are rooted at vertices $v_a$ and $v_b$ respectively. We define input gates of $F$ to have a height of one, so the trees in Figure 3 satisfy the first invariant. The remaining two invariants can be verified by examining every case of Figure 3.

**AND Gate.** Given an input gate $g = (g_1 \land g_2)$, and the trees $T_a^1, T_b^1$ and $T_a^2, T_b^2$ corresponding to gates $g_1$ and $g_2$ respectively, we wish to construct trees $T_a^g$ and $T_b^g$ so that $T_a^g$ is contained in $T_b^g$ if and only if gate $g$ has output 1 on input $(a, b)$. By our third invariant it suffices to ensure that $T_a^g$ is contained in $T_b^g$ if and only if $T_a^1$ is contained in $T_b^1$ AND $T_a^2$ is contained in $T_b^2$. We construct trees $T_a^g$ and $T_b^g$ as in Figure 4. The trees are rooted at vertices $v_a^0$ and $v_b^0$ respectively. We now verify that all invariants are satisfied.
we call a universal subtree. We design $U_g$. These trees are rooted at vertices

g gates only if $T_a$ and $T_b$ have height at most $4(h-1)$, and so trees $T_a^g$ and $T_b^g$ have height at most $4(h-1) + 4 = 4h$.

**Invariant 2.** We assume that the construction of trees $T_a^1$ and $T_a^2$ is independent of $b$, and the trees $T_b^1$ and $T_b^2$ are independent of $a$. Then it can be easily verified that tree $T_a^g$ does not depend on $b$, and tree $T_b^g$ does not depend on $a$.

**Invariant 3.** We must show that tree $T_a^g$ is contained in tree $T_b^g$ if and only if $g$ evaluates to 1 on bit assignment $(a, b)$. By our inductive hypothesis, it suffices to show that $T_a^g$ is contained in $T_b^g$ if and only if $T_a^1$ is contained in $T_b^1$ AND $T_a^2$ is contained in $T_b^2$. The ‘if’ direction is immediate from our construction: just map vertex $v_a^i$ in $T_a^g$ to vertex $v_b^i$ in $T_b^g$ for $i \in [0, 4]$, and map trees $T_a^1$ and $T_a^2$ to subtrees of $T_b^1$ and $T_b^2$ respectively.

For the ‘only if’ direction we must prove that subtree $T_a^1$ can only map to a subtree of $T_b^1$, and subtree $T_a^2$ can only map to a subtree of $T_b^2$. First note that since trees $T_a^g$ and $T_b^g$ have the same height, every isomorphism between $T_a^g$ and a subtree $T_b^g$ must map the root vertex $v_a^0$ of $T_a^g$ to the root vertex $v_b^0$ of $T_b^g$. Now suppose $T_a^1$ is mapped to $T_b^2$ in some isomorphism between $T_a^g$ and a subtree of $T_b^g$. Then vertex $v_a^3$ would be mapped to vertex $v_b^1$, and the path of length two hanging off $v_a^3$ would have nowhere to map to. It immediately follows that in every valid subtree isomorphism, $T_a^1$ is mapped to $T_b^1$, and $T_a^2$ is mapped to $T_b^2$. Then $T_a^g$ is contained in $T_b^g$ if and only if $T_a^1$ is contained in $T_b^1$ and $T_a^2$ is contained in $T_b^2$.

**OR Gate.** Given an input gate $g = (g_1 \lor g_2)$, and the trees $T_a^1, T_b^1$ and $T_a^2, T_b^2$ corresponding to gates $g_1$ and $g_2$ respectively, we will construct trees $T_a^g$ and $T_b^g$ so that $T_a^g$ is contained in $T_b^g$ if and only if $T_a^1$ is contained in $T_b^1$ OR $T_a^2$ is contained in $T_b^2$. We construct trees $T_a^g$ and $T_b^g$ as in Figure 5. These trees are rooted at vertices $v_a^0$ and $v_b^0$ respectively. Tree $T_b^g$ contains a subtree $U_g$, which we call a universal subtree. We design $U_g$ so that it contains both tree $T_a^1$ and tree $T_a^2$ for every bit assignment $a$. This will allow either $T_a^1$ or $T_a^2$ to match with $U_g$, thus achieving the OR gate logic.

We now construct our universal subtree $U_g$. First, observe that for any gate $g$ and any two bit assignments $a, a' \in A$, the only difference between trees $T_a^g$ and $T_a^g$ is in the input gate subtrees.

![Figure 5: The trees $T_a^g$ (left) and $T_b^g$ (right) corresponding to OR gate $g = (g_1 \lor g_2)$.](image-url)
There are two different input gate subtrees in $T^a_g$: the $a_i = 0$ subtree composed of a root vertex and two leaves, and the $a_i = 1$ subtree composed of a root vertex with a single leaf (see Figure 3). Note that the $a_i = 0$ input subtree contains the $a_i = 1$ input subtree. Then if we define a bit assignment $u = 0^m$, it follows that for every $a \in A$, the tree $T^a_g$ is contained within the tree $T^u_g$. Then for trees $T^1_a$ and $T^2_a$ we construct trees $T^1_g$ and $T^2_g$ so that $T^1_a$ is contained in $T^1_u$ and $T^2_a$ is contained in $T^2_u$ for all $a \in A$. We define our universal subtree $U_g$ as the tree created by merging the root vertex of $T^1_u$ with the root vertex of $T^2_u$. By construction, this tree $U_g$ contains $T^1_a$ and $T^2_a$ for all $a \in A$ as intended. We now verify that all invariants are satisfied.

- **Invariant 1.** This invariant holds by an argument identical to that of the AND gate construction.

- **Invariant 2.** A similar argument as with the AND gate will show that $T^a_g$ does not depend on bit assignment $b$. Likewise, tree $T^g_b$ does not depend on bit assignment $a$; the construction of universal subtree $U_g$ is independent of $a$ as detailed in its construction.

- **Invariant 3.** By our inductive hypothesis, it suffices to show that $T^a_g$ is contained in $T^b_g$ if and only if $T^1_a$ is contained in $T^1_b$ OR $T^2_a$ is contained in $T^2_b$. The ‘if’ direction can be seen by observing that if $T^1_a$ is contained in $T^1_b$, then we can align $T^1_a$ with $T^1_b$ and align $T^2_a$ with $U_g$, which is guaranteed to contain $T^2_a$; the case where $T^2_a$ is contained in $T^1_b$ is identical.

The ‘only if’ direction follows from a similar argument given for the AND construction. First note that since trees $T^a_g$ and $T^b_g$ have the same height, every subtree isomorphism must map the root vertex $v^a_0$ of $T^a_g$ to the root vertex $v^b_0$ of $T^b_g$. Additionally, it is immediate from construction that exactly one subtree $T^1_a$ or $T^2_a$ can be aligned with universal subtree $U_g$. Then we simply need to verify that there is no valid subtree isomorphism between $T^a_g$ and $T^b_g$ that maps $T^1_a$ to $T^2_b$ or $T^2_a$ to $T^1_b$. Suppose that $T^1_a$ was mapped to a subtree of $T^2_b$ (the other case is symmetric). Then vertex $v^a_3$ would map to vertex $v^b_5$, and the path of length two hanging off $v^a_3$ would have nowhere to map to. We conclude that subtree $T^1_a$ must map to subtree $T^1_b$ or subtree $T^2_a$ must map to subtree $T^2_b$ in any subtree isomorphism from $T^a_g$ to $T^b_g$. The invariant is maintained.
3.3 Completing the Reduction

The final trees are constructed using the technique provided in [1]. The construction is shown in Figure 6 and described next.

- For the final tree $T_A$, start with a complete binary tree where the number of leaves is the smallest power of 2 that is greater or equal to $N$, say $2^x$. From each of the $2^x$ leaves, attach a path of length $x$. Let the first $N$ leaves at the ends of these paths be numbered 1 to $N$. For $1 \leq i \leq N$, replace leaf $i$ with root of $T_{a_i}$. For the remaining $2^x - N$ leaves at the end of paths, replace the leaf with the roots of $2^x - N$ copies of $T_{a_{N}}$.

- For the final tree $T_B$, again start with a complete binary tree with $2^x$ leaves. From the first $2^x - 1$ leaves, attach a path of length $x$. Replace the end of each of the paths with the root of a universal tree $U$, which is $T_u$ with input bit assignment $u = 0^m$. From the remaining leaf in the complete binary tree, replace this leaf with the root of another complete binary tree, again with $2^x$ leaves. Let the first $N$ leaves of this second complete binary tree be numbered 1 to $N$. For $1 \leq i \leq N$, replace leaf $i$ with the root of $T_{b_i}$.

To see why this works, consider that for $T_A$ to be isomorphic to a subtree of $T_B$, the root of $T_A$ must be mapped onto the root of $T_B$. Then, one of $T_A$'s $2^x$ paths hanging from the leaves of its complete binary tree must traverse down the lower complete binary tree in $T_B$. From here, a subtree rooted at the end of one of these paths in $T_A$ must have to be isomorphic to one of the subtrees hanging from the leaves of the second binary tree in $T_B$. This is possible if and only if for some $a \in A$ and $b \in B$ we have that $T_a$ is isomorphic to a subtree of $T_b$. By the invariants proven above, such a pair $a \in A$ and $b \in B$ exists iff the starting formula $F$ evaluates to true on the assignment $(a, b)$.

The final tree $T_A$ is of size $O(Ns)$. This is because there are $N$ trees $T_a$ in $T_A$, and each tree $T_a$ is of size $O(s)$. The upper bound on the size of $T_a$ follows from the fact that formula $F$ has $s$ gates, and each gate contributes constantly many vertices to $T_a$. The final tree $T_B$ is of size $O(Ns^2)$. To see this, fix a particular assignment $(a, b)$, and consider the tree $T_b$. Each AND gate contributes a constant number of vertices to $T_b$. Each OR gate appends a universal subtree $U$ of size at most the size of $T_a$ to $T_b$. Since the size of $T_a$ is $O(s)$ and there are $s$ gates in formula $F$, we have that $T_b$ is of size $O(s^2)$.

4 Discussion

The key property highlighted by the two reductions is that both problems we reduced to allow for the construction of two independent objects $O_A$ and $O_B$, where $O_A$ is constructed independently from the partial input assignments in $B$, and $O_B$ is constructed independently from the partial input assignments in $A$.

In order to construct these objects, both reductions start by fixing an input assignment $(a, b)$. Then, two new objects for each gate $g$ are constructed using the objects for the circuits that are input into $g$. The aim of this construction is to maintain the invariant that whichever desired property we want our objects to have (e.g., the pattern occurring in a graph, or having an isomorphic subtree) holds iff $(a, b)$ satisfy the circuit with output gate $g$. This is accomplished by supposing (i) we are adding the gate $g = g_1 * g_2$ where $* \in \{\wedge, \vee\}$, (ii) the objects $O_a^{g_1}$ and $O_b^{g_1}$ have the desired property iff $(a, b)$ evaluates to true on the circuit with output gate $g_1$, and (iii) the objects $O_a^{g_2}$ and $O_b^{g_2}$
have the desired property iff \((a, b)\) evaluate to true on the circuit with output gate \(g_2\). The task is then to construct \(O_g^a\) from only \(O_g^{a_1}\) and \(O_g^{a_2}\), and \(O_g^b\) from only \(O_g^{b_1}\) and \(O_g^{b_2}\), such that \(O_g^a\) and \(O_g^b\) have the desired property iff \(g = g_1 \ast g_2\) evaluates to true. By the invariant, this is equivalent when \(\ast = \land\) to \(O_g^{a_1}\) and \(O_g^{b_1}\) having the desired property, and \(O_g^{a_2}\) and \(O_g^{b_2}\) having the desired property. In the case of \(\ast = \lor\), only one of the pairs \(O_g^{a_1}\), \(O_g^{b_1}\) or \(O_g^{a_2}\), \(O_g^{b_2}\) needs to have the property.

In the last step, the final objects \(O_A\) and \(O_B\) are constructed by combining all \(O_{a_i}\), \(1 \leq i \leq N\) to form \(O_A\), and \(O_{b_j}\), \(1 \leq j \leq N\) to form \(O_B\). These final objects must allow for selection between different partial assignments. Additionally, the final objects satisfy the desired property iff at least one object pair \(O_{a_i}\) and \(O_{b_j}\) together satisfy the desired property.

The above outlines, on a high level, the approach used in reductions from Formula-SAT to polynomial-time problems that appear here, and in \cite{2, 25}. The techniques presented in \cite{3} instead start with the problem of the satisfiability of branching programs, but they work similarly in the sense that they must model the logical gates AND and OR (this time connecting logical statements about reachability). The authors also take similar steps in order to build two independent objects based on a fixed input assignment \((a, b)\).

References

\cite{1} A. Abboud, A. Backurs, T. D. Hansen, V. V. Williams, and O. Zamir. Subtree isomorphism revisited. ACM Trans. Algorithms, 14(3):27:1–27:23, 2018.

\cite{2} A. Abboud and K. Bringmann. Tighter connections between formula-sat and shaving logs. In 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic, pages 8:1–8:18, 2018.

\cite{3} A. Abboud, T. D. Hansen, V. V. Williams, and R. Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016, pages 375–388, 2016.

\cite{4} T. Akutsu. A linear time pattern matching algorithm between a string and a tree. In Combinatorial Pattern Matching, 4th Annual Symposium, CPM 93, Padova, Italy, June 2-4, 1993, Proceedings, pages 1–10, 1993.

\cite{5} A. Amir, M. Lewenstein, and N. Lewenstein. Pattern matching in hypertext. J. Algorithms, 35(1):82–99, 2000.

\cite{6} T. Brüggemann and W. Kern. An improved local search algorithm for 3-sat. Electron. Notes Discret. Math., 17:69–73, 2004.

\cite{7} R. Chen. Satisfiability algorithms and lower bounds for boolean formulas over finite bases. In Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II, pages 223–234, 2015.

\cite{8} R. Chen, V. Kabanets, A. Kolokolova, R. Shaltiel, and D. Zuckerman. Mining circuit lower bound proofs for meta-algorithms. Comput. Complex., 24(2):333–392, 2015.
[9] M. Chung. O(n^{(2.55)}) time algorithms for the subgraph homeomorphism problem on trees. *J. Algorithms*, 8(1):106–112, 1987.

[10] R. Cole and R. Hariharan. Tree pattern matching to subset matching in linear time. *SIAM J. Comput.*, 32(4):1056–1066, 2003.

[11] M. Equi, R. Grossi, V. Mäkinen, and A. I. Tomescu. On the complexity of string matching for graphs. In C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[12] T. D. Hansen, H. Kaplan, O. Zamir, and U. Zwick. Faster k-sat algorithms using biased-ppsz. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 578–589, 2019.

[13] R. Impagliazzo, W. Matthews, and R. Paturi. A satisfiability algorithm for ac0. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 961–972. SIAM, 2012.

[14] R. Impagliazzo, R. Paturi, and S. Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 479–488, 2013.

[15] C. Jain, H. Zhang, Y. Gao, and S. Aluru. On the complexity of sequence to graph alignment. In L. J. Cowen, editor, *Research in Computational Molecular Biology - 23rd Annual International Conference, RECOMB 2019, Washington, DC, USA, May 5-8, 2019, Proceedings*, volume 11467 of *Lecture Notes in Computer Science*, pages 85–100. Springer, 2019.

[16] I. Komargodski, R. Raz, and A. Tal. Improved average-case lower bounds for demorgan formula size. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 588–597, 2013.

[17] A. Lingas. An application of maximum bipartite c-matching to subtree isomorphism. In *CAAP’83, Trees in Algebra and Programming, 8th Colloquium, L’Aquila, Italy, March 9-11, 1983, Proceedings*, pages 284–299, 1983.

[18] A. Lingas and M. Karpinski. Subtree isomorphism is NC reducible to bipartite perfect matching. *Inf. Process. Lett.*, 30(1):27–32, 1989.

[19] U. Manber and S. Wu. Approximate string matching with arbitrary costs for text and hypertext. In *Advances In Structural And Syntactic Pattern Recognition*, pages 22–33. World Scientific, 1992.

[20] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discret. Appl. Math.*, 10(3):287–295, 1985.

[21] G. Navarro. Improved approximate pattern matching on hypertext. *Theor. Comput. Sci.*, 237(1-2):455–463, 2000.
A Proving the implications of logarithmically faster algorithms for Subtree Isomorphism

Theorem 3. Let \( n \leq S(n) \leq 2^{o(n)} \) be time constructible and monotone non-decreasing. Let \( \mathcal{C} \) be a class of circuits. Suppose there is an SAT algorithm for \( n \)-input circuits which are \( \text{ANDs} \) of \( O(S(n)) \) arbitrary functions of three \( O(S(n)) \)-size circuits from \( \mathcal{C} \), that runs in \( O(2^n/n^{10}) \) time. Then \( \text{E}^{\text{NP}} \) does not have \( S(n) \)-size circuits.
Theorem 4 (3). Suppose there is a satisfiability algorithm for bounded fan-in formulas of size $n^k$ running in $O(2^n/n^k)$ time, for all constants $k > 0$. Then NTIME[$2^{O(n^k)}$] is not contained in non-uniform NC$^1$.

Corollary 1. The existence of a strongly subquadratic time algorithm for PMLG (or Subtree Isomorphism) would imply the class $E_{NP}$ (1) does not have non-uniform $2^{o(n^k)}$-size Boolean formulas and (2) does not have non-uniform $o(n)$-depth circuits of bounded fan-in. It also implies that NTIME[$2^{O(n^k)}$] is not in non-uniform NC.

Proof. Note that the condition in Theorem 3 that the SAT-algorithm works on $n$-input circuits which are ANDs of $O(S(n))$ arbitrary functions of three $O(S(n))$-size circuits is trivially satisfied by a solver that works over Boolean formula. By Theorem 3 (Theorem 2 resp.), for circuits (or equivalently formulas) of size $S(n) = 2^{o(n)}$, a strongly subquadratic time algorithm for PMLG (Subtree Isomorphism resp.) would imply a SAT algorithm running in time

$$O(n^{1+o(1)} \cdot |E||P|^{-1-\varepsilon}) = O(n^{1+o(1)} \cdot 2^{n-\varepsilon n/2}S(n)^4)$$

which is $O(2^n/n^{10})$; the $n^{1+o(1)}$ factor is introduced when moving from a word size of $\Theta(\log n)$ to $\Theta(n)$. Thus, Theorem 3 implies (1). Part (2) is implied as well since a $o(n)$-depth circuit of bounded fan-in can be expressed as a formula of size $S(n) = 2^{o(n)}$. The last statement follows from Theorem 3 and the fact that on circuits of size $n^k$, our subquadratic algorithm would run in time $O(n^{1+o(1)} \cdot 2^{n-\varepsilon n/2}n^{2k})$ which is $O(2^n/n^k)$.

Corollary 2. If PMLG (or Subtree Isomorphism) can be solved in time $O(|E||P|/\log |E|)$ or $O(|E||P|/\log |P|)$ or $O(|T_1||T_2|/\log |T_1|)$ or $O(|T_1||T_2|/\log |T_2|)$ resp. for all $c = \Theta(1)$, then NTIME[$2^{O(n^k)}$] does not have non-uniform polynomial-size log-depth circuits.

Proof. We prove this for PMLG, the proof for Subtree Isomorphism is similar. By Theorem 3, it suffices to show that for all $k$, there exists an algorithm to check satisfiability of all bounded fan-in formulas of size $n^k$ running in time $O(2^n/n^k)$. Suppose that for all $c = \Theta(1)$, there exists an algorithm running in time $O(|E||P|/\log |P|)$ or $O(|E||P|/\log |P|)$. Then by Theorem 3 if we let $c > 4k+1$ we obtain an algorithm running in time

$$\frac{n^{1+o(1)} \cdot 2^n s^3}{\log^c (2^n s^2)} = \frac{n^{1+o(1)} \cdot 2^n s^{3k}}{\log^c (2^n s^{2k})} \leq \frac{n^{1+o(1)} \cdot 2^n s^{3k}}{(\frac{c}{4})^c} = \frac{2^n + c}{n^{c-3k-1-o(1)}} = O \left( \frac{2^n}{n^k} \right)$$

Corollary 3. $E_{NP}$ cannot be computed by non-uniform formulas of cubic size if PMLG (or Subtree Isomorphism) can be solved in time $O \left( \frac{|E||P|}{\log^{2+\varepsilon} |E|} \right)$ or $O \left( \frac{|E||P|}{\log^{2+\varepsilon} |P|} \right)$ for $\varepsilon > 0$, where $G$ is a deterministic DAG of maximum degree three (or $O \left( \frac{|T_1||T_2|}{\log^{2+\varepsilon} |T_1|} \right)$ or $O \left( \frac{|T_1||T_2|}{\log^{2+\varepsilon} |T_2|} \right)$ for $\varepsilon > 0$ resp.).

Proof. Theorem 3 as given in 3 says that solving Formula-SAT in time $O(2^n/n^{10})$ on formulas of size $s = O(n^{3+\varepsilon})$ implies that there is a function in class $E_{NP}$ that cannot be computed by formulas of size $O(n^{3+\varepsilon})$. Then via a proof identical to that of Corollary 3 we have the above result. $\square$