Known Algorithms on Graphs of Bounded Treewidth are Probably Optimal

Daniel Lokshtanov*, Dániel Marx†, Saket Saurabh‡

Abstract
We obtain a number of lower bounds on the running time of algorithms solving problems on graphs of bounded treewidth. We prove the results under the Strong Exponential Time Hypothesis of Impagliazzo and Paturi. In particular, assuming that SAT cannot be solved in time \((2 - \epsilon)^{3^{tw(G)}}\), we show that for any \(\epsilon > 0\):

- **INDEPENDENT SET** cannot be solved in time \((2 - \epsilon)^{tw(G)}|V(G)|^{O(1)}\),
- **DOMINATING SET** cannot be solved in time \((3 - \epsilon)^{tw(G)}|V(G)|^{O(1)}\),
- **MAX CUT** cannot be solved in time \((2 - \epsilon)^{tw(G)}|V(G)|^{O(1)}\),
- **ODD CYCLE TRANSVERSAL** cannot be solved in time \((3 - \epsilon)^{tw(G)}|V(G)|^{O(1)}\),
- For any \(q \geq 3\), **q-COLORING** cannot be solved in time \((q - \epsilon)^{tw(G)}|V(G)|^{O(1)}\),
- **PARTITION INTO TRIANGLES** cannot be solved in time \((2 - \epsilon)^{tw(G)}|V(G)|^{O(1)}\).

Our lower bounds match the running times for the best known algorithms for the problems, up to the \(\epsilon\) in the base.

1 Introduction
It is well-known that many NP-hard graph problems can be solved efficiently if the treewidth \((tw(G))\) of the input graph \(G\) is bounded. For an example, an expository algorithm to solve VERTEX COVER and INDEPENDENT SET running in time \(O^*(4^{tw(G)})\) is described in the algorithms textbook by Kleinberg and Tardos [15] (the \(O^*\) notation suppresses factors polynomial in the input size), while the book of Niedermeier [21] on fixed-parameter algorithms presents an algorithm with running time \(O^*(2^{tw(G)})\). Similar algorithms, with running times on the form \(O^*(\epsilon^{tw(G)})\) for a constant \(c\), are known for many other graph problems such as DOMINATING SET, q-COLORING and ODD CYCLE TRANSVERSAL [1, 9, 10, 28]. Algorithms for graph problems on bounded treewidth graphs have found many uses as subroutines in approximation algorithms [7, 8], parameterized algorithms [6, 20, 27], and exact algorithms [12, 24, 29].

In this paper, we show that any improvement over the currently best known algorithms for a number of well-studied problems on graphs of bounded treewidth would yield a faster algorithm for SAT. In particular, we show if there exists an \(\epsilon > 0\) such that

- **INDEPENDENT SET** can be solved in time \(O^*((2 - \epsilon)^{tw(G)})\) or
- **DOMINATING SET** can be solved in time \(O^*((3 - \epsilon)^{tw(G)})\) or
- **MAX CUT** can be solved in time \(O^*((2 - \epsilon)^{tw(G)})\) or
- **ODD CYCLE TRANSVERSAL** can be solved in time \(O^*((3 - \epsilon)^{tw(G)})\) or
- there is a \(q \geq 3\) such that **q-COLORING** can be solved in time \(O^*((q - \epsilon)^{tw(G)})\) or
- **PARTITION INTO TRIANGLES** can be solved in time \(O^*((2 - \epsilon)^{tw(G)})\)

then SAT can be solved in \(O^*((2 - \delta)^n)\) time for some \(\delta > 0\). Here \(n\) is the number of variables in the input formula to SAT. Such an algorithm would violate the Strong Exponential Time Hypothesis (SETH) of Impagliazzo and Paturi [13]. Thus, assuming SETH, the known algorithms for the mentioned problems on graphs of bounded treewidth are essentially the best possible.

To show our results we give polynomial time many-one reductions that transform \(n\)-variable boolean formulas \(\phi\) to instances of the problems in question. Such reductions are well-known, but for our results we need to carefully control the treewidth of the graphs that our reductions output. A typical reduction creates \(n\) gadgets corresponding to the \(n\) variables; each gadget has a small constant number of vertices. In most cases, this implies that the treewidth can be bounded by \(O(n)\). However, to prove the a lower bound of the form \(O^*((2 - \epsilon)^{tw(G)})\), we need that the treewidth of the
constructed graph is $(1 + o(1))n$. Thus we can afford to increase the treewidth by at most one per variable. For lower bounds above $O^*(n^{(2-\epsilon)\text{tw}(G)})$, we need even more economical constructions. To understand the difficulty, consider the Dominating Set problem, here we want to say that if Dominating Set admits an algorithm with running time $O^*((3-\epsilon)^{\text{tw}(G)})$, we can solve SAT on input formulas with $n$-variables in time $O^*(2^{3-\epsilon \text{tw}(G)})$ for some $\epsilon > 0$, then we can construct graphs on bounded pathwidth graphs.

In other words, each variable should increase the treewidth by less than one. The main challenge in our reductions is to squeeze out as many combinatorial possibilities per increase of treewidth as possible. In order to control the treewidth of the graphs we construct, we upper bound the pathwidth ($\text{pw}(G)$) of the constructed instances and use the fact that for any graph $G$, $\text{tw}(G) \leq \text{pw}(G)$. Thus all of our lower bounds also hold for problems on graphs of bounded pathwidth.

Complexity Assumption: The Exponential Time Hypothesis (ETH) and its strong variant (SETH) are conjectures about the exponential time complexity of $k$-SAT. The $k$-SAT problem is a restriction of SAT, where every clause in input boolean formula $\phi$ has at most $k$ literals. Let $s_k = \inf \{ \delta : k$-SAT can be solved in $2^{s_k \cdot n} \}$ time. The Exponential Time Hypothesis conjectured by Impagliazzo, Paturi and Zane [14] is that $s_k > 0$. In [14] it is shown that ETH is robust, that is $s_k > 0$ if and only if there is a $k \geq 3$ such that $s_k > 0$. In the same year it was shown that assuming ETH the sequence $\{s_k\}$ increases infinitely often [13]. Since SAT has a $O^*(2^n)$ time algorithm, $\{s_k\}$ is bounded by above by one, and Impagliazzo and Paturi [13] conjecture that 1 is indeed the limit of this sequence. In a subsequent paper [3], this conjecture is coined as SETH.

While ETH is now a widely believed assumption, and has been used as a starting point to prove running time lower bounds for numerous problems [5, 4, 11, 19, 18], SETH remains largely untouched (with one exception [22]). The reason for this is two-fold. First, the assumption that $\lim_{k \to \infty} s_k = \infty$ is a very strong one. Second, when proving lower bounds under ETH we can utilize the Sparsification Lemma [14] which allows us to reduce from instances of 3-SAT where the number of clauses is linear in the number of variables. Such a tool does not exist for SETH, and this seems to be a major obstruction for showing running time lower bounds for interesting problems under SETH. We overcome this obstruction by circumventing it — in order to show running time lower bounds for algorithms on bounded treewidth graphs sparsification is simply not required.

Related Work. Despite of the importance of fast algorithms on graphs of bounded treewidth or pathwidth, there is no known natural graph problem for which we know an algorithm outperforming the naïve approach on bounded pathwidth graphs. For treewidth, the situation is slightly better: Alber et al. [1] gave a $O^*(4^{\text{tw}(G)})$ time algorithm for Dominating Set, improving over the natural $O^*(9^{\text{tw}(G)})$ algorithm of Telle and Proskurowski [26]. Recently, van Rooij et al. [28] observed that one could use fast subset convolution [2] to improve the running time of algorithms on graphs of bounded treewidth. Their results include a $O^*(3^{\text{tw}(G)})$ algorithm for Dominating Set and a $O^*(2^{\text{tw}(G)})$ time algorithm for Partition Into Triangles. Interestingly, the effect of applying subset convolution was that the running time for several graph problems on bounded treewidth graphs became the same as the running time for the problems on graphs of bounded pathwidth.

In [28], van Rooij et al. believe that their algorithms are probably optimal, since the running times of their algorithms match the size of the dynamic programming table, and that improving the size of the table without losing time should be very difficult. Our results prove them right: improving their algorithm is at least as hard as giving an improved algorithm for SAT.

2 Preliminaries

In this section we give various definitions which we make use of in the paper. Let $G$ be a graph with vertex set $V(G)$ and edge set $E(G)$. A graph $G'$ is a subgraph of $G$ if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. For subset $V' \subseteq V(G)$, the subgraph $G'[V']$ of $G$ is called a subgraph induced by $V'$. By $N(u)$ we denote (open) neighborhood of $u$ in graph $G$ that is the set of all vertices adjacent to $u$ and by $N[u] = N(u) \cup \{u\}$. Similarly, for a subset $D \subseteq V$, we define $N[D] = \bigcup_{v \in D} N[v]$.

A tree decomposition of a graph $G$ is a pair $(X, T)$ where $T$ is a tree and $A = \{X_i \mid i \in V(T)\}$ is a collection of subsets of $V$ such that: 1. $\bigcup_{i \in V(T)} X_i = V(G)$, 2. for each edge $xy \in E(G)$, $\{x, y\} \subseteq X_i$ for some $i \in V(T)$; 3. for each $x \in V(G)$ the set $\{i \mid x \in X_i\}$ induces a connected subtree of $T$. The width of the tree decomposition is $\max_{i \in V(T)} \{|X_i| - 1\}$. The treewidth of a graph is the minimum width over all tree decompositions of $G$. We denote by $\text{tw}(G)$ the treewidth of graph $G$. If in the definition of treewidth we restrict the tree $T$ to be a path then we get the notion of pathwidth and denote it by $\text{pw}(G)$. For our purpose we need an equivalent definition of pathwidth via mixed search games.
In a mixed search game, a graph $G$ is considered as a system of tunnels. Initially, all edges are contaminated by a gas. An edge is cleared by placing searchers (like a pebble placed on a node) at both its end-points simultaneously or by sliding a searcher along the edge. A cleared edge $e$ is re-contaminated if there is a path $P$ containing $e$ and a contaminated edge and no internal vertex of $P$ contains a searcher. A search is a sequence of operations that can be of the following types: (a) placement of a new searcher on a vertex; (b) removal of a searcher from a vertex; (c) sliding a searcher on a vertex along an incident edge and placing the searcher on the other end. A search strategy is winning if after its termination all edges are cleared. The mixed search number of a graph $G$, denoted by $ms(G)$, is the minimum number of searchers required for a winning strategy of mixed searching on $G$. Takahashi, Ueno and Kajitani [25] obtained the following relationship by placing searchers.

**Proposition 2.1.** ([25]) For a graph $G$, $pw(G) \leq ms(G) \leq pw(G) + 1$.

An instance to SAT always consists of a boolean formula $\phi = C_1 \land \cdots \land C_m$ over $n$ variables $\{v_1, \ldots, v_n\}$ where each clause $C_i$ is OR of one or more literals of variables. We also denote a clause $C_i$ by the set $\{\ell_1, \ell_2, \ldots, \ell_c\}$ of its literals and denote by $|C_i|$ the number of literals in $C_i$. An assignment $\tau$ to the variables is an element of $\{0, 1\}^n$, and it satisfies the formula $\phi$ if for every clause $C_i$ there is literal that is assigned 1 by $\tau$. We say that a variable $v_i$ satisfies a clause $C_j$ if there exists a literal corresponding to $v_i$ in $\{\ell_1, \ell_2, \ldots, \ell_c\}$ and it is set to 1 by $\tau$. A group of variables satisfies a clause $C_j$ if there is a variable that satisfies the clause $C_j$. All the sections in this paper follow the following pattern: definition of the problem; statement of the lower bound; construction used in the reduction; correctness of the reduction; and the upper bound on the pathwidth of the resultant graph.

### 3 Independent Set

An independent set of a graph $G$ is a set $S \subseteq V(G)$ such that $G[S]$ contains no edges. In the Independent Set problem we are given a graph $G$ and the objective is to find an independent set of maximum size.

**Theorem 3.1.** If Independent Set can be solved in $O^*((2 - \epsilon)^{pw(G)})$ for some $\epsilon > 0$ then SAT can be solved in $O^*((2 - \delta)^n)$ time for some $\delta > 0$.

**Construction.** Given an instance $\phi$ to SAT we construct a graph $G$ as follows. We assume that every clause has an even number of variables, if not we can add a single variable to all odd size clauses and force this variable to false. First we describe the construction of clause gadgets. For a clause $C = \{\ell_1, \ell_2, \ldots, \ell_c\}$ we make a gadget $\tilde{C}$ as follows. We take two paths, $CP = cp_1, cp_2, \ldots, cp_c$ and $CP' = cp'_1, cp'_2, \ldots, cp'_c$ having $c$ vertices each, and connect $cp_i$ with $cp'_i$ for every $i$. For each literal $\ell_i$ we make a vertex $\ell_i$ in $\tilde{C}$ and make it adjacent to $cp_i$ and $cp'_i$. Finally we add two vertices $c_{\text{start}}$ and $c_{\text{end}}$, such that $c_{\text{start}}$ is adjacent to $cp_1$ and $c_{\text{end}}$ is adjacent to $cp_c$. Observe that the size of the maximum independent set of $\tilde{C}$ is $c + 2$. Also, since $c$ is even, any independent set of size $c + 2$ in $\tilde{C}$ must contain at least one vertex in $C = \{\ell_1, \ell_2, \ldots, \ell_c\}$. Finally, notice that for any $i$, there is an independent set of size $c + 2$ in $\tilde{C}$ that contains $\ell_i$ and none of $\ell_j$ for $j \neq i$.

We first construct a graph $G_1$. We make $n$ paths $P_1, \ldots, P_n$, each path of length $2m$. Let the vertices of the path $P_i$ be $p_{i0}^1, \ldots, p_{i2m}^1$. The path $P_i$ corresponds to the variable $v_i$. For every clause $C_i$ of $\phi$ we make a gadget $\tilde{C}_i$. Now, for every variable $v_i$, if $v_i$ occurs positively in $C_j$, we add an edge between $p_{i2j}$ and the literal corresponding to $v_i$ in $\tilde{C}_j$. If $v_i$ occurs negatively in $C_j$, we add an edge between $p_{i2j}$ and the literal corresponding to $v_i$ in $\tilde{C}_j$. Now we construct the graph $G$ as follows. We take $n + 1$ copies of $G_1$, call them $G_1, \ldots, G_{n+1}$. For every $i \leq n$ we connect $G_i$ and $G_{i+1}$ by connecting $p_{i2m}$ in $G_i$ with $p_{i1}^1$ in $G_{i+1}$ for every $j \leq n$. This concludes the construction of $G$.

**Lemma 3.1.** If $\phi$ is satisfiable, then $G$ has an indepen-
dent set of size \((mn + \sum_{i \leq m} |C_i| + 2)(n + 1)\).

**Proof.** Consider a satisfying assignment to \(\phi\). We construct an independent set \(I\) in \(G\). For every variable \(v_i\) if \(v_i\) is set to true, then pick all the vertices on odd positions from all copies of \(P_i\), that is \(p_{i,1}^2, p_{i,1}^3, p_{i,1}^5\) and so on. If \(v_i\) is false then pick all the vertices on even positions from all copies of \(P_i\), that is \(p_{i,2}^2, p_{i,2}^3, p_{i,2}^5\) and so on. It is easy to see that this is an independent set of size \(mn(n + 1)\) containing vertices from all the paths. We will now consider the gadget \(\tilde{C}_j\) corresponding to a clause \(C_j\). We will only consider the copy of \(\tilde{C}_j\) in \(G_1\) as the other copies can be dealt identically. Let use choose a true literal \(\ell_a\) in \(C_j\) and let \(v_i\) be the corresponding variable. Consider the vertex \(\ell_a\) in \(\tilde{C}_j\).

If \(v_i\) occurs positively in \(C_j\) then \(v_i\) is true. Then \(I\) does not contain \(p_{i,1}^{2j}\), the only neighbour of \(\ell_a\) outside of \(\tilde{C}_j\). On the other hand if \(v_i\) occurs negatively in \(C_j\) then \(v_i\) is false. In this case \(I\) does not contain \(p_{i,1}^{2j-1}\), the only neighbour of \(\ell_a\) outside of \(\tilde{C}_j\).

There is an independent set of size \(|C_j| + 2\) in \(\tilde{C}\) that contains \(\ell_a\) and none out of \(\ell_b, b \neq a\). We add this independent set to \(I\) and proceed in this manner for every clause gadget. By the end of the process \((\sum_{i \leq m} |C_i| + 2)(n + 1)\) vertices from clause gadgets are added to \(I\), yielding that the size of \(I\) is \((mn + \sum_{i \leq m} |C_i| + 2)(n + 1)\), concluding the proof. \(\square\)

**Lemma 3.2.** If \(G\) has an independent set of size \((mn + \sum_{i \leq m} |C_i| + 2)(n + 1)\), then \(\phi\) is satisfiable.

**Proof.** Consider an independent set \(G\) of size \((mn + \sum_{i \leq m} |C_i| + 2)(n + 1)\). The set \(I\) can contain at most \(m\) vertices from each copy of \(P_i\) for every \(i \leq n\) and at most \(|C_j| + 2\) vertices from each copy of the gadget \(C_j\). Since \(I\) must contain at least these many vertices from each path and clause gadget in order to contain at least \((mn + \sum_{i \leq m} |C_i| + 2)(n + 1)\) vertices, it follows that \(I\) has exactly \(m\) vertices in each copy of each path \(P_i\) and exactly \(|C_j| + 2\) vertices in each copy of each clause gadget \(\tilde{C}_j\). For a fixed \(j\), consider the \(n + 1\) copies of the path \(P_j\). Since \(P_j\) in \(G_i\) is attached to \(P_j\) in \(G_{i+1}\) these \(n + 1\) copies of \(P_j\) together form a path \(P\) having \(2m(n + 1)\) vertices. Since \(|I \cap P| = m(n + 1)\) it follows that \(I \cap P\) must contain every second vertex of \(P\), except possibly in one position where \(I \cap P\) skips two vertices of \(P\). There are only \(n\) paths and \(n + 1\) copies of \(G_1\), hence the pigeon-hole principle yields that in some copy \(G_y\) of \(G_1\), \(I\) contains every second vertex on every path \(P_i\). From now onwards we only consider such a copy \(G_y\).

In \(G_y\), for every \(i \leq n\), \(I\) contains every second vertex of \(P_i\). We make an assignment to the variables of \(\phi\) as follows. If \(I\) contains all the odd numbered vertices of \(P_i\) then \(v_i\) is set to true, otherwise \(I\) contains all the even numbered vertices of \(P_i\) and \(v_i\) is set to false. We argue that this assignment satisfies \(\phi\). Indeed, consider any clause \(C_j\), and look at the gadget \(\tilde{C}_j\). We know that \(I\) contains \(|C_j| + 2\) vertices from \(\tilde{C}_j\) and hence \(I\) must contain a vertex \(\ell_a\) in corresponding to a literal of \(C_j\).

Suppose \(\ell_a\) is a literal of \(v_i\). Since \(I\) contains \(\ell_a\), if \( \ell_a \) occurs positively in \(C_j\), then \(I\) can not contain \(p_{i,1}^{2j}\) and hence \(v_i\) is true. Similarly, if \(\ell_a\) occurs negatively in \(C_j\) then \(I\) can not contain \(p_{i,1}^{2j-1}\) and hence \(v_i\) is false. In both cases \(v_i\) satisfies \(C_j\) and hence all clauses of \(\phi\) are satisfied by the assignment. \(\square\)

**Lemma 3.3.** \(\text{pw}(G) \leq n + 4\).

**Proof.** We give a mixed search strategy to clean \(G\) using \(n + 3\) searchers. For every \(i\) we place a searcher on the first vertex of \(P_i\) in \(G_1\). The \(n\) searchers slide along the paths \(P_1, \ldots, P_n\) in \(m\) rounds. In round \(j\) each searcher \(i\) starts on \(P_i^{2j-1}\). Then, for every variable \(v_i\) that occurs positively in \(C_j\), the searcher \(i\) slide forward to \(P_i^{2j}\). Observe that at this point there is a searcher on every neighbour of the gadget \(\tilde{C}_j\). This gadget can now be cleaned with 3 additional searchers. After \(\tilde{C}_j\) is clean, the additional 3 searchers are removed, and each of the \(n\) searchers on the paths \(P_1, \ldots, P_n\) slide forward along these paths, such that searcher \(i\) stands on \(p_i^{2j+1}\). At that point, the next round commences. When the searchers have cleaned \(G_1\) they slide onto the first vertex of \(P_1, \ldots, P_n\) in \(G_2\). Then they proceed to clean \(G_2, \ldots, G_{n+1}\) in the same way that \(G_1\) was cleaned. Now applying Proposition 2.1 we get that \(\text{pw}(G) \leq n + 4\). \(\square\)

The construction, together with Lemmata 3.1, 3.2 and 3.3 proves Theorem 3.1.

### 4 Dominating Set

A **dominating set** of a graph \(G\) is a set \(S \subseteq V(G)\) such that \(V(G) = N[S]\). In the **Dominating Set** problem we are given a graph \(G\) and the objective is to find a dominating set of minimum size.

**Theorem 4.1.** If **Dominating Set can be solved in** \(O^*(3 - \epsilon)^{\text{pw}(G)}\) time for some \(\epsilon > 0\) then **SAT can be solved in** \(O^*((2 - \delta)^n)\) time for some \(\delta > 0\).

**Construction.** Given \(\epsilon < 1\) and an instance \(\phi\) to SAT we construct a graph \(G\) as follows. We first choose an integer \(p\) depending only on \(\epsilon\). Exactly how \(p\) is chosen will be discussed in the proof of Theorem 4.1. We group the variables of \(\phi\) into groups \(F_1, F_2, \ldots, F_t\), each of size at most \(\beta = \lceil \log 3p \rceil\). Hence \(t = \lceil n/\beta \rceil\). We
now proceed to describe a “group gadget” \( \tilde{B} \), which is central in our construction.

To build the group gadget \( \tilde{B} \) we make \( p \) paths \( P_1, \ldots, P_p \), where the path \( P_i \) contains the vertices \( p_1^i \), \( p_2^i \) and \( p_3^i \). To each path \( P_i \) we attach two guards \( g_i \) and \( g'_i \), both of which are neighbours to \( p_1^i \), \( p_2^i \) and \( p_3^i \). When the gadgets are attached to each other, the guards will not have any neighbours outside of their own gadget \( \tilde{B} \), and will ensure that at least one vertex out of \( p_1^i \), \( p_2^i \) and \( p_3^i \) are chosen in any minimum size dominating set of \( G \). Let \( P \) be a vertex set containing all the vertices on the paths \( P_1, \ldots, P_p \). For every subset \( S \) of \( P \) that picks exactly one vertex from each path \( P_i \) we make two vertices \( x_S \) and \( x'_S \), where \( x_S \) is adjacent to all vertices of \( P \setminus S \) (all those vertices that are on paths and not in \( S \)) and \( x'_S \) is only adjacent to \( x_S \). We conclude the construction of \( \tilde{B} \) by making all the vertices \( x'_S \) (for every set \( S \)) adjacent to each other, that is making them into a clique, and adding a guard \( x \) adjacent to \( x'_S \) for every set \( S \). Essentially \( x'_S \)'s together with \( x \) forms a clique and all the neighbors of \( x \) reside in this clique.

We construct the graph \( G \) as follows. For every group \( F_i \) of variables we make \( m(2pt+1) \) copies of the gadget \( \tilde{B} \), call them \( \tilde{B}_i^j \) for \( 1 \leq j \leq m(2pt+1) \). For every fixed \( i \leq t \) we connect the gadgets \( \tilde{B}_1^i, \tilde{B}_2^i, \ldots, \tilde{B}_m^i(2pt+1) \) in a path-like manner. In particular, for every \( j < m(2pt+1) \) and every \( \ell \leq p \) we make an edge between \( p_3^j \) in the gadget \( \tilde{B}_1^i \) with \( p_1^j \) in the gadget \( \tilde{B}_l^i+1 \). Now we make two new vertices \( h \) and \( h' \), with \( h \) adjacent to \( h' \), \( p_j^i \) in \( \tilde{B}_i^j \) for every \( i \leq t, j \leq p \) and to \( p_j^i \) in \( \tilde{B}_i^{m(2pt+1)} \) for every \( i \leq t, j \leq p \). That is, for all \( 1 \leq i \leq t, h \) is adjacent to first and last vertices of “long paths” obtained after connecting the gadgets \( \tilde{B}_1^i, \tilde{B}_2^i, \ldots, \tilde{B}_m^i(2pt+1) \) in a path-like manner.

For every \( 1 \leq i \leq t \), and to every assignment of the variables in the group \( F_i \), we designate a subset \( S \) of \( P \) in the gadget \( \tilde{B} \) that picks exactly one vertex from each path \( P_i \). Since there are at most \( 2^3 \) different assignments to the variables in \( F_i \), and there are \( 3p \geq 2^3 \) such sets \( S \), we can assign a unique set to each assignment. Of course, the same set \( S \) can correspond to one assignment of the group \( F_1 \) and some another assignment of the group \( F_2 \). Recall that the clauses of \( \phi \) are \( C_1, \ldots, C_m \). For every clause \( C_j \) we make \( 2pt+1 \) vertices \( c_j^l \), one for each \( 0 \leq \ell < 2pt+1 \). The vertex \( c_j^l \) will be connected to the gadgets \( \tilde{B}_i^{m+1} \) for every \( 1 \leq i \leq t \). In particular, for every assignment of the variables in the group \( F_i \) that satisfy the clause \( C_j \), we consider the subset \( S \) of \( P \) that corresponds to the assignment. For every \( 0 \leq \ell < 2n+1 \), we make \( x'_S \) in \( \tilde{B}_i^{m+1} \) adjacent to \( c_j^l \). The best way to view this is that every clause \( C_j \) has \( 2pt+1 \) private gadgets, \( \tilde{B}_1^i, \tilde{B}_2^i, \ldots, \tilde{B}_m^i(2pt+1) \), in every group of gadgets corresponding to \( F_i \)'s. Now we have \( 2pt+1 \) vertices corresponding to the clause \( C_j \), one each for one gadget from each group gadgets corresponding to \( F_i \)'s. This concludes the construction of \( G \).

**Lemma 4.1.** If \( \phi \) has a satisfying assignment, then \( G \) has a dominating set of size \( (p+1)tm(2pt+1) + 1 \).
Proof. Given a satisfying assignment to $\phi$ we construct a dominating set $D$ of $G$ that contains the vertex $h$ and exactly $p+1$ vertices in each gadget $\mathring{B}_i^\ell$. For each group $F_i$ of variables we consider the set $S$ which corresponds to the restriction of the assignment to the variables in $F_i$. From each gadget $\mathring{B}_i^\ell$ we add the set $S$ to $D$ and also the vertex $x'_S$ to $D$. It remains to argue that $D$ is indeed a dominating set. Clearly the size is bounded by $(p+1)tm(2pt+1)+1$, as the number of gadgets is $tm(2pt+1)$.

For a fixed $i \leq t$ and $j$ consider the vertices on the path $P_j$ in the gadgets $\mathring{B}_i^\ell$ for every $\ell \leq m(2pt+1)$. Together these vertices form a path of length $3m(2pt+1)$ and every third vertex of this path is in $S$. Thus, all vertices on this path are dominated by other vertices on the path, except for the first and last one. Both these vertices, however, are dominated by $h$.

Now, fix some $i \leq t$ and $\ell \leq m(2pt+1)$ and consider the gadget $\mathring{B}_i^\ell$. Since $D$ contains some vertex on the path $P_j$, we have that for every $j$ both $g_j$ and $g'_j$ are dominated. Furthermore, for every set $S^*$ not equal to $S$ that picks exactly one vertex from each $P_j$, vertex $x_S$ is dominated by some vertex on some $P_j$—namely by all vertices in $S \setminus S^* \neq \emptyset$. The last assertion follows since $x_S$ is connected to all the vertices on paths except $S^*$. On the other hand, $x_S$ is dominated by $x'_S$, and $x'_S$ also dominates all the other vertices $x''_S$, for $S \neq S$ and the guard $x$.

The only vertices not yet accounted for are the vertices $c_{j, \ell}$ for every $j \leq m$ and $\ell < 2pt+1$. Fix a $j$ and $\ell$ and consider the clause $C_j$. This clause contains a literal set to true, and this literal corresponds to a variable in the group $F_i$ for some $i \leq t$. Of course, the assignment to $F_i$ satisfies $C_j$. Let $S$ be the set corresponding to this assignment of $F_i$. By the construction of $D$, the dominating set contains $x_S$ in $\mathring{B}_i^{mt+\ell}$ and $x'_S$ is adjacent to $c_{j, \ell}$. This concludes the proof. \qed

Lemma 4.2. If $G$ has a dominating set of size $(p+1)tm(2pt+1)+1$, then $\phi$ has a satisfying assignment. 
Proof. Let $D$ be a dominating set of $G$ of size at most $(p+1)tm(2pt+1)+1$. Since $D$ must dominate $h'$, hence without loss of generality we can assume that $D$ contains $h$. Furthermore, inside every gadget $\mathring{B}_i^\ell$, $D$ must dominate all the guards, namely $g_j$ and $g'_j$ for every $j \leq p$, and also $x$. Thus $D$ contains at least $p+1$ vertices from each gadget $\mathring{B}_i^\ell$ which in turn implies that $D$ contains exactly $p+1$ vertices from each gadget $\mathring{B}_i^\ell$. The only way $D$ can dominate $g_j$ and $g'_j$ for every $j$ and in addition dominate $x$ with only $p+1$ vertices if $D$ has one vertex from each $P_j$, $j \leq p$ and in addition contains some vertex in $N[x]$. Let $S$ be $D \cap P$ in $\mathring{B}_i^\ell$. Observe that $x_S$ is not dominated by $D \cap S$. The only vertex in $N[x]$ that dominates $x_S$ is $x'_S$ and hence $D$ contains $x'_S$.

Now we want to show that for every $1 \leq i \leq t$ there exists one $0 \leq \ell \leq 2pt$ such that for fixed $i$, $D \cap P$ is same in all the gadgets $\mathring{B}_i^{mt+\ell}$, $1 \leq \ell \leq m$. Consider a gadget $\mathring{B}_i^\ell$ and its follower, $\mathring{B}_i^{\ell+1}$. Let $S$ be $D \cap P$ in $\mathring{B}_i^\ell$ and $S' \cap P$ in $\mathring{B}_i^{\ell+1}$. Observe that if $S$ contains $p_i^p$ in $\mathring{B}_i^\ell$ and $p_i^b$ in $\mathring{B}_i^{\ell+1}$ then we must have $b \leq a$. We call a consecutive pair bad if for some $j \leq p$, $D$ contains $p_i^j$ in $\mathring{B}_i^\ell$ and $p_i^b$ in $\mathring{B}_i^{\ell+1}$ and $b < a$. Hence for a fixed $i$, we can at most have $2p$ consecutive bad pairs. Now we mark all the bad pairs that occur among the gadgets corresponding to some $F_i$. This way we can mark only $2tp$ bad pairs. Thus, by the pigeon hole principle, there exists an $\ell \in \{0, \ldots, 2tp\}$ such that there are no bad pairs in $\mathring{B}_i^{mt+\ell}$ for all $1 \leq i \leq t$ and $1 \leq r \leq m$.

We make an assignment $\phi$ by reading off $D \cap P$ in each gadget $\mathring{B}_i^{mt+\ell}$. In particular, for every group $F_i$, we consider $S = D \cap P$ in the gadget $\mathring{B}_i^{mt+\ell}$. This set $S$ corresponds to an assignment of $F_i$, and this is the assignment of $F_i$ that we use. It remains to argue that every clause $C_r$ is satisfied by this assignment.

Consider the vertex $c_{r, \ell}$. We know that it is dominated by some $x'_S$ in a gadget $\mathring{B}_i^{mt+\ell}$. The set $S$ corresponds to an assignment of $F_i$ that satisfies the clause $C_r$. Because $D \cap P$ remains unchanged in all gadgets from $\mathring{B}_i^{mt+\ell}$ to $\mathring{B}_i^{mt+\ell+1}$, this is exactly the assignment $\phi$ restricted to the group $F_i$. This concludes the proof. \qed

Lemma 4.3. $\text{pw}(G) \leq tp + O(3^p)$
Proof. We give a mixed search strategy to clean the graph with $tp + O(3^p)$ searchers. For a gadget $\mathring{B}$ we call the vertices $p_i^j$ and $p_i^b$, $1 \leq j \leq p$, as entry vertices and exit vertices respectively. We search the graph in $m(2pt+1)$ rounds. In the beginning of round $\ell$ there are searchers on the entry vertices of the gadgets $\mathring{B}_i^\ell$ for every $i \leq t$. Let $1 \leq a \leq m$ and $0 \leq b < 2pt+1$ be integers such that $\ell = a + mb$. We place a searcher on $c_{r, \ell}$. Then, for each $i$ between 1 and $p$ in turn we first put searchers on all vertices of $\mathring{B}_i^\ell$ and then remove all the searchers from $\mathring{B}_i^\ell$ except for the ones standing on the exit vertices. After all gadgets $\mathring{B}_1^\ell \ldots \mathring{B}_p^\ell$ have been cleaned in this manner, we can remove the searcher from $c_{r, \ell}$. To commence the next round, the searchers slide from the exit positions of $\mathring{B}_i^\ell$ to the entry positions of $\mathring{B}_i^{\ell+1}$ for every $i$. In total, at most $tp + |V(\mathring{B})| + 1 \leq tp + O(3^p)$ searchers are used simultaneously. This together with Proposition 2.1 give the desired upperbound on the pathwidth. \qed

Proof. [Proof (of Theorem 4.1)] Suppose DOMINATING SET can be solved in $O^*(\bar{(3 - \epsilon)^{\text{pw}(G)}}) = O^*(3^{\text{pw}(G)})$
time, where \( \lambda = \log_3 (3 - \epsilon) < 1 \). We choose \( p \) large enough such that \( \lambda \cdot \frac{p}{(p \log 3)} = \frac{\delta'}{\log 3} \) for some \( \delta' < 1 \).

Given an instance of SAT we construct an instance of DOMINATING SET using the above construction and the chosen value of \( p \). Then we solve the DOMINATING SET instance using the \( O^*(3^{\lambda \text{pw}(G)}) \) time algorithm. Correctness is ensured by Lemmata 4.1 and 4.2. Lemma 4.3 yields that the total time taken is upper bounded by \( O^*(3^{\lambda \text{pw}(G)}) \leq O^*(3^{\lambda \text{pw}+f(\lambda)}) \leq O^*(3^3 \text{pw}) \leq O^*(3^{2^n}) = O^*((2 - \delta)^n) \), for some \( \delta, \delta' < 1 \). This concludes the proof. \( \Box \)

5 Max Cut

A cut in a graph \( G \) is a partition of \( V(G) \) into \( V_0 \) and \( V_1 \). The cut-set of the cut is the set of edges whose one end point is in \( V_0 \) and the other in \( V_1 \). We say that an edge is crossing this cut if it has one endpoint in \( V_0 \) and one in \( V_1 \), that is, the edge is in the cut-set. The size of the cut is the number of edges in \( G \) which are crossing this cut. If the edges of \( G \) have positive integer weights then the weight of the cut is the sum of the weights of edges which are crossing the cut. In the MAX CUT problem we are given a graph \( G \) together with an integer \( t \) and asked whether there is a cut of \( G \) of size at least \( t \). In the WEIGHTED MAX CUT problem every edge has a positive integer weight and the objective is to find a cut of weight at least \( t \). In this section we show the following theorem.

**Theorem 5.1.** If MAX CUT can be solved in \( O^*((2 - \epsilon)^{\text{pw}(G)}) \) for some \( \epsilon > 0 \) then SAT can be solved in \( O^*((2 - \delta)^n) \) time for some \( \delta > 0 \).

**Construction.** Given an instance \( \phi \) of SAT we first construct an instance \( G_w \) of WEIGHTED MAX CUT as follows. We later explain how to obtain an instance of unweighted MAX CUT from here.

We start with making a vertex \( x_0 \). Without loss of generality, we will assume that \( x_0 \in V_0 \) in every solution. We make a vertex \( \hat{v}_i \) for each variable \( v_i \). For every clause \( C_j \) we make a gadget as follows. We make a path \( P_j \) having \( |V(C_j)| \) vertices. All the edges on \( P_j \) have weight \( 3n \). Now, we make the first and last vertex of \( P_j \) adjacent to \( x_0 \) with an edge of weight \( 3n \). Thus the path \( P_j \) plus the edges from the first and last vertex of \( P_j \) to \( x_0 \) form an odd cycle \( C_j \). We will say that the first, third, fifth, etc., vertices are on odd positions on \( P_j \) while the remaining vertices are on even positions. For every variable \( v_i \) that appears positively in \( C_j \) we select a vertex \( p \) at an even position (but not the last vertex) on \( P_j \) and make \( \hat{v} \) adjacent to \( p \) and \( p \)'s successor on \( P_j \) with edges of weight 1. For every variable \( v_i \) that appears negatively in \( C_j \) we select a vertex \( p \) at an odd position on \( P_j \) and make \( \hat{v} \) adjacent to \( p \) and \( p \)'s successor on \( P_j \) with edges of weight 1. We make sure that each vertex on \( P_j \) receives an edge at most once in this process. There are more than enough vertices on \( P_j \) to accommodate all the edges incident to vertices corresponding to variables in the clause \( C_j \). We create such a gadget for each clause and set \( t = 1 + (12n + 1) \sum_{j=1}^m |C_j| \). This concludes the construction.

**Lemma 5.1.** If \( \phi \) is satisfiable, then \( G_w \) has a cut of weight at least \( t \).

**Proof.** Suppose \( \phi \) is satisfiable. We put \( x_0 \) in \( V_0 \) and for every variable \( v_i \) we put \( \hat{v}_i \) in \( V_1 \) if \( v_i \) is true and \( \hat{v}_i \) in \( V_0 \) if \( v_i \) is false. For every clause \( C_j \) we proceed as follows. Let us choose a true literal of \( C_j \) and suppose that this literal corresponds to a vertex \( p_j \) on \( P_j \). We put the first vertex on \( P_j \) in \( V_1 \), the second in \( V_0 \) and then we proceed along \( P_j \) putting every second vertex into \( V_1 \) and \( V_0 \) until we reach \( p_j \). The successor \( p'_j \) of \( p_j \) on \( P_j \) is put into the same set as \( p_j \). Then we continue along \( P_j \) putting every second vertex in \( V_1 \) and \( V_0 \). Notice that even though \( C_j \) may contain more than one literal that is set to true, we only select one vertex \( p_j \) from the path \( P_j \) and put \( p_j \) and its successor on the same side of the partition. It remains to argue that this cut has weight at least \( t \).

For every clause \( C_j \) all edges on the path \( P_j \) except for \( p_j p'_j \) are crossing, and the two edges to \( x_0 \) from the first and last vertex of \( P_j \) are crossing as well. These edges contribute \( 12n |C_j| \) to the weight of the cut. We know that \( p_j \) corresponds to a literal that is set to true, and this literal corresponds to a variable \( v_i \). If \( v_i \) occurs positively in \( C_j \) then \( v_i \in V_1 \) and \( p_j \) is on an even position of \( P_j \). Thus both \( p_j \) and his successor \( p'_j \) are in \( V_0 \) and hence both \( v_i p_j \) and \( v_i p'_j \) are crossing, contributing 2 to the weight of the cut. For each of the remaining variables \( v_{i'} \) appearing in \( C_j \), one of the two neighbours of \( \hat{v}_{i'} \) on \( P_j \) appear in \( V_0 \) and one in \( V_1 \), so exactly one edge from \( v_{i'} \) to \( P_j \) is crossing. Thus the total weight of the cut is \( t = \sum_{j=1}^m 12n |C_j| + |C_j| + 1 = m + (12n + 1) \sum_{j=1}^m |C_j| \). This completes the proof. \( \Box \)

**Lemma 5.2.** If \( G_w \) has a cut of weight at least \( t \), then \( \phi \) is satisfiable.

**Proof.** Let \( (V_0, V_1) \) be a cut of \( G \) of maximum weight, hence the weight of this cut is at least \( t \). Without loss of generality, let \( x_0 \in V_0 \). For every clause \( C_j \) at least one edge of the odd cycle \( C_j \) is not crossing. If more than
one edge of this cycle is not crossing, then the total weight of the cut edges incident to the path $P_j$ is at most $3n|4|C_j| + 1| + 2n < 12|C_j|$. In this case we could change the partition $(V_0, V_1)$ such that all edges of $P_j$ are crossing and the first vertex of $P_j$ is in $V_1$. Using the new partition the weight of the crossing edges in the cycle $C_j$ is at least $12|C_j|$ and the edges not incident to $P_j$ are unaffected by the changes. This contradicts that $(V_0, V_1)$ was a maximum weight cut. Thus it follows that exactly one edge of $C_j$ is not crossing.

Given the cut $(V_0, V_1)$ we set each variable $v_i$ to true if $\hat{v}_i \in V_1$ and $v_i$ to false otherwise. Consider a clause $C_j$ and a variable $v_i$ that appears in $C_j$. Let $w$ be the edge of $C_j$ that is not crossing. If there is a variable $\hat{v}_i$ adjacent to both $u$ and $v$, then it is possible that both $\hat{v}_i u$ and $\hat{v}_i v$ are crossing. For every other variable $v_j$ in $C_j$, at most one of the edges from $v_j$ to $P_j$ is crossing.

Thus, the weight of the edges that are crossing in the gadget $C_j$ is at most $(12n + 1)|C_j| + 1$. Hence, to find a cut-set of weight at least $t$ in $G$, we need to have crossing edges in $C_j$ with sum of their weights exactly equal to $12n|C_j| + |C_j| + 1$. It follows that there is a vertex $\hat{v}_i$ adjacent to both $u$ and $v$ such that both $\hat{v}_i u$ and $\hat{v}_i v$ are crossing.

If $v_i$ occurs in $C_j$ positively then $u$ is on an even position and hence, $u \in V_0$. Since $\hat{v}_i u$ is crossing it follows that $v_i$ is true and $C_j$ is satisfied. On the other hand, if $v_i$ occurs in $C_j$ negatively then $u$ is on an odd position and hence, $u \in V_1$. Since $\hat{v}_i u$ is crossing it follows that $v_i$ is false and $C_j$ is satisfied. As this holds for each clause individually, this concludes the proof. □

For every edge $e \in E(G_w)$, let $w_e$ be the weight of $e$ in $G_w$. We construct an unweighted graph $G$ from $G_w$ by replacing every edge $e = uv$ by $w_e$ paths from $u$ to $v$ on three edges. Let $W$ be the sum of the edge weights of all edges in $G_w$.

**Lemma 5.3.** $G$ has a cut of size $2W + t$ if and only if $G_w$ has a cut of weight at least $t$.

**Proof.** Given a partition of $V(G_w)$ we partition $V(G)$ as follows. The vertices of $G$ that also are vertices of $V(G)$ are partitioned in the same way as in $V(G_w)$. On each path of length 3, if the endpoints of the path are in different sets we can partition the middle vertices of the path such that all edges are cut. If the endpoints are in the same set we can only partition the middle vertices such that 2 out of the 3 edges are cut. The reverse direction is similar. □

**Lemma 5.4.** $\text{pw}(G) \leq n + 5$.

**Proof.** We give a search strategy to clean $G$ with $n + 5$ searchers. We place one searcher on each vertex $\hat{v}_i$ and one searcher on $x_0$. Then one can search the gadgets $\hat{H}_j$ one by one. In $G_w$ it is sufficient to use 2 searchers for each $\hat{H}_j$, whereas in $G$ after the edges have been replaced by multiple paths on three edges, we need 4 searchers. This combined with Proposition 2.1 gives the desired upper bound on the pathwidth of the graph. □

The construction, together with Lemmata 5.1, 5.2, 5.3 and 5.4 proves Theorem 5.1.

6 Graph Coloring

A $q$-coloring of $G$ is a function $\mu : V(G) \rightarrow [q]$. A $q$-coloring $\mu$ of $G$ is proper if for every edge $uv \in E(G)$ we have $\mu(u) \neq \mu(v)$. In the $q$-COLORING problem we are given as input a graph $G$ and the objective is to decide whether $G$ has a proper $q$-coloring. In the LIST COLORING problem, every vertex $v$ is given a list $\mu(v) \subseteq [q]$ of admissible colors. A proper list coloring of $G$ is a function $\mu : V(G) \rightarrow [q]$ such that $\mu$ is a proper coloring of $G$ that satisfies $\mu(v) \subseteq L(v)$ for every $v \in V(G)$. In the $q$-LIST COLORING problem we are given a graph $G$ together with a list $L(v) \subseteq [q]$ for every vertex $v$. The task is to determine whether there exists a proper list coloring of $G$.

A feedback vertex set of a graph $G$ is a set $S \subseteq V(G)$ such that $G \setminus S$ is a forest; we denote by $\text{fvs}(G)$ the size of the smallest such set. It is well-known that $\text{tw}(G) \leq \text{fvs}(G) + 1$. Unlike in the other sections, where we give lower bounds for algorithms parameterized by $\text{pw}(G)$, the following theorem gives also a lower bound for algorithms parameterized by $\text{fvs}(G)$. Such a lower bound follows very naturally from the construction we are doing here, but not from the constructions in the other sections. It would be interesting to explore whether it is possible to prove tight bounds parameterized by $\text{fvs}(G)$ for the problems considered in the other sections.
Theorem 6.1. If \( q \)-Coloring can be solved in \( \mathcal{O}^*(p) \) time for some \( p > 0 \), then SAT can be solved in \( \mathcal{O}^*(2^\sqrt{n}) \) time for some \( \delta > 0 \).

Construction. We will show the result for List Coloring first, and then give a simple reduction that demonstrates that \( q \)-Coloring can be solved in \( \mathcal{O}^*((q - \epsilon)\log q) \) time if and only if \( q \)-List Coloring can.

Depending on \( \epsilon \) and \( q \) we choose a parameter \( p \). Now, given an instance \( \phi \) to SAT we will construct a graph \( G \) with a list \( L(v) \) for every \( v \), such that \( G \) has a proper list-coloring if and only if \( \phi \) is satisfiable. Throughout the construction we will call color 1-red, color 2-white and color 3-black.

We start by grouping the variables of \( \phi \) into \( t \) groups \( F_1, \ldots, F_t \) of size \( \lceil \log q \rceil \). Thus \( t = \lceil \frac{n \log q^p}{q^p} \rceil \). We will call an assignment of truth values to the variables in \( G \) a group assignment. We will say that a group assignment satisfies a clause \( C_j \) of \( \phi \) if \( C_j \) contains at least one literal which is set to true by the group assignment. Notice that \( C_j \) can be satisfied by a group assignment of a group \( F_i \), even though \( C_j \) also contains variables that are not in \( F_i \).

For each group \( F_i \), we make a set \( V_i \) of \( p \) vertices \( v_{i1}, \ldots, v_{ip} \). The vertices in \( V_i \) get full lists, that is, they can be colored by any color in \( [q] \). The coloring of the vertices in \( V_i \) will encode the group assignment of \( F_i \). There are \( q^p \geq 2^{n/\epsilon} \) possible group assignments, and that \( q \) and \( p \) are constants independent of the input \( \phi \). The list of every vertex on \( V_i \) will encode the group assignment of \( F_i \). Thus, to each possible group assignment of \( F_i \) we attach a unique coloring of \( V_i \). Notice that some colorings of \( V_i \) may not correspond to any group assignment of \( F_i \).

For each clause \( C_j \) of \( \phi \), we make a gadget \( \hat{C}_j \). The main part of \( \hat{C}_j \) is a long path \( \hat{P}_j \) that has one vertex for each group assignment that satisfies \( \hat{C}_j \). Notice that there are at most \( tq^p \) possible group assignments, and that \( q \) and \( p \) are constants independent of the input \( \phi \). The list of every vertex on \( \hat{P}_j \) is \{red, white, black\}. We attach two vertices \( p_{j}^{\text{start}} \) and \( p_{j}^{\text{end}} \) to the start and end of \( \hat{P}_j \) respectively, and the two vertices are not counted as vertices of the path \( \hat{P}_j \) itself. The list of \( p_{j}^{\text{start}} \) is \{white\}. If \( |V(\hat{P}_j)| \) is even, then the list of \( p_{j}^{\text{end}} \) is \{white\}, whereas if \( |V(\hat{P}_j)| \) is odd then the list of \( p_{j}^{\text{end}} \) is \{black\}. The intention is that to properly color \( \hat{P}_j \) one needs to use the color red at least once, and that once is sufficient. The position of the red colored vertex on the path \( \hat{P}_j \) encodes how the clause \( C_j \) is satisfied.

For every vertex \( v \) on \( \hat{P}_j \) we proceed as follows. The vertex \( v \) corresponds to a group assignment to \( F_i \) that satisfies the clause \( C_j \). This assignment in turn corresponds to a coloring of the vertices of \( V_i \). Let this coloring be \( \mu_i \). We build a connector whose role is to enforce that \( v \) can be red only if coloring \( \mu_i \) appears on \( V_i \). To build the connector, for each vertex \( v^i_j \in V_i \) and color \( x \in [q] \setminus \{\mu_i(v^i_j)\} \) we do the following.

- If \( x \) is red, then we add one vertex \( w_x \) for every color \( y \) except for red. We make \( w_x \) adjacent to \( v^i_j \) and the list of \( w_x \) is \{red, \( y \)\}. Then we add a vertex \( w \) which is adjacent to all vertices \( v^i_j \) and \( v \), and whose list is all of \( [q] \). Notice that in the above construction we have reused the names \( w \), \( w_x \) and \( w_y \) for many different vertices: in each connector, there is a separate vertex \( w \) for each vertex \( v^i_j \in V_i \) and color \( x \in [q] \setminus \{\mu_i(v^i_j)\} \). Building a connector for each vertex on \( \hat{P}_j \) concludes the construction of the clause gadget \( \hat{C}_j \), and creating one such gadget for each clause concludes the construction of \( G \). The following lemma summarizes the most important properties of the connector.

Lemma 6.1. Consider the connector corresponding a vertex \( v \) on \( \hat{P}_j \) and coloring \( \mu_i \) of \( V_i \).

1. Any coloring on \( V_i \) and any color \( c \in \{\text{white}, \text{black}\} \) on \( v \) can be extended to the rest of the connector.
2. Coloring \( \mu_i \) on \( V_i \) and any color \( c \in \{\text{red}, \text{white}, \text{black}\} \) on \( v \) can be extended to the rest of the connector.
3. In any coloring of the connector, if \( v \) is red, then \( \mu_i \) appears on \( V_i \).

Proof. 1. For each vertex \( v^i_j \in V_i \) and color \( x \in [q] \setminus \{\mu_i(v^i_j)\} \) we do the following.
• If $x$ is red then in the construction of $\hat{C}_j$ we added a vertex $w_x$ with list \{y, red\} for every color $y \neq$ red adjacent to $v_i'$, and a vertex $w$ with list \{q\} adjacent to $w_y$ for every $y \neq$ red. If $w_i'$ is colored red, then we color each vertex $w_y$ with $y$ and $w$ with red. Notice that $w$ is adjacent to $v$, but $v$ is colored either white or black, so it is safe to color $w$ red. If, on the other hand, $w_i'$ is not colored red, we can color $w_y$ red for every $y$. Then all the neighbours of $w_i$ have been colored with red, except for $v$ which has been colored white or black. Thus it is safe to color $w$ with the color out of black and white which was not used to color $v$.

• If $x$ is not red, then in the construction of $\hat{C}_j$ we added two vertices $w_y$ and $w_y'$ for each color $y$ except for red, and also added a vertex $w$. The vertices $w_y$ are adjacent to $v_i'$ and for every $y \neq$ red the vertex $w_i'$ is adjacent to $w_y$. Finally $w$ is adjacent to all the vertices $w_y'$ and to $v$. For every $y$ the list of $w_y$ is \{x, red\} while the list of $w_y'$ is \{y, red\}. The list of $w$ is \{q\}. If $v_i'$ is colored with $x$, then we let $w_y$ take color red and $w_y'$ take color $y$ for every $y \neq$ red. We color $w$ with red. In the case that $v_i'$ is colored with a color different from $x$, we let $w_y$ be colored with $x$ and $w_y'$ be colored red for every $y \neq$ red. Finally, all the neighbours of $w$ except for $v$ have been colored red, while $v$ is colored with either black or white. According to the color of $v$ we can either color $w$ black or white.

2. We can assume that $v$ is red, otherwise we are done by the previous statement. For each vertex $v_i' \in V_i$ and color $x \in \{q\} \setminus \{\mu_i(v_i')\}$ we do the following.

• If $x$ is red then in the construction of $\hat{C}_j$ we added a vertex $w_x$ with list \{y, red\} for every color $y \neq$ red adjacent to $v_i'$, and a vertex $w$ with list \{q\} adjacent to $w_y$ for every $y \neq$ red. Since $v_i'$ is not colored red by $\mu_i$, we can color $w_y$ red for every $y$. Then all the neighbours of $w$ including $v$ have been colored with red and it is safe to color $w$ with white.

• If $x$ is not red, then in the construction of $\hat{C}_j$ we added two vertices $w_y$ and $w_y'$ for each color $y$ except for red, and also added a vertex $w$. The vertices $w_y$ are adjacent to $v_i'$ and for every $y \neq$ red the vertex $w_i'$ is adjacent to $w_y$. Finally $w$ is adjacent to all the vertices $w_y'$ and to $v$. For every $y$ the list of $w_y$ is \{x, red\} while the list of $w_y'$ is \{y, red\}. The list of $w$ is \{q\}. Since $\mu_i$ colors $v_i'$ with a color different from $x$ we let $w_y$ be colored with $x$ and $w_y'$ be colored red for every $y \neq$ red. Finally, all the neighbours of $w$ including $v$ have been colored red so it is safe to color $w$ white.

3. Suppose for contradiction that $v$ is red, but some vertex $v_i' \in V_i$ has been colored with a color $x \neq \mu_i(v_i')$. There are two cases. If $x$ is red, then in the construction we added vertices $w_y$ adjacent to $v_i'$ for every color $y \neq$ red. Also we added a vertex $v$ adjacent to $v$ and to $w_y$ for every $y \neq$ red. The list of $w_y$ is \{red, y\} and since $w_y$ must have been colored $y$ for every $y \neq$ red. But then $w$ is adjacent to $v$ which is colored red, and to $w_y$ which is colored $y$ for every $y \neq$ red. Thus vertex $w$ has all colors in its neighborhood, a contradiction. In the case when $x$ is not red, then in the construction we added two vertices $w_y$ and $w_y'$ for each $y \neq$ red. Each $w_y$ was adjacent to $v_i'$ and had \{x, red\} as its list. Since $v_i'$ is colored $x$, all the $w_y$ vertices must be colored red. For every $y \neq$ red, we have that $w_y'$ is adjacent to $w_y$ and has \{red, y\} as its list. Hence for every $y \neq$ red the vertex $w_y'$ is colored with $y$. But, in the construction we also added a vertex $w$ adjacent to $v$ and to $w_y'$ for each $y \neq$ red. Thus again, vertex $w$ has all colors in its neighbourhood, a contradiction. □

**Lemma 6.2.** If $\phi$ is satisfiable, then $G$ has a proper list-coloring.

**Proof.** Starting from a satisfying assignment of $\phi$ we construct a coloring $\gamma$ of $G$. The assignment $\phi$ corresponds to a group assignment to each group $F_i$. Each group assignment corresponds to a coloring of $V_i$. For every $i$, we let $\gamma$ color the vertices of $V_i$ using the coloring corresponding to the group assignment of $F_i$.

Now we show how to complete this coloring to a proper coloring of $G$. Since the gadgets $\hat{C}_j$ are pairwise disjoint, and there are no edges going between them, it is sufficient to show that we can complete the coloring for every gadget $\hat{C}_j$. Consider the clause $C_j$. The clause contains a literal that is set to true, and this literal belongs to a variable in the group $F_i$. The group assignment of $F_i$ satisfies the clause $C_j$. Thus, there is a vertex $v$ on $\hat{P}_j$ that corresponds to this assignment. We set $\gamma(v)$ as red (that is, $\gamma$ colors $v$ red), $p_{j}^{\text{start}}$ is colored white and $p_{j}^{\text{end}}$ is colored with its only admissible color, namely black if $|V(\hat{P}_j)|$ is even and white if $|V(\hat{P}_j)|$ is odd. The remaining vertices of $\hat{P}_j$ are colored alternatingly white or black. By Lemma 6.1(2), the coloring can be extended to every vertex of the connector between $V_i$ and $v$: the coloring appearing on $V_i$ is the coloring $\mu_i$ corresponding to the group assignment $F_i$. For every other vertex $u$ on $\hat{P}_j$, the color of $u$ is black or white, thus Lemma 6.1(1) ensures that the coloring can be extended to any connector on $u$.

As this procedure can be repeated to color the gadget $\hat{C}_j$ for every clause $C_j$, we can complete $\gamma$ to a proper list-coloring of $G$. □
Lemma 6.3. If $G$ has a proper list-coloring $\gamma$, then $\phi$ is satisfiable.

Proof. Given $\gamma$ we construct an assignment to the variables of $\phi$ as follows. For every group $F_i$ of variables, if $\gamma$ colors $V_i$ with a coloring that corresponds to a group assignment of $F_i$ then we set this assignment for the variables in $F_i$. Otherwise we set all the variables in $F_i$ to false. We need to argue that this assignment satisfies all the clauses of $\phi$.

Consider a clause $C_j$ and the corresponding gadget $\hat{C}_j$. By a simple parity argument, $\hat{P}_j$ can not be colored using only the colors black and white. Thus, some vertex $v$ on $\hat{P}_j$ is colored red. The vertex $v$ corresponds to a group assignment of some group $F_i$ that satisfies $\hat{C}_j$. As $v$ is red, Lemma 6.1(3) implies that $V_i$ is colored with the coloring $\mu_i$ that corresponds to this assignment. The construction then implies that our chosen assignment satisfies $C_j$. As this is true for every clause, this concludes the proof. □

Observation 1. The vertices $\bigcup_{i \leq t} V_i$ form a feedback vertex set of $G$. Furthermore, $pw(G) \leq pt + 4$

Proof. Observe that after removing $\bigcup_{i \leq t} V_i$, all that is left are the gadgets $\hat{C}_j$ which do not have any edges between each other. Each such gadget is a tree and hence $\bigcup_{i \leq t} V_i$ form a feedback vertex set of $G$. If we place a searcher on each vertex of $\bigcup_{i \leq t} V_i$ it is easy to see that each gadget $\hat{C}_j$ can be searched with 4 searches. The pathwidth bound on $G$ follows using Proposition 2.1. □

Lemma 6.4. If $q$-List Coloring can be solved in $O^*((q - \epsilon)^{fvs(G)})$ time for some $\epsilon < 1$, then SAT can be solved in $O^*((2 - \delta)^n)$ time for some $\delta < 1$.

Proof. Let $O^*((q - \epsilon)^{fvs(G)}) = O^*((\lambda)^{fvs(G)})$ time, where $\lambda = \log_{q}(q - \epsilon) < 1$. We choose a sufficiently large $p$ such that $\delta' = \lambda \frac{p}{p-1} < 1$. Given an instance $\phi$ of SAT we construct a graph $G$ using the construction above, and run the assumed $q$-List Coloring. Correctness follows from Lemmata 6.2 and 6.3. By Observation 1 the graph $G$ has a feedback vertex set of size $p\left\lfloor \frac{n}{p \log q} \right\rfloor$. The choice of $p$ implies that

$$\lambda p \left\lfloor \frac{n}{p \log q} \right\rfloor \leq \lambda p \frac{n}{(p-1) \log q} + p \leq \frac{n}{\log q} + p \leq \delta'' n,$$

for some $\delta'' < 1$. Hence SAT can be solved in time $O^*(2^{\delta'' n}) = O^*((2 - \delta)^n)$, for some $\delta > 0$. □

Finally, observe that we can reduce $q$-List-Coloring to $q$-Coloring by adding a clique $Q = \{q_1, \ldots, q_q\}$ on $q$ vertices to $G$ and making $q_i$ adjacent to $v$ when $i \notin L(v)$. Any coloring of $Q$ must use $q$ different colors, and without loss of generality $q_i$ is colored with color $i$. Then one can complete the coloring if and only if one can properly color $G$ using a color from $L(v)$ for each $v$. We can add the clique $Q$ to the feedback vertex set—this increases the size of the minimum feedback vertex set by $q$. Since $q$ is a constant independent of the input, this yields Theorem 6.1.

7 Odd Cycle Transversal
An equivalent formulation of MAX CUT is to delete the minimum number of edges to make the graph bipartite. We can also consider the vertex deletion version of the problem. An odd cycle transversal of a graph $G$ is a subset $S \subseteq V(G)$ such that $G \setminus S$ is bipartite. In the ODD CYCLE TRANSVERSAL problem we are given a graph $G$ together with an integer $k$ and asked whether $G$ has an odd cycle transversal of size $k$. The proof of the following theorem can be found in [16].

Theorem 7.1. If ODD CYCLE TRANSVERSAL can be solved in $O^*((3 - \epsilon)^{pw(G)})$ time for $\epsilon > 0$, then SAT can be solved in $O^*((2 - \delta)^n)$ time for some $\delta > 0$.

8 Partition Into Triangles
A triangle packing in a graph $G$ is a collection of pairwise disjoint vertex sets $S_1, S_2, \ldots, S_t$ in $G$ such that $S_i$ induces a triangle in $G$ for every $i$. The size of the packing is $t$. If $V(G) = \bigcup_{i \leq t} S_i$ then the collection $S_1 \ldots S_t$ is a partition of $G$ into triangles. In the TRIANGLE PACKING problem we are given a graph $G$ and an integer $t$ and asked whether there is a triangle packing in $G$ of size at least $t$. In the PARTITION INTO TRIANGLES problem we are given a graph $G$ and asked whether $G$ can be partitioned into triangles. Notice that since PARTITION INTO TRIANGLES is the special case of TRIANGLE PACKING when the number of triangles is the number of vertices divided by 3, the bound of Theorem 8.1 holds for TRIANGLE PACKING as well. A proof of Theorem 8.1 can be found in [16].

Theorem 8.1. If PARTITION INTO TRIANGLES can be solved in $O^*((2 - \epsilon)^{pw(G)})$ for $\epsilon > 0$ then SAT can be solved in $O^*((2 - \delta)^n)$ time for some $\delta > 0$.

9 Conclusion
We have shown that for a number of basic graph problems, the best known algorithms parameterized by treewidth are optimal in the sense that base of the exponential dependence on treewidth is best possible. Recall that for DOMINATING SET and PARTITION INTO TRIANGLES, this running time was obtained quite recently using the new technique of fast subset sum convolu-
tions [28]. Thus it could have been a real possibility that the running time is improved for some other problems as well.

The results are proved under the Strong Exponential Time Hypothesis (SETH). While this hypothesis is relatively recent and might not be accepted by everyone, our results at least make a connection between rather specific graph problems and the very basic issue of better SAT algorithms. Our results suggest that one should not try to find better algorithms on bounded treewidth graphs for the problems considered in the paper: as this would disprove SETH, such an effort is better spent on trying to disprove SETH directly in the domain of satisfiability. Finally, we suggest the following open questions for future work:

- Can we prove similar tight lower bounds under the restriction that the graph is planar? Or is it possible to find improved algorithms on bounded treewidth planar graphs?
- Can we prove tight lower bounds for problems parameterized not by treewidth, but by something else? Naturally, one should look at problems where the algorithm or the the running time suggests that the best known algorithm is optimal. Possible candidates are the $O(2^k)$ time algorithm for STEINER Tree with $k$ terminals [2], and the $O(2^k)$ (resp., $O(3^k)$) time algorithms for EDGE BIPARTITION (resp., ODD CYCLE TRANSVERSAL) [17, 23].
- For the $q$-COLORING problem, we were able to prove lower bounds parameterized by the feedback vertex set number. Can we prove such bounds for the other problems as well?

References


