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 $(2-\epsilon)^n m^{\mathcal{O}(1)}$ time, we show that for any $\epsilon > 0$;

- Independent Set cannot be solved in time $(2-\epsilon)^{\mathbf{tw}(G)}|V(G)|^{\mathcal{O}(1)}$,
- Dominating Set cannot be solved in time $(3-\epsilon)^{\mathbf{tw}(G)}|V(G)|^{\mathcal{O}(1)}$,
- Max Cut cannot be solved in time $(2-\epsilon)^{\mathbf{tw}(G)}|V(G)|^{\mathcal{O}(1)}$,
- Odd Cycle Transversal cannot be solved in time $(3-\epsilon)^{\mathbf{tw}(G)}|V(G)|^{\mathcal{O}(1)}$,
- For any $q \geq 3$, q-Coloring cannot be solved in time $(q-\epsilon)^{\mathbf{tw}(G)}|V(G)|^{\mathcal{O}(1)}$,
- Partition Into Triangles cannot be solved in time $(2-\epsilon)^{\mathbf{tw}(G)}|V(G)|^{\mathcal{O}(1)}$.

Our lower bounds match the running times for the best known algorithms for the problems, up to the ϵ in the base.

1 Introduction

It is well-known that many NP-hard graph problems can be solved efficiently if the *treewidth* ($\mathbf{tw}(G)$) of the input graph G is bounded. For an example, an expository algorithm to solve Vertex Cover and Independent Set running in time $\mathcal{O}^*(4^{\mathbf{tw}(G)})$ is described in the algorithms textbook by Kleinberg and Tardos [15] (the $\mathcal{O}^*$ notation suppresses factors polynomial in the input size), while the book of Niedermeier [21] on fixed-parameter algorithms presents an algorithm with running time $\mathcal{O}^*(2^{\mathbf{tw}(G)})$. Similar algorithms, with running times on the form $\mathcal{O}^*(c^{\mathbf{tw}(G)})$ 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 $\mathcal{O}^*((2-\epsilon)^{\mathbf{tw}(G)})$ or
- Dominating Set can be solved in time $\mathcal{O}^*((3-\epsilon)^{\mathbf{tw}(G)})$ or
- Max Cut can be solved in time $\mathcal{O}^*((2-\epsilon)^{\mathbf{tw}(G)})$ or
- Odd Cycle Transversal can be solved in time $\mathcal{O}^*((3-\epsilon)^{\mathbf{tw}(G)})$ or
- there is a $q \geq 3$ such that q-Coloring can be solved in time $\mathcal{O}^*((q-\epsilon)^{\mathbf{tw}(G)})$ or
- Partition Into Triangles can be solved in time $\mathcal{O}^*((2-\epsilon)^{\mathbf{tw}(G)})$

then SAT can be solved in $\mathcal{O}^*((2-\delta)^n)$ 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 ϕ 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 $\mathcal{O}^*((2-\epsilon)^{\mathbf{tw}(G)})$, we need that the treewidth of the

[*]Department of Computer Science and Engineering, University of California, San Diego, USA. dlokshtanov@cs.ucsd.edu

[†]Institut für Informatik, Humboldt-Universität zu Berlin, Germany. Email: dmarx@cs.bme.hu. Supported in part by ERC Advanced Grant DMMCA and Hungarian National Research Fund OTKA 67651.

[‡]The Institute of Mathematical Sciences, India. saket@imsc.res.in

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 $\mathcal{O}^*((2-\epsilon)^{\mathbf{tw}(G)})$, 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 $\mathcal{O}^*((3-\epsilon)^{\mathbf{tw}(G)}) = \mathcal{O}^*(2^{\log(3-\epsilon)\mathbf{tw}(G)})$ for some $\epsilon > 0$, then we can solve SAT on input formulas with n-variables in time $\mathcal{O}^*((2-\delta)^n)$ for some $\delta > 0$. Therefore by naïvely equating the exponent in the previous sentence we get that we need to construct an instance for Dominating Set whose treewidth is essentially $\frac{n}{\log 3}$. In other words, each variable should increase 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* ($\mathbf{pw}$(G)) of the constructed instances and use the fact that for any graph G, $\mathbf{tw}(G) \leq \mathbf{pw}(G)$. 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 ϕ has at most k literals. Let $s_k = \inf\{\delta : k\text{-SAT } \textit{can be solved in } 2^{\delta n} \textit{ time}\}$. The Exponential Time Hypothesis conjectured by Impagliazzo, Paturi and Zane [14] is that $s_3 > 0$. In [14] it is shown that ETH is robust, that is $s_3 > 0$ 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 $\mathcal{O}^*(2^n)$ 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 $\mathcal{O}^*(4^{\mathbf{tw}(G)})$ time algorithm for Dominating Set, improving over the natural $\mathcal{O}^*(9^{\mathbf{tw}(G)})$ 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 $\mathcal{O}^*(3^{\mathbf{tw}(G)})$ algorithm for Dominating Set and a $\mathcal{O}^*(2^{\mathbf{tw}(G)})$ 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 $G' = G[V']$ of G is called a *subgraph induced by* V' if $E(G') = \{uv \in E(G) \mid u, v \in V'\}$. 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 $N[D] = \cup_{v\in D} N[v]$.

A *tree decomposition* of a graph G is a pair $(\mathcal{X}, T)$ where T is a tree and $\mathcal{X} = \{X_i \mid i \in V(T)\}$ 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 G is the minimum width over all tree decompositions of G. We denote by $\mathbf{tw}(G)$ 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 $\mathbf{pw}(G)$. 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 $\mathbf{ms}(G)$, 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 between $\mathbf{pw}(G)$ and $\mathbf{ms}(G)$, which we use for bounding the pathwidth of the graphs obtained in reduction.

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

An instance to SAT always consists of a boolean formula $\phi = C_1 \wedge \cdots \wedge C_m$ 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 τ to the variables is an element of $\{0,1\}^n$, and it satisfies the formula ϕ if for every clause C_i there is literal that is assigned 1 by τ. 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 τ. A group of variables satisfy 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 $\mathcal{O}^*((2-\epsilon)^{\mathbf{tw}(G)})$ for some $\epsilon > 0$ then* SAT *can be solved in $\mathcal{O}^*((2-\delta)^n)$ time for some $\delta > 0$.*

Construction. Given an instance ϕ to SAT we construct a graph G as follows. We assume that every

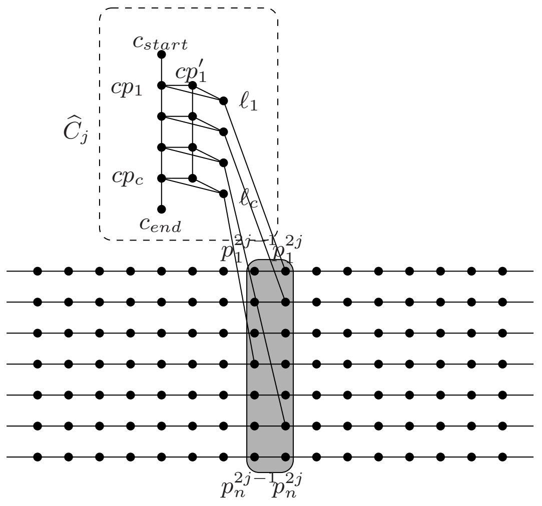

Figure 1: Reduction to Independent Set: clause gadget $\widehat{C}_j$ attached to the n paths representing the variables.

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 $\widehat{C}$ 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 ℓ_i we make a vertex ℓ_i in $\widehat{C}$ and make it adjacent to cp_i and cp'_i. Finally we add two vertices c_{start} and c_{end}, such that c_{start} is adjacent to cp_1 and c_{end} is adjacent to cp_c. Observe that the size of the maximum independent set of $\widehat{C}$ is $c+2$. Also, since c is even, any independent set of size $c+2$ in $\widehat{C}$ 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 $\widehat{C}$ that contains ℓ_i and none of ℓ_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 $p_i^1 \ldots p_i^{2m}$. The path P_i corresponds to the variable v_i. For every clause C_i of ϕ we make a gadget $\widehat{C}_i$. Now, for every variable v_i, if v_i occurs positively in C_j, we add an edge between p_i^{2j} and the literal corresponding to v_i in $\widehat{C}_j$. If v_i occurs negatively in C_j, we add an edge between p_i^{2j-1} and the literal corresponding to v_i in $\widehat{C}_j$. 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_j^{2m} in G_i with p_j^1 in G_{i+1} for every $j \leq n$. This concludes the construction of G.

Lemma 3.1. *If ϕ is satisfiable, then G has an indepen-*

dent set of size $(mn + \sum_{i \le m} |C_i| + 2)(n+1)$.

Proof. Consider a satisfying assignment to ϕ. 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, p_i^3, p_i^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, p_i^4, p_i^6 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 $\widehat{C}_j$ corresponding to a clause C_j. We will only consider the copy of $\widehat{C}_j$ in G_1 as the other copies can be dealt identically. Let use choose a true literal ℓ_a in C_j and let v_i be the corresponding variable. Consider the vertex ℓ_a in $\widehat{C}_j$. If v_i occurs positively in C_j then v_i is true. Then I does not contain p_i^{2j}, the only neighbour of ℓ_a outside of $\widehat{C}_j$. 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^{2j-1}, the only neighbour of ℓ_a outside of $\widehat{C}_j$. There is an independent set of size $|C_j|+2$ in $\widehat{C}$ that contains ℓ_a and none out of ℓ_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 $(\sum_{i \le m} |C_i| + 2)(n+1)$ vertices from clause gadgets are added to I, yielding that the size of I is $(mn + \sum_{i \le m} |C_i| + 2)(n+1)$, concluding the proof. □

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

Proof. Consider an independent set of G of size $(mn + \sum_{i \le m} |C_i| + 2)(n+1)$. The set I can contain at most m vertices from each copy of P_i for every $i \le n$ 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 $(mn + \sum_{i \le m} |C_i| + 2)(n+1)$ 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 $\widehat{C}_j$. 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_i 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 $i \le n$, I contains every second vertex of P_i. We make an assignment to the variables of ϕ 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 ϕ. Indeed, consider any clause C_j, and look at the gadget $\widehat{C}_j$. We know that I contains $|C_j|+2$ vertices from $\widehat{C}_j$ and hence I must contain a vertex ℓ_a in corresponding to a literal of C_j. Suppose ℓ_a is a literal of v_i. Since I contains ℓ_a, if ℓ_a occurs positively in C_j, then I can not contain p_i^{2j} and hence v_i is true. Similarly, if ℓ_a occurs negatively in C_j then I can not contain p_i^{2j-1} and hence v_i is false. In both cases v_i satisfies C_j and hence all clauses of ϕ are satisfied by the assignment. □

Lemma 3.3. $\mathbf{pw}(G) \le n+4$.

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 $\widehat{C}_j$. This gadget can now be cleaned with 3 additional searchers. After $\widehat{C}_j$ 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 $p_i^{2(j+1)}$. 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 $\mathbf{pw}(G) \le n+4$. □

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 $\mathcal{O}^*((3-\epsilon)^{\mathbf{pw}(G)})$ time for some $\epsilon > 0$ then* SAT *can be solved in $\mathcal{O}^*((2-\delta)^n)$ time for some $\delta > 0$.*

Construction. Given $\epsilon < 1$ and an instance ϕ to SAT we construct a graph G as follows. We first chose an integer p depending only on ϵ. Exactly how p is chosen will be discussed in the proof of Theorem 4.1. We group the variables of ϕ into groups $F_1, F_2, \ldots, F_t$, each of size at most $\beta = \lfloor \log 3^p \rfloor$. Hence $t = \lceil n/\beta \rceil$. We

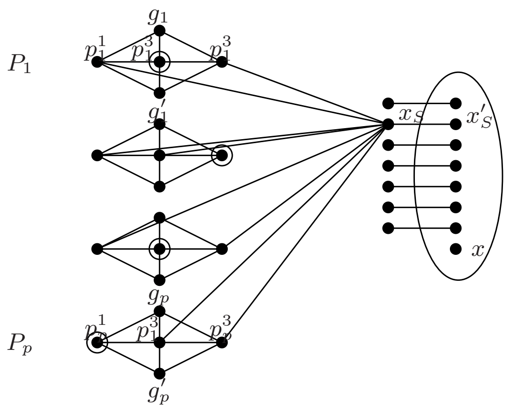

Figure 2: Reduction to DOMINATING SET: group gadget $\widehat{B}$. The set S is shown by the circled vertices.

now proceed to describe a "group gadget" $\widehat{B}$, which is central in our construction.

To build the group gadget $\widehat{B}$ we make p paths $P_1, \ldots, P_p$, where the path P_i contains the vertices p_i^1, p_i^2 and p_i^3. To each path P_i we attach two *guards* g_i and g_i', both of which are neighbours to p_i^1, p_i^2 and p_i^3. When the gadgets are attached to each other, the guards will not have any neighbours outside of their own gadget $\widehat{B}$, and will ensure that at least one vertex out of p_i^1, p_i^2 and p_i^3 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 $\widehat{B}$ 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 $\widehat{B}$, call them $\widehat{B}_i^j$ for $1 \leq j \leq m(2pt{+}1)$. For every fixed $i \leq t$ we connect the gadgets $\widehat{B}_i^1, \widehat{B}_i^2 \ldots, \widehat{B}_i^{m(2pt+1)}$ in a path-like manner. In particular, for every $j < m(2pt+1)$ and every $\ell \leq p$ we make an edge between p_ℓ^3 in the gadget $\widehat{B}_i^j$ with p_ℓ^1 in the gadget $\widehat{B}_i^{j+1}$. Now we make two new vertices h and h', with h adjacent to h', p_j^1 in $\widehat{B}_i^1$ for every $i \leq t$, $j \leq p$ and to p_j^3 in $\widehat{B}_i^{m(2pt+1)}$ 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 $\widehat{B}_i^1, \widehat{B}_i^2 \ldots, \widehat{B}_i^{m(2pt+1)}$ in a path-like manner.

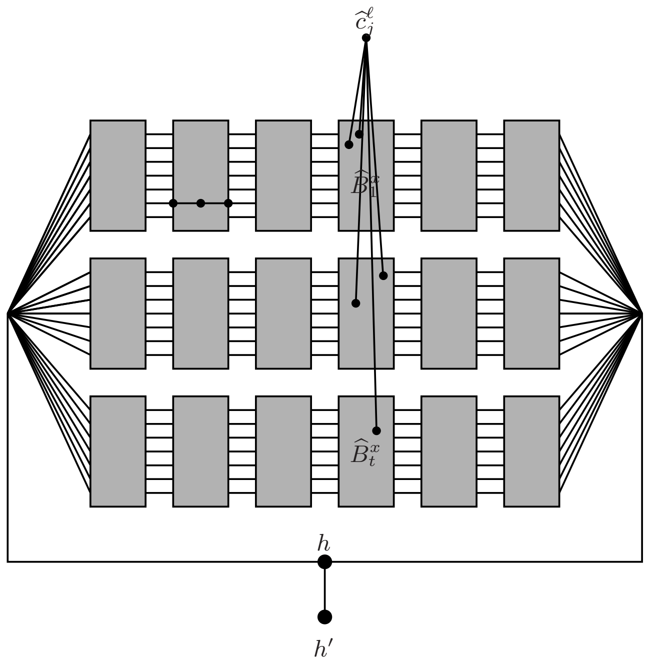

Figure 3: Reduction to DOMINATING SET: arranging the group gadgets. Note that $x = m\ell + j$, thus $\widehat{c}_j^\ell$ is attached to vertices in $\widehat{B}_1^x$, ..., $\widehat{B}_t^x$.

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 $\widehat{B}$ that picks exactly one vertex from each path P_j. Since there are at most 2^β different assignments to the variables in F_i, and there are $3^p \geq 2^\beta$ 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 ϕ are $C_1, \ldots, C_m$. For every clause C_j we make $2pt+1$ vertices $\widehat{c}_j^\ell$, one for each $0 \leq \ell < 2pt+1$. The vertex $\widehat{c}_j^\ell$ will be connected to the gadgets $\widehat{B}_i^{m\ell+j}$ 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 $\widehat{B}_i^{m\ell+j}$ adjacent to $\widehat{c}_j^\ell$. The best way to view this is that every clause C_j has $2pt+1$ private gadgets, $\widehat{B}_i^j, \widehat{B}_i^{m+j}, \ldots, \widehat{B}_i^{m2pt+j}$, 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 ϕ has a satisfying assignment, then G has a dominating set of size $(p+1)tm(2pt+1)+1$.*

Proof. Given a satisfying assignment to ϕ we construct a dominating set D of G that contains the vertex h and *exactly* $p+1$ vertices in each gadget $\widehat{B}_i^j$. 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 $\widehat{B}_i^j$ 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 $i \le t$ and j consider the vertices on the path P_j in the gadgets $\widehat{B}_i^\ell$ for every $\ell \le m(2pt+1)$. 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 $i \le t$ and $l \le m(2pt+1)$ and consider the gadget $\widehat{B}_i^\ell$. 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 $\widehat{c}_j^\ell$ for every $j \le m$ and $\ell < 2pt+1$. Fix a j and a ℓ 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 $i \le t$. 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 $\widehat{B}_i^{m\ell+j}$ and x'_S is adjacent to $\widehat{c}_j^\ell$. This concludes the proof. □

Lemma 4.2. *If G has a dominating set of size $(p+1)tm(2pt+1)+1$, then ϕ 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 $\widehat{B}_i^\ell$, D must dominate all the guards, namely g_j and g'_j for every $j \le p$, and also x. Thus D contains at least $p+1$ vertices from each gadget $\widehat{B}_i^\ell$ which in turn implies that D contains exactly $p+1$ vertices from each gadget $\widehat{B}_i^\ell$. The only way D can dominate g_j and g'_j for every j and in addition dominate x with only $p+1$ verticesis if D has one vertex from each P_j, $j \le p$ and in addition contains some vertex in $N[x]$. Let S be $D \cap P$ in $\widehat{B}_i^\ell$. 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 $1 \le i \le t$ there exists *one* $0 \le \ell \le 2tp$ such that for fixed i, $D \cap P$ is same in all the gadgets $\widehat{B}_i^{m\ell+r}$, $1 \le r \le m$. Consider a gadget $\widehat{B}_i^\ell$ and its follower, $\widehat{B}_i^{\ell+1}$. Let S be $D \cap P$ in $\widehat{B}_i^\ell$ and S' be $D \cap P$ in $\widehat{B}_i^{\ell+1}$. Observe that if S contains p_j^a in $\widehat{B}_i^\ell$ and p_j^b in $\widehat{B}_i^{\ell+1}$ then we must have $b \le a$. We call a consecutive pair *bad* if for some $j \le p$, D contains p_j^a in $\widehat{B}_i^\ell$ and p_j^b in $\widehat{B}_i^{\ell+1}$ 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 $\ell \in \{0, \dots, 2tp\}$ such that there are no bad pairs in $\widehat{B}_i^{m\ell+r}$ for all $1 \le i \le t$ and $1 \le r \le m$.

We make an assignment ϕ by reading off $D \cap P$ in each gadget $\widehat{B}_i^{m\ell+1}$. In particular, for every group F_i, we consider $S = D \cap P$ in the gadget $\widehat{B}_i^{m\ell+1}$. 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 $\widehat{c}_\ell^r$. We know that it is dominated by some x'_S in a gadget $\widehat{B}_i^{m\ell+r}$. 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 $\widehat{B}_i^{m\ell+1}$ to $\widehat{B}_i^{m\ell+r}$, this is exactly the assignment ϕ restricted to the group F_i. This concludes the proof. □

Lemma 4.3. $\mathbf{pw}(G) \le tp + \mathcal{O}(3^p)$

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

Proof. [Proof (of Theorem 4.1)] Suppose Dominating Set can be solved in $\mathcal{O}^*((3-\epsilon)^{\mathbf{pw}(G)}) = \mathcal{O}^*(3^{\lambda \mathbf{pw}(G)})$

time, where $\lambda = \log_3(3-\epsilon) < 1$. We choose p large enough such that $\lambda \cdot \frac{p}{\lceil p \log 3 \rceil} = \frac{\delta'}{\log 3}$ 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 $\mathcal{O}^*(3^{\lambda \mathbf{pw}(G)})$ 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 $\mathcal{O}^*(3^{\lambda \mathbf{pw}(G)}) \leq \mathcal{O}^*(3^{\lambda(tp+f(\lambda))}) \leq \mathcal{O}^*(3^{\lambda \frac{np}{\lceil p \log 3 \rceil}}) \leq \mathcal{O}^*(3^{\delta' \frac{n}{\log 3}}) \leq \mathcal{O}^*(2^{\delta'' n}) = \mathcal{O}^*((2-\delta)^n)$, for some $\delta'', \delta < 1$. This concludes the proof. □

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

Construction. Given an instance ϕ 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 $\widehat{v}_i$ for each variable v_i. For every clause C_j we make a gadget as follows. We make a path $\widehat{P}_j$ having $4|C_j|$ vertices. All the edges on $\widehat{P}_j$ have weight $3n$. Now, we make the first and last vertex of $\widehat{P}_j$ adjacent to x_0 with an edge of weight $3n$. Thus the path $\widehat{P}_j$ plus the edges from the first and last vertex of $\widehat{P}_j$ to x_0 form an odd cycle $\widehat{C}_j$. We will say that the first, third, fifth, etc, vertices are on *odd positions* on $\widehat{P}_j$ 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 $\widehat{P}_j$ and make $\widehat{v}$ adjacent to p and p's successor on $\widehat{P}_j$ 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 $\widehat{P}_j$ and make $\widehat{v}$ adjacent to p and p's successor on $\widehat{P}_j$ with edges of weight 1. We make sure that each vertex on $\widehat{P}_j$ receives an edge at most once in this process. There are more than enough vertices on $\widehat{P}_j$ 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 ϕ is satisfiable, then G_w has a cut of weight at least t.*

Proof. Suppose ϕ is satisfiable. We put x_0 in V_0 and for every variable v_i we put $\widehat{v}_i$ in V_1 if v_i is true and $\widehat{v}_i$ 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 $\widehat{P}_j$. We put the first vertex on $\widehat{P}_j$ in V_1, the second in V_0 and then we proceed along $\widehat{P}_j$ putting every second vertex into V_1 and V_0 until we reach p_j. The successor p'_j of p_j on $\widehat{P}_j$ is put into the same set as p_j. Then we continue along $\widehat{P_j}$ 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 $\widehat{P}_j$ 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 $\widehat{P}_j$ except for $p_jp'_j$ are crossing, and the two edges to x_0 from the first and last vertex of $\widehat{P}_j$ 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 $\widehat{P}_j$. Thus both p_j and his successor p'_j are in V_0 and hence both v_ip_j and $v_ip'_j$ 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 $\widehat{v}_{i'}$ on $\widehat{P}_j$ appear in V_0 and one in V_1, so exactly one edge from $v_{i'}$ to $\widehat{P}_j$ 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. □

Lemma 5.2. *If G_w has a cut of weight at least t, then ϕ 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 $\widehat{C}_j$ 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 $\widehat{P}_j$ 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 $\widehat{P}_j$ are crossing and the first vertex of $\widehat{P}_j$ is in V_1. Using the new partition the weight of the crossing edges in the cycle $\widehat{C}_j$ is at least $12|C_j|$ and the edges not incident to $\widehat{P}_j$ 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 $\widehat{C}_j$ is not crossing.

Given the cut (V_0, V_1) we set each variable v_i to true if $\widehat{v}_i \in V_1$ and v_i to false otherwise. Consider a clause C_j and a variable v_i that appears in C_j. Let uv be the edge of $\widehat{C}'_j$ that is not crossing. If there is a variable $\widehat{v}_i$ adjacent to both u and v, then it is possible that both $\widehat{v}_i u$ and $\widehat{v}_i v$ are crossing. For every other variable $v_{i'}$ in C_j, at most one of the edges from $\widehat{v}_{i'}$ to $\widehat{P}_j$ is crossing. Thus, the weight of the edges that are crossing in the gadget $\widehat{C}_j$ 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 $\widehat{C}_j$ with sum of their weights exactly equal to $12n|C_j|+|C_j|+1$. It follows that there is a vertex $\widehat{v}_i$ adjacent to both u and v such that both $\widehat{v}_i u$ and $\widehat{v}_i v$ are crossing.

If v_i occurs in C_j positively then u is on an even position and hence, $u \in V_0$. Since $\widehat{v}_i u$ 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 negated then u is on an odd position and hence, $u \in V_1$. Since $\widehat{v}_i u$ 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. $\mathbf{pw}(G) \le n + 5$.

Proof. We give a search strategy to clean G with $n+5$ searchers. We place one searcher on each vertex $\widehat{v}_i$ and one searcher on x_0. Then one can search the gadgets $\widehat{H}_j$ one by one. In G_w it is sufficient to use 2 searchers for each $\widehat{H}_j$, 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. □

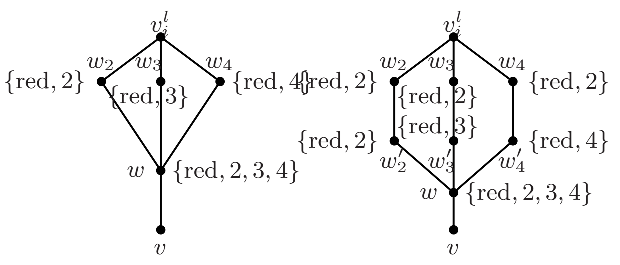

Figure 4: Reduction to q-Coloring: the way the connector connects a vertex v_i^l with v for a particular "bad color" $x \in [q] \setminus \{\mu_i(v_i^l)\}$. The left side shows the case $x = \text{red} = 1$, the right side $x = 2$ $(q = 4)$.

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 $\mu : V(G) \to [q]$. A q-coloring μ 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 $L(v) \subseteq [q]$ of admissible colors. A *proper list coloring* of G is a function $\mu : V(G) \to [q]$ such that μ is a proper coloring of G that satisfies $\mu(v) \in L(v)$ 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 $\mathbf{fvs}(G)$ the size of the smallest such set. It is well-known that $\mathbf{tw}(G) \le \mathbf{fvs}(G) + 1$. Unlike in the other sections, where we give lower bounds for algorithms parameterized by $\mathbf{pw}(G)$, the following theorem gives also a lower bound for algorithms parameterized by $\mathbf{fvs}(G)$. 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 $\mathbf{fvs}(G)$ for the problems considered in the other sections.

Theorem 6.1. *If q-Coloring can be solved in $\mathcal{O}^*((q-\epsilon)^{\mathbf{fvs}(G)})$ or $\mathcal{O}^*((3-\epsilon)^{\mathbf{pw}(G)})$ time for some $\epsilon > 0$, then* SAT *can be solved in $\mathcal{O}^*((2-\delta)^n)$ 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 $\mathcal{O}^*((q-\epsilon)^{\mathbf{fvs}(G)})$ time if and only if q-List Coloring can.

Depending on ϵ and q we choose a parameter p. Now, given an instance ϕ 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 ϕ 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 ϕ into t groups $F_1, \ldots, F_t$ of size $\lfloor \log q^p \rfloor$. Thus $t = \lceil \frac{n}{\lfloor \log q^p \rfloor} \rceil$. We will call an assignment of truth values to the variables in a group F_i a *group assignment*. We will say that a group assignment satisfies a clause C_j of ϕ 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 $v_i^1, \ldots, v_i^p$. 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 $q^p \geq 2^{|F_i|}$ possible colorings of V_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 assignments of F_i.

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

For every vertex v on $\widehat{P}_j$ 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 μ_i. We build a *connector* whose role is to enforce that v can be red only if coloring μ_i appears on V_i. To build the connector, for each vertex $v_i^l \in V_i$ and color $x \in [q] \setminus \{\mu_i(v_i^l)\}$ we do the following.

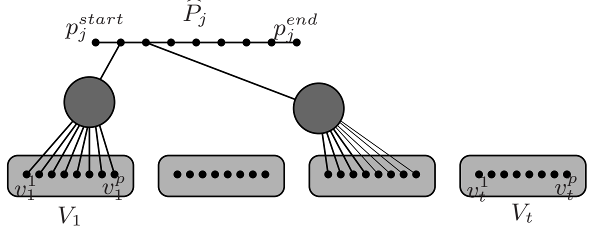

Figure 5: Reduction to q-Coloring. The t groups of vertices $V_1, \ldots, V_t$ represent the t groups of variables $F_1, \ldots, F_t$ (each of size $\lceil \log q^p \rceil$). Each vertex of the clause path $\widehat{P}_j$ is connected to one group V_i via a connector.

- If x is red, then we add one vertex w_y for every color y except for red. We make w_y adjacent to v_i^l and the list of w_y is $\{\text{red}, y\}$. Then we add a vertex w which is adjacent to all vertices w_y and v, and whose list is all of $[q]$.

- If x is not red, we add two vertices w_y and w'_y for each color y except for red. We make w_y adjacent to v_i^l and w'_y adjacent to w_y. The list of w_y is $\{x, \text{red}\}$ while the list of w'_y is $\{y, \text{red}\}$. Finally we add a vertex w adjacent to w'_y for all y and to v. The list of w is all of $[q]$.

Notice that in the above construction we have reused the names w, w_y and w'_y for many different vertices: in each connector, there is a separate vertex w for each vertex $v_i^l \in V_i$ and color $x \in [q] \setminus \{\mu_i(v_i^l)\}$. Building a connector for each vertex v on $\widehat{P}_j$ concludes the construction of the clause gadget $\widehat{C}_j$, 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 $\widehat{P}_j$ and coloring μ_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 μ_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 μ_i appears on V_i.*

Proof. 1. For each vertex $v_i^l \in V_i$ and color $x \in [q] \setminus \{\mu_i(v_i^l)\}$ we do the following.

- If x is red then in the construction of $\widehat{C}_j$ we added a vertex w_y with list $\{y, \text{red}\}$ for every color $y \neq$ red adjacent to v_i^l, and a vertex w with list $[q]$ adjacent to w_y for every $y \neq$ red. If v_i^l 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, v_i^l is not colored red, we can color w_y red for every y. Then all the neighbours of w 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 $\widehat{C}_j$ 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^l and for every $y \neq$ red the vertex w'_y is adjacent to w_y. Finally w is adjacent to al the vertices w'_y and to v. For every y the list of w_y is $\{x, \text{red}\}$ while the list of w'_y is $\{y, \text{red}\}$. The list of w is $[q]$. If v_i^l 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^l 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 neighours 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 $v_i^l \in V_i$ and color $x \in [q] \setminus \{\mu_i(v_i^l)\}$ we do the following.

- If x is red then in the construction of $\widehat{C}_j$ we added a vertex w_y with list $\{y, \text{red}\}$ for every color $y \neq$ red adjacent to v_i^l, and a vertex w with list $[q]$ adjacent to w_y for every $y \neq$ red. Since $v_{i'}^l$ is not colored red by μ_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 $\widehat{C}_j$ 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^l and for every $y \neq$ red the vertex w'_y 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 $\{x, \text{red}\}$ while the list of w'_y is $\{y, \text{red}\}$. The list of w is $[q]$. Since μ_i colors v_i^l 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 neighours 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 $v_i^l \in V_i$ has been colored with a color $x \neq \mu_i(v_i^l)$. There are two cases. If x is red, then in the construction we added vertices w_y adjacent to v_i^l for every color $y \neq$ red. Also we added a vertex w adjacent to v and to w_y for each $y \neq$ red. The list of w_y is $\{\text{red}, y\}$ and hence 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^l and had $\{x, \text{red}\}$ as its list. Since v_i^l 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 $\{\text{red}, y\}$ 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 ϕ is satisfiable, then G has a proper list-coloring.*

Proof. Starting from a satisfying assignment of ϕ we construct a coloring γ of G. The assignment to ϕ 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 γ 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 $\widehat{C}_j$ 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 $\widehat{C}_j$. 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 $\widehat{P}_j$ that corresponds to this assignment. We set $\gamma(v)$ as red (that is, γ colors v red), p_j^{start} is colored white and p_j^{end} is colored with its only admissible color, namely black if $|V(\widehat{P}_j)|$ is even and white if $|V(\widehat{P}_j)|$ is odd. The remaining vertices of $\widehat{P}_j$ 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 μ_i corresponding to the group assignment F_i. For every other vertex u on $\widehat{P}_j$, 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 $\widehat{C}_j$ for every clause C_j, we can complete γ to a proper list-coloring of G. □

LEMMA 6.3. *If G has a proper list-coloring γ, then ϕ is satisfiable.*

Proof. Given γ we construct an assignment to the variables of ϕ as follows. For every group F_i of variables, if γ 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 ϕ.

Consider a clause C_j and the corresponding gadget $\widehat{C}_j$. By a simple parity argument, $\widehat{P}_j$ can not be colored using only the colors black and white. Thus, some vertex v on $\widehat{P}_j$ is colored red. The vertex v corresponds to a group assignment of some group F_i that satisfies $\widehat{C}_j$. As v is red, Lemma 6.1(3) implies that V_i is colored with the coloring μ_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 $\bigcup_{i \le t} V_i$ form a feedback vertex set of G. Furthermore, $\mathbf{pw}(G) \le pt + 4$*

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

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

Proof. Let $\mathcal{O}^*((q-\epsilon)^{\mathbf{fvs}(G)})= O^*(q^{\lambda \mathbf{fvs}(G)})$ 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 ϕ 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 $p\lceil \frac{n}{\lfloor p \log q \rfloor} \rceil$. The choice of p implies that

$$\lambda p \lceil \frac{n}{\lfloor p \log q \rfloor} \rceil \le \lambda p \frac{n}{(p-1)\log q} + p \le \delta' \frac{n}{\log q} + p \le \delta'' n,$$

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

Finally, observe that we can reduce q-LIST-COLORING to q-COLORING by adding a clique $Q = \{q_1, \ldots, q_c\}$ 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 $\mathcal{O}^*((3-\epsilon)^{\mathbf{pw}(G)})$ time for $\epsilon > 0$, then* SAT *can be solved in $\mathcal{O}^*((2-\delta)^n)$ 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 $V(G) = \bigcup_{i \le t} S_i$ 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 $\mathcal{O}^*((2-\epsilon)^{\mathbf{pw}(G)})$ for $\epsilon > 0$ then* SAT *can be solved in $\mathcal{O}^*((2-\delta)^n)$ time for some $\delta > 0$.*

9 Conclusion

We have showed 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 convolutions [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 $\mathcal{O}(2^k)$ time algorithm for STEINER TREE with k terminals [2], and the $\mathcal{O}(2^k)$ (resp., $\mathcal{O}(3^k)$) time algorithms for EDGE BIPARTIZATION (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

[1] J. ALBER AND R. NIEDERMEIER, *Improved tree decomposition based algorithms for domination-like problems*, in LATIN, 2002, pp. 613–628.

[2] A. BJÖRKLUND, T. HUSFELDT, P. KASKI, AND M. KOIVISTO, *Fourier meets möbius: fast subset convolution*, in STOC, 2007, pp. 67–74.

[3] C. CALABRO, R. IMPAGLIAZZO, AND R. PATURI, *The complexity of satisfiability of small depth circuits*, in IWPEC, 2009, pp. 75–85.

[4] J. CHEN, X. HUANG, I. A. KANJ, AND G. XIA, *On the computational hardness based on linear FPT-reductions*, J. Comb. Optim., 11 (2006), pp. 231–247.

[5] ———, *Strong computational lower bounds via parameterized complexity*, J. Comput. Syst. Sci., 72 (2006), pp. 1346–1367.

[6] E. D. DEMAINE, F. V. FOMIN, M. T. HAJIAGHAYI, AND D. M. THILIKOS, *Subexponential parameterized algorithms on bounded-genus graphs and -minor-free graphs*, J. ACM, 52 (2005), pp. 866–893.

[7] E. D. DEMAINE AND M. HAJIAGHAYI, *The bidimensionality theory and its algorithmic applications*, Comput. J., 51 (2008), pp. 292–302.

[8] D. EPPSTEIN, *Diameter and treewidth in minor-closed graph families*, Algorithmica, 27 (2000), pp. 275–291.

[9] S. FIORINI, N. HARDY, B. A. REED, AND A. VETTA, *Planar graph bipartization in linear time*, Discrete Applied Mathematics, 156 (2008).

[10] J. FLUM AND M. GROHE, *Parameterized Complexity Theory*, Springer, Berlin, 2006.

[11] F. FOMIN, P. GOLOVACH, D. LOKSHTANOV, AND S. SAURABH, *Algorithmic lower bounds for problems parameterized by clique-width*, in SODA, 2010, pp. 493–502.

[12] F. V. FOMIN, S. GASPERS, S. SAURABH, AND A. A. STEPANOV, *On two techniques of combining branching and treewidth*, Algorithmica, 54 (2009), pp. 181–207.

[13] R. IMPAGLIAZZO AND R. PATURI, *On the complexity of k-sat*, J. Comput. Syst. Sci., 62 (2001), pp. 367–375.

[14] R. IMPAGLIAZZO, R. PATURI, AND F. ZANE, *Which problems have strongly exponential complexity?*, J. Comput. Syst. Sci., 63 (2001), pp. 512–530.

[15] J. KLEINBERG AND E. TARDOS, *Algorithm Design*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[16] D. LOKSHTANOV, D. MARX, AND S. SAURABH, *Known algorithms on graphs of bounded treewidth are probably optimal*, CoRR, abs/1007.5450 (2010).

[17] D. LOKSHTANOV, S. SAURABH, AND S. SIKDAR, *Simpler parameterized algorithm for oct*, in IWOCA, 2009, pp. 380–384.

[18] D. MARX, *Can you beat treewidth?*, in FOCS, 2007, pp. 169–179.

[19] ———, *On the optimality of planar and geometric approximation schemes*, in FOCS, 2007, pp. 338–348.

[20] D. MÖLLE, S. RICHTER, AND P. ROSSMANITH, *Enumerate and expand: Improved algorithms for connected vertex cover and tree cover*, Theory Comput. Syst., 43 (2008), pp. 234–253.

[21] R. NIEDERMEIER, *Invitation to fixed-parameter algorithms*, vol. 31 of Oxford Lecture Series in Mathematics and its Applications, Oxford University Press, Oxford, 2006.

[22] M. PĂTRAŞCU AND R. WILLIAMS, *On the possibility of faster sat algorithms*, in Proc. 21st ACM/SIAM Symposium on Discrete Algorithms (SODA), 2010. To appear.

[23] B. REED, K. SMITH, AND A. VETTA, *Finding odd cycle transversals*, Operations Research Letters, 32 (2004), pp. 299–301.

[24] A. D. SCOTT AND G. B. SORKIN, *Linear-programming design and analysis of fast algorithms for max 2-csp*, Discrete Optimization, 4 (2007), pp. 260–287.

[25] A. TAKAHASHI, S. UENO, AND Y. KAJITANI, *Mixed searching and proper-path-width*, Theor. Comput. Sci., 137 (1995), pp. 253–268.

[26] J. A. Telle and A. Proskurowski, *Practical algorithms on partial k-trees with an application to domination-like problems*, in WADS, 1993, pp. 610–621.

[27] D. M. Thilikos, M. J. Serna, and H. L. Bodlaender, *Cutwidth i: A linear time fixed parameter algorithm*, J. Algorithms, 56 (2005), pp. 1–24.

[28] J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith, *Dynamic programming on tree decompositions using generalised fast subset convolution*, in ESA, 2009, pp. 566–577.

[29] J. M. M. van Rooij, J. Nederlof, and T. C. van Dijk, *Inclusion/exclusion meets measure and conquer*, in ESA, 2009, pp. 554–565.