DM898 Parameterized Algorithms
Fall 2025

Lars Rohwedder
rohwedder@sdu.dk

Exponential Time Hypothesis

Lecture Notes

The purpose of these notes is to give a proof that there is no FPT algorithm for the Clique problem assuming that the Exponential Time Hypothesis (ETH) holds.

This is proven by a combination of the Sparsification Lemma and a reduction of 3-SAT to an instance of Clique with clique size roughly $1/\varepsilon$ and graph size roughly $2^{\varepsilon m}$. The Sparsification Lemma is from [2].

The emphasis of these notes is on simplicity and self-containedness, for which we accept slightly worse bounds and lower generality. Specifically, while the paper above and most other sources give a proof for k-SAT (or k-CSP), we only prove it for 3-SAT as this suffices for our purposes.

If you notice errors, please send them by email.

3-SAT and Exponential Time Hypothesis

Recall that a 3-SAT formular consists of n Boolean variables $x_1, \dots, x_n$ and a logical AND over m clauses, each forming a logical OR of at most 3 literals. A literal is a variable x_i or a negated variable $\neg x_i$. The 3-SAT problem asks for a given 3-SAT formular to find an assignment of variables satisfying every clause. Depending on preferences one might require each clause to have exactly 3 literals. This does not make a difference for our purposes and here we make no such assumption. As an example consider the formular:

$$(x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2)$$

A satisfying assignment is $x_1 = \text{false}, x_2 = \text{true}, x_3 = \text{true}, x_4 = \text{true}$. A trivial enumeration algorithm achieves a running time of $O(2^n)$. We have also seen an algorithm in the exercises that achieves a slightly faster exponential running time, namely $O(3^{n/2})$.

As shown by the Cook-Levin Theorem, the infamous hypothesis P≠NP is equivalent to the hypothesis that 3-SAT has no polynomial time algorithm. While this has become a standard assumption in complexity theory and is heavily discussed, we are in fact not even close to polynomial running time: there are many natural running times between polynomial and exponential, for example, $O(n^{\log n}), O(2^{\sqrt{n}})$ or $O(2^{n/\log\log n})$. None of them have been achieved for the 3-SAT problem. The best algorithm so far runs in roughly $O(2^{0.387n})$ time, that is, better only by a constant factor in the exponent than the trivial enumeration algorithm.

This may lead us to believe that perhaps exponential time is required to solve 3-SAT. We formalize it in the following hypothesis:

Definition 1 (Exponential Time Hypothesis (ETH))**.** There exists some constant $\varepsilon > 0$ such that there is no $O(2^{\varepsilon n})$ time algorithm for 3-SAT.

In other words, while we expect that some further constant improvement in the exponent may be possible through additional tricks as the previous improvements, there could be an inherent limit preventing us from any super-constant improvements in the exponent.

The ETH is unproven and even more controversal than P≠NP. The ETH would of course imply P≠NP, so there is little hope to see a proof of it any time soon. It still serves an important purpose: If we are willing to believe in the ETH, then a lot of interesting lower bounds follow for a wide range of problems. Even if we do not believe it, we can acknowledge that it is not an easy

hypothesis to refute and therefore any consequences we prove based on it will also be difficult to refute.

We will see in the following how the ETH implies that there is no FPT algorithm for the clique problem, with the size of the largest clique as a natural parameter. After this, we will have a starting point for parameterized reductions that we can use to show that all kinds of problems do not have FPT algorithms assuming ETH. Note that the standard toolbox from parameterized complexity is typically based on the W-hierarchy, which we will not detail here. This is closely related and the hypotheses used there, e.g. W[1] $\neq$ FPT, are implied by the ETH. Hence, if we are willing to accept the ETH (the stronger assumption), then the long list of consequences based on W-hierarchy also follows. We refer to the textbook [1] for more information. A nice aspect of the proof via the ETH is that it gives us evidence against the existence of FPT algorithms based on an assumption that is not inherently about parameterization, but naturally relates to the existing complexity assumption of P$\neq$NP.

Sparsification Lemma

Before we can show hardness for Clique, we need an important result that essentially says that with respect to exponential time, a 3-SAT instance with very many clauses is not harder than one with only $O(n)$ clauses. The latter is called a *sparse* instance.

Theorem 2 (Sparsification Lemma)**.** *If the ETH is true, then there is some $\varepsilon' > 0$ such that there is no $O(2^{\varepsilon'(n+m)})$ time algorithm for 3-SAT.*

Note that in general the number of clauses (without repetitions) can be as large as $O(n^3)$. Hence, an algorithm with running time $O(2^{\varepsilon m})$ would not trivially refute the ETH.

Assume that for every constant $\varepsilon' > 0$ there is an algorithm Sparse$_{\varepsilon'}$ that solves 3-SAT in time $O(2^{\varepsilon'(n+m)})$ We will show that this implies that also for every constant $\varepsilon > 0$ there exists an algorithm for 3-SAT with running time $O(2^{\varepsilon n})$; hence refuting the ETH.

Towards this, we define an intricate branching algorithm that in its leafs has only sparse instances where it will call algorithm Sparse$_{\varepsilon}$ as a subroutine. The number of leafs will be roughly $2^{\varepsilon n}$. The algorithm uses a constant $0 < \delta \leq 1/10$ that will influence the running time and we think of it as very small. We will specify its value later. Roughly speaking, the smaller δ is, the faster the algorithm.

Pseudo-sunflowers. We consider the clauses to be sets of literals. Then a set of clauses $S_1, \ldots, S_k$ of the same size $|S_1| = \cdots = |S_k|$ and with non-empty intersection $Y = \bigcap_{i=1}^{k} S_i \neq \emptyset$ is called a pseudo-sunflower. We call Y the core and $S_1 \setminus Y, \ldots, S_k \setminus Y$ the petals. Note that this structure differs from the sunflowers we saw earlier in kernelization algorithms. The differences are that here we require Y to be non-empty and we allow that the petals overlap; hence the name *pseudo*-sunflower. In a 3-SAT formular only three types of pseudo-sunflowers can occur, see Figure 1: a *small* sunflower with core size $|Y| = 1$ and petal size $|S_1 \setminus Y| = \cdots = |S_k \setminus Y| = 1$; a *medium* sunflower with core size $|Y| = 2$ and petal size $|S_1 \setminus Y| = \cdots = |S_k \setminus Y| = 1$; or a *large* sunflower with core size $|Y| = 1$ and petal size $|S_1 \setminus Y| = \cdots = |S_k \setminus Y| = 2$;

Branching over sunflowers. We devise a branching algorithm that carefully selects a sunflower $S_1, \ldots, S_k, Y$ and then branches on whether Y is true in the satisfying assignment (assuming there is one) or not. If a satisfying assignment also satisfies Y, then we can safely add Y to the set of clauses and the formular will still be satisfiable. If, on the other hand, Y is not satisfied by a satisfying assignment, then the assignment needs to satisfy all petals $S_1 \setminus Y, \ldots, S_k \setminus Y$. Thus, we can add $S_1 \setminus Y, \ldots, S_k \setminus Y$ to the clauses. If we had a NO-instance, then neither of these two branches can turn the instance into a YES-instance, since we are only adding clauses. Adding clauses seems counterproductive for our goal of making the instance more sparse by branching. However, we can now remove all clauses that strictly contain another clause. In particular, we can remove $S_1, \ldots, S_k$ in either case. The algorithm chooses a careful priority and bounds on the

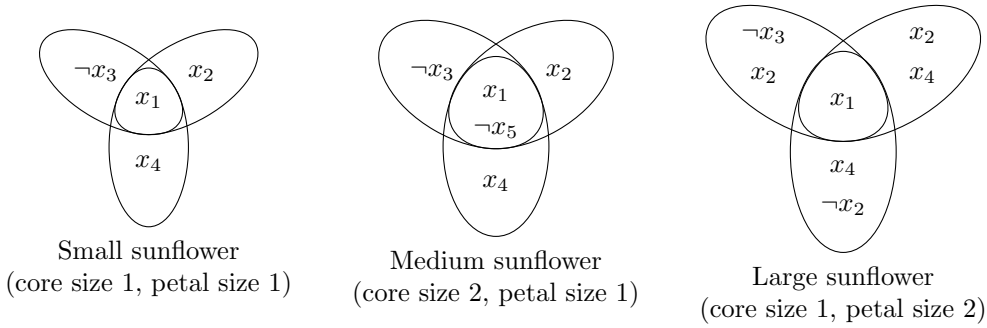

Figure 1: Types of pseudo-sunflowers

number of petals to choose the sunflower to branch on, see Figure 2 for the full algorithm. These details are very important for the analysis to work.

Analysis

It is straight-forward that the branching strategy is safe in the sense that the clauses of one node are satisfiable if and only if at least one of the clause sets of children constructed by branching is satisfiable. Removing clauses that strictly contain other clauses also does not affect satisfiability and returning *NO* in case that two contradictory singleton clauses exist is clearly the correct answer. Furthermore, the Sparse algorithm is only called with clause sets of linear size.

Lemma 3. *When* Sparse *is called, then* $|\mathcal{C}| < 6n/\delta^2$.

Proof. Suppose that $|\mathcal{C}| \geq 6n/\delta^2$. Since there are $2n$ different literals, there is a literal that appears in at least $3/\delta^2$ clauses. Out of these clauses at least $1/\delta^2$ many must have the same size. These clauses form a sunflower of size at least $1/\delta^2$; hence, the algorithm would branch instead of calling Sparse. □

To bound the running time and to prove that the algorithm terminates at all, we need to bound the number of leafs of the enumeration tree. Towards this, we consider a path P from the root to a leaf of the tree. It is natural to hope that the length of any such path P is bounded by δn (or similar), which would give a bound of $2^{\delta n}$ on the number of leafs. We call the algorithm Sparse_{δ^3} with at most $6n/\delta^2$ clauses; hence its running time is $O(2^{\delta^3(n+6n/\delta^2)})$. The total running time would therefore be $n^{O(1)} \cdot 2^{8\delta n}$, which by choosing $\delta = \varepsilon/9$ would solve the 3-SAT instance in time $O(2^{\varepsilon n})$. Unfortunately, the length of P can be large in general.

We will carefully analyze the number of core branches (where Y is satisfied) and petal branches (where Y is not satisfied) in P. We will show that although the number of core branches can be large, the number of petal branches is very small, which will suffice to give a good bound on the number of leafs.

Therefore, we look at the six combinations of small, medium, large branches with the core and petal variant in each. We start with some simple bounds.

Lemma 4. *The number of small and large core branches in any path P is at most n.*

Proof. In either case we have that $|Y| = 1$ and Y is added to the clause set, which means the number of singleton clauses increases, essentially fixing the truth value of one variable. Since each variable can only occur once in a singleton clause and we never remove any singleton clauses, the number of small and large core branches is at most n. □

Input: variables V and clauses $\mathcal{C}$, where each clause is a set of at most 3 literals.
Output: YES if there exists truth assignment that satisfies all clauses $\mathcal{C}$ or NO otherwise.

While there are clauses $C, C' \in \mathcal{C}$ with $C \subsetneq C'$

- $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C'\}$ // *C implies C'*

If $\{x_i\} \in \mathcal{C}$ and $\{\neg x_i\} \in \mathcal{C}$ for some $x_i \in V$

- return No

If there is a small sunflower $Y, S_1, \ldots, S_k$ with $k \geq 1/\delta$

- return Alg(V, $\mathcal{C} \cup \{Y\}$) or Alg(V, $\mathcal{C} \cup \{S_1 \setminus Y, \ldots, S_k \setminus Y\}$)

else if there is a medium sunflower $Y, S_1, \ldots, S_k$ with $k \geq 1/\delta$

- return Alg(V, $\mathcal{C} \cup \{Y\}$) or Alg(V, $\mathcal{C} \cup \{S_1 \setminus Y, \ldots, S_k \setminus Y\}$)

else if there is a large sunflower $Y, S_1, \ldots, S_k$ with $k \geq 1/\delta^2$

- return Alg(V, $\mathcal{C} \cup \{Y\}$) or Alg(V, $\mathcal{C} \cup \{S_1 \setminus Y, \ldots, S_k \setminus Y\}$)

else

- return $\text{Sparse}_{\delta^3}(V, \mathcal{C})$

Figure 2: Branching algorithm

Lemma 5. *The number of small and medium petal branches is at most δn.*

Proof. In either case we have that $|S_i \setminus Y| = 1$ for each petal and the number of petals is at least $1/\delta$. All petals are added to the clause set. Therefore, the number of singleton clauses increases by $1/\delta$. As in the previous lemma, each singleton clause can only be added once. □

For the last two types of branches these simple arguments are not sufficient. To handle them, it helps to take a closer look at *new* 2-clauses, that is, clauses consisting of exactly two literals that have not been in the original clause set. In other words, 2-clauses that were added by some branching within the path P. This is relevant, because both of the remaining cases add new 2-clauses.

Lemma 6. *Let $Y, S_1, \ldots, S_k$ be a small sunflower (i.e., $|Y| = |Y \setminus S_1| = \cdots = |Y \setminus S_k| = 1$) consisting only of new 2-clauses. Then $k \leq 2/\delta$.*

Proof. We will show that this is an invariant that is maintained by each branching. Clearly, in the initial clause set it holds, since there are no new clauses. Let k the the maximal size of a small sunflower consisting of new 2-clauses.

Case 1: $k \geq 1/\delta$. Then the next branch is either a small core branch or a small petal branch, since small branches take priority over all others. Either variant does not add a 2-clause. Thus, k does not increase.

Case 2: $k \leq 1/\delta$. In this case we may add new 2-clauses through either a medium core branch or a large petal branch. It suffices to show that one such branch cannot add more than $1/\delta$ many 2-clauses with a literal in common. Then the size of a small sunflower consisting of new 2-clauses can still only have size $2/\delta$: at most $1/\delta$ from before the branch and at most $1/\delta$ added through the branch.

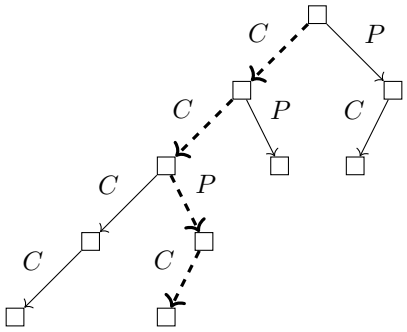

Figure 3: Leaf in enumeration tree identified by core and petal branches

A medium core branch only adds a single 2-clause. Hence, consider a large petal branch on a sunflower $Y', S'_1, \ldots, S'_h$. Assume that the large petal branch adds more than $1/\delta$ many 2-clauses that have a literal in common. In other words, there are $i_1 < i_2 < \cdots < i_{h'}$ with $Y'' := \bigcap_{j=1}^{h'} S'_{i_j} \setminus Y' \neq \emptyset$ and $h' > 1/\delta$. In this case $Y' \cup Y'', S'_{i_1}, \ldots, S'_{i_{h'}}$ forms a medium sunflower with more than $1/\delta$ petals, which would have taken priority over the large sunflower. A contradiction. □

The invariant above lets us bound the total number of new 2-clauses added throughout P.

Lemma 7. *For any path P, at most $4/\delta \cdot n$ many new 2-clauses are added.*

Proof. Note that at any leaf there are at most $2n/\delta$ many 2-clauses, old or new. Otherwise, one literal would appear in at least $1/\delta$ many 2-clauses, resulting in a sufficiently large small sunflower to branch on. Hence, the number of added 2-clauses is at most $2n/\delta$ plus the number of removed 2-clauses during P. Only when we add a singleton clause, we remove a 2-clause and each added singleton claus can only remove $2/\delta$ many 2-clauses because of Lemma 6. Hence the number of new 2-clauses added in the entire path can be at most

$$2n/\delta + n \cdot 2/\delta = 4/\delta \cdot n \ .$$ □

Lemma 8. *The number of medium core branches is at most $4/\delta \cdot n$.*

Proof. Each medium core branch adds a new 2-clause. Only $4/\delta \cdot n$ new 2-clauses can be added. □

Lemma 9. *The number of large petal branches is at most $4\delta \cdot n$.*

Proof. Each large petal branch adds $1/\delta^2$ many 2-clauses. Only $4/\delta \cdot n$ many 2-clauses can be added. □

Thus, the total number of petal branches is at most $\delta n + \delta n + 4\delta n \leq 6\delta n$. The total number of core branches is at most $n + n + 4/\delta \cdot n \leq 6/\delta \cdot n$. Each leaf is uniquely identified by how many petal and core branches are taken and in which order on the path to the leaf, see Figure 3. It follows that the number of leafs of the branching tree is at most

$$\sum_{i=0}^{\lfloor 6/\delta \cdot n \rfloor} \sum_{k=0}^{\lfloor 6\delta n \rfloor} \binom{i+k}{k} \ .$$

The following is a fairly good bound of the binomial coefficient. It is tight up to the factor of e in the basis.

Lemma 10. *For all $a \geq b$,*

$$\binom{a}{b} \leq \left(\frac{ia}{b}\right)^b \ .$$

Furthermore, the right-hand side of equation above is clearly non-decreasing in a. It is also non-decreasing in b for $b \in [0, ea/2]$. The proof is relatively simple, but we omit it here. Thus, each element from the previous sum can be bounded as

$$\binom{i+k}{k} \leq \left(\frac{e(i+k)}{k}\right)^k \leq \left(\frac{e \cdot 7/\delta \cdot n}{6\delta n}\right)^{6\delta n} \leq \left(\frac{3}{\delta^2}\right)^{6\delta n} \leq 2^{12\log(3/\delta)\cdot\delta\cdot n} .$$

The total running time of the algorithm, including the calls to $\mathrm{Sparse}_{\delta^3}$ is therefore

$$n^{O(1)} \cdot 2^{12\log(3/\delta)\cdot\delta\cdot n} \cdot 2^{\delta^3(n+6n/\delta^2)} \leq 2^{19\log(3/\delta)\cdot\delta\cdot n} .$$

We choose $\delta > 0$ such that $19\log(3/\delta)\cdot\delta \leq \varepsilon$, resulting in a running time of $O(2^{\varepsilon n})$. This finishes the analysis.

ETH-hardness of Clique

We are now equipped to prove the hardness of Clique.

Theorem 11. *Unless the ETH fails, there is no FPT algorithm for the Clique problem parameterized by clique size.*

Proof. We will assume that there is an FPT algorithm for the Clique problem and then derive a fast algorithm for 3-SAT from it. Towards this, assume that there is an algorithm that solves Clique in time $f(k) \cdot n^C$, where $C \geq 1$ is a constant and f is a computable function.

Let $\varepsilon > 0$. We will devise an algorithm solving 3-SAT in time $O(2^{\varepsilon m})$. Using the Sparsification Lemma and the fact that we can choose ε arbitrarily, this refutes the ETH (under our assumptions). Compute an arbitrary partition P of the clauses of the 3-SAT formular into sets of $\lfloor \varepsilon m/(6C) \rfloor$ many clauses each and possibly one set with fewer clauses. Then the number of these clause sets is $|P| = \lceil m/\lfloor \varepsilon m/(6C) \rfloor \rceil \leq O(1/\varepsilon)$.

Since each clause set $A \in P$ contains at most $\varepsilon m/(6C)$ clauses and each clause contains at most 3 literals, the total number of different variables $X(A)$ that appear in A is at most $|X(A)| \leq \varepsilon m/(2C)$. Let us now enumerate every possible partial variable assignment of variables appearing in A. Denote by $\Lambda(A)$ the set of all partial variable assignments $\lambda : X(A) \to \{\text{true}, \text{false}\}$ that satisfy all clauses in A. Then $|\Lambda(A)| \leq 2^{\varepsilon m/(2C)}$.

Now create the following instance of the Clique problem. For every $A \in P$ and every $\lambda \in \Lambda(A)$ we introduce one vertex $v(A, \lambda)$. Then for every pair of vertices $u = v(A, \lambda), w = v(A', \lambda')$, where $A \neq A'$, we add an edge (u, w) if and only if $\lambda(i) = \lambda'(i)$ for all $i \in X(A) \cap X(A')$, that is, the two partial variable assignments have no variable on which they disagree. There are no edges between vertices $v(A, x), v(A, x')$ corresponding to the same clause set $A \in P$.

We now prove that the graph has a clique of size $|P|$ if and only if the 3-SAT formular is satisfiable. Assume that the 3-SAT formular is satisfiable with a variable assignment λ. For each $A \in P$ let $\lambda_{|X(A)} : X(A) \to \{\text{true}, \text{false}\}, i \mapsto \lambda(i)$ be the partial variable assignment that is consistent with λ. Then the vertices $v(A, \lambda_{|X(A)})$, $A \in P$, are a clique of size $|P|$.

For the other direction, assume that there exists a clique of size $|P|$. Since there is no edge between any vertices of the same set $A \in P$, the clique must contain exactly one vertex $v(A, \lambda^A)$ for each $A \in P$. We define a variable assignment λ where $\lambda(i) = \lambda^A(i)$ for an arbitrary $A \in P$ with $i \in X(A)$. Note that the choice of A does not matter, since all clause sets that contain variable i will have the same truth assignment of i. The assignment λ is satisfying because every clause appears in some clause set $A \in P$ and one of its literals is true in λ^A. This is the same value that the variable has in λ; hence the clause is also satisfied by λ.

The graph can be constructed in time $O((2^{\varepsilon m/(2C)})^2) \leq O(2^{\varepsilon m})$. Using the FPT algorithm we can check in time $f(|P|) \cdot (2^{\varepsilon m/(2C)})^C \leq O(2^{\varepsilon m})$ whether there exists a clique of size $|P|$. Note that because $|P| = O(1/\varepsilon)$ is a constant, also $f(|P|)$ is constant. Thus, the overal running time for solving the 3-SAT instance is $O(2^{\varepsilon m})$. □

References

[1] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

[2] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.