ETH and SETH lower bounds

Michał Pilipczuk

Institute of Informatics, University of Warsaw, Poland

Parameterized Complexity Summer School

Vienna, September 1st, 2017

Motivation

- Take the Hamiltonian Cycle problem.

Motivation

- Take the HAMILTONIAN CYCLE problem.
- **Algorithms**:

Motivation

- Take the HAMILTONIAN CYCLE problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.

Motivation

- Take the Hamiltonian Cycle problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.

Motivation

- Take the HAMILTONIAN CYCLE problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
  - Currently fastest: $\mathcal{O}(1.657^n)$ [Björklund, 2010].

Motivation

- Take the HAMILTONIAN CYCLE problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
  - Currently fastest: $\mathcal{O}(1.657^n)$ [Björklund, 2010].
- **Lower bounds**:

Motivation

- Take the Hamiltonian Cycle problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
  - Currently fastest: $\mathcal{O}(1.657^n)$ [Björklund, 2010].
- **Lower bounds**:
  - No poly-time algorithm under $\mathrm{P} \neq \mathrm{NP}$.

Motivation

- Take the HAMILTONIAN CYCLE problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
  - Currently fastest: $\mathcal{O}(1.657^n)$ [Björklund, 2010].
- **Lower bounds**:
  - No poly-time algorithm under $\mathrm{P} \neq \mathrm{NP}$.
  - How much can you improve the constant 1.657?

Motivation

- Take the HAMILTONIAN CYCLE problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
  - Currently fastest: $\mathcal{O}(1.657^n)$ [Björklund, 2010].
- **Lower bounds**:
  - No poly-time algorithm under $\mathrm{P} \neq \mathrm{NP}$.
  - How much can you improve the constant 1.657?
  - Can you do *significantly* better, e.g. $2^{\mathcal{O}(n/\log n)}$ or $2^{\mathcal{O}(\sqrt{n})}$?

Motivation

- Take the HAMILTONIAN CYCLE problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
  - Currently fastest: $\mathcal{O}(1.657^n)$ [Björklund, 2010].
- **Lower bounds**:
  - No poly-time algorithm under P $\neq$ NP.
  - How much can you improve the constant 1.657?
  - Can you do *significantly* better, e.g. $2^{\mathcal{O}(n/\log n)}$ or $2^{\mathcal{O}(\sqrt{n})}$?
- Assumption P $\neq$ NP seems too weak to answer these questions.

Motivation

- Take the HAMILTONIAN CYCLE problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
  - Currently fastest: $\mathcal{O}(1.657^n)$ [Björklund, 2010].
- **Lower bounds**:
  - No poly-time algorithm under $\mathrm{P} \neq \mathrm{NP}$.
  - How much can you improve the constant 1.657?
  - Can you do *significantly* better, e.g. $2^{\mathcal{O}(n/\log n)}$ or $2^{\mathcal{O}(\sqrt{n})}$?
- Assumption $\mathrm{P} \neq \mathrm{NP}$ seems too weak to answer these questions.
- **Parameterized complexity**:

Motivation

- Take the Hamiltonian Cycle problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
  - Currently fastest: $\mathcal{O}(1.657^n)$ [Björklund, 2010].
- **Lower bounds**:
  - No poly-time algorithm under $\mathrm{P} \neq \mathrm{NP}$.
  - How much can you improve the constant 1.657?
  - Can you do *significantly* better, e.g. $2^{\mathcal{O}(n/\log n)}$ or $2^{\mathcal{O}(\sqrt{n})}$?
- Assumption $\mathrm{P} \neq \mathrm{NP}$ seems too weak to answer these questions.
- **Parameterized complexity**:
  - **FPT**: $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$, $\mathcal{O}^\star(2^{\mathcal{O}(k)})$, $\mathcal{O}^\star(k^{\mathcal{O}(k^2)})$, $\mathcal{O}^\star(2^{2^{\mathcal{O}(k)}})$, ...

Motivation

- Take the HAMILTONIAN CYCLE problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
  - Currently fastest: $\mathcal{O}(1.657^n)$ [Björklund, 2010].
- **Lower bounds**:
  - No poly-time algorithm under $\mathrm{P} \neq \mathrm{NP}$.
  - How much can you improve the constant 1.657?
  - Can you do *significantly* better, e.g. $2^{\mathcal{O}(n/\log n)}$ or $2^{\mathcal{O}(\sqrt{n})}$?
- Assumption $\mathrm{P} \neq \mathrm{NP}$ seems too weak to answer these questions.
- **Parameterized complexity**:
  - **FPT**: $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$, $\mathcal{O}^\star(2^{\mathcal{O}(k)})$, $\mathcal{O}^\star(k^{\mathcal{O}(k^2)})$, $\mathcal{O}^\star(2^{2^{\mathcal{O}(k)}})$, ...
  - **XP**: $n^{\mathcal{O}(\log k)}$, $n^{\mathcal{O}(\sqrt{k})}$, $n^{\mathcal{O}(k)}$, $n^{2^{\mathcal{O}(k)}}$, ...

Motivation

- Take the Hamiltonian Cycle problem.
- **Algorithms**:
  - Brute-force $\mathcal{O}(n!)$.
  - $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
  - Currently fastest: $\mathcal{O}(1.657^n)$ [Björklund, 2010].
- **Lower bounds**:
  - No poly-time algorithm under $\mathrm{P} \neq \mathrm{NP}$.
  - How much can you improve the constant 1.657?
  - Can you do *significantly* better, e.g. $2^{\mathcal{O}(n/\log n)}$ or $2^{\mathcal{O}(\sqrt{n})}$?
- Assumption $\mathrm{P} \neq \mathrm{NP}$ seems too weak to answer these questions.
- **Parameterized complexity**:
  - **FPT**: $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$, $\mathcal{O}^\star(2^{\mathcal{O}(k)})$, $\mathcal{O}^\star(k^{\mathcal{O}(k^2)})$, $\mathcal{O}^\star(2^{2^{\mathcal{O}(k)}})$, ...
  - **XP**: $n^{\mathcal{O}(\log k)}$, $n^{\mathcal{O}(\sqrt{k})}$, $n^{\mathcal{O}(k)}$, $n^{2^{\mathcal{O}(k)}}$, ...
- Assumption $\mathrm{FPT} \neq \mathrm{W}[1]$ seems too weak to separate these.

$3\mathrm{SAT}$: complexity status

- $3\mathrm{SAT}$: given formula φ in 3CNF with n variables and m clauses, decide whether φ is satisfiable.

3SAT: complexity status

- 3SAT: given formula φ in 3CNF with n variables and m clauses, decide whether φ is satisfiable.
  - 3CNF: φ is a conjunction of clauses, where each clause is a disjunction of at most 3 literals — variables or their negations.

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

3SAT: complexity status

- $\mathrm{3SAT}$: given formula φ in 3CNF with n variables and m clauses, decide whether φ is satisfiable.
    - 3CNF: φ is a conjunction of clauses, where each clause is a disjunction of at most 3 literals — variables or their negations.

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

- **Trivial**: $\mathcal{O}^\star(2^n)$

$\mathrm{3SAT}$: complexity status

- $\mathrm{3SAT}$: given formula φ in 3CNF with n variables and m clauses, decide whether φ is satisfiable.
  - 3CNF: φ is a conjunction of clauses, where each clause is a disjunction of at most 3 literals — variables or their negations.

$$(x_1 \lor x_2 \lor \neg x_3) \land (\neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_3)$$

- **Trivial**: $\mathcal{O}^\star(2^n)$
- **Smarter**:

$\mathrm{3SAT}$: complexity status

- $\mathrm{3SAT}$: given formula φ in 3CNF with n variables and m clauses, decide whether φ is satisfiable.
  - 3CNF: φ is a conjunction of clauses, where each clause is a disjunction of at most 3 literals — variables or their negations.

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

- **Trivial**: $\mathcal{O}^\star(2^n)$
- **Smarter**:
  - Take any clause not satisfied so far, and branch on the evaluations of the variables: there are at most 7 options for a 3-clause.

3SAT: complexity status

- 3SAT: given formula φ in 3CNF with n variables and m clauses, decide whether φ is satisfiable.
  - 3CNF: φ is a conjunction of clauses, where each clause is a disjunction of at most 3 literals — variables or their negations.

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

- **Trivial**: $\mathcal{O}^\star(2^n)$
- **Smarter**:
  - Take any clause not satisfied so far, and branch on the evaluations of the variables: there are at most 7 options for a 3-clause.
  - Hence the running time is $\mathcal{O}^\star(7^{n/3}) = \mathcal{O}(1.913^n)$.

3SAT: complexity status

- 3SAT: given formula φ in 3CNF with n variables and m clauses, decide whether φ is satisfiable.
  - 3CNF: φ is a conjunction of clauses, where each clause is a disjunction of at most 3 literals — variables or their negations.

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

- **Trivial**: $\mathcal{O}^\star(2^n)$
- **Smarter**:
  - Take any clause not satisfied so far, and branch on the evaluations of the variables: there are at most 7 options for a 3-clause.
  - Hence the running time is $\mathcal{O}^\star(7^{n/3}) = \mathcal{O}(1.913^n)$.
- **Currently fastest**: $\mathcal{O}(1.308^n)$ [PPSZ]

$\mathrm{3SAT}$: complexity status

- $\mathrm{3SAT}$: given formula φ in 3CNF with n variables and m clauses, decide whether φ is satisfiable.
  - 3CNF: φ is a conjunction of clauses, where each clause is a disjunction of at most 3 literals — variables or their negations.

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

- **Trivial**: $\mathcal{O}^\star(2^n)$
- **Smarter**:
  - Take any clause not satisfied so far, and branch on the evaluations of the variables: there are at most 7 options for a 3-clause.
  - Hence the running time is $\mathcal{O}^\star(7^{n/3}) = \mathcal{O}(1.913^n)$.
- **Currently fastest**: $\mathcal{O}(1.308^n)$ [PPSZ]
- For every q you can do $(2-\varepsilon_q)^n$ for $q\mathrm{SAT}$, but $\lim_{q\to\infty} \varepsilon_q = 0$.

Intuitive statements

- **Questions**:

Intuitive statements

- **Questions**:
  - Can you do *significantly* better, that is, obtain running time $2^{o(n)}$?

Intuitive statements

- **Questions**:
  - Can you do *significantly* better, that is, obtain running time $2^{o(n)}$?
  - If you do not have a bound on the clause length, can you do anything smarter than brute-force?

Intuitive statements

- **Questions**:
  - Can you do *significantly* better, that is, obtain running time $2^{o(n)}$?
  - If you do not have a bound on the clause length, can you do anything smarter than brute-force?

ETH and SETH, first attempt

ETH: $\mathrm{3SAT}$ cannot be solved in time $2^{o(n)}$.
SETH: $\mathrm{CNF\text{-}SAT}$ cannot be solved in time $\mathcal{O}(\alpha^n)$ for any $\alpha < 2$.

Actual statements

$$\delta_q = \inf\{c : \text{There is an } \mathcal{O}(2^{cn}) \text{ algorithm for } q\text{SAT}\}$$

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

There is a $c > 0$ such that $3\mathrm{SAT}$ cannot be solved in $\mathcal{O}(2^{cn})$ time.

Strong Exponential Time Hypothesis, SETH

$$\lim_{q \to \infty} \delta_q = 1.$$

Actual statements

$$\delta_q = \inf\{c : \text{There is an } \mathcal{O}(2^{cn}) \text{ algorithm for } q\mathrm{SAT}\}$$

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

There is a $c > 0$ such that $3\mathrm{SAT}$ cannot be solved in $\mathcal{O}(2^{cn})$ time.

Strong Exponential Time Hypothesis, SETH

$$\lim_{q \to \infty} \delta_q = 1.$$

- These statements are **stronger** than our first attempts.

Actual statements

$$\delta_q = \inf\{c : \text{There is an } \mathcal{O}(2^{cn}) \text{ algorithm for } q\text{SAT}\}$$

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

There is a $c > 0$ such that $3\mathrm{SAT}$ cannot be solved in $\mathcal{O}(2^{cn})$ time.

Strong Exponential Time Hypothesis, SETH

$$\lim_{q \to \infty} \delta_q = 1.$$

- These statements are **stronger** than our first attempts.
  - Often the weaker statements suffice, but the above are more robust.

Actual statements

$$\delta_q = \inf\{c : \text{There is an } \mathcal{O}(2^{cn}) \text{ algorithm for } q\text{SAT}\}$$

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

There is a $c > 0$ such that $3\mathrm{SAT}$ cannot be solved in $\mathcal{O}(2^{cn})$ time.

Strong Exponential Time Hypothesis, SETH

$$\lim_{q \to \infty} \delta_q = 1.$$

- These statements are **stronger** than our first attempts.
  - Often the weaker statements suffice, but the above are more robust.
- **Note**: Usually we allow two-sided error algorithms in the definition.

Actual statements

$$\delta_q = \inf\{c : \text{There is an } \mathcal{O}(2^{cn}) \text{ algorithm for } q\text{SAT}\}$$

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

There is a $c > 0$ such that $3\mathrm{SAT}$ cannot be solved in $\mathcal{O}(2^{cn})$ time.

Strong Exponential Time Hypothesis, SETH

$$\lim_{q \to \infty} \delta_q = 1.$$

- These statements are **stronger** than our first attempts.
  - Often the weaker statements suffice, but the above are more robust.
- **Note**: Usually we allow two-sided error algorithms in the definition.
- SETH $\Rightarrow$ ETH (nontrivial)

Actual statements

$$\delta_q = \inf\{c : \text{There is an } \mathcal{O}(2^{cn}) \text{ algorithm for } q\text{SAT}\}$$

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

There is a $c > 0$ such that $3\mathrm{SAT}$ cannot be solved in $\mathcal{O}(2^{cn})$ time.

Strong Exponential Time Hypothesis, SETH

$$\lim_{q \to \infty} \delta_q = 1.$$

- These statements are **stronger** than our first attempts.
  - Often the weaker statements suffice, but the above are more robust.
- **Note**: Usually we allow two-sided error algorithms in the definition.
- SETH $\Rightarrow$ ETH (nontrivial)
- Formulated by Impagliazzo, Paturi, and Zane in 2001.

Actual statements

$$\delta_q = \inf\{c : \text{There is an } \mathcal{O}(2^{cn}) \text{ algorithm for } q\mathrm{SAT}\}$$

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

There is a $c > 0$ such that $3\mathrm{SAT}$ cannot be solved in $\mathcal{O}(2^{cn})$ time.

Strong Exponential Time Hypothesis, SETH

$$\lim_{q \to \infty} \delta_q = 1.$$

- These statements are **stronger** than our first attempts.
  - Often the weaker statements suffice, but the above are more robust.
- **Note**: Usually we allow two-sided error algorithms in the definition.
- SETH $\Rightarrow$ ETH (nontrivial)
- Formulated by Impagliazzo, Paturi, and Zane in 2001.
- Nowadays, standard assumptions for fine-grained complexity theory.

Actual statements

$$\delta_q = \inf\{c : \text{There is an } \mathcal{O}(2^{cn}) \text{ algorithm for } q\text{SAT}\}$$

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

There is a $c > 0$ such that $3\mathrm{SAT}$ cannot be solved in $\mathcal{O}(2^{cn})$ time.

Strong Exponential Time Hypothesis, SETH

$$\lim_{q\to\infty} \delta_q = 1.$$

- These statements are **stronger** than our first attempts.
  - Often the weaker statements suffice, but the above are more robust.
- **Note**: Usually we allow two-sided error algorithms in the definition.
- SETH $\Rightarrow$ ETH (nontrivial)
- Formulated by Impagliazzo, Paturi, and Zane in 2001.
- Nowadays, standard assumptions for fine-grained complexity theory.
- ETH is widely believed, SETH is disputed.

Reductions

- We would like to transfer lower bounds via **reductions**.

Reductions

- We would like to transfer lower bounds via **reductions**.
- A reduction $\mathrm{3SAT} \to L$ and a too fast algorithm for L would give a too fast algorithm for $\mathrm{3SAT}$.

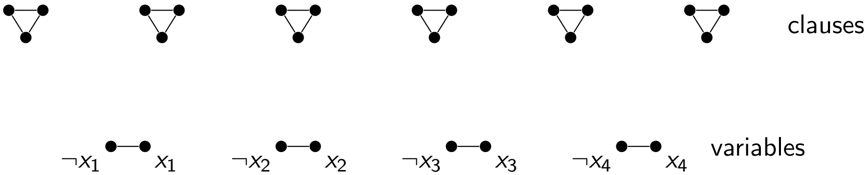
Reductions

- We would like to transfer lower bounds via **reductions**.
- A reduction $\mathrm{3SAT} \rightarrow L$ and a too fast algorithm for L would give a too fast algorithm for $\mathrm{3SAT}$.

Vertex Cover

I: graph G and $k \in \mathbb{N}$
Q: Is there a set $X \subseteq V(G)$ with $|X| \leqslant k$ such that every edge of G has at least one endpoint in X?

Let us inspect the standard reduction from 3SAT to Vertex Cover.

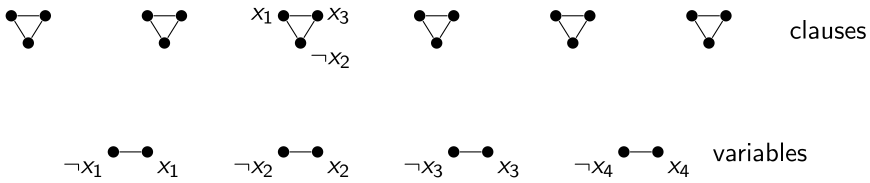


Variable gadgets $\rightsquigarrow$ Edges

Clause gadgets $\rightsquigarrow$ Triangles

Vertex Cover reduction

Let us inspect the standard reduction from 3SAT to Vertex Cover.

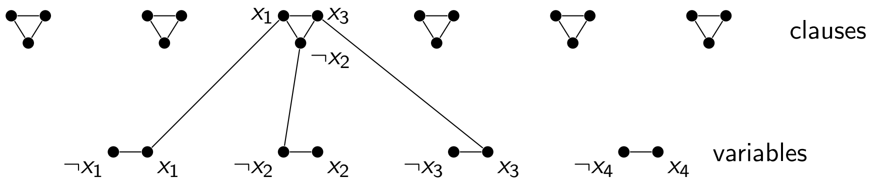


Variable gadgets $\rightsquigarrow$ Edges

Clause gadgets $\rightsquigarrow$ Triangles

Vertex Cover reduction

Let us inspect the standard reduction from 3SAT to Vertex Cover.

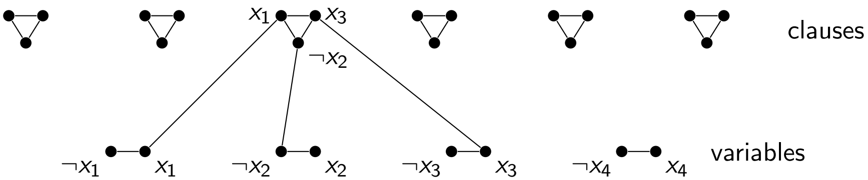


Variable gadgets $\rightsquigarrow$ Edges

Clause gadgets $\rightsquigarrow$ Triangles

Vertex Cover reduction

Let us inspect the standard reduction from 3SAT to Vertex Cover.



Variable gadgets $\rightsquigarrow$ Edges

Clause gadgets $\rightsquigarrow$ Triangles

φ is satisfiable iff the created graph has a vertex cover of size $n + 2m$.

Vertex Cover reduction, analysis

- If $N = 2n + 3m$ is the number of vertices of the output graph, then $N = \mathcal{O}(n+m) = \mathcal{O}(n^3)$.

VERTEX COVER reduction, analysis

- If $N = 2n + 3m$ is the number of vertices of the output graph, then $N = \mathcal{O}(n + m) = \mathcal{O}(n^3)$.
- Hence an $2^{o(N^{1/3})}$ algorithm for VC would give an $2^{o(n)}$ algorithm for 3-SAT, contradicting ETH.

$\textsc{Vertex Cover}$ reduction, analysis

- If $N = 2n + 3m$ is the number of vertices of the output graph, then $N = \mathcal{O}(n + m) = \mathcal{O}(n^3)$.
- Hence an $2^{o(N^{1/3})}$ algorithm for VC would give an $2^{o(n)}$ algorithm for $3\text{-}\mathrm{SAT}$, contradicting ETH.
- **Gap**: between upper bound $2^{\mathcal{O}(N)}$ and $2^{o(N^{1/3})}$.

Vertex Cover reduction, analysis

- If $N = 2n + 3m$ is the number of vertices of the output graph, then $N = \mathcal{O}(n+m) = \mathcal{O}(n^3)$.
- Hence an $2^{o(N^{1/3})}$ algorithm for VC would give an $2^{o(n)}$ algorithm for 3-SAT, contradicting ETH.
- **Gap**: between upper bound $2^{\mathcal{O}(N)}$ and $2^{o(N^{1/3})}$.
- If we started with an instance of 3-SAT that was **sparse**, that is, $m = \mathcal{O}(n)$, then a $2^{o(N)}$ lower bound would follow.

Sparsification Lemma

Sparsification Lemma informally

For every constant q, there is a subexponential-time algorithm which reduces the number of clauses of a qCNF formula to $\mathcal{O}(n)$.

Sparsification Lemma

Sparsification Lemma informally

For every constant q, there is a subexponential-time algorithm which reduces the number of clauses of a qCNF formula to $\mathcal{O}(n)$.

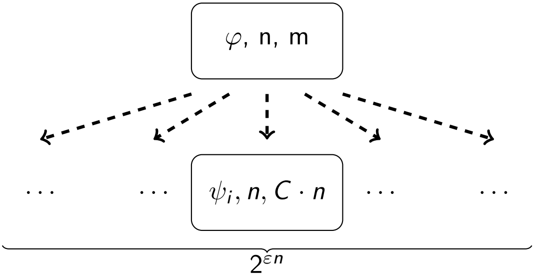
Sparsification Lemma [IPZ, 2001]

For any $q \geq 3$ and $\varepsilon > 0$, there is a constant $C = C(q, \varepsilon)$ such that any qCNF formula φ can be expressed as $\varphi = \bigvee_{i=1}^{t} \psi_i$, where $t \leq 2^{\varepsilon n}$ and each ψ_i is a qCNF formula with the same set of variables and at most $C \cdot n$ clauses. Such disjunction can be computed in time $\mathcal{O}^\star(2^{\varepsilon n})$.

Sparsification Lemma

Sparsification Lemma [IPZ, 2001]

For any $q \geq 3$ and $\varepsilon > 0$, there is a constant $C = C(q, \varepsilon)$ such that any qCNF formula φ can be expressed as $\varphi = \bigvee_{i=1}^{t} \psi_i$, where $t \leq 2^{\varepsilon n}$ and each ψ_i is a qCNF formula with the same set of variables and at most $C \cdot n$ clauses. Such disjunction can be computed in time $\mathcal{O}^\star(2^{\varepsilon n})$.



Sparsification Lemma: corollary

Theorem

Unless ETH fails, there is a constant $c > 0$, such that no algorithm solves $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.

Sparsification Lemma: corollary

Theorem

Unless ETH fails, there is a constant $c > 0$, such that no algorithm solves $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.

Proof by contradiction:

- Suppose that for every $c > 0$ there is an algorithm $\mathcal{A}_c$ solving $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.

Sparsification Lemma: corollary

Theorem

Unless ETH fails, there is a constant $c > 0$, such that no algorithm solves $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.

Proof by contradiction:

- Suppose that for every $c > 0$ there is an algorithm $\mathcal{A}_c$ solving $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.
- Consider any $d > 0$. We want to solve $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{dn})$.

Sparsification Lemma: corollary

Theorem

Unless ETH fails, there is a constant $c > 0$, such that no algorithm solves $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.

Proof by contradiction:

- Suppose that for every $c > 0$ there is an algorithm $\mathcal{A}_c$ solving $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.
- Consider any $d > 0$. We want to solve $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{dn})$.
- Use Sparsification Lemma for $\varepsilon = d/2$. Denote $C = C(3, \varepsilon)$.

Sparsification Lemma: corollary

Theorem

Unless ETH fails, there is a constant $c > 0$, such that no algorithm solves $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.

Proof by contradiction:

- Suppose that for every $c > 0$ there is an algorithm $\mathcal{A}_c$ solving $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.
- Consider any $d > 0$. We want to solve $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{dn})$.
- Use Sparsification Lemma for $\varepsilon = d/2$. Denote $C = C(3, \varepsilon)$.
- Solve each ψ_i by $\mathcal{A}_{c'}$, where $c' = \frac{d}{2(C+1)}$.

Sparsification Lemma: corollary

Theorem

Unless ETH fails, there is a constant $c > 0$, such that no algorithm solves $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.

Proof by contradiction:

- Suppose that for every $c > 0$ there is an algorithm $\mathcal{A}_c$ solving $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.
- Consider any $d > 0$. We want to solve $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{dn})$.
- Use Sparsification Lemma for $\varepsilon = d/2$. Denote $C = C(3, \varepsilon)$.
- Solve each ψ_i by $\mathcal{A}_{c'}$, where $c' = \frac{d}{2(C+1)}$.
- The total running time is

$$\mathcal{O}^\star(2^{\varepsilon n}) + \mathcal{O}^\star(2^{\varepsilon n} \cdot 2^{\frac{d}{2(C+1)} \cdot (C+1)n}) = \mathcal{O}^\star(2^{dn}).$$

Theorem

Unless ETH fails, there is a constant $c > 0$, such that no algorithm solves $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.

Proof by contradiction:

- Suppose that for every $c > 0$ there is an algorithm $\mathcal{A}_c$ solving $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{c(n+m)})$.
- Consider any $d > 0$. We want to solve $3\mathrm{SAT}$ in time $\mathcal{O}^\star(2^{dn})$.
- Use Sparsification Lemma for $\varepsilon = d/2$. Denote $C = C(3, \varepsilon)$.
- Solve each ψ_i by $\mathcal{A}_{c'}$, where $c' = \frac{d}{2(C+1)}$.
- The total running time is

$$\mathcal{O}^\star(2^{\varepsilon n}) + \mathcal{O}^\star(2^{\varepsilon n} \cdot 2^{\frac{d}{2(C+1)} \cdot (C+1)n}) = \mathcal{O}^\star(2^{dn}).$$

Corollary

Under ETH, there is no $2^{o(n+m)}$-time algorithm for $3\mathrm{SAT}$, so also no $2^{o(N+M)}$-time algorithm for Vertex Cover.

Transfer of hardness

Hardness under ETH via reductions

Suppose there is a reduction from $3\mathrm{SAT}$ to a problem L that produces an instance of size $N \leqslant f(n+m)$.

Then we can exclude an algorithm for L with running time $2^{o(f^{-1}(N))}$.

Transfer of hardness

Hardness under ETH via reductions

Suppose there is a reduction from $3\mathrm{SAT}$ to a problem L that produces an instance of size $N \leqslant f(n+m)$.

Then we can exclude an algorithm for L with running time $2^{o(f^{-1}(N))}$.

- The following problems admit **linear** reductions from $3\mathrm{SAT}$:
  - Feedback Vertex Set,
  - Dominating Set,
  - 3-Coloring,
  - Hamiltonian Cycle, ...

Transfer of hardness

Hardness under ETH via reductions

Suppose there is a reduction from $3\mathrm{SAT}$ to a problem L that produces an instance of size $N \leqslant f(n+m)$.

Then we can exclude an algorithm for L with running time $2^{o(f^{-1}(N))}$.

- The following problems admit **linear** reductions from $3\mathrm{SAT}$:
  - Feedback Vertex Set,
  - Dominating Set,
  - 3-Coloring,
  - Hamiltonian Cycle, ...
- Let's see tight bounds for another form of running time.

Planar problems

- Consider PLANAR VERTEX COVER.

Planar problems

- Consider Planar Vertex Cover.
- NP-**hardness**: Take an instance of general VC, and embed it in the plane replacing every crossing with:

Planar problems

- Consider PLANAR VERTEX COVER.
- NP-**hardness**: Take an instance of general VC, and embed it in the plane replacing every crossing with:

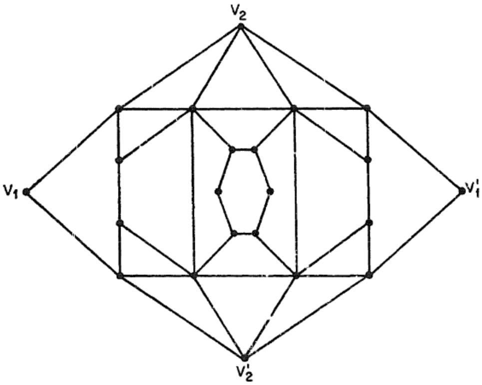


Fig. 11. Crossover H for Theorem 2.7.

PLVC: analysis

- Reduction takes an instance of VC with n vertices and m edges, and outputs an instance with $N = \mathcal{O}((n+m)^2)$ vertices.

$\textsc{PlVC}$: analysis

- Reduction takes an instance of VC with n vertices and m edges, and outputs an instance with $N = \mathcal{O}((n+m)^2)$ vertices.
- Hence, an $2^{o(\sqrt{N})}$-time algorithm for $\textsc{PlVC}$ would give an $2^{o(n+m)}$-time algorithm for VC, contradicting ETH.

$\textsc{PlVC}$: analysis

- Reduction takes an instance of VC with n vertices and m edges, and outputs an instance with $N = \mathcal{O}((n+m)^2)$ vertices.
- Hence, an $2^{o(\sqrt{N})}$-time algorithm for $\textsc{PlVC}$ would give an $2^{o(n+m)}$-time algorithm for VC, contradicting ETH.
- $\textsc{PlVC}$ actually **has** an algorithm working in time $2^{\mathcal{O}(\sqrt{N})}$.

PLVC: analysis

- Reduction takes an instance of VC with n vertices and m edges, and outputs an instance with $N = \mathcal{O}((n+m)^2)$ vertices.
- Hence, an $2^{o(\sqrt{N})}$-time algorithm for PLVC would give an $2^{o(n+m)}$-time algorithm for VC, contradicting ETH.
- PLVC actually **has** an algorithm working in time $2^{\mathcal{O}(\sqrt{N})}$:
    - Treewidth DP + planar graph on N vertices has treewidth $\mathcal{O}(\sqrt{N})$.

PlVC: analysis

- Reduction takes an instance of VC with n vertices and m edges, and outputs an instance with $N = \mathcal{O}((n+m)^2)$ vertices.
- Hence, an $2^{o(\sqrt{N})}$-time algorithm for PlVC would give an $2^{o(n+m)}$-time algorithm for VC, contradicting ETH.
- PlVC actually **has** an algorithm working in time $2^{\mathcal{O}(\sqrt{N})}$:
  - Treewidth DP $+$ planar graph on N vertices has treewidth $\mathcal{O}(\sqrt{N})$.
- The $\sqrt{N}$ term in the exponent is not a coincidence!

ETH implies FPT $\neq$ W[1]

- Consider the CLIQUE problem:

ETH implies FPT $\neq$ W[1]

- Consider the CLIQUE problem:
  - Given graph G and $k \in \mathbb{N}$, is there a clique on k vertices in G.

ETH implies FPT $\neq$ W[1]

- Consider the CLIQUE problem:
  - Given graph G and $k \in \mathbb{N}$, is there a clique on k vertices in G.
  - **Trivial**: $\mathcal{O}^\star(n^k)$.

ETH implies FPT $\neq$ W[1]

- Consider the Clique problem:
  - Given graph G and $k \in \mathbb{N}$, is there a clique on k vertices in G.
  - **Trivial**: $\mathcal{O}^\star(n^k)$.
  - **Fastest known**: $\mathcal{O}^\star(n^{\omega k/3}) = \mathcal{O}^\star(n^{0.791k})$ [Nešetřil, Poljak]

ETH implies FPT $\neq$ W[1]

- Consider the CLIQUE problem:
  - Given graph G and $k \in \mathbb{N}$, is there a clique on k vertices in G.
  - **Trivial**: $\mathcal{O}^\star(n^k)$.
  - **Fastest known**: $\mathcal{O}^\star(n^{\omega k/3}) = \mathcal{O}^\star(n^{0.791k})$ [Nešetřil, Poljak]
  - Under FPT $\neq$ W[1], no algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$.

ETH implies FPT $\neq$ W[1]

- Consider the CLIQUE problem:
  - Given graph G and $k \in \mathbb{N}$, is there a clique on k vertices in G.
  - **Trivial**: $\mathcal{O}^\star(n^k)$.
  - **Fastest known**: $\mathcal{O}^\star(n^{\omega k/3}) = \mathcal{O}^\star(n^{0.791k})$ [Nešetřil, Poljak]
  - Under FPT $\neq$ W[1], no algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$.
- We now prove the following:

ETH implies FPT $\neq$ W[1]

- Consider the CLIQUE problem:
  - Given graph G and $k \in \mathbb{N}$, is there a clique on k vertices in G.
  - **Trivial**: $\mathcal{O}^\star(n^k)$.
  - **Fastest known**: $\mathcal{O}^\star(n^{\omega k/3}) = \mathcal{O}^\star(n^{0.791k})$ [Nešetřil, Poljak]
  - Under FPT $\neq$ W[1], no algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$.
- We now prove the following:

ETH hardness of CLIQUE [Chen, Huang, Kanj, Xia]

Unless ETH fails, CLIQUE cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

ETH implies FPT $\neq$ W[1]

- Consider the CLIQUE problem:
  - Given graph G and $k \in \mathbb{N}$, is there a clique on k vertices in G.
  - **Trivial**: $\mathcal{O}^\star(n^k)$.
  - **Fastest known**: $\mathcal{O}^\star(n^{\omega k/3}) = \mathcal{O}^\star(n^{0.791k})$ [Nešetřil, Poljak]
  - Under FPT $\neq$ W[1], no algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$.
- We now prove the following:

ETH hardness of CLIQUE [Chen, Huang, Kanj, Xia]

Unless ETH fails, CLIQUE cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

- **Corollary**: ETH $\Rightarrow$ FPT $\neq$ W[1]

ETH implies FPT $\neq$ W[1]

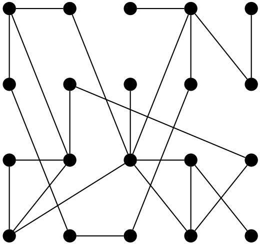
- Consider the CLIQUE problem:
  - Given graph G and $k \in \mathbb{N}$, is there a clique on k vertices in G.
  - **Trivial**: $\mathcal{O}^\star(n^k)$.
  - **Fastest known**: $\mathcal{O}^\star(n^{\omega k/3}) = \mathcal{O}^\star(n^{0.791k})$ [Nešetřil, Poljak]
  - Under FPT $\neq$ W[1], no algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$.
- We now prove the following:

ETH hardness of CLIQUE [Chen, Huang, Kanj, Xia]

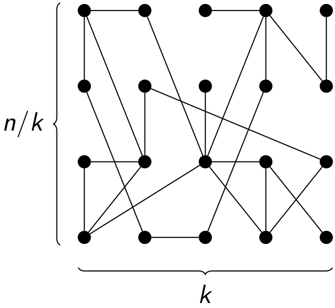
Unless ETH fails, CLIQUE cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

- **Corollary**: ETH $\Rightarrow$ FPT $\neq$ W[1]
- We start from an instance of 3-COLORING, for which we already have a $2^{o(n)}$ lower bound.

ETH hardness for CLIQUE

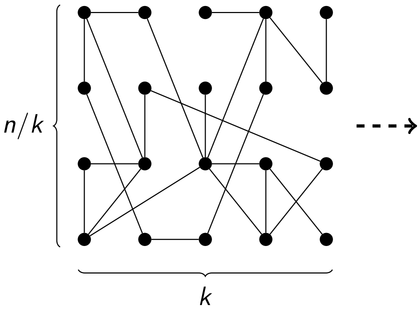


ETH hardness for CLIQUE

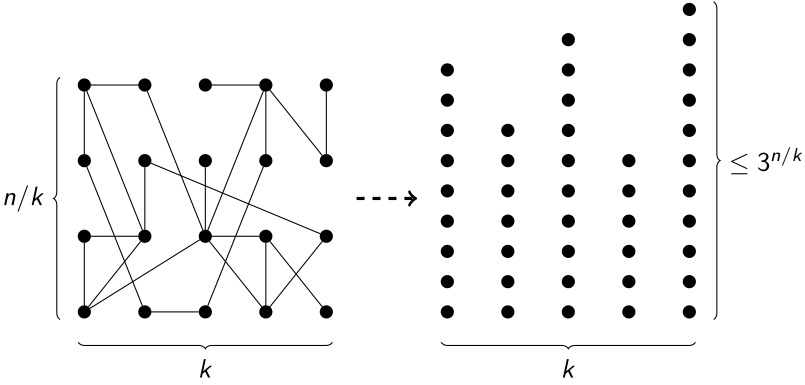


Partition vertices into k groups with n/k vertices each.

ETH hardness for CLIQUE

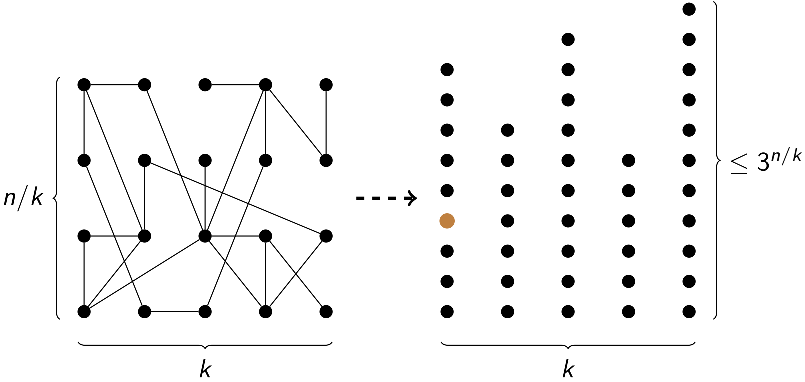


ETH hardness for CLIQUE



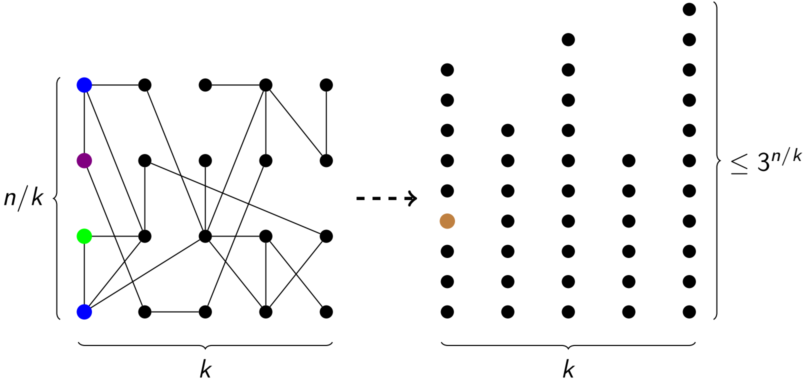
Create one vertex per each **consistent coloring** of each group.

ETH hardness for CLIQUE



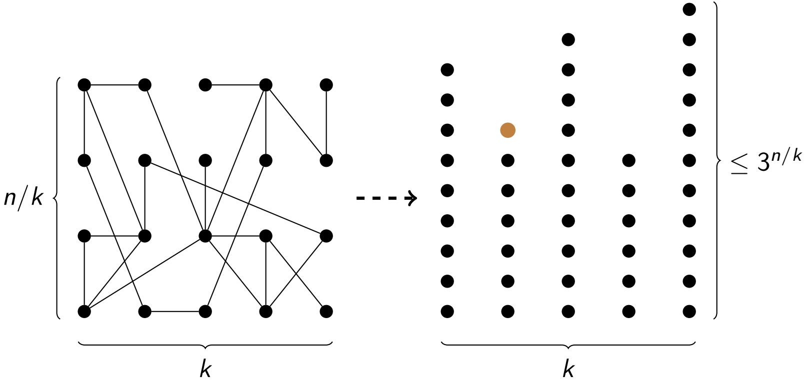
Create one vertex per each **consistent coloring** of each group.

ETH hardness for Clique



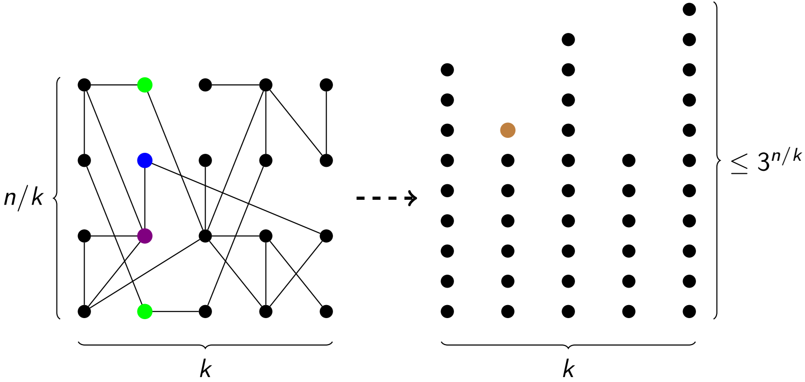
Create one vertex per each **consistent coloring** of each group.

ETH hardness for CLIQUE



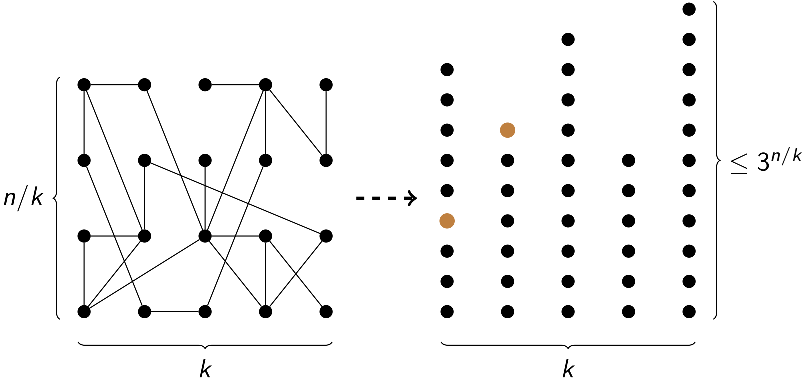
Create one vertex per each **consistent coloring** of each group.

ETH hardness for CLIQUE



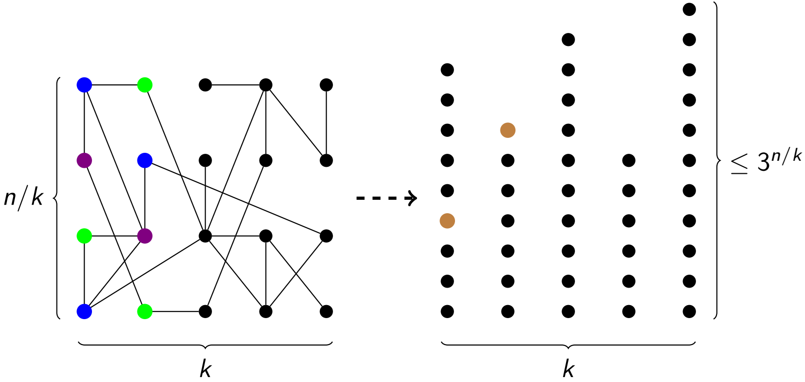
Create one vertex per each **consistent coloring** of each group.

ETH hardness for CLIQUE



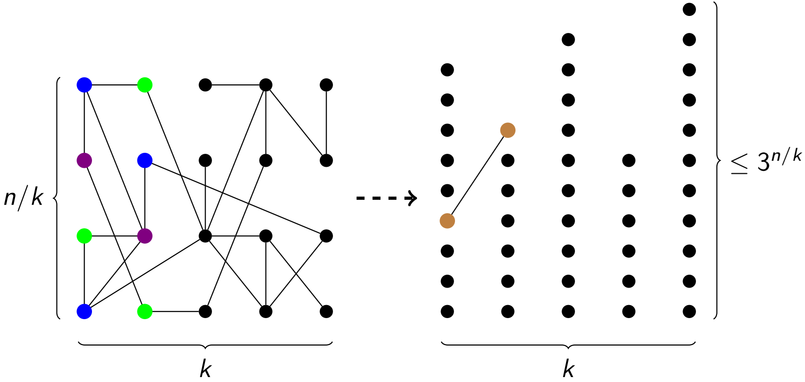
Connect two colorings if they are consistent.

ETH hardness for Clique



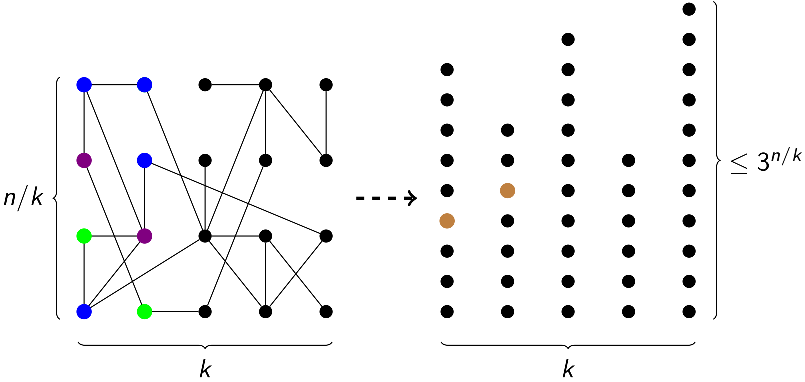
Connect two colorings if they are consistent.

ETH hardness for CLIQUE



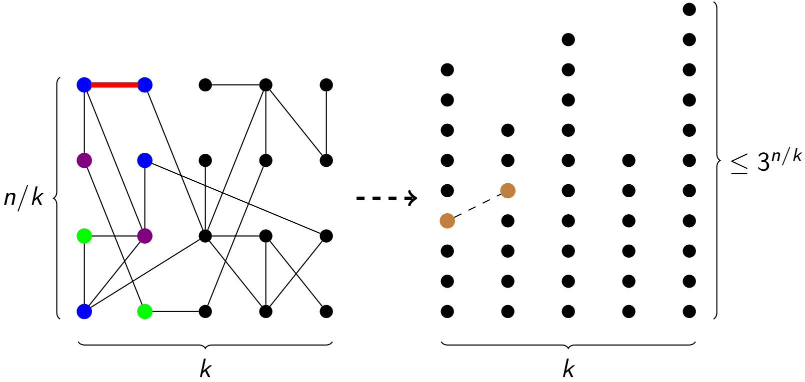
Connect two colorings if they are consistent.

ETH hardness for CLIQUE



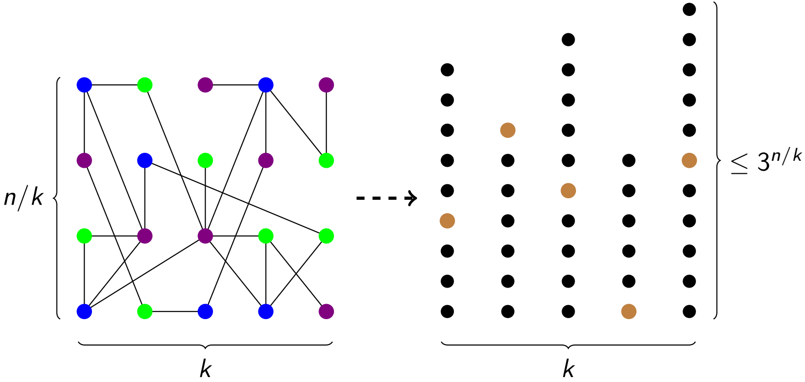
Inconsistent pairs of colorings remain nonadjacent.

ETH hardness for CLIQUE



Inconsistent pairs of colorings remain nonadjacent.

ETH hardness for CLIQUE



A 3-coloring on the left corresponds to a k-clique on the right.

Left graph admits 3-coloring iff right graph contains k-clique.

Hardness for CLIQUE

ETH hardness of CLIQUE

Unless ETH fails, CLIQUE cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

Proof continued:

- The output graph has a k-clique iff the input graph is 3-colorable.

Hardness for CLIQUE

ETH hardness of CLIQUE

Unless ETH fails, CLIQUE cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

Proof continued:

- The output graph has a k-clique iff the input graph is 3-colorable.
- We have introduced $N \leqslant k \cdot 3^{n/k}$ vertices.

Hardness for CLIQUE

ETH hardness of CLIQUE

Unless ETH fails, CLIQUE cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

Proof continued:

- The output graph has a k-clique iff the input graph is 3-colorable.
- We have introduced $N \leqslant k \cdot 3^{n/k}$ vertices.
- Let's try $k = \log n$, suppose CLIQUE can be solved in time $2^k \cdot N^{o(k)}$.

Hardness for CLIQUE

ETH hardness of CLIQUE

Unless ETH fails, CLIQUE cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

Proof continued:

- The output graph has a k-clique iff the input graph is 3-colorable.
- We have introduced $N \leqslant k \cdot 3^{n/k}$ vertices.
- Let's try $k = \log n$, suppose CLIQUE can be solved in time $2^k \cdot N^{o(k)}$.
- $2^k \cdot N^{o(k)} = n \cdot (\log n)^{o(\log n)} \cdot 3^{n \cdot o(\log n)/\log n} = 2^{o(n)}$.

Hardness for Clique

ETH hardness of Clique

Unless ETH fails, Clique cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

Proof continued:

- The output graph has a k-clique iff the input graph is 3-colorable.
- We have introduced $N \leqslant k \cdot 3^{n/k}$ vertices.
- Let's try $k = \log n$, suppose Clique can be solved in time $2^k \cdot N^{o(k)}$.
- $2^k \cdot N^{o(k)} = n \cdot (\log n)^{o(\log n)} \cdot 3^{n \cdot o(\log n)/\log n} = 2^{o(n)}$.
- Similarly $k = \log \log n$ implies no $2^{2^k} \cdot N^{o(k)}$ time algorithm.

Hardness for CLIQUE

ETH hardness of CLIQUE

Unless ETH fails, CLIQUE cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

Proof continued:

- The output graph has a k-clique iff the input graph is 3-colorable.
- We have introduced $N \leqslant k \cdot 3^{n/k}$ vertices.
- Let's try $k = \log n$, suppose CLIQUE can be solved in time $2^k \cdot N^{o(k)}$.
- $2^k \cdot N^{o(k)} = n \cdot (\log n)^{o(\log n)} \cdot 3^{n \cdot o(\log n)/\log n} = 2^{o(n)}$.
- Similarly $k = \log \log n$ implies no $2^{2^k} \cdot N^{o(k)}$ time algorithm.
- To exclude all computable $f(k)$, one needs roughly $k = f^{-1}(n)$ (technical difficulties omitted).

Hardness for CLIQUE

ETH hardness of CLIQUE

Unless ETH fails, CLIQUE cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

Proof continued:

- The output graph has a k-clique iff the input graph is 3-colorable.
- We have introduced $N \leqslant k \cdot 3^{n/k}$ vertices.
- Let's try $k = \log n$, suppose CLIQUE can be solved in time $2^k \cdot N^{o(k)}$.
- $2^k \cdot N^{o(k)} = n \cdot (\log n)^{o(\log n)} \cdot 3^{n \cdot o(\log n)/\log n} = 2^{o(n)}$.
- Similarly $k = \log\log n$ implies no $2^{2^k} \cdot N^{o(k)}$ time algorithm.
- To exclude all computable $f(k)$, one needs roughly $k = f^{-1}(n)$ (technical difficulties omitted).
- **Intuition**: We embed the 3^n solution space of 3-COLORING into the N^k solution space of CLIQUE.

Corollaries

- By reductions from CLIQUE, one can prove lower bounds on the running times of XP algorithms.

Corollaries

- By reductions from CLIQUE, one can prove lower bounds on the running times of XP algorithms.
- **Parameter blow-up**: parameter k in a CLIQUE instance is transformed into parameter $k' \leqslant g(k)$ for the output instance.

Corollaries

- By reductions from CLIQUE, one can prove lower bounds on the running times of XP algorithms.
- **Parameter blow-up**: parameter k in a CLIQUE instance is transformed into parameter $k' \leqslant g(k)$ for the output instance.
  - Linear blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(k')}$ lower bound

Corollaries

- By reductions from $\textsc{Clique}$, one can prove lower bounds on the running times of XP algorithms.
- **Parameter blow-up**: parameter k in a $\textsc{Clique}$ instance is transformed into parameter $k' \leqslant g(k)$ for the output instance.
  - Linear blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(k')}$ lower bound
  - Quadratic blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(\sqrt{k'})}$ lower bound

Corollaries

- By reductions from CLIQUE, one can prove lower bounds on the running times of XP algorithms.
- **Parameter blow-up**: parameter k in a CLIQUE instance is transformed into parameter $k' \leqslant g(k)$ for the output instance.
  - Linear blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(k')}$ lower bound
  - Quadratic blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(\sqrt{k'})}$ lower bound

PLANAR SCATTERED SET

I: An edge-weighted planar graph G, $d \in \mathbb{R}^+$, $k \in \mathbb{N}$
Q: Are there are k vertices pairwise at distance $\geqslant d$ from each other?

Corollaries

- By reductions from CLIQUE, one can prove lower bounds on the running times of XP algorithms.
- **Parameter blow-up**: parameter k in a CLIQUE instance is transformed into parameter $k' \leqslant g(k)$ for the output instance.
  - Linear blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(k')}$ lower bound
  - Quadratic blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(\sqrt{k'})}$ lower bound

PLANAR SCATTERED SET

I: An edge-weighted planar graph G, $d \in \mathbb{R}^+$, $k \in \mathbb{N}$
Q: Are there are k vertices pairwise at distance $\geqslant d$ from each other?

- There is an algorithm with running time $n^{\mathcal{O}(\sqrt{k})}$.

Corollaries

- By reductions from CLIQUE, one can prove lower bounds on the running times of XP algorithms.
- **Parameter blow-up**: parameter k in a CLIQUE instance is transformed into parameter $k' \leqslant g(k)$ for the output instance.
  - Linear blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(k')}$ lower bound
  - Quadratic blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(\sqrt{k'})}$ lower bound

PLANAR SCATTERED SET

I: An edge-weighted planar graph G, $d \in \mathbb{R}^+$, $k \in \mathbb{N}$
Q: Are there are k vertices pairwise at distance $\geqslant d$ from each other?

- There is an algorithm with running time $n^{\mathcal{O}(\sqrt{k})}$.
- Under ETH, there is no algorithm with running time $f(k) \cdot n^{o(\sqrt{k})}$.

Corollaries

- By reductions from CLIQUE, one can prove lower bounds on the running times of XP algorithms.
- **Parameter blow-up**: parameter k in a CLIQUE instance is transformed into parameter $k' \leqslant g(k)$ for the output instance.
  - Linear blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(k')}$ lower bound
  - Quadratic blow-up $\rightsquigarrow$ $f(k') \cdot n^{o(\sqrt{k'})}$ lower bound

PLANAR SCATTERED SET

I: An edge-weighted planar graph G, $d \in \mathbb{R}^+$, $k \in \mathbb{N}$

Q: Are there are k vertices pairwise at distance $\geqslant d$ from each other?

- There is an algorithm with running time $n^{\mathcal{O}(\sqrt{k})}$.
- Under ETH, there is no algorithm with running time $f(k) \cdot n^{o(\sqrt{k})}$.
  - The reduction transforms an instance of CLIQUE with parameter k into an instance of PLANAR SCATTERED SET with parameter $\mathcal{O}(k^2)$.

ETH and parameterized complexity

- **Goal**: Tight asymptotic bounds on $f(k)$ in the $f(k) \cdot n^{\mathcal{O}(1)}$ running times for FPT problems.

ETH and parameterized complexity

- **Goal**: Tight asymptotic bounds on $f(k)$ in the $f(k) \cdot n^{\mathcal{O}(1)}$ running times for FPT problems.
  - $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $2^{o(N)}$ algorithm. $\lightning$

ETH and parameterized complexity

- **Goal**: Tight asymptotic bounds on $f(k)$ in the $f(k) \cdot n^{\mathcal{O}(1)}$ running times for FPT problems.
  - $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $2^{o(N)}$ algorithm. $\lightning$
  - $\mathcal{O}^\star(2^{o(\sqrt{k})})$ algorithm for PLVC would be also a $2^{o(\sqrt{N})}$ algorithm. $\lightning$

ETH and parameterized complexity

- **Goal**: Tight asymptotic bounds on $f(k)$ in the $f(k) \cdot n^{\mathcal{O}(1)}$ running times for FPT problems.
  - $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $2^{o(N)}$ algorithm.
  - $\mathcal{O}^\star(2^{o(\sqrt{k})})$ algorithm for PLVC would be also a $2^{o(\sqrt{N})}$ algorithm.
  - These are tight, as there are $\mathcal{O}^\star(2^k)$ and $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ time algorithms, respectively.

ETH and parameterized complexity

- **Goal**: Tight asymptotic bounds on $f(k)$ in the $f(k) \cdot n^{\mathcal{O}(1)}$ running times for FPT problems.
  - $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $2^{o(N)}$ algorithm. $\lightning$
  - $\mathcal{O}^\star(2^{o(\sqrt{k})})$ algorithm for PLVC would be also a $2^{o(\sqrt{N})}$ algorithm. $\lightning$
  - These are tight, as there are $\mathcal{O}^\star(2^k)$ and $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ time algorithms, respectively.
- **Note**: For the reduction from $3\mathrm{SAT}$ to, say, VC, it would be sufficient to have the **parameter** bounded linearly in $n + m$, while the output instance size may be polynomial.

ETH and parameterized complexity

- **Goal**: Tight asymptotic bounds on $f(k)$ in the $f(k) \cdot n^{\mathcal{O}(1)}$ running times for FPT problems.
  - $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $2^{o(N)}$ algorithm. ⚡
  - $\mathcal{O}^\star(2^{o(\sqrt{k})})$ algorithm for PLVC would be also a $2^{o(\sqrt{N})}$ algorithm. ⚡
  - These are tight, as there are $\mathcal{O}^\star(2^k)$ and $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ time algorithms, respectively.
- **Note**: For the reduction from $3\mathrm{SAT}$ to, say, VC, it would be sufficient to have the **parameter** bounded linearly in $n + m$, while the output instance size may be polynomial.
  - If $N = (n+m)^{\mathcal{O}(1)}$ and $k = \mathcal{O}(n+m)$, then $2^{o(k)} \cdot N^{\mathcal{O}(1)} = 2^{o(n+m)}$.

ETH and parameterized complexity

- **Goal**: Tight asymptotic bounds on $f(k)$ in the $f(k) \cdot n^{\mathcal{O}(1)}$ running times for FPT problems.
  - $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $2^{o(N)}$ algorithm. $\lightning$
  - $\mathcal{O}^\star(2^{o(\sqrt{k})})$ algorithm for $\mathrm{P}{\small\mathrm{L}}\mathrm{VC}$ would be also a $2^{o(\sqrt{N})}$ algorithm. $\lightning$
  - These are tight, as there are $\mathcal{O}^\star(2^k)$ and $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ time algorithms, respectively.
- **Note**: For the reduction from $3\mathrm{SAT}$ to, say, VC, it would be sufficient to have the **parameter** bounded linearly in $n+m$, while the output instance size may be polynomial.
  - If $N = (n+m)^{\mathcal{O}(1)}$ and $k = \mathcal{O}(n+m)$, then $2^{o(k)} \cdot N^{\mathcal{O}(1)} = 2^{o(n+m)}$.

Lower bounds for parameterized problems under ETH

Suppose L admits a polynomial-time reduction from $3\mathrm{SAT}$ with output parameter $k \leqslant f(n+m)$.

Then L does not admit an $\mathcal{O}^\star(2^{o(f^{-1}(k))})$ time algorithm unless ETH fails.

ETH and parameterized complexity

- For many parameterized problems the situation is simple:

ETH and parameterized complexity

- For many parameterized problems the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ exists.

ETH and parameterized complexity

- For many parameterized problems the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ exists.
  - The existence of a **subexponential parameterized algorithm**, with running time $\mathcal{O}^\star(2^{o(k)})$, can be excluded under ETH using known NP-hardness reductions.

ETH and parameterized complexity

- For many parameterized problems the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ exists.
  - The existence of a **subexponential parameterized algorithm**, with running time $\mathcal{O}^\star(2^{o(k)})$, can be excluded under ETH using known NP-hardness reductions.
  - For planar problems, runtime $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ is often tight under ETH.

ETH and parameterized complexity

- For many parameterized problems the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ exists.
  - The existence of a **subexponential parameterized algorithm**, with running time $\mathcal{O}^\star(2^{o(k)})$, can be excluded under ETH using known NP-hardness reductions.
  - For planar problems, runtime $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ is often tight under ETH.
- Tight bounds for more exotic running times, e.g. $\mathcal{O}^\star(2^{2^{o(k)}})$ lower bound for Edge Clique Cover.

ETH and parameterized complexity

- For many parameterized problems the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ exists.
  - The existence of a **subexponential parameterized algorithm**, with running time $\mathcal{O}^\star(2^{o(k)})$, can be excluded under ETH using known NP-hardness reductions.
  - For planar problems, runtime $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ is often tight under ETH.
- Tight bounds for more exotic running times, e.g. $\mathcal{O}^\star(2^{2^{o(k)}})$ lower bound for EDGE CLIQUE COVER.
- Let's examine *slightly super-exponential* FPT algorithms.

$$\text{Slightly super-exponential} = \mathcal{O}^\star(2^{\mathcal{O}(k \log k)}) = \mathcal{O}^\star(k^{\mathcal{O}(k)})$$

ETH and parameterized complexity

- For many parameterized problems the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ exists.
  - The existence of a **subexponential parameterized algorithm**, with running time $\mathcal{O}^\star(2^{o(k)})$, can be excluded under ETH using known NP-hardness reductions.
  - For planar problems, runtime $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ is often tight under ETH.
- Tight bounds for more exotic running times, e.g. $\mathcal{O}^\star(2^{2^{o(k)}})$ lower bound for Edge Clique Cover.
- Let's examine *slightly super-exponential* FPT algorithms.

$$\text{Slightly super-exponential} = \mathcal{O}^\star(2^{\mathcal{O}(k \log k)}) = \mathcal{O}^\star(k^{\mathcal{O}(k)})$$

- Appears naturally:

ETH and parameterized complexity

- For many parameterized problems the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ exists.
  - The existence of a **subexponential parameterized algorithm**, with running time $\mathcal{O}^\star(2^{o(k)})$, can be excluded under ETH using known NP-hardness reductions.
  - For planar problems, runtime $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ is often tight under ETH.
- Tight bounds for more exotic running times, e.g. $\mathcal{O}^\star(2^{2^{o(k)}})$ lower bound for Edge Clique Cover.
- Let's examine *slightly super-exponential* FPT algorithms.

$$\text{Slightly super-exponential} = \mathcal{O}^\star(2^{\mathcal{O}(k \log k)}) = \mathcal{O}^\star(k^{\mathcal{O}(k)})$$

- Appears naturally:
  (a) Iterate through $k!$ possibilities.

ETH and parameterized complexity

- For many parameterized problems the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ exists.
  - The existence of a **subexponential parameterized algorithm**, with running time $\mathcal{O}^\star(2^{o(k)})$, can be excluded under ETH using known NP-hardness reductions.
  - For planar problems, runtime $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ is often tight under ETH.
- Tight bounds for more exotic running times, e.g. $\mathcal{O}^\star(2^{2^{o(k)}})$ lower bound for Edge Clique Cover.
- Let's examine *slightly super-exponential* FPT algorithms.

$$\text{Slightly super-exponential} = \mathcal{O}^\star(2^{\mathcal{O}(k \log k)}) = \mathcal{O}^\star(k^{\mathcal{O}(k)})$$

- Appears naturally:
  (a) Iterate through $k!$ possibilities.
  (b) A branching procedure branches $\mathcal{O}(k)$ times, each time choosing one of $\mathrm{poly}(k)$ options.

ETH and parameterized complexity

- For many parameterized problems the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ exists.
  - The existence of a **subexponential parameterized algorithm**, with running time $\mathcal{O}^\star(2^{o(k)})$, can be excluded under ETH using known NP-hardness reductions.
  - For planar problems, runtime $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ is often tight under ETH.
- Tight bounds for more exotic running times, e.g. $\mathcal{O}^\star(2^{2^{o(k)}})$ lower bound for Edge Clique Cover.
- Let's examine *slightly super-exponential* FPT algorithms.

$$\text{Slightly super-exponential} = \mathcal{O}^\star(2^{\mathcal{O}(k \log k)}) = \mathcal{O}^\star(k^{\mathcal{O}(k)})$$

- Appears naturally:
  (a) Iterate through $k!$ possibilities.
  (b) A branching procedure branches $\mathcal{O}(k)$ times, each time choosing one of $\mathrm{poly}(k)$ options.
  (c) A treewidth DP has partitions of the bag as the states.

ETH and parameterized complexity

- For many parameterized problems the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ exists.
  - The existence of a **subexponential parameterized algorithm**, with running time $\mathcal{O}^\star(2^{o(k)})$, can be excluded under ETH using known NP-hardness reductions.
  - For planar problems, runtime $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ is often tight under ETH.
- Tight bounds for more exotic running times, e.g. $\mathcal{O}^\star(2^{2^{o(k)}})$ lower bound for Edge Clique Cover.
- Let's examine *slightly super-exponential* FPT algorithms.

$$\text{Slightly super-exponential} = \mathcal{O}^\star(2^{\mathcal{O}(k \log k)}) = \mathcal{O}^\star(k^{\mathcal{O}(k)})$$

- Appears naturally:
  (a) Iterate through $k!$ possibilities.
  (b) A branching procedure branches $\mathcal{O}(k)$ times, each time choosing one of $\mathrm{poly}(k)$ options.
  (c) A treewidth DP has partitions of the bag as the states.
- We focus on (b), since this is the most typical behavior in parameterized algorithms.

$k \times k$-Clique

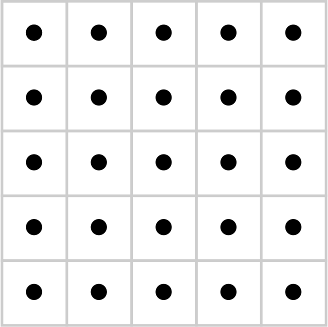
$k \times k$-Clique

I: A graph H on vertex set $\{1, \ldots, k\} \times \{1, \ldots, k\}$

Q: Is there a k-clique in H that contains exactly one vertex from each row?

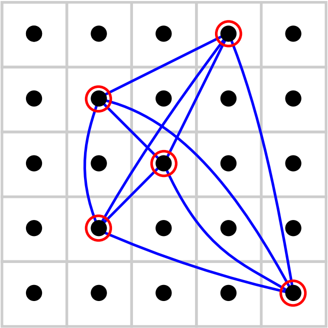
$k \times k$-CLIQUE

I: A graph H on vertex set $\{1, \ldots, k\} \times \{1, \ldots, k\}$
Q: Is there a k-clique in H that contains exactly one vertex from each row?



$k \times k$-Clique

I: A graph H on vertex set $\{1,\ldots,k\} \times \{1,\ldots,k\}$
Q: Is there a k-clique in H that contains exactly one vertex from each row?



- **Note**: the input to the problem is of size $\mathcal{O}(k^4)$.

Lower bound for $k \times k$-Clique

- **Note**: the input to the problem is of size $\mathcal{O}(k^4)$.
- Trivial $\mathcal{O}^\star(k^k)$ algorithm: verify all the choices.

Lower bound for $k \times k$-Clique

- **Note**: the input to the problem is of size $\mathcal{O}(k^4)$.
- Trivial $\mathcal{O}^\star(k^k)$ algorithm: verify all the choices.
- **Intuition**: extracts the idea of having k independent 1-in-k choices, similarly as for Clique.

Lower bound for $k \times k$-Clique

- **Note**: the input to the problem is of size $\mathcal{O}(k^4)$.
- Trivial $\mathcal{O}^\star(k^k)$ algorithm: verify all the choices.
- **Intuition**: extracts the idea of having k independent 1-in-k choices, similarly as for Clique.

Hardness for $k \times k$-Clique

There is a reduction from 3-Coloring to $k \times k$-Clique that for an input instance with n vertices, outputs an instance with parameter $k = \mathcal{O}(n/\log n)$.

Lower bound for $k \times k$-Clique

- **Note**: the input to the problem is of size $\mathcal{O}(k^4)$.
- Trivial $\mathcal{O}^\star(k^k)$ algorithm: verify all the choices.
- **Intuition**: extracts the idea of having k independent 1-in-k choices, similarly as for Clique.

Hardness for $k \times k$-Clique

There is a reduction from 3-Coloring to $k \times k$-Clique that for an input instance with n vertices, outputs an instance with parameter $k = \mathcal{O}(n/\log n)$.

- **Note**: the input to the problem is of size $\mathcal{O}(k^4)$.
- Trivial $\mathcal{O}^\star(k^k)$ algorithm: verify all the choices.
- **Intuition**: extracts the idea of having k independent 1-in-k choices, similarly as for Clique.

Hardness for $k \times k$-Clique

There is a reduction from 3-Coloring to $k \times k$-Clique that for an input instance with n vertices, outputs an instance with parameter $k = \mathcal{O}(n/\log n)$.

- **Proof sketch**: Apply the reduction for Clique for $k = n/\log n$.

- **Note**: the input to the problem is of size $\mathcal{O}(k^4)$.
- Trivial $\mathcal{O}^\star(k^k)$ algorithm: verify all the choices.
- **Intuition**: extracts the idea of having k independent 1-in-k choices, similarly as for Clique.

Hardness for $k \times k$-Clique

There is a reduction from 3-Coloring to $k \times k$-Clique that for an input instance with n vertices, outputs an instance with parameter $k = \mathcal{O}(n/\log n)$.

- **Proof sketch**: Apply the reduction for Clique for $k = n/\log n$.

Corollary

Unless ETH fails, there is no algorithm for $k \times k$-Clique with running time $\mathcal{O}^\star(2^{o(k \log k)})$.

Tight bounds for slightly super-exponential time

- Further reductions give more tightness results for slightly super-exponential time.

Tight bounds for slightly super-exponential time

- Further reductions give more tightness results for slightly super-exponential time.

Closest String

I: Strings $x_1, x_2, \ldots, x_n \in \Sigma^L$, $d \in \mathbb{N}$
Q: Is there $y \in \Sigma^L$ that is at Hamming distance $\leqslant d$ for each x_i?

Tight bounds for slightly super-exponential time

- Further reductions give more tightness results for slightly super-exponential time.

Closest String

I: Strings $x_1, x_2, \ldots, x_n \in \Sigma^L$, $d \in \mathbb{N}$
Q: Is there $y \in \Sigma^L$ that is at Hamming distance $\leqslant d$ for each x_i?

- **Algorithms** with running time $\mathcal{O}^\star(d^{\mathcal{O}(d)})$ and $\mathcal{O}^\star(|\Sigma|^{\mathcal{O}(d)})$.

Tight bounds for slightly super-exponential time

- Further reductions give more tightness results for slightly super-exponential time.

Closest String

I: Strings $x_1, x_2, \ldots, x_n \in \Sigma^L$, $d \in \mathbb{N}$
Q: Is there $y \in \Sigma^L$ that is at Hamming distance $\leqslant d$ for each x_i?

- **Algorithms** with running time $\mathcal{O}^\star(d^{\mathcal{O}(d)})$ and $\mathcal{O}^\star(|\Sigma|^{\mathcal{O}(d)})$.
- **Lower bounds** excluding running time $\mathcal{O}^\star(2^{o(d \log d)})$ or $\mathcal{O}^\star(2^{o(d \log |\Sigma|)})$ under ETH.

Tight bounds for slightly super-exponential time

- Further reductions give more tightness results for slightly super-exponential time.

Closest String

I: Strings $x_1, x_2, \ldots, x_n \in \Sigma^L$, $d \in \mathbb{N}$
Q: Is there $y \in \Sigma^L$ that is at Hamming distance $\leqslant d$ for each x_i?

- **Algorithms** with running time $\mathcal{O}^\star(d^{\mathcal{O}(d)})$ and $\mathcal{O}^\star(|\Sigma|^{\mathcal{O}(d)})$.
- **Lower bounds** excluding running time $\mathcal{O}^\star(2^{o(d \log d)})$ or $\mathcal{O}^\star(2^{o(d \log |\Sigma|)})$ under ETH.
- **Further**: treewidth dynamic programming, e.g. Cycle Packing:

Tight bounds for slightly super-exponential time

- Further reductions give more tightness results for slightly super-exponential time.

Closest String

I: Strings $x_1, x_2, \ldots, x_n \in \Sigma^L$, $d \in \mathbb{N}$

Q: Is there $y \in \Sigma^L$ that is at Hamming distance $\leqslant d$ for each x_i?

- **Algorithms** with running time $\mathcal{O}^\star(d^{\mathcal{O}(d)})$ and $\mathcal{O}^\star(|\Sigma|^{\mathcal{O}(d)})$.
- **Lower bounds** excluding running time $\mathcal{O}^\star(2^{o(d \log d)})$ or $\mathcal{O}^\star(2^{o(d \log |\Sigma|)})$ under ETH.
- **Further**: treewidth dynamic programming, e.g. Cycle Packing:
  - Pack the maximum number of vertex-disjoint cycles in a graph.

Tight bounds for slightly super-exponential time

- Further reductions give more tightness results for slightly super-exponential time.

Closest String

I: Strings $x_1, x_2, \ldots, x_n \in \Sigma^L$, $d \in \mathbb{N}$
Q: Is there $y \in \Sigma^L$ that is at Hamming distance $\leqslant d$ for each x_i?

- **Algorithms** with running time $\mathcal{O}^\star(d^{\mathcal{O}(d)})$ and $\mathcal{O}^\star(|\Sigma|^{\mathcal{O}(d)})$.
- **Lower bounds** excluding running time $\mathcal{O}^\star(2^{o(d \log d)})$ or $\mathcal{O}^\star(2^{o(d \log |\Sigma|)})$ under ETH.
- **Further**: treewidth dynamic programming, e.g. Cycle Packing:
  - Pack the maximum number of vertex-disjoint cycles in a graph.
  - $\mathcal{O}^\star(2^{\mathcal{O}(t \log t)})$-time algorithm by remembering a matching.

Tight bounds for slightly super-exponential time

- Further reductions give more tightness results for slightly super-exponential time.

Closest String

I: Strings $x_1, x_2, \ldots, x_n \in \Sigma^L$, $d \in \mathbb{N}$
Q: Is there $y \in \Sigma^L$ that is at Hamming distance $\leqslant d$ for each x_i?

- **Algorithms** with running time $\mathcal{O}^\star(d^{\mathcal{O}(d)})$ and $\mathcal{O}^\star(|\Sigma|^{\mathcal{O}(d)})$.
- **Lower bounds** excluding running time $\mathcal{O}^\star(2^{o(d \log d)})$ or $\mathcal{O}^\star(2^{o(d \log |\Sigma|)})$ under ETH.
- **Further**: treewidth dynamic programming, e.g. Cycle Packing:
  - Pack the maximum number of vertex-disjoint cycles in a graph.
  - $\mathcal{O}^\star(2^{\mathcal{O}(t \log t)})$-time algorithm by remembering a matching.
  - Running time $\mathcal{O}^\star(2^{o(t \log t)})$ would contradict ETH.

Tight bounds for slightly super-exponential time

- Further reductions give more tightness results for slightly super-exponential time.

Closest String

I: Strings $x_1, x_2, \ldots, x_n \in \Sigma^L$, $d \in \mathbb{N}$
Q: Is there $y \in \Sigma^L$ that is at Hamming distance $\leqslant d$ for each x_i?

- **Algorithms** with running time $\mathcal{O}^\star(d^{\mathcal{O}(d)})$ and $\mathcal{O}^\star(|\Sigma|^{\mathcal{O}(d)})$.
- **Lower bounds** excluding running time $\mathcal{O}^\star(2^{o(d \log d)})$ or $\mathcal{O}^\star(2^{o(d \log |\Sigma|)})$ under ETH.
- **Further**: treewidth dynamic programming, e.g. Cycle Packing:
  - Pack the maximum number of vertex-disjoint cycles in a graph.
  - $\mathcal{O}^\star(2^{\mathcal{O}(t \log t)})$-time algorithm by remembering a matching.
  - Running time $\mathcal{O}^\star(2^{o(t \log t)})$ would contradict ETH.
  - **Note**: For Hamiltonian Path you can get running time $\mathcal{O}^\star(4^t)$.

Strong ETH

- **Recall**: SETH $\Leftrightarrow$ if $\mathcal{O}(2^{\delta_q n})$ is optimum for $q\mathrm{SAT}$, then $\delta_q \to 0$.

Strong ETH

- **Recall**: SETH $\Leftrightarrow$ if $\mathcal{O}(2^{\delta_q n})$ is optimum for $q\mathrm{SAT}$, then $\delta_q \to 0$.
- ETH $\rightsquigarrow$ Asymptotics of the exponent

Strong ETH

- **Recall**: SETH $\Leftrightarrow$ if $\mathcal{O}(2^{\delta_q n})$ is optimum for $q\mathrm{SAT}$, then $\delta_q \to 0$.
- ETH $\rightsquigarrow$ Asymptotics of the exponent
- SETH $\rightsquigarrow$ Exact value of the coefficient in the exponent

Strong ETH

- **Recall**: SETH $\Leftrightarrow$ if $\mathcal{O}(2^{\delta_q n})$ is optimum for $q\mathrm{SAT}$, then $\delta_q \to 0$.
- ETH $\rightsquigarrow$ Asymptotics of the exponent
- SETH $\rightsquigarrow$ Exact value of the coefficient in the exponent
- Reductions for ETH hardness:
  we care only about the **asymptotics** of the parameter blow-up.

Strong ETH

- **Recall**: SETH $\Leftrightarrow$ if $\mathcal{O}(2^{\delta_q n})$ is optimum for $q\mathrm{SAT}$, then $\delta_q \to 0$.
- ETH $\rightsquigarrow$ Asymptotics of the exponent
- SETH $\rightsquigarrow$ Exact value of the coefficient in the exponent
- Reductions for ETH hardness:
  we care only about the **asymptotics** of the parameter blow-up.
- Reductions for SETH hardness:
  we need to know **precisely** how the parameter is transformed.

Strong ETH

- **Recall**: SETH $\Leftrightarrow$ if $\mathcal{O}(2^{\delta_q n})$ is optimum for $q\mathrm{SAT}$, then $\delta_q \to 0$.
- ETH $\rightsquigarrow$ Asymptotics of the exponent
- SETH $\rightsquigarrow$ Exact value of the coefficient in the exponent
- Reductions for ETH hardness:
  we care only about the **asymptotics** of the parameter blow-up.
- Reductions for SETH hardness:
  we need to know **precisely** how the parameter is transformed.
- Lower bounds under SETH are more delicate and much scarcer.

Strong ETH

- **Recall**: SETH $\Leftrightarrow$ if $\mathcal{O}(2^{\delta_q n})$ is optimum for $q\mathrm{SAT}$, then $\delta_q \to 0$.
- ETH $\rightsquigarrow$ Asymptotics of the exponent
- SETH $\rightsquigarrow$ Exact value of the coefficient in the exponent
- Reductions for ETH hardness:
  we care only about the **asymptotics** of the parameter blow-up.
- Reductions for SETH hardness:
  we need to know **precisely** how the parameter is transformed.
- Lower bounds under SETH are more delicate and much scarcer.
  - Also, technically more challenging.

SETH and treewidth DPs

- Many classic problems can be solved in time $\mathcal{O}^\star(c^t)$, where c is a constant and t is the width of a given tree decomposition.

SETH and treewidth DPs

- Many classic problems can be solved in time $\mathcal{O}^\star(c^t)$, where c is a constant and t is the width of a given tree decomposition.
  - VC and IS in $\mathcal{O}^\star(2^t)$, DS and OCT in $\mathcal{O}^\star(3^t)$.

SETH and treewidth DPs

- Many classic problems can be solved in time $\mathcal{O}^\star(c^t)$, where c is a constant and t is the width of a given tree decomposition.
  - VC and IS in $\mathcal{O}^\star(2^t)$, DS and OCT in $\mathcal{O}^\star(3^t)$.

Theorem

Assume that $\mathrm{CNF\text{-}SAT}$ cannot be solved in time $\mathcal{O}^\star(c^n)$ for any $c < 2$. Then for every $\varepsilon > 0$:

- Independent Set cannot be solved in time $\mathcal{O}^\star((2-\varepsilon)^p)$;
- Dominating Set cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$;
- Odd Cycle Traversal cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$;

where p is the width of a given path decomposition of the input graph.

SETH and treewidth DPs

- Many classic problems can be solved in time $\mathcal{O}^\star(c^t)$, where c is a constant and t is the width of a given tree decomposition.
  - VC and IS in $\mathcal{O}^\star(2^t)$, DS and OCT in $\mathcal{O}^\star(3^t)$.

Theorem

Assume that $\mathrm{CNF\text{-}SAT}$ cannot be solved in time $\mathcal{O}^\star(c^n)$ for any $c < 2$. Then for every $\varepsilon > 0$:

- Independent Set cannot be solved in time $\mathcal{O}^\star((2-\varepsilon)^p)$;
- Dominating Set cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$;
- Odd Cycle Traversal cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$;

where p is the width of a given path decomposition of the input graph.

Theorem [Cygan, Kratsch, Nederlof]

Hamiltonian Path can be solved in time $\mathcal{O}^\star((2+\sqrt{2})^p)$, but an algorithm with running time $\mathcal{O}^\star((2+\sqrt{2}-\varepsilon)^p)$ would contradict SETH.

- Many classic problems can be solved in time $\mathcal{O}^\star(c^t)$, where c is a constant and t is the width of a given tree decomposition.
  - VC and IS in $\mathcal{O}^\star(2^t)$, DS and OCT in $\mathcal{O}^\star(3^t)$.

Theorem

Assume that $\mathrm{CNF\text{-}SAT}$ cannot be solved in time $\mathcal{O}^\star(c^n)$ for any $c < 2$. Then for every $\varepsilon > 0$:

- Independent Set cannot be solved in time $\mathcal{O}^\star((2-\varepsilon)^p)$;
- Dominating Set cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$;
- Odd Cycle Traversal cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$;

where p is the width of a given path decomposition of the input graph.

Theorem [Cygan, Kratsch, Nederlof]

Hamiltonian Path can be solved in time $\mathcal{O}^\star((2+\sqrt{2})^p)$, but an algorithm with running time $\mathcal{O}^\star((2+\sqrt{2}-\varepsilon)^p)$ would contradict SETH.

SETH and treewidth DPs

- Many classic problems can be solved in time $\mathcal{O}^\star(c^t)$, where c is a constant and t is the width of a given tree decomposition.
  - VC and IS in $\mathcal{O}^\star(2^t)$, DS and OCT in $\mathcal{O}^\star(3^t)$.

Theorem

Assume that $\mathrm{CNF\text{-}SAT}$ cannot be solved in time $\mathcal{O}^\star(c^n)$ for any $c < 2$. Then for every $\varepsilon > 0$:

- Independent Set cannot be solved in time $\mathcal{O}^\star((2-\varepsilon)^p)$;
- Dominating Set cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$;
- Odd Cycle Traversal cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$;

where p is the width of a given path decomposition of the input graph.

Theorem [Cygan, Kratsch, Nederlof]

Hamiltonian Path can be solved in time $\mathcal{O}^\star((2+\sqrt{2})^p)$, but an algorithm with running time $\mathcal{O}^\star((2+\sqrt{2}-\varepsilon)^p)$ would contradict SETH.

- **Open**: The algorithmic result **does not** work for treewidth.

SETH and covering problems

Set Cover

I: Universe U, set family $\mathcal{F} \subseteq 2^U$, integer k
Q: Is there a subfamily $\mathcal{G} \subseteq \mathcal{F}$ with $|\mathcal{G}| \leqslant k$ s.t. $\bigcup \mathcal{G} = U$?

SETH and covering problems

Set Cover

I: Universe U, set family $\mathcal{F} \subseteq 2^U$, integer k
Q: Is there a subfamily $\mathcal{G} \subseteq \mathcal{F}$ with $|\mathcal{G}| \leqslant k$ s.t. $\bigcup \mathcal{G} = U$?

- Denote $n = |U|$ and $m = |\mathcal{F}|$.

SETH and covering problems

Set Cover

I: Universe U, set family $\mathcal{F} \subseteq 2^U$, integer k
Q: Is there a subfamily $\mathcal{G} \subseteq \mathcal{F}$ with $|\mathcal{G}| \leqslant k$ s.t. $\bigcup \mathcal{G} = U$?

- Denote $n = |U|$ and $m = |\mathcal{F}|$.
- **Brute-force**: time $\mathcal{O}^\star(2^m)$. **DP on subsets**: time $\mathcal{O}^\star(2^n)$.

SETH and covering problems

Set Cover

I: Universe U, set family $\mathcal{F} \subseteq 2^U$, integer k
Q: Is there a subfamily $\mathcal{G} \subseteq \mathcal{F}$ with $|\mathcal{G}| \leqslant k$ s.t. $\bigcup \mathcal{G} = U$?

- Denote $n = |U|$ and $m = |\mathcal{F}|$.
- **Brute-force**: time $\mathcal{O}^\star(2^m)$. **DP on subsets**: time $\mathcal{O}^\star(2^n)$.
- Under SETH, no $\mathcal{O}^\star(\alpha^m)$ algorithm for Set Cover for any $\alpha < 2$.

SETH and covering problems

Set Cover

I: Universe U, set family $\mathcal{F} \subseteq 2^U$, integer k

Q: Is there a subfamily $\mathcal{G} \subseteq \mathcal{F}$ with $|\mathcal{G}| \leqslant k$ s.t. $\bigcup \mathcal{G} = U$?

- Denote $n = |U|$ and $m = |\mathcal{F}|$.
- **Brute-force**: time $\mathcal{O}^\star(2^m)$. **DP on subsets**: time $\mathcal{O}^\star(2^n)$.
- Under SETH, no $\mathcal{O}^\star(\alpha^m)$ algorithm for Set Cover for any $\alpha < 2$.
- Breaking 2 in the base for many covering problems is **equivalent** to breaking it for Set Cover$/n$.

SETH and covering problems

Set Cover

I: Universe U, set family $\mathcal{F} \subseteq 2^U$, integer k
Q: Is there a subfamily $\mathcal{G} \subseteq \mathcal{F}$ with $|\mathcal{G}| \leqslant k$ s.t. $\bigcup \mathcal{G} = U$?

- Denote $n = |U|$ and $m = |\mathcal{F}|$.
- **Brute-force**: time $\mathcal{O}^\star(2^m)$. **DP on subsets**: time $\mathcal{O}^\star(2^n)$.
- Under SETH, no $\mathcal{O}^\star(\alpha^m)$ algorithm for Set Cover for any $\alpha < 2$.
- Breaking 2 in the base for many covering problems is **equivalent** to breaking it for Set Cover$/n$.
  - Steiner Tree, Conn Vertex Cover, Set Partitioning

SETH and covering problems

SET COVER

I: Universe U, set family $\mathcal{F} \subseteq 2^U$, integer k

Q: Is there a subfamily $\mathcal{G} \subseteq \mathcal{F}$ with $|\mathcal{G}| \leqslant k$ s.t. $\bigcup \mathcal{G} = U$?

- Denote $n = |U|$ and $m = |\mathcal{F}|$.
- **Brute-force**: time $\mathcal{O}^\star(2^m)$. **DP on subsets**: time $\mathcal{O}^\star(2^n)$.
- Under SETH, no $\mathcal{O}^\star(\alpha^m)$ algorithm for SET COVER for any $\alpha < 2$.
- Breaking 2 in the base for many covering problems is **equivalent** to breaking it for SET COVER/n.
  - STEINER TREE, CONN VERTEX COVER, SET PARTITIONING
- No known reductions between SET COVER/n and SET COVER/m.

SETH and covering problems

Set Cover

I: Universe U, set family $\mathcal{F} \subseteq 2^U$, integer k
Q: Is there a subfamily $\mathcal{G} \subseteq \mathcal{F}$ with $|\mathcal{G}| \leqslant k$ s.t. $\bigcup \mathcal{G} = U$?

- Denote $n = |U|$ and $m = |\mathcal{F}|$.
- **Brute-force**: time $\mathcal{O}^\star(2^m)$. **DP on subsets**: time $\mathcal{O}^\star(2^n)$.
- Under SETH, no $\mathcal{O}^\star(\alpha^m)$ algorithm for Set Cover for any $\alpha < 2$.
- Breaking 2 in the base for many covering problems is **equivalent** to breaking it for Set Cover/n.
  - Steiner Tree, Conn Vertex Cover, Set Partitioning
- No known reductions between Set Cover/n and Set Cover/m.

Set Cover Conjecture

Let λ_q be the infimum of the set of constants c such that q-Set Cover can be solved in time $\mathcal{O}^\star(2^{cn})$, where n is the size of the universe. Then $\lim_{q\to\infty} \lambda_q = 1$. In particular, there is no algorithm for the general Set Cover problem that runs in $\mathcal{O}^\star(\alpha^n)$ for any $\alpha < 2$.

- SETH can be also used to give lower bounds for polynomial-time solvable problems.

Hardness in **P**

- SETH can be also used to give lower bounds for polynomial-time solvable problems.

Orthogonal Vectors

I: Vectors $v_1, \ldots, v_N \in \{0,1\}^d$
Q: Are there i, j such that $\langle v_i, v_j \rangle = 0$?

Hardness in **P**

- SETH can be also used to give lower bounds for polynomial-time solvable problems.

Orthogonal Vectors

I: Vectors $v_1, \ldots, v_N \in \{0,1\}^d$
Q: Are there i, j such that $\langle v_i, v_j \rangle = 0$?

Theorem

Unless SETH fails, for every $\delta > 0$ there is $c \in \mathbb{N}$, such that no algorithm for OV with dimension $d = c \log N$ can achieve running time $\mathcal{O}(N^{2-\delta})$.

Hardness in P

- SETH can be also used to give lower bounds for polynomial-time solvable problems.

Orthogonal Vectors

I: Vectors $v_1, \ldots, v_N \in \{0,1\}^d$
Q: Are there i, j such that $\langle v_i, v_j \rangle = 0$?

Theorem

Unless SETH fails, for every $\delta > 0$ there is $c \in \mathbb{N}$, such that no algorithm for OV with dimension $d = c \log N$ can achieve running time $\mathcal{O}(N^{2-\delta})$.

- **Note**: Further hardness results obtained via reductions.

Hardness in P

- SETH can be also used to give lower bounds for polynomial-time solvable problems.

Orthogonal Vectors

I: Vectors $v_1, \ldots, v_N \in \{0,1\}^d$
Q: Are there i, j such that $\langle v_i, v_j \rangle = 0$?

Theorem

Unless SETH fails, for every $\delta > 0$ there is $c \in \mathbb{N}$, such that no algorithm for OV with dimension $d = c \log N$ can achieve running time $\mathcal{O}(N^{2-\delta})$.

- **Note**: Further hardness results obtained via reductions.
- **Example**: Longest Common Subsequence in strongly subquadratic time.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.
- Partition variables into V_1 and V_2, each of size $n/2$.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.
- Partition variables into V_1 and V_2, each of size $n/2$.
- For every assignment $\eta\colon V_1 \to \{\mathbf{T}, \mathbf{F}\}$, create a vector $u_\eta \in \{0,1\}^m$:

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.
- Partition variables into V_1 and V_2, each of size $n/2$.
- For every assignment $\eta\colon V_1 \to \{\mathbf{T}, \mathbf{F}\}$, create a vector $u_\eta \in \{0,1\}^m$:
  - At position corresponding to clause c, put 1 iff η **does not** satisfy c.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.
- Partition variables into V_1 and V_2, each of size $n/2$.
- For every assignment $\eta \colon V_1 \to \{\mathbf{T}, \mathbf{F}\}$, create a vector $u_\eta \in \{0,1\}^m$:
  - At position corresponding to clause c, put 1 iff η **does not** satisfy c.
  - **Note**: We create $2^{n/2}$ such vectors.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.
- Partition variables into V_1 and V_2, each of size $n/2$.
- For every assignment $\eta\colon V_1 \to \{\mathbf{T}, \mathbf{F}\}$, create a vector $u_\eta \in \{0,1\}^m$:
  - At position corresponding to clause c, put 1 iff η **does not** satisfy c.
  - **Note**: We create $2^{n/2}$ such vectors.
- Do the same for V_2, creating vectors v_η for $\eta\colon V_2 \to \{\mathbf{T}, \mathbf{F}\}$.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.
- Partition variables into V_1 and V_2, each of size $n/2$.
- For every assignment $\eta\colon V_1 \to \{\mathbf{T}, \mathbf{F}\}$, create a vector $u_\eta \in \{0,1\}^m$:
  - At position corresponding to clause c, put 1 iff η **does not** satisfy c.
  - **Note**: We create $2^{n/2}$ such vectors.
- Do the same for V_2, creating vectors v_η for $\eta\colon V_2 \to \{\mathbf{T}, \mathbf{F}\}$.
- **Note**: This takes time $\mathcal{O}^\star(2^{n/2})$.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.
- Partition variables into V_1 and V_2, each of size $n/2$.
- For every assignment $\eta\colon V_1 \to \{\mathbf{T}, \mathbf{F}\}$, create a vector $u_\eta \in \{0,1\}^m$:
  - At position corresponding to clause c, put 1 iff η **does not** satisfy c.
  - **Note**: We create $2^{n/2}$ such vectors.
- Do the same for V_2, creating vectors v_η for $\eta\colon V_2 \to \{\mathbf{T}, \mathbf{F}\}$.
- **Note**: This takes time $\mathcal{O}^\star(2^{n/2})$.
- Append 01 to vectors u_η and 10 to vectors v_η.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.
- Partition variables into V_1 and V_2, each of size $n/2$.
- For every assignment $\eta \colon V_1 \to \{\mathbf{T}, \mathbf{F}\}$, create a vector $u_\eta \in \{0,1\}^m$:
  - At position corresponding to clause c, put 1 iff η **does not** satisfy c.
  - **Note**: We create $2^{n/2}$ such vectors.
- Do the same for V_2, creating vectors v_η for $\eta \colon V_2 \to \{\mathbf{T}, \mathbf{F}\}$.
- **Note**: This takes time $\mathcal{O}^\star(2^{n/2})$.
- Append 01 to vectors u_η and 10 to vectors v_η.
- **Claim**: There is a pair of orthogonal vectors iff φ is satisfiable.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.
- Partition variables into V_1 and V_2, each of size $n/2$.
- For every assignment $\eta \colon V_1 \to \{\mathbf{T}, \mathbf{F}\}$, create a vector $u_\eta \in \{0,1\}^m$:
  - At position corresponding to clause c, put 1 iff η **does not** satisfy c.
  - **Note**: We create $2^{n/2}$ such vectors.
- Do the same for V_2, creating vectors v_η for $\eta \colon V_2 \to \{\mathbf{T}, \mathbf{F}\}$.
- **Note**: This takes time $\mathcal{O}^\star(2^{n/2})$.
- Append 01 to vectors u_η and 10 to vectors v_η.
- **Claim**: There is a pair of orthogonal vectors iff φ is satisfiable.
- An algorithm for OV with running time $\mathcal{O}(N^{1.98}) = \mathcal{O}(2^{0.99n})$ would contradict SETH.

Hardness for OV

- Take input formula φ of $q\mathrm{SAT}$, with n variables and m clauses.
  - **Goal**: decide satisfiability of φ in time $\mathcal{O}(2^{0.99n})$.
  - By Sparsification Lemma, we may assume $m \leqslant Cn$.
- Partition variables into V_1 and V_2, each of size $n/2$.
- For every assignment $\eta\colon V_1 \to \{\mathbf{T}, \mathbf{F}\}$, create a vector $u_\eta \in \{0,1\}^m$:
  - At position corresponding to clause c, put 1 iff η **does not** satisfy c.
  - **Note**: We create $2^{n/2}$ such vectors.
- Do the same for V_2, creating vectors v_η for $\eta\colon V_2 \to \{\mathbf{T}, \mathbf{F}\}$.
- **Note**: This takes time $\mathcal{O}^\star(2^{n/2})$.
- Append 01 to vectors u_η and 10 to vectors v_η.
- **Claim**: There is a pair of orthogonal vectors iff φ is satisfiable.
- An algorithm for OV with running time $\mathcal{O}(N^{1.98}) = \mathcal{O}(2^{0.99n})$ would contradict SETH.
- **Note**: dimension is $m \leqslant Cn \leqslant 2C \log N$.

Conclusions

- Using ETH we can estimate the asymptotics of the exponents.

Conclusions

- Using ETH we can estimate the asymptotics of the exponents.
  - Applies to exponential-time, FPT, XP algorithms, and many more.

Conclusions

- Using ETH we can estimate the asymptotics of the exponents.
  - Applies to exponential-time, FPT, XP algorithms, and many more.
- SETH may give a precise bound on the base of the exponent, but is less plausible and less applicable.

Conclusions

- Using ETH we can estimate the asymptotics of the exponents.
  - Applies to exponential-time, FPT, XP algorithms, and many more.
- SETH may give a precise bound on the base of the exponent, but is less plausible and less applicable.
  - Can be also used to give tight bounds within **P**.

Conclusions

- Using ETH we can estimate the asymptotics of the exponents.
  - Applies to exponential-time, FPT, XP algorithms, and many more.
- SETH may give a precise bound on the base of the exponent, but is less plausible and less applicable.
  - Can be also used to give tight bounds within **P**.
- **Fine-grained complexity theory**: obtaining matching lower and upper bounds on the complexity of problems of interest.

Conclusions

- Using ETH we can estimate the asymptotics of the exponents.
  - Applies to exponential-time, FPT, XP algorithms, and many more.
- SETH may give a precise bound on the base of the exponent, but is less plausible and less applicable.
  - Can be also used to give tight bounds within **P**.
- **Fine-grained complexity theory**: obtaining matching lower and upper bounds on the complexity of problems of interest.
  - **Motivation**: Really tight bounds give us a better understanding of the studied problem, or of its specific parameterization.

Conclusions

- Using ETH we can estimate the asymptotics of the exponents.
  - Applies to exponential-time, FPT, XP algorithms, and many more.
- SETH may give a precise bound on the base of the exponent, but is less plausible and less applicable.
  - Can be also used to give tight bounds within **P**.
- **Fine-grained complexity theory**: obtaining matching lower and upper bounds on the complexity of problems of interest.
  - **Motivation**: Really tight bounds give us a better understanding of the studied problem, or of its specific parameterization.
  - Stronger assumptions lead to tighter lower bounds.

Conclusions

- Using ETH we can estimate the asymptotics of the exponents.
  - Applies to exponential-time, FPT, XP algorithms, and many more.
- SETH may give a precise bound on the base of the exponent, but is less plausible and less applicable.
  - Can be also used to give tight bounds within **P**.
- **Fine-grained complexity theory**: obtaining matching lower and upper bounds on the complexity of problems of interest.
  - **Motivation**: Really tight bounds give us a better understanding of the studied problem, or of its specific parameterization.
  - Stronger assumptions lead to tighter lower bounds.
- **Thank you for your attention!**