T C C C

V. A

Institute of Mathematical Sciences, CIT Campus, Taramani
Chennai 600113, India
arvind@imsc.res.in
http://www.imsc.res.in/~arvind

The quest for fast exact exponential-time algorithms and fast parameterized algorithms for NP-hard problems has been an exciting and fruitful area of research over the last decade. There is an accompanying theory of hardness based on the Exponential Time Hypothesis (ETH) and the Strong Exponential Time Hypothesis (SETH). In this interesting article, Daniel Lokshtanov, Dániel Marx, and Saket Saurabh describe recent progress on algorithmic lower bound results based on these hypotheses.

L E
T H

Daniel Lokshtanov* Dániel Marx† Saket Saurabh‡

Abstract

In this article we survey algorithmic lower bound results that have been obtained in the field of exact exponential time algorithms and parameterized complexity under certain assumptions on the running time of algorithms

*Department of Computer Science and Engineering, University of California, USA. dlokshtanov@ucsd.edu

†Institut für Informatik, Humboldt-Universiät , Berlin, Germany. dmarx@cs.bme.hu

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

solving CNF-S , namely Exponential time hypothesis (ETH) and Strong Exponential time hypothesis (SETH).

1 Introduction

The theory of NP-hardness gives us strong evidence that certain fundamental combinatorial problems, such as 3SAT or 3-C , are unlikely to be polynomial-time solvable. However, NP-hardness does not give us any information on what kind of super-polynomial running time is possible for NP-hard problems. For example, according to our current knowledge, the complexity assumption P $\neq$ NP does not rule out the possibility of having an $n^{O(\log n)}$ time algorithm for 3SAT or an $n^{O(\log \log k)}$ time algorithm for k-C , but such incredibly efficient super-polynomial algorithms would be still highly surprising. Therefore, in order to obtain qualitative lower bounds that rule out such algorithms, we need a complexity assumption stronger than P $\neq$ NP.

Impagliazzo, Paturi, and Zane [36, 35] introduced the Exponential Time Hypothesis (ETH) and the stronger variant, the Strong Exponential Time Hypothesis (SETH). These complexity assumptions state lower bounds on how fast satisfiability problems can be solved. These assumptions can be used as a basis for qualitative lower bounds for other concrete computational problems.

The goal of this paper is to survey lower bounds that can be obtained by assuming ETH or SETH. We consider questions of the following form (we employ the O^* notation which suppresses factors polynomials in input size):

- C N on an n-vertex graph can be solved in time $O^*(2^n)$. Can this be improved to $2^{o(n)}$ or to $(2-\epsilon)^n$?
- k-I S on an n-vertex graph can be solved in time $n^{O(k)}$. Can this be improved to $n^{o(k)}$?
- k-I S on an n-vertex *planar* graph can be solved in time $O^*(2^{O(\sqrt{k})})$. Can this be improved to $2^{o(\sqrt{k})}$?
- D S on a graph with treewidth w can be solved in time $O^*(3^w)$. Can this be improved to $O^*((3-\epsilon)^w)$?
- H S over an n-element universe is solvable in time $O^*(2^n)$. Can this be improved to $2^{o(n)}$ or to $(2-\epsilon)^n$?

As we shall see, if ETH or SETH hold then many of these questions can be answered negatively. In many cases, these lower bounds are tight: they match (in

some sense) the running time of the best known algorithm for. Such results provide evidence that the current best algorithms are indeed the best possible.

The main focus of this survey is to state consequences of ETH and SETH for concrete computational problems. We avoid in-depth discussions of how believable these conjectures are, what complexity-theoretic consequences they have, or what other techniques could be used to obtain similar results.

Parameterized complexity. Many of the questions raised above can be treated naturally in the framework of parameterized complexity introduced by Downey and Fellows [22]. The goal of parameterized complexity is to find ways of solving NP-hard problems more efficiently than brute force, by restricting the "combinatorial explosion" in the running time to a parameter that for reasonable inputs is much smaller than the input size. Parameterized complexity is basically a two-dimensional generalization of "P vs. NP" where in addition to the overall input size n, one studies how a relevant secondary measurement affects the computational complexity of problem instances.

This additional information can be the size or quality of the output solution sought for, or a structural restriction on the input instances considered, such as a bound on the treewidth of the input graph. Parameterization can be employed in many different ways; for general background on the theory, the reader is referred to the monographs [22, 26, 48].

The two-dimensional analogue (or generalization) of P, is solvability within a time bound of $O(f(k)n^c)$, where n is the total input size, k is the parameter, f is some (usually computable) function, and c is a constant that does not depend on k or n. Parameterized decision problems are defined by specifying the input, the parameter, and the question to be answered. A parameterized problem that can be solved in $O(f(k)n^c)$ time is said to be *fixed-parameter tractable* (FPT). Just as NP-hardness is used as evidence that a problem probably is not polynomial time solvable, there exists a hierarchy of complexity classes above FPT, and showing that a parameterized problem is hard for one of these classes gives evidence that the problem is unlikely to be fixed-parameter tractable. The main classes in this hierarchy are:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq XP$$

The principal analogue of the classical intractability class NP is $W[1]$, which is a strong analogue, because a fundamental problem complete for $W[1]$ is the k-S H P N T M (with unlimited nondeterminism and alphabet size) — this completeness result provides an analogue of Cook's Theorem in classical complexity. In particular this means that an FPT algorithm for any $W[1]$ hard problem would yield a $O(f(k)n^c)$ time algorithm for k-S H P N T M . A conve-

nient source of $W[1]$-hardness reductions is provided by the result that k-C is complete for $W[1]$. Other highlights of the theory include that k-D S , by contrast, is complete for $W[2]$. XP is the class of all problems that are solvable in time $O(n^{g(k)})$.

There is also a long list of NP-hard problems that are FPT under various parameterizations: finding a vertex cover of size k, finding a cycle of length k, finding a maximum independent set in a graph of treewidth at most k, etc. The form of the function $f(k)$ in the running time of these algorithms vary drastically. In some cases, for example in results obtained from Graph Minors theory, the function $f(k)$ is truly humongous (a tower of exponentials), making the result purely of theoretical interest. On the other hand, in many cases $f(k)$ is a moderately growing exponential function: for example, $f(k)$ is 1.2738^k in the current fastest algorithm for finding a vertex cover of size k [14], which can be further improved to 1.1616^k in the special case of graphs with maximum degree 3 [55]. For some problems, $f(k)$ can be even subexponential (e.g., $c^{\sqrt{k}}$) [21, 20, 19, 2]. For more background on parameterized algorithms, the reader is referred to the monographs [22, 26, 48].

Exact algorithms. At this point we take a slight detour and talk about the exact exponential algorithms, which will be central to this survey. Every problem in NP can be solved in time $2^{n^{O(1)}}$ by brute force - i.e by enumerating all candidates for the witness. While we do not believe that *polynomial* time algorithms for NP-complete problems exist, many NP-complete problems admit exponential time algorithms that are dramatically faster than the brute force algorithms. For some classical problems, such as S S , G C or H C such algorithms [32, 40, 3] were known even before the discovery of NP-completeness. Over the last decade, a subfield of algorithms devoted to developing faster exponential time algorithms for NP-hard problems has emerged. A myriad of problems have been shown to be solvable much faster than by brute force, and a variety of algorithm design techniques for exponential time algorithms has been developed. Some problems, such as I S and D S have seen a chain of improvements [28, 53, 50, 39], each new improvement being smaller than the previous. For these problems the running time of the algorithms on graphs on n vertices seems to converge towards $O(c^n)$ for some unknown c. For other problems, such as G C or T S , non-trivial solutions have been found, but improving these algorithms further seems to be out of reach [5]. For other problems yet, such as CNF-S or H S , no algorithms faster than brute force have been discovered. We would refer to the book of Fomin and Kratsch for more information on exact exponential time algorithms [29].

For the purpose of this survey we will not distinguish between exact and parameterized algorithms. Instead we will each time explicitly specify in terms of which parameter the running time is measured. For an example an exact exponential time algorithm for I S could be viewed as parameterized algorithm where the parameter is the number of vertices in the input graph. Such a perspective allows us to discuss complexity theory for both exact and parameterized algorithms in one go.

Organization. The survey is organized as follows. In Section 2, we give all the necessary definitions and introduce our complexity theory assumptions. We have organized the results topic wise. In Sections 3 and 4 we give various algorithmic lower bounds on problems in the field of exact algorithms and parameterized algorithms, respectively. We look at lower bounds for problems that are known to be W[1]-hard in Section 5. Section 6 deals with structural parameterizations. More precisely, in this section we give lower bounds results on problems parameterized by the treewidth of the input graph. Finally, we conclude with some remarks and open problems in Section 7.

Notation. We use $G = (V, E)$ to denote a graph on vertex set V and the edge set E. For a subset S of V, the *subgraph of G induced by S* is denoted by $G[S]$ and it is defined as the subgraph of G with vertex set S and edge set $\{(u, v) : u, v \in S\}$. By $N_G(u)$ we denote the (open) neighborhood of u, that is, the set of all vertices adjacent to u. Similarly, for a subset $T \subseteq V$, we define $N_G(T) = (\cup_{v \in T} N_G(v)) \setminus T$. A r-CNF formula $\phi = c_1 \wedge \cdots \wedge c_m$ is a boolean formula where each clause is a disjunction of literals and has size at most r. By $[k]$ we denote the set $\{1, 2, \ldots, k\}$.

2 Complexity Theory Assumptions

In this section we outline the complexity theory assumptions that is central to this survey. We start with a few definitions from parametrized complexity. We mainly follow the notation of Flum and Grohe [26]. We describe decision problems as languages over a finite alphabet Σ.

Definition 2.1. *Let Σ be a finite alphabet.*

(1) A parameterization *of Σ^* is a polynomial time computable mapping $\kappa : \Sigma^* \to \mathbb{N}$.*

(2) A parameterized problem *(over Σ) is a pair (Q, κ) consisting of a set $Q \subseteq \Sigma^*$ of strings over Σ and a parameterization κ of Σ^*.*

For a parameterized problem (Q, κ) over alphabet Σ, we call the strings $x \in \Sigma^*$ the *instances* of Q or (Q, κ) and the number of $\kappa(x)$ the corresponding *parameters*. We usually represent a parameterized problem on the form

Instance:	$x \in \Sigma^*$.
Parameter:	$\kappa(x)$.
Problem:	Decide whether $x \in Q$.

Very often the parameter is also a part of the instance. For example, consider the following parameterized version of the minimum feedback vertex set problem, where the instance consists of a graph G and a positive integer k, the problem is to decide whether G has a feedback vertex set, a set of vertices whose removal destroys all cycles in the graph, of k elements.

F V S	
Instance:	A graph G, and a non-negative integer k.
Parameter:	k.
Problem:	Decide whether G has a feedback vertex set with at most k elements.

In this problem the instance is the string (G, k) and $\kappa(G, k) = k$. When the parameterization κ is defined as $\kappa(x, k) = k$, the parameterized problem can be defined as subsets of $\Sigma^* \times \mathbb{N}$. Here the parameter is the second component of the instance. In this survey we use both notations for parameterized problems.

Definition 2.2. *A parameterized problem (Q, κ) is* fixed-parameter tractable *if there exists an algorithm that decides in $f(\kappa(x)) \cdot n^{O(1)}$ time whether $x \in Q$, where $n := |x|$ and f is a computable function that does not depend on n. The algorithm is called a* fixed-parameter algorithm *for the problem. The complexity class containing all fixed-parameter tractable problems is called FPT.*

A common way to obtain lower bounds is by reductions. A reduction from one problem to another is just a proof that a "too fast" solution for the latter problem would transfer to a too fast solution for the former. The specifics of the reduction varies based on what we mean by "too fast". The next definition is of a kind of reductions that preserves fixed-parameter tractability.

Definition 2.3. *Let (Q, κ) and (Q', κ') be two parameterized problems over the alphabet Σ and Σ', respectively. An* fpt reduction *(more precisely fpt many-one reduction) from (Q, κ) to (Q', κ') is a mapping $R : \Sigma^* \to (\Sigma')^*$ such that:*

1. *For all $x \in \Sigma^*$ we have $x \in Q$ if and only if $R(x) \in Q'$.*

2. *R is computable by an fpt-algorithm (with respect to κ).*

3. *There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.*

It can be verified that fpt reductions work as expected: if there is an fpt reduction from (Q, κ) to (Q', κ') and $(Q', \kappa') \in$ FPT, then $(Q, \kappa) \in$ FPT as well. We now define the notion of *subexponential* time algorithms.

Definition 2.4. SUBEPT *is the class of parameterized problems (P, κ) where P can be solved in time $2^{\frac{\kappa(x)}{s(\kappa(x))}}|x|^{O(1)} = 2^{o(\kappa(x))}|x|^{O(1)}$. Here, $s(k)$ is a monotonically increasing unbounded function. A problem P in* SUBEPT *is said to have subexponential algorithms.*

A useful observation is that an "arbitrarily good" exponential time algorithm implies a subexponential time algorithm and vice versa.

Proposition 2.5 ([26])**.** *A parameterized problem (P, κ) is in SUBEPT if and only if there is an algorithm that for every fixed $\epsilon > 0$ solves instances x of P in time $2^{\epsilon\kappa(x)}|x|^c$ where c is independent of x and ϵ.*

The r-CNF-S problem is a central problem in computational complexity, as it is the canonical NP-complete problem. We will use this problem as a basis for our complexity assumptions.

r-CNF-S	
Instance:	A r-CNF formula F on n variables and m clauses.
Parameter 1:	n.
Parameter 2:	m.
Problem:	Decide whether there exists a $\{0, 1\}$ assignment to the variables of F such that it is satisfiable?.

It is trivial to solve 3-CNF-S it time $2^n \cdot (n + m)^{O(1)}$. There are better algorithms for 3-CNF-S , but all of them have running time of the form $c^n \cdot (n+m)^{O(1)}$ for some constant $c > 1$ (the current best algorithm runs in time $O(1.30704^n)$ [33]). Our first complexity hypothesis, formulated by Impagliazzo, Paturi and Zane [37], states that every algorithm for 3-CNF-S has this running time, that is, the problem has no subexponential time algorithms.

> **Exponential Time Hypothesis (ETH)** [37]: There is a positive real s such that 3-CNF-S with parameter n cannot be solved in time $2^{sn}(n + m)^{O(1)}$.

In particular, ETH states that 3-CNF-S with parameter n cannot be solved in $2^{o(n)}(n+m)^{O(1)}$ time. We will use this assumption to show that several other problems do not have subexponential-time algorithms either. To transfer this hardness assumption to other problems, we need a notion of reduction that preserves solvability in subexponential time. It is easy to see that a polynomial-time fpt-reduction that increases the parameter only linearly (that is, $\kappa'(R(x)) = O(\kappa(x))$ holds for every instance x) preserves subexponential-time solvability: if the target problem (Q', κ') is in SUBEPT, then so is the source problem (Q, κ). Most of the reductions in this survey are on this form. However, it turns out that sometimes a more general form of subexponential time reductions, introduced by Impagliazzo, Paturi, and Zane [37], are required. Essentially, we allow the running time of the reduction to be subexponential and the reduction to be a Turing reduction rather than a many-one reduction:

Definition 2.6. *A* SERF-T *reduction from parameterized problem* (A_1, κ_1) *to a parameterized problem* (A_2, κ_2) *is a Turing reduction M from A_1 to A_2 that has the following properties.*

1. *Given an $\epsilon > 0$ and an instance x of A_1, M runs in time $O(2^{\epsilon\kappa_1(x)}|x|^{O(1)})$.*
2. *For any query $M(x)$ makes to A_2 with the input x',*
   (a) $|x'| = |x|^{O(1)}$,
   (b) $\kappa_2(x') = \alpha\kappa_1(x)$.

   The constant α may depend on ϵ while the constant hidden in the $O()$-notation in the bound for $|x'|$ may not.

It can easily be shown that SERF-T reductions are transitive. We now prove that SERF-T reductions work as expected and indeed preserve solvability in subexponential time.

Proposition 2.7. *If there is a* SERF-T *reduction from* (A_1, κ_1) *to* (A_2, κ_2) *and A_2 has a subexponential time algorithm then so does A_1.*

Proof. By Proposition 2.5 there is an algorithm for (A_2, κ_2) that for every $\epsilon > 0$ can solve an instance x in time $O(2^{\epsilon\kappa_2(x)}|x|^c)$ for some c independent of x and ϵ. We show that such an algorithm also exists for (A_1, κ_1).

Given an $\epsilon > 0$ we need to make an algorithm running in time $O(2^{\epsilon\kappa_1(x)}|x|^{c'})$ for some c' independent of x and ϵ. We choose $\epsilon' = \epsilon/2$ and run the SERF-T reduction from (A_1, κ_1) to (A_2, κ_2) with parameter ϵ'. This reduction makes at most $O(2^{\epsilon'\kappa_1(x)}|x|^{O(1)})$ calls to instances x' of A_2, each with $|x'| \leq |x|^{O(1)}$ and $\kappa_2(x') \leq \alpha\kappa_1(x)$. Each such instance can be solved in time $2^{\epsilon\kappa_1(x)/2}|x|^{O(1)}$. Hence the total running time for solving x is $2^{\epsilon\kappa_1(x)}|x|^{c'}$ for some c' independent of x and ϵ. By Proposition 2.5 this means that (A_1, κ_1) is in SUBEPT. □

Since every variable appears in some clause it follows that $n \leq rm$, and hence r-CNF-S with parameter m (the number of clauses) is SERF-T reducible to r-CNF-S with parameter n. However, there is no equally obvious SERF-T reduction from r-CNF-S with parameter n to r-CNF-S with parameter m. Nevertheless, Impagliazzo, Paturi and Zane [37] established such a reduction, whose core argument is called the *sparsification lemma* stated below.

Lemma 2.8 ([37])**. (Sparsification Lemma)** *For every $\epsilon > 0$ and positive integer r, there is a constant $C = O((\frac{r}{\epsilon})^{3r})$ so that any r-CNF formula F with n variables, can be expressed as $F = \vee_{i=1}^{t} Y_i$, where $t \leq 2^{\epsilon n}$ and each Y_i is an r-CNF formula with every variables appearing in at most C clauses. Moreover, this disjunction can be computed by an algorithm running in time $2^{\epsilon n} n^{O(1)}$.*

Lemma 2.8 directly gives a SERF-T reduction from r-CNF-S with parameter n to r-CNF-S with parameter m. Thus the following proposition is a direct consequence of the sparsification lemma.

Proposition 2.9 ([37])**.** *Assuming ETH, there is a positive real s' such that* 3-CNF-S *with parameter m cannot be solved in time $O(2^{s'm})$. That is, there is no $2^{o(m)}$ algorithm for* 3-CNF-S *with parameter m.*

Proposition 2.9 has far-reaching consequences: as we shall see, by reductions from from 3-CNF-S with parameter m, we can show lower bounds for a wide range of problems. Moreover, we can even show that several NP-complete problems are equivalent with respect to solvability in subexponential time. For an example, every problem in SNP and size-constrained SNP (see [37] for definitions of these classes) can be shown to have SERF-T reductions to r-CNF-S with parameter n for some $r \geq 3$. The SNP and size-constrained SNP problem classes contain several important problems such as r-CNF-S with parameter n and I S , V C and C parameterized by the number of vertices in the input graph. This gives some evidence that a subexponential time algorithm for r-CNF-S with parameter n is unlikely to exist, giving some credibility to ETH.

It is natural to ask how the complexity of r-CNF-S evolves as r grows. For all $r \geq 3$, define,

$$s_r = \inf \Big\{ \delta : \text{there exists an } O^*(2^{\delta n}) \text{ algorithm solving } r\text{-CNF-S with parameter } n \Big\}.$$

$$s_\infty = \lim_{r \to \infty} s_r.$$

Since r-CNF-S easily reduces to $(r+1)$-SAT it follows that $s_r \leq s_{r+1}$. However, saying anything else non-trivial about this sequence is difficult. ETH is equivalent to conjecturing that $s_3 > 0$. Impagliazzo, Paturi and Zane [37] present the following relationships between the s_r's and the solvability of problems in SNP in subexponential time. The theorem below is essentially a direct consequence of Lemma 2.8

Theorem 2.10 ([37]). *The following statements are equivalent*

1. *For all* $r \geq 3$, $s_r > 0$.
2. *For some* r, $s_r > 0$.
3. $s_3 > 0$.
4. SNP $\not\subseteq$ SUBEPT.

The equivalence above offers some intuition that r-CNF-S with parameter n may not have a subexponential time algorithm and thus strengthens the credibility of ETH. Impagliazzo and Paturi [37, 10] studied the sequence of s_r's and obtained the following results.

Theorem 2.11 ([37, 10]). *Assuming ETH, the sequence* $\{s_r\}_{r\geq 3}$ *is increasing infinitely often. Furthermore,* $s_r \leq s_\infty(1 - \frac{d}{r})$ *for some constant* $d > 0$

A natural question to ask is what is s_∞? As of today the best algorithms for r-CNF-S all use time $O(2^{n(1-\frac{c}{r})})$ for some constant c independent of r and n. This, together with Theorem 2.11 hints at $s_\infty = 1$. The conjecture that this is indeed the case is known as the Strong Exponential Time Hypothesis.

Strong Exponential Time Hypothesis (SETH) [37, 10]: $s_\infty = 1$.

An immediate consequence of SETH is that that SAT with parameter n (here the input formula F could have arbitrary size clauses) cannot be solved in time $(2-\epsilon)^n(n+m)^{O(1)}$. In order to justify SETH one needs to link the existence of a faster satisfiability algorithm to known complexity assumptions, or at least give an analogy of Theorem 2.10 for SETH. In a recent manuscript, Cygan et al. [17] show that for several basic problems their brute force algorithm can be improved if and only if SETH fails. Thus there is at least a small class of problems whose exponential time complexity stands and falls with SETH.

Theorem 2.12 ([17]). *The following are equivalent.*

- $\exists \delta < 1$ *such that* r-CNF-S *with parameter* n *is solvable in* $O(2^{\delta n})$ *time for all* r.
- $\exists \delta < 1$ *such that* H S *for set systems with sets of size at most* k *is solvable in* $O(2^{\delta n})$ *time for all* k. *Here* n *is the size of the universe.*

- $\exists \delta < 1$ *such that* S S *for set systems with sets of size at most k is solvable in* $O(2^{\delta n})$ *time for all k,. Here n is the size of the universe.*
- $\exists \delta < 1$ *such that k*-NAE-S *is solvable in* $O(2^{\delta n})$ *time for all k. Here n is the number of variables.*
- $\exists \delta < 1$ *such that satisfiability of cn size series-parallel circuits is solvable in* $O(2^{\delta n})$ *time for all c.*

This immediately implies that a $2^{\delta n}$ time algorithm for any of the above problems without the restrictions on clause width or set size would violate SETH. All of the problems above have $O(2^n)$ time brute force algorithms, and hence these bounds are tight.

It is tempting to ask whether it is possible to show a "strong" version of the Sparsification Lemma, implying that under SETH there is no algorithm for *r*-CNF-S with running time $O((2-\epsilon)^m)$ for any $\epsilon > 0$. However, faster algorithms for SAT parameterized by the number of clauses *do* exist. In particular, the currently fastest known algorithm [34] for SAT parameterized by the number of clauses runs in time $O(1.239^m)$.

3 Lower Bounds for Exact Algorithms

In this section we outline algorithmic lower bounds obtained using ETH and SETH on the running time of exact exponential time algorithms.

In order to prove that a too fast algorithm for a certain problem P contradicts ETH, one can give a reduction from 3-CNF-S to P and argue that a too fast algorithm for L would solve 3-CNF-S in time $2^{o(m)}$. This together with Lemma 2.9 would imply that 3-CNF-S can be solved in time $2^{o(n)}$, contradicting ETH. Quite often the known NP-completeness reduction from 3-CNF-S to the problem in question give the desired running time bounds. We illustrate this for the case of 3-C . We use the well-known fact that there is a polynomial-time reduction from 3-CNF-S to 3-C where the number of vertices of the graph is linear in the size of the formula.

Proposition 3.1 ([51])**.** *Given a 3SAT formula* ϕ *with n-variables and m-clauses, it is possible to construct a graph G with* $O(n+m)$ *vertices in polynomial time such that G is 3-colorable if and only if* ϕ *is satisfiable.*

Proposition 3.1 implies that the number of vertices in the graph G is linear in the number of variables and clauses. Thus, an algorithm for 3-C with running time subexponential in the number of vertices would gives a $2^{o(m)}$ time algorithm for 3-CNF-S . This together with Lemma 2.9 imply the following.

Theorem 3.2. *Assuming ETH, there is no $2^{o(n)}$ time algorithm for* 3-C .

Similarly, one can show that various graph problems such as D S , I S , V C and H P do not have $2^{o(n)}$ time algorithms unless ETH fails.

Theorem 3.3. *Assuming ETH, there is no $2^{o(n)}$ time algorithm for* D S , I S , V C *or* H P .

The essential feature of the reductions above is that the number of vertices is linear in the number of clauses of the original 3SAT formula. If the reduction introduces some blow up, that is, the number of vertices is more than linear, then we still get lower bounds, but weaker than $2^{o(n)}$. For example, such blow up is very common in reductions to planar problems. We outline one such lower bound result for P H C . Here, the input consists of planar graph G on n vertices and the objective to check whether there is a hamiltonian cycle in G.

Proposition 3.4 ([30])**.** *Given a 3SAT formula ϕ with n-variables and m-clauses, it is possible to construct a planar graph G with $O(m^2)$ vertices and edges in polynomial time such that G has a hamiltonian cycle if and only if ϕ is satisfiable.*

Proposition 3.4 implies that the number of vertices in the graph G is quadratic in the number of clauses. Thus, an algorithm for P H C with running time $2^{o(\sqrt{n})}$ would gives a $2^{o(m)}$ time algorithm for 3-CNF-S . This together with Lemma 2.9 imply the following.

Theorem 3.5. *Assuming ETH, there is no $2^{o(\sqrt{n})}$ time algorithm for* P H - C .

One can prove similar lower bounds for P V C , P D - S and various other problems on planar graphs and other kind of geometric graphs. Note that many of these results are tight: for example, P H C can be solved in time $2^{O(\sqrt{n})}$.

While reductions from SAT can be used to prove many interesting bounds under ETH, such an approach has an inherent limitation; the obtained bounds can only distinguish between the asymptotic behaviour, and not between different constants in the exponents. Most efforts in Exact Exponential time algorithms have concentrated exactly on decreasing the constants in the exponents of the running times, and hence, in order to have a good complexity theory for Exact Exponential Time Algorithms one needs a tool to rule out $O(c^n)$ time algorithms for problems for concrete constants c. Assuming that SETH holds and reducing from r-CNF-S allows us to rule out $O(c^n)$ time algorithms for at least some problems (see Theorem 2.10). However, the complexity theory of Exact Exponential Time Algorithms is still at a nascent stage, with much left unexplored.

4 Lower Bounds for FPT Algorithms

Once it is established that a parameterized problem is FPT, that is, can be solved in time $f(\kappa(x)) \cdot |x|^{O(1)}$, the next obvious goal is to design algorithms where the function f is as slowly growing as possible. Depending on the type of problem considered and the algorithmic technique employed, the function f comes in all sizes and shapes. It can be an astronomical tower of exponentials for algorithms using Robertson and Seymour's Graph Minors theory; it can be c^k for some nice small constant c (e.g., 1.2738^k for vertex cover [14]); it can be even subexponential (e.g., $2^{\sqrt{k}}$). It happens very often that by understanding a problem better or by using more suitable techniques, better and better fpt algorithms are developed for the same problem and a kind of "race" is established to make f as small as possible. Clearly, it would be very useful to know if the current best algorithm can be improved further or it has already hit some fundamental barrier. Cai and Juedes [9] were first to examine the existence of $2^{o(k)}$ or $2^{o(\sqrt{k})}$ algorithms for various parameterized problems solvable in time $2^{O(k)}$ or $2^{O(\sqrt{k})}$, respectively. They showed that for variety of problems assuming ETH, there is no $2^{o(k)}$ or $2^{o(\sqrt{k})}$ algorithms possible. In this section, we survey how ETH can be used to obtain lower bounds on the function f for various FPT problems.

We start with a simple example. We first define the V C problem.

V C	
Instance:	A graph G, and a non-negative integer k.
Parameter:	k.
Problem:	Decide whether G has a vertex cover with at most k elements.

Since $k \leq n$, a $2^{o(k)}n^c$ time algorithm directly implies a $2^{o(n)}$ time algorithm for V C . However, by Theorem 3.3 we know that V C does not have an algorithm with running time $2^{o(n)}$ unless ETH fails. This immediately implies the following theorem.

Theorem 4.1 ([9]). *Assuming ETH, there is no $2^{o(k)}n^{O(1)}$ time algorithm for* V - C .

Similarly, assuming ETH, we can show that several other problems parameterized by the solution size, such as F V S or L P do not have $2^{o(k)}n^{O(1)}$ time algorithms.

Theorem 4.2 ([9]). *Assuming ETH, there is no $2^{o(k)}n^{O(1)}$ time algorithm for* F - V S *or* L P .

Similar arguments yield tight lower bounds for parameterized problems on special graph classes, such as planar graphs. As we have seen in the previous section, for many problems we can rule out algorithms with running time $2^{o(\sqrt{n})}$ even when the input graph is restricted to be planar. If the solution to such a problem is a subset of the vertices (or edges), then the problem parameterized by solution size cannot be solved in time $2^{o(\sqrt{k})}n^{O(1)}$ on planar graphs, unless ETH fails.

Theorem 4.3 ([9]). *Assuming ETH, there is no* $2^{o(\sqrt{k})}n^{O(1)}$ *time algorithm for* P - V C .

Results similar to Theorem 4.3 are possible for several other graph problems on planar graphs. It is worth to mention that many of these lower bounds on these problems are tight. That is, many of the mentioned problems admit both $2^{O(k)}n^{O(1)}$ time algorithms on general graphs and $2^{O(\sqrt{k})}n^{O(1)}$ time algorithms on planar graphs.

Obtaining lower bounds of the form $2^{o(k)}n^{O(1)}$ or $2^{o(\sqrt{k})}n^{O(1)}$ on parameterized problems generally follows from the known NP-hardness reduction. However, there are several parameterized problems where $f(k)$ is "slightly superexponential" in the best known running time: $f(k)$ is of the form $k^{O(k)} = 2^{O(k \log k)}$. Algorithms with this running time naturally occur when a search tree of height at most k and branching factor at most k is explored, or when all possible permutations, partitions, or matchings of a k element set are enumerated. Recently, for a number of such problems lower bounds of the form $2^{o(k \log k)}$ were obtained under ETH [43]. We show how such a lower bound can be obtained for an artificial variant of the C problem. In this problem the vertices are the elements of a $k \times k$ table, and the clique we are looking for has to contain exactly one element from each row.

$k \times k$ C	
Input:	A graph G over the vertex set $[k] \times [k]$
Parameter:	k
Question:	Is there a k-clique in G with exactly one element from each row?

Note that the graph G in the $k \times k$ C instance has $O(k^2)$ vertices at most $O(k^4)$ edges, thus the size of the instance is $O(k^4)$.

Theorem 4.4 ([43]). *Assuming ETH, there is no* $2^{o(k \log k)}$ *time algorithm for* $k \times k$ C .

Proof. Suppose that there is an algorithm $\mathbb{A}$ that solves $k \times k$ C in $2^{o(k \log k)}$ time. We show that this implies that 3-C on a graph with n vertices can be solved in time $2^{o(n)}$, which contradicts ETH by Theorem 3.2.

Let H be a graph with n vertices. Let k be the smallest integer such that $3^{n/k+1} \leq k$, or equivalently, $n \leq k \log_3 k - k$. Note that such a finite k exists for every n and it is easy to see that $k \log k = O(n)$ for the smallest such k. Intuitively, it will be useful to think of k as a value somewhat larger than $n/\log n$ (and hence n/k is somewhat less than $\log n$).

Let us partition the vertices of H into k groups $X_1, \ldots, X_k$, each of size at most $\lceil n/k \rceil$. For every $1 \leq i \leq k$, let us fix an enumeration of all the proper 3-colorings of $H[X_i]$. Note that there are most $3^{\lceil n/k \rceil} \leq 3^{n/k+1} \leq k$ such 3-colorings for every i. We say that a proper 3-coloring c_i of $H[X_i]$ and a proper 3-coloring c_j of $H[X_j]$ are *compatible* if together they form a proper coloring of $H[X_i \cup X_j]$: for every edge uv with $u \in X_i$ and $v \in X_j$, we have $c_i(u) \neq c_j(v)$. Let us construct a graph G over the vertex set $[k] \times [k]$ where vertices (i_1, j_1) and (i_2, j_2) with $i_1 \neq i_2$ are adjacent if and only if the j_1-th proper coloring of $H[X_{i_1}]$ and the j_2-th proper coloring of $H[X_{i_2}]$ are compatible (this means that if, say, $H[X_{i_1}]$ has less than j_1 proper colorings, then (i_1, j_1) is an isolated vertex).

We claim that G has a k-clique having exactly one vertex from each row if and only if H is 3-colorable. Indeed, a proper 3-coloring of H induces a proper 3-coloring for each of $H[X_1], \ldots, H[X_k]$. Let us select vertex (i, j) if and only if the proper coloring of $H[X_i]$ induced by c is the j-th proper coloring of $H[X_i]$. It is clear that we select exactly one vertex from each row and they form a clique: the proper colorings of $H[X_i]$ and $H[X_j]$ induced by c are clearly compatible. For the other direction, suppose that $(1, \rho(1)), \ldots, (k, \rho(k))$ form a k-clique for some mapping $\rho : [k] \to [k]$. Let c_i be the $\rho(i)$-th proper 3-coloring of $H[X_i]$. The colorings $c_1, \ldots, c_k$ together define a coloring c of H. This coloring c is a proper 3-coloring: for every edge uv with $u \in X_{i_1}$ and $v \in X_{i_2}$, the fact that $(i_1, \rho(i_1))$ and $(i_2, \rho(i_2))$ are adjacent means that c_{i_1} and c_{i_2} are compatible, and hence $c_{i_1}(u) \neq c_{i_2}(v)$.

Running the assumed algorithm $\mathbb{A}$ on G decides the 3-colorability of H. Let us estimate the running time of constructing G and running algorithm $\mathbb{A}$ on G. The graph G has k^2 vertices and the time required to construct G is polynomial in k: for each X_i, we need to enumerate at most k proper 3-colorings of $G[X_i]$. Therefore, the total running time is $2^{o(k \log k)} \cdot k^{O(1)} = 2^{o(n)}$ (using that $k \log k = O(n)$). It follows that we have a $2^{o(n)}$ time algorithm for 3-C , contradicting ETH. □

In [43], Lokshtanov et al. first define other problems similar in flavor to $k \times k$ C : basic problems artificially modified in such a way that they can be solved by brute force in time $2^{O(k \log k)}|I|^{O(1)}$. It is then shown that assuming ETH, these problems do not admit a $2^{o(k \log k)}$ time algorithm. Finally, combining the lower

bounds on the variants of basic problems with suitable reductions one can obtain lower bounds for natural problems. One example is the bound for the C S problem.

C S	
Input:	Strings $s_1, \ldots, s_t$ over an alphabet Σ of length L each, an integer d
Parameter:	d
Question:	Is there a string s of length L such $d(s, s_i) \leq d$ for every $1 \leq i \leq t$?

Here $d(s, s_i)$ is the *Hamming distance* between the strings s and s_i, that is, the number of positions where s and s_i differ. Gramm et al. [31] showed that C S is fixed-parameter tractable parameterized by d: they gave an algorithm with running time $O(d^d \cdot |I|)$. The algorithm works over an arbitrary alphabet Σ (i.e., the size of the alphabet is part of the input). For fixed alphabet size, single-exponential dependence on d can be achieved: algorithms with running time of the form $|\Sigma|^{O(d)} \cdot |I|^{O(1)}$ were presented in [44, 54, 15]. It is an obvious question if the running time can be improved to $2^{O(d)} \cdot |I|^{O(1)}$, i.e., single-exponential in d, even for arbitrary alphabet size. However, the following result shows that the running times of the cited algorithms have the best possible form:

Theorem 4.5 ([43])**.** *Assuming ETH, there is no* $2^{o(d \log d)} \cdot |I|^{O(1)}$ *or* $2^{o(d \log |\Sigma|)} \cdot |I|^{O(1)}$ *time algorithm for* C S .

Using similar methods one can also give tight running time lower bounds for the D problem. Here we are given a graph G and parameter d. The objective is to determine whether there exists a map f from the vertices of G to $\mathbb{N}$ such that for every pair of vertices u and v in G, if the distance between u and v in G is δ then $\delta \leq |f(u) - f(v)| \leq d\delta$. This problem belongs to a broader range of "metric embedding" problems where one is looking for a map from a complicated distance metric into a simple metric while preserving as many properties of the original metric as possible. Fellows et al. [24] give a $O(d^d n^{O(1)})$ time algorithm for D . The following theorem shows that under ETH the dependence on d of this algorithm cannot be significantly improved.

Theorem 4.6 ([43])**.** *Assuming ETH, there is no* $2^{o(d \log d)} \cdot n^{O(1)}$ *time algorithm for* D .

5 W[1]-Hard problems

The complexity assumption ETH can be used not only to obtain running time lower bounds on problems that are FPT, but also on problems that are known to

be W[1]-hard in parameterized complexity. For an example I S and D S are known to be W[1]-complete and W[2]-complete, respectively. Under the standard parameterized complexity assumption that $FPT \neq W[1]$, this immediately rules out the possibility of having an fpt algorithm for C , I - S and D S . However, knowing that no algorithm of the form $f(k)n^{O(1)}$ exists, that these results do not rule out the possibility of an algorithm with running time, say, $n^{O(\log \log k)}$. As the best known algorithms for these problems take $n^{O(k)}$ time, there is huge gap between the upper and lower bounds obtained this way.

Chen et al. [11] were the first to consider the possibility of showing sharper running time lower bounds for W[1]-hard problems. They show that lower bounds of the form $n^{o(k)}$ can be achieved for several W[2]-hard problems such as D - S , under the assumption that FPT $\neq W[1]$. However, for problems that are W[1]-hard rather than W[2]-hard, such as I S , we need ETH in order to show lower bounds. Later, Chen et al. [13, 12] strengthened their lower bounds to also rule out $f(k)n^{o(k)}$ time algorithms (rather than just $n^{o(k)}$ time algorithms). We outline one such lower bound result here and then transfer it to other problems using reductions.

Theorem 5.1 ([11, 13])**.** *Assuming ETH, there is no $f(k)n^{o(k)}$ time algorithm for* C *or* I S .

Proof. We give a proof sketch. We will show that if there is an $f(k)n^{o(k)}$ time algorithm for C , then ETH fails. Suppose that C can be solved in time $f(k)n^{k/s(k)}$, where $s(k)$ is a monotone increasing unbounded function. We use this algorithm to solve 3-C on an n-vertex graph G in time $2^{o(n)}$. Let $f^{-1}(n)$ be the largest integer i such that $f(i) \leq n$. Function $f^{-1}(n)$ is monotone increasing and unbounded. Let $k := f^{-1}(n)$. Split the vertices of G into k groups. Let us build a graph H where each vertex corresponds to a proper 3-coloring of one of the groups. Connect two vertices if they are not conflicting. That is, if the union of the colorings corresponding to these vertices corresponds to a valid coloring of the graph induced on the vertices of these two groups, then connect the two vertices. A k-clique of H corresponds to a proper 3-coloring of G. A 3-coloring of G can be found in time $f(k)n^{k/s(k)} \leq n(3^{n/k})^{k/s(k)} = n3^{n/s(f^{-1}(n))} = 2^{o(n)}$. This completes the proof.

Since a graph G has a clique of size k if and only the complement of G has an independent set of size k. Thus, as a simple corollary to the result of C , we get that I S does not have any $f(k)n^{o(k)}$ time algorithm unless ETH fails. □

A colored version of clique problem, called M C has been proven to be very useful in showing hardness results in Parameterized Complexity.

An input to M C consists of a graph G and a proper coloring of vertices with $\{1, \ldots, k\}$ and the objective is to check whether there exists a k-sized clique containing a vertex from each color class. A simple reduction from I S shows the following theorem.

Theorem 5.2. *Assuming ETH, there is no $f(k)n^{o(k)}$ time algorithm for* M - C .

Proof. We reduce from the I S problem. Given an instance (G, k) to I S we construct a new graph $G' = (V', E')$ as follows. For each vertex $v \in V$ we make k copies of v in V' with the i'th copy being colored with the i'th color. For every pair u,$v \in V$ such that $uv \notin E$ we add edges between all copies of u and all copies of v with different colors. It is easy to see that G has an independent set of size k if and only if G' contains a clique of size k. Furthermore, running a $f(k)n^{o(k)}$ time algorithm on G' would take time $f(k)(nk)^{o(k)} = f'(k)n^{o(k)}$. This concludes the proof. □

One should notice that the reduction produces instances with a quite specific structure to M C . In particular, all color classes have the same size and the number of edges between every pair of color classes is the same. It is often helpful to exploit this fact when reducing from M C to a specific problem. We now give an example of a slightly more involved reduction that will show a lower bound on D S .

Theorem 5.3. *Assuming ETH, there is no $f(k)n^{o(k)}$ time algorithm for* D S .

Proof. We reduce from the M C problem. Given an instance (G, k) to M C we construct a new graph G'. For every $i \leq k$ let V_i be the set of vertices in G colored i and for every pair of distinct integers $i, j \leq k$ let $E_{i,j}$ be the set of edges in $G[V_i \cup V_j]$. We start making G' by taking a copy of V_i for every $i \leq k$ and making this copy into a clique. Now, for every $i \leq k$ we add a set S_i of $k + 1$ vertices and make them adjacent to all vertices of V_i. Finally, for every pair of distinct integers $i, j \leq k$ we consider the edges in $E_{i,j}$.For every pair of vertices $u \in V_i$ and $v \in V_j$ such that $uv \notin E_{i,j}$ we add a vertex x_{uv} and make it adjacent to all vertices in $V_i \setminus \{u\}$ and all vertices in $V_j \setminus \{v\}$. This concludes the construction. We argue that G contains a k-clique if and only if G' has a dominating set of size at most k.

If G contains a k-clique C then C is a dominating set of G'. In the other direction, suppose G' has a dominating set S of size at most k. If for some i, $S \cap V_i = \emptyset$ then $S_i \subseteq S$, contradicting that S has size at most k. Hence for every $i \leq k$, $S \cap V_i \neq \emptyset$ and thus S contains exactly one vertex v_i from V_i for each i, and S contains no other vertices. Finally, we argue that S is a clique in G. Suppose

that $v_iv_j \notin E_{i,j}$. Then there is a vertex x in $V(G')$ with neighbourhood $V_i \setminus \{u\}$ and $V_j \setminus \{v\}$. This x is not in S and has no neighbours in S contradicting that S is a dominating set of G'.

The above reduction together with Theorem 5.2 imply the result. □

The proof in Theorem 5.3 could be viewed as a fpt reduction from I S to D S . The first fpt reduction from I S to D S is due to Fellows [23]. The reduction presented here is somewhat simpler than the original proof and is due to Lokshtanov [41].

W[1]-hardness proofs are typically done by a parameterized reductions from C . It is easy to observe that a parameterized reduction itself gives strong lower bounds under ETH for the target problem, with the exact form of the lower bound depending on the way the reduction changes the parameter. In the case of the reduction to D S above, the parameter changes only linearly (actually, does not change at all) and therefore we obtain the same lower bound $f(k)n^{o(k)}$ for this problem as well. If the reduction increases the parameter more than linearly, then we get only weaker lower bounds. For an example, Marx [45] showed the following.

Proposition 5.4 ([45])**.** *Given a graph G and a positive integer k, it is possible to construct a unit disk graph G' in polynomial time such that G has a clique of size k if and only if G' has a dominating set of size $O(k^2)$.*

Observe that Proposition 5.4 gives a W[1]-hardness proof for D S on unit disk graphs, starting from C . The reduction squares the parameter k and this together with Theorem 5.1 gives the following theorem.

Theorem 5.5 ([45])**.** *Assuming ETH, there is no $f(k)n^{o(\sqrt{k})}$ time algorithm for* D S *on unit disk graphs.*

As D S on unit disk graphs can be solved in time $n^{O(\sqrt{k})}$ [1], Theorem 5.5 is tight.

C S (a generalization of C S) is an extreme example where reductions increase the parameter exponentially or even double exponentially, and therefore we obtain very weak lower bounds. This problem is defined as follows:

C S	
Input:	Strings $s_1, \ldots, s_t$ over an alphabet Σ, integers L and d
Parameter:	d, t
Question:	Is there a string s of length L such that s_i has a substring s'_i of length L with $d(s, s_i) \leq d$? for every $1 \leq i \leq t$?

Let us restrict our attention to the case where the alphabet is of constant size, say binary. Marx [46] gave a reduction from C to C S where $d = 2^{O(k)}$ and $t = 2^{2^{O(k)}}$ in the constructed instance (k is the size of the clique we are looking for in the original instance). Therefore, we get weak lower bounds with only $o(\log d)$ and $o(\log \log k)$ in the exponent. Interestingly, these lower bounds are actually tight, as there are algorithms matching these bounds.

Theorem 5.6 ([46])**.** C S *over an alphabet of constant size can be solved in time* $f(d)n^{O(\log d)}$ *or in* $f(d,k)n^{O(\log\log k)}$.

Furthermore, assuming ETH, there are no algorithms for the problem with running time $f(k,d)n^{o(\log d)}$ *or* $f(k,d)n^{o(\log\log k)}$.

As we have seen, it is very important to control the increase of the parameter in the reductions if our aim is to obtain strong lower bounds. Many of the more involved reductions from C use edge selection gadgets (see e.g., [25, 27, 45]). As a clique of size k has $\Theta(k^2)$ edges, this means that the reduction typically increases the parameter to $\Theta(k^2)$ at least and we can conclude that there is no $f(k)n^{o(\sqrt{k})}$ time algorithm for the target problem (unless ETH fails). If we want to obtain stronger bounds on the exponent, then we have to avoid the quadratic blow up of the parameter and do the reduction from a different problem. Many of the reductions from C can be turned into a reduction from S I (Given two graphs $G = (V,E)$ and H, decide if G is a subgraph of H). In a reduction from S I , we need $|E|$ edge selection gadgets, which usually implies that the new parameter is $\Theta(|E|)$, leading to an improved lower bounds compared to those coming from the reduction from C . Thus the following lower bound on S I , parameterized by the number of edges in G, could be a new source of lower bounds for various problems.

Theorem 5.7 ([47])**.** *If* S I *can be solved in time* $f(k)n^{o(k/\log k)}$*, where* f *is an arbitrary function and* $k = |E|$ *is the number of edges of the smaller graph* G*, then ETH fails.*

We remark that it is an interesting open question if the factor $\log k$ in the exponent can be removed, making this result tight.

While the results in Theorems 5.1, 5.3 are asymptotically tight, they do not tell us the exact form of the exponent, that is, we do not know what the smallest c is such that the problems can be solved in time n^{ck}. However, assuming SETH, stronger bounds of this form can be obtained. Specifically, Pătraşcu and Williams [49] obtained the following bound for D S under SETH.

Theorem 5.8 ([49])**.** *Assuming SETH, there is no* $O(n^{k-\epsilon})$ *time algorithm for* D - S *for any* $\epsilon > 0$.

Theorem 5.8 is almost tight as it is known that for $k \geq 7$, D S can be solved in time $n^{k+o(1)}$ [49]. Interestingly, Pătraşcu and Williams [49] do not believe SETH holds and state Theorem 5.8 as a route to obtain faster satisfiability algorithms through better algorithms for D S .

6 Parameterization by Treewidth

The notion of *treewidth* has emerged as a popular structural graph parameter, defined independently in a number of contexts. It is convenient to think of treewidth as a measure of the "tree-likeness" of a graph, so that the smaller the treewidth of a graph, the more tree-like properties it has. Just as a number of NP-complete problems are polynomial time solvable on trees, a number of problems can be solved efficiently on graphs of small treewidth. Often, the strategies that work for trees can be generalized smoothly to work over *tree decompositions* instead. Very few natural problems are W[1]-hard under this parameter, and the literature is rich with algorithms and algorithmic techniques that exploit the small treewidth of input instances (see e.g., [7, 6, 38]). Formally, treewidth is defined as follows:

Definition 6.1. *A* tree decomposition *of a graph $G = (V, E)$ is a pair*

$$(T = (V_T, E_T), \mathcal{X} = \{X_t \ : \ X_t \subseteq V\}_{t \in T_T})$$

such that

1. $\cup_{t \in V(T)} X_t = V$,
2. *for every edge $(x, y) \in E$ there is a $t \in V_T$ such that $\{x, y\} \subseteq X_t$, and*
3. *for every vertex $v \in V$ the subgraph of T induced by the set $\{t \mid v \in X_t\}$ is connected.*

The *width* of a tree decomposition is $(\max_{t \in V(T)} |X_t|) - 1$ and the *treewidth* of G, denoted by $\mathbf{tw}(G)$, is the minimum width over all tree decompositions of G.

It is well known that several graph problems parameterized by the treewidth of the input graph are FPT. See Table 1 for the time complexity of some known algorithms for problems parameterized by the treewidth of the input graph. Most of the algorithms on graphs of bounded treewidth are based on simple dynamic programming on the tree decomposition, although for some problems a recently discovered technique called fast subset convolution [52, 4] needs to be used to obtain the running time shown in Table 1.

An obvious question is how fast these algorithms can be. We can easily rule out the existence of $2^{o(t)}$ algorithm for many of these problems assuming ETH.

Problem Name	$f(t)$ in the best known algorithms
V C	2^t
D S	3^t
O C T	3^t
P I T	2^t
M C	2^t
C N	$2^{O(t \log t)}$
D P	$2^{O(t \log t)}$
C P	$2^{O(t \log t)}$

Table 1: The table gives the $f(t)$ bound in the running time of various problems parameterized by the treewidth of the input graph.

Recall that, Theorem 3.3 shows that assuming ETH, the I S problem parameterized by the number of vertices in the input graph does not admit a $2^{o(n)}$ algorithm. Since the treewidth of a graph is clearly at most the number of vertices, it is in fact a "stronger" parameter, and thus the lower bound carries over. Thus, we trivially have that I S does not admit a subexponential algorithm when parameterized by treewidth. Along the similar lines we can show the following theorem.

Theorem 6.2. *Assuming ETH,* I S , D S *and* O C T *parameterized by the treewidth of the input graph do not admit an algorithm with running time* $2^{o(t)}n^{O(1)}$. *Here, n is the number of vertices in the input graph to these problems.*

For the problems C N , C P , and D P , the natural dynamic programming approach gives $2^{O(t \log t)}n^{O(1)}$ time algorithms. As these problems can be solved in time $2^{O(n)}$ on n-vertex graphs, the easy arguments of Theorem 6.2 cannot be used to show the optimality of the $2^{O(t \log t)}n^{O(1)}$ time algorithms. However, as reviewed in Section 4, Lokshtanov et al. [43] developed a machinery for obtaining lower bounds of the form $2^{o(k \log k)}n^{O(1)}$ for parameterized problems and we can apply this machinery in the case of parameterization by treewidth as well.

Theorem 6.3 ([43, 18])**.** *Assuming ETH,* C N , C P , D P *parameterized by the treewidth of the input graph do not admit an algorithm with running time* $2^{o(t \log t)}n^{O(1)}$. *Here, n is the number of vertices in the input graph to these problems.*

The lower bounds obtained by Theorem 6.2 are quite weak: they tell us that $f(t)$ cannot be improved to $2^{o(t)}$, but they do not tell us whether the numbers 2 and

3 appearing as the base of exponentials in Table 1 can be improved. Just as we saw for Exact Algorithms, ETH seems to be too weak an assumption to show a lower bound that concerns the base of the exponent. Assuming the SETH, however, much tighter bounds can be shown. In [42] it is established that any non-trivial improvement over the best known algorithms for a variety of basic problems on graphs of bounded treewidth would yield a faster algorithm for SAT.

Theorem 6.4 ([42])**.** *If there exists an* $\epsilon > 0$ *such that*

- I S *can be solved in* $(2-\epsilon)^{tw(G)}n^{O(1)}$ *time, or*
- D S *can be solved in* $(3-\epsilon)^{tw(G)}n^{O(1)}$ *time, or*
- M C *can be solved in* $(2-\epsilon)^{tw(G)}n^{O(1)}$ *time, or*
- O C T *can be solved in* $(3-\epsilon)^{tw(G)}n^{O(1)}$ *time, or*
- *there is a* $q \geq 3$ *such that* q-C *can be solved in* $(q-\epsilon)^{tw(G)}n^{O(1))}$ *time, or*
- P I T *can be solved in* $(2-\epsilon)^{tw(G)}n^{O(1)}$ *time,*

then SETH fails.

Thus, assuming SETH, the known algorithms for the mentioned problems on graphs of bounded treewidth are essentially the best possible. To show these results, polynomial time many-one reductions are devised, and these transform n-variable boolean formulas ϕ to instances of the problems in question, while carefully controlling the treewidth of the graphs that the 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 a lower bound of the form $O((2-\epsilon)^{\mathbf{tw}(G)}n^{O(1)})$, 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 $O((2-\epsilon)^{\mathbf{tw}(G)}n^{O(1)})$, we need even more economical constructions. To understand the difficulty, consider the D S problem, here we want to say that if D S admits an algorithm with running time $O((3-\epsilon)^{\mathbf{tw}(G)}n^{O(1)}) = O(2^{\log(3-\epsilon)\mathbf{tw}(G)}n^{O(1)})$ for some $\epsilon > 0$, then we can solve SAT on input formulas with n-variables in time $O((2-\delta)^n)$ for some $\delta > 0$. Therefore by naïvely equating the exponents in the previous sentence we get that we need to construct an instance for D S whose treewidth is essentially $\frac{n}{\log 3}$. In other words, each variable should increase treewidth by *less than one.* The main challenge in these reductions is to squeeze out as many combinatorial possibilities per increase of treewidth as possible.

While most natural graph problems are fixed parameter tractable when parameterized by the treewidth of the input graph, there are a few problems for which

the best algorithms are stuck at $O(n^{O(t)})$ time, where t is the treewidth of the input graph. Under ETH one can show that the algorithms for several of these problems cannot be improved to $f(t)n^{o(t)}$. Just as for the problems that are FPT parameterized by treewidth, the lower bounds are obtained by reductions that carefully control the treewidth of the graphs they output. We give one such reduction as an illustration.

L C	
Instance:	A graph $G = (V, E)$ of treewidth at most t, and for each vertex $v \in V$, a list $L(v)$ of permitted colors.
Parameter :	t.
Problem:	Is there a proper vertex coloring c with $c(v) \in L(v)$ for each v?

We show that the L C problem on graphs of treewidth t cannot have an algorithm with running time $f(t)n^{o(t)}$. This means that tre treewidth parameterization of L C is much harder than the closely related C N , which has a $2^{O(t \log t)}n$ time algorithm.

Theorem 6.5 ([25])**.** *Assuming ETH,* L C *on graphs of treewidth t cannot be solved in time* $f(t)n^{o(t)}$.

Proof. We give a reduction from M C to L C where the treewidth of the graph produced by the reduction is bounded by k, the size of the clique in the M C instance. This together with Theorem 5.2 implies the result.

Given an instance G of the M C problem, we construct an instance G' of L C that admits a proper choice of color from each list if and only if the source instance G contains a k-clique. The colors on the lists of vertices in G' have a one to one correspondence with the vertices of G. For simplicity of arguments we *do not* distinguish between a vertex v of G and the color v which appears in the list assigned to some of the vertices of G'.

Recall that every vertex v in G is given a color from 1 to k as a part of the input for the M C instance. Let V_i be the set of vertices in G with color i. The vertices of G' on the other hand do not get colors assigned a priori - however a solution to the constructed L C instance is a coloring of the vertices of G'. The instance G' is constructed as follows.

1. There are k vertices $v[i]$ in G', $i = 1, \ldots, k$, one for each color class of G, and the list assigned to $v[i]$ consists of the colors corresponding to the vertices in G of color i. That is, $L_{v[i]} = \{V_i\}$.

2. For $i \neq j$, there is a degree two vertex in G' adjacent to $v[i]$ and $v[j]$ for each pair x, y of *nonadjacent* vertices in G, where x has color i and y has color j. This vertex is labeled $v_{i,j}[x, y]$ and has $\{x, y\}$ as its list.

This completes the construction.

The treewidth of G' is bounded by k since (a) removing the k vertices $v[i]$, $1 \leq i \leq k$, from G' yields an edgeless graph, (b) edgeless graphs have treewidth 0 and (c) removing a single vertex from a graph decreases treewidth by at most one. If G has a multicolored clique K then we can easily list color G'. Let $K = \{c_1, c_2, \ldots c_k\}$ where $c_i \in V_i$. Color $v[i]$ with c_i, namely the vertex in K from V_i. It is easy to see that every degree 2 vertex in G' has at least one color free in its list, as the pair of colors in the list correspond to non-adjacent vertices in G. For the other direction, suppose that G' can be properly colored such that each vertex is assigned a color from its list. Let $K = \{c_1, \ldots, c_k\}$ be the set of vertices in G that correspond to the colors assigned to the $v[i]$'s in this coloring of G', such that $v[i]$ is colored with c_i. We prove that K is a clique, and to do this it is sufficient to show that c_i and c_j are adjacent for every $i \neq j$. However c_i and c_j can't be non-adjacent because then there would be a degree 2 vertex adjacent to $v[i]$ and $v[j]$ which only can be colored with c_i or c_j, but can't be colored with either. This completes the proof of the theorem. □

The reason why L C is hard is that when doing dynamic programming over the tree decomposition, then (as Theorem 6.5 suggests) the number of possible colorings that we have to keep track of is $n^{\Omega(t)}$. More generally, we encounter a similar difficulty when solving constraint satisfaction problems over a large domain. Constraint satisfaction is a general framework that includes many standard algorithmic problems such as satisfiability, database queries, graph coloring, , etc. A constraint satisfaction problem (CSP) consists of a set V of variables, a domain D, and a set C of constraints, where each constraint is a relation on a subset of the variables. The task is to assign a value from D to each variable in such a way that every constraint is satisfied. For example, 3-SAT can be interpreted as a CSP instance where the domain is $\{0, 1\}$ and the constraints in C correspond to the clauses (thus the arity of each constraint is 3). Another example is vertex coloring or list coloring, which can be interpreted as a CSP instance where the variables correspond to the vertices, the domain corresponds to the set of colors, and there is a binary disequality constraint corresponding to each edge. The primal graph (or Gaifman graph) of a CSP instance is defined to be a graph on the variables of the instance such that there is an edge between two variables if and only if they appear together in some constraint. If the treewidth of the primal graph is t, then CSP can be solved in time $n^{O(t)}$. Since L C can be interpreted as a CSP problem, Theorem 6.5 immediately implies that we cannot improve the exponent to $o(t)$.

Theorem 6.6. *Assuming ETH, CSP cannot be solved in time $f(t)n^{o(t)}$, where t is the treewidth of the primal graph.*

This result seems to suggest that there is no faster way of solving CSP than using the algorithm based on tree decompositions. However, this result does not rule out the possibility that there is a class of graphs (say, planar graphs, bounded degree graphs, expanders, etc.) such that an $n^{o(t)}$ algorithm is possible if the primal graphs is in this class. We would like to have a lower bound that says something about each particular class of graphs. To make this formal, for a class $\mathcal{G}$ of graphs, let CSP($\mathcal{G}$) be the class of all CSP instances where the primal graph of the instance is in $\mathcal{G}$. In [47], Marx showed a lower bound on CSP($\mathcal{G}$) for every fixed class $\mathcal{G}$.

Theorem 6.7 ([47])**.** *If there is a recursively enumerable class $\mathcal{G}$ of graphs with unbounded treewidth and a function f such that binary CSP($\mathcal{G}$) can be solved in time $f(G)|I|^{o(\mathbf{tw}(G)/\log \mathbf{tw}(G))}$ for instances I with primal graph $G \in \mathcal{G}$, then ETH fails.*

Binary CSP($\mathcal{G}$) is the special case of CSP($\mathcal{G}$) where every constraint is binary, that is, it involves two variables. Note that adding this restriction makes the statement of Theorem 6.7 stronger.

Other structural parameters. If we restrict ourselves to paths rather than trees in the definition of tree decompositions, then this corresponds to *path decomposition* and the minimum width over all path decompositions of G is called *pathwidth* of G, denoted by $\mathbf{pw}(G)$. Clearly, $\mathbf{pw}(G) \geq \mathbf{tw}(G)$ and actually pathwidth can be unbounded even for trees. Therefore, it is somewhat surprising that the reductions in the proof of Theorem 6.4 constrain not only the treewidth of the constructed graphs but also the pathwidth. It follows that all the lower bounds mentioned in Theorem 6.4 also hold for problems on graphs of bounded pathwidth.

There are also other kinds of graph decomposition and corresponding width measures, like cliquewidth and rankwidth, that can be much smaller than treewidth. Several algorithms for NP-hard problems parameterized by these width measures have been obtained [16]. For various basic problems like M C and E D S that are W[1]-hard parameterized by cliquewidth, lower bounds of form $n^{O(w)}$, where w is the cliquewidth of the input graph, was obtained in [27]. Broersma et al. [8] gave lower bounds for some problems that are FPT parameterized by cliquewidth.

7 Conclusion

In this article we surveyed algorithmic lower bound results that have been obtained in the field of exact exponential time algorithms and parameterized complexity using ETH and SETH. For a wide range of problems, these lower bounds give useful

information about what kind of algorithms are possible, in many cases even showing the optimality of the current best algorithms. However, all these results have to be taken with caution: there is no universal consensus about accepting ETH and especially SETH (compared to say, accepting P ≠ NP or FPT ≠ W[1]). However, if one is reluctant to accept these hypotheses, the lower bounds following from them still carry a very useful message. These lower bounds say that going beyond these barriers implies an improved algorithm not only for this specific problem at hand but also for satisfiability. Therefore, the tight lower bounds discussed in this paper can be interpreted as saying that instead of trying to improve the current best algorithm, one's effort is better spent at trying to improve satisfiability algorithms directly. In particular, we cannot expect that some problem-specific idea related to the concrete problem can help, as we eventually need ideas that improve satisfiability.

We did not touch all the lower bound results obtained under the assumption of ETH and SETH. For an example, assuming ETH, it is possible to prove lower bound on the form of running time of (efficient) polynomial time approximation schemes ((E)PTAS) for several problems. We refer to [45] for further details. We conclude the survey with several intriguing questions which remain open.

1. Could we relate ETH and SETH to some other known complexity theory assumptions?

2. Could we use SETH to obtain lower bound on the base of the exponent of problems parameterized by the solution size?

3. Could we use ETH to show that running time of the form $2^{O(k^2)} \cdot n^{O(1)}$ is best possible for some natural parameterized problem?

4. Could we use SETH to obtain a lower bound of form c^n for some fixed constant c for problems like D S and I S when parameterized by the number of vertices of the input graph?

5. Could we use SETH to show that the C N of a graph on n vertices cannot be solved in time $(2 - \epsilon)^n$ for any fixed $\epsilon > 0$?

References

[1] J. A J. F , *Geometric separation and exact solutions for the parameterized independent set problem on disk graphs*, J. Algorithms, 52 (2004), pp. 134–151.

[2] N. A , D. L , S. S , *Fast fast*, in ICALP (1), 2009, pp. 49–58.

[3] R. B , *Dynamic programming treatment of the travelling salesman problem*, J. ACM, 9 (1962), pp. 61–63.

[4] A. B ¨ , T. H , P. K , M. K , *Fourier meets möbius: fast subset convolution*, in STOC, 2007, pp. 67–74.

[5] A. B ¨ , T. H , M. K , *Set partitioning via inclusion-exclusion*, SIAM J. Comput., 39 (2009), pp. 546–563.

[6] H. L. B , *A tourist guide through treewidth*, Acta Cybernet., 11 (1993), pp. 1–21.

[7] H. L. B A. M. C. A. K , *Combinatorial optimization on graphs of bounded treewidth*, Comput. J., 51 (2008), pp. 255–269.

[8] H. B , P. A. G , V. P , *Tight complexity bounds for FPT subgraph problems parameterized by clique-width*. Accepted to IPEC 2011.

[9] L. C D. W. J , *On the existence of subexponential parameterized algorithms*, J. Comput. Syst. Sci., 67 (2003), pp. 789–807.

[10] C. C , R. I , R. P , *The complexity of satisfiability of small depth circuits*, in IWPEC, 2009, pp. 75–85.

[11] J. C , B. C , M. F , X. H , D. W. J , I. A. K , G. X , *Tight lower bounds for certain parameterized np-hard problems*, Inf. Comput., 201 (2005), pp. 216–231.

[12] J. C , X. H , I. A. K , G. X , *On the computational hardness based on linear fpt-reductions*, J. Comb. Optim., 11 (2006), pp. 231–247.

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

[14] J. C , I. A. K , G. X , *Improved parameterized upper bounds for vertex cover*, in MFCS, 2006, pp. 238–249.

[15] Z.-Z. C , B. M , L. W , *A three-string approach to the closest string problem*. Accepted to COCOON 2010.

[16] B. C , J. A. M , U. R , *Linear time solvable optimization problems on graphs of bounded clique-width*, Theory Comput. Syst., 33 (2000), pp. 125–150.

[17] M. C , H. D , D. L , D. M , J. N , Y. O , R. P , S. S , M. W ¨ , *On problems as hard as cnfsat*. July 2011.

[18] M. C , J. N , M. P , M. P , J. M. M. R , J. O. W , *Solving connectivity problems parameterized by treewidth in single exponential time*, To appear in FOCS, abs/1103.0534 (2011).

[19] E. D. D , F. V. F , M. H , D. M. T , *Subexponential parameterized algorithms on graphs of bounded-genus and H-minor-free graphs*, Journal of the ACM, 52 (2005), pp. 866–893.

[20] E. D. D M. H , *Fast algorithms for hard graph problems: Bidimensionality, minors, and local treewidth*, in Proceedings of the 12th International Symposium on Graph Drawing (GD 2004), vol. 3383 of Lecture Notes in Computer Science, Harlem, New York, September 29–October 2 2004, pp. 517–533.

[21] ——, *Bidimensionality: new connections between fpt algorithms and ptass*, in SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, 2005, Society for Industrial and Applied Mathematics, pp. 590–601.

[22] R. G. D M. R. F , *Parameterized Complexity*, Monographs in Computer Science, Springer, New York, 1999.

[23] M. R. F , *Personal communication*, 2011.

[24] M. R. F , F. V. F , D. L , E. L , F. A. R , S. S , *Distortion is fixed parameter tractable*, in ICALP (1), 2009, pp. 463–474.

[25] M. R. F , F. V. F , D. L , F. A. R , S. S , S. S , C. T , *On the complexity of some colorful problems parameterized by treewidth*, Inf. Comput., 209 (2011), pp. 143–153.

[26] J. F M. G , *Parameterized Complexity Theory*, Springer, Berlin, 2006.

[27] F. V. F , P. A. G , D. L , S. S , *Algorithmic lower bounds for problems parameterized with clique-width*, in SODA '10: Proceedings of the twenty first annual ACM-SIAM symposium on Discrete algorithms, 2010, pp. 493–502.

[28] F. V. F , F. G , D. K , *A measure & conquer approach for the analysis of exact algorithms*, J. ACM, 56 (2009).

[29] F. V. F D. K , *Exact Exponential Algorithms*, Springer, 2010.

[30] M. R. G , D. S. J , R. E. T , *The planar hamiltonian circuit problem is np-complete*, SIAM J. Comput., 5 (1976), pp. 704–714.

[31] J. G , R. N , P. R , *Fixed-parameter algorithms for closest string and related problems*, Algorithmica, 37 (2003), pp. 25–42.

[32] M. H R. M. K , *A dynamic programming approach to sequencing problems*, 10 (1962), pp. 196–210.

33 , 37 mr1894519

[33] T. H , *3-SAT faster and simpler - unique-sat bounds for ppsz hold in general*, to appear in FOCS, abs/1103.2165 (2011).

[34] E. A. H , *New worst-case upper bounds for sat*, J. Autom. Reasoning, 24 (2000), pp. 397–420.

[35] R. I R. P , *On the complexity of k-sat*, J. Comput. Syst. Sci., 62 (2001), pp. 367–375.

[36] R. I , R. P , F. Z , *Which problems have strongly exponential complexity?*, Journal of Computer and System Sciences, 63 (2001), pp. 512–530.

[37] R. I , R. P , F. Z , *Which problems have strongly exponential complexity?*, J. Comput. System Sci., 63 (2001), pp. 512–530.

[38] T. K , *Treewidth, Computations and Approximations*, vol. 842 of Lecture Notes in Computer Science, Springer, 1994.

[39] J. K , A. L , P. R , *A fine-grained analysis of a simple independent set algorithm*, in FSTTCS, vol. 4 of LIPIcs, 2009, pp. 287–298.

[40] E. L. L , *A note on the complexity of the chromatic number problem*, Inf. Process. Lett., 5 (1976), pp. 66–67.

[41] D. L , *New Methods in Parameterized Algorithms and Complexity.*, PhD thesis, University of Bergen, 2009.

[42] D. L , D. M , S. S , *Known algorithms on graphs on bounded treewidth are probably optimal*, in SODA '11: Proceedings of the twenty second annual ACM-SIAM symposium on Discrete algorithms, 2011, pp. 777–789.

[43] ——, *Slightly superexponential parameterized problems*, in SODA '11: Proceedings of the twenty second annual ACM-SIAM symposium on Discrete algorithms, 2011, pp. 760–776.

[44] B. M X. S , *More effcient algorithms for closest string and substring problems*, SIAM J. Comput., 39 (2009), pp. 1432–1443.

[45] D. M , *On the optimality of planar and geometric approximation schemes*, in FOCS, IEEE Computer Society, 2007, pp. 338–348.

[46] ——, *Closest substring problems with small distances*, SIAM Journal on Computing, 38 (2008), pp. 1382–1410.

[47] ——, *Can you beat treewidth?*, Theory of Computing, 6 (2010), pp. 85–112.

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

[49] M. P R. W , *On the possibility of faster SAT algorithms*, in SODA '10: Proceedings of the twenty first annual ACM-SIAM symposium on Discrete algorithms, 2010, pp. 1065–1075.

[50] J. M. R , *Algorithms for maximum independent sets*, J. Algorithms, 7 (1986), pp. 425–440.

[51] M. S , *Introduction to the theory of computation*, PWS Publishing Company, 1997.

[52] J. M. M. R , H. L. B , P. R , *Dynamic programming on tree decompositions using generalised fast subset convolution*, in ESA, 2009, pp. 566–577.

[53] J. M. M. R , J. N , T. C. D , *Inclusion/exclusion meets measure and conquer*, in ESA, vol. 5757 of Lecture Notes in Computer Science, 2009, pp. 554–565.

[54] L. W B. Z , *Effcient algorithms for the closest string and distinguishing string selection problems*, in FAW, 2009, pp. 261–270.

[55] M. X , *Algorithms for multiterminal cuts*, in CSR, 2008, pp. 314–325.