Lower bounds based on ETH and SETH

Michał Pilipczuk

Institute of Informatics, University of Warsaw, Poland

Workshop on Exact Algorithms and Lower Bounds, IIT Delhi,
December 13th, 2014

Motivation

- Under P $\neq$ NP, Hamiltonian Cycle cannot be solved in polynomial time.

Motivation

- Under P $\neq$ NP, Hamiltonian Cycle cannot be solved in polynomial time.
- Best known algorithms:

Motivation

- Under P $\neq$ NP, Hamiltonian Cycle cannot be solved in polynomial time.
- Best known algorithms:
  - Brute-force $\mathcal{O}(n!)$.

Motivation

- Under P $\neq$ NP, Hamiltonian Cycle cannot be solved in polynomial time.
- Best known algorithms:
  - Brute-force $\mathcal{O}(n!)$.
  - For a long time $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.

Motivation

- Under P $\neq$ NP, Hamiltonian Cycle cannot be solved in polynomial time.
- Best known algorithms:
  - Brute-force $\mathcal{O}(n!)$.
  - For a long time $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
    - $\mathcal{O}^\star(f(n)) = f(n) \cdot \mathrm{poly}(|\mathrm{input}|)$.

Motivation

- Under P $\neq$ NP, Hamiltonian Cycle cannot be solved in polynomial time.
- Best known algorithms:
  - Brute-force $\mathcal{O}(n!)$.
  - For a long time $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
    - $\mathcal{O}^\star(f(n)) = f(n) \cdot \mathrm{poly}(|\mathrm{input}|)$.
  - Today: $\mathcal{O}^\star(1.657^n)$ of Björklund (2010).

Motivation

- Under P $\neq$ NP, Hamiltonian Cycle cannot be solved in polynomial time.
- Best known algorithms:
  - Brute-force $\mathcal{O}(n!)$.
  - For a long time $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
    - $\mathcal{O}^\star(f(n)) = f(n) \cdot \mathrm{poly}(|\mathrm{input}|)$.
  - Today: $\mathcal{O}^\star(1.657^n)$ of Björklund (2010).
- How much can you improve the constant 1.657?

Motivation

- Under P $\neq$ NP, Hamiltonian Cycle cannot be solved in polynomial time.
- Best known algorithms:
  - Brute-force $\mathcal{O}(n!)$.
  - For a long time $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
    - $\mathcal{O}^\star(f(n)) = f(n) \cdot \mathrm{poly}(|\mathrm{input}|)$.
  - Today: $\mathcal{O}^\star(1.657^n)$ of Björklund (2010).
- How much can you improve the constant 1.657?
- Can you do *significantly* better, e.g. $\mathcal{O}^\star(2^{\mathcal{O}(n/\log n)})$ or $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{n})})$?

Motivation

- Under P $\neq$ NP, Hamiltonian Cycle cannot be solved in polynomial time.
- Best known algorithms:
  - Brute-force $\mathcal{O}(n!)$.
  - For a long time $\mathcal{O}^\star(2^n)$ Held-Karp dynamic programming.
    - $\mathcal{O}^\star(f(n)) = f(n) \cdot \mathrm{poly}(|\mathrm{input}|)$.
  - Today: $\mathcal{O}^\star(1.657^n)$ of Björklund (2010).
- How much can you improve the constant 1.657?
- Can you do *significantly* better, e.g. $\mathcal{O}^\star(2^{\mathcal{O}(n/\log n)})$ or $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{n})})$?
- Assumption P $\neq$ NP seems too week to answer these questions.

Case study: 3-SAT

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

Case study: 3-SAT

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

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

Case study: 3-SAT

- 3-SAT: given formula φ in 3-CNF with n variables and m clauses, decide whether φ is satisfiable.
- 3-CNF: 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**: $\mathcal{O}^\star(2^n)$

Case study: 3-SAT

- 3-SAT: given formula φ in 3-CNF with n variables and m clauses, decide whether φ is satisfiable.
- 3-CNF: 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**:

Case study: 3-SAT

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

Case study: 3-SAT

- 3-SAT: given formula φ in 3-CNF with n variables and m clauses, decide whether φ is satisfiable.
- 3-CNF: 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 $\mathcal{O}^\star(7^{n/3}) = \mathcal{O}^\star(1.913^n)$.

Case study: 3-SAT

- 3-SAT: given formula φ in 3-CNF with n variables and m clauses, decide whether φ is satisfiable.
- 3-CNF: 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 $\mathcal{O}^\star(7^{n/3}) = \mathcal{O}^\star(1.913^n)$.
- Current record: $\mathcal{O}^\star(1.308^n)$ (Paturi, Pudlák, Saks, Zane; Hertli)

First try

- Can you do *significantly* better, i.e. have running time $\mathcal{O}^\star(2^{o(n)})$?

First try

- Can you do *significantly* better, i.e. have running time $\mathcal{O}^\star(2^{o(n)})$?
- If you do not have a bound on the clause length, can you do anything smarter than brute-force?

First try

- Can you do *significantly* better, i.e. have running time $\mathcal{O}^\star(2^{o(n)})$?
- If you do not have a bound on the clause length, can you do anything smarter than brute-force?
- With current knowledge, we are faaaaar from answering positively any of these two questions.

First try

- Can you do *significantly* better, i.e. have running time $\mathcal{O}^\star(2^{o(n)})$?
- If you do not have a bound on the clause length, can you do anything smarter than brute-force?
- With current knowledge, we are faaaaar from answering positively any of these two questions.

ETH and SETH, first try

- 3-SAT cannot be solved in time $\mathcal{O}^\star(2^{o(n)})$.
- General CNF-SAT cannot be solved in time $\mathcal{O}^\star(c^n)$ for any $c < 2$.

Actual statements

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

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

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

Strong Exponential Time Hypothesis, SETH

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

Actual statements

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

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

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

Strong Exponential Time Hypothesis, SETH

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

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

Actual statements

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

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

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

Strong Exponential Time Hypothesis, SETH

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

- These statements are *stronger* than the first attempts.
- Formulated by Impagliazzo, Paturi and Zane in 2001.

Actual statements

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

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

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

Strong Exponential Time Hypothesis, SETH

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

- These statements are *stronger* than the first attempts.
- Formulated by Impagliazzo, Paturi and Zane in 2001.
- Usually we allow two-sided error algorithms in the definition.

Actual statements

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

Exponential Time Hypothesis, ETH

$$\delta_3 > 0.$$

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

Strong Exponential Time Hypothesis, SETH

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

- These statements are *stronger* than the first attempts.
- Formulated by Impagliazzo, Paturi and Zane in 2001.
- Usually we allow two-sided error algorithms in the definition.
- SETH $\Rightarrow$ ETH (nontrivial)

Reductions

- As usual with lower bounds, we would like to transfer them via reductions.

Reductions

- As usual with lower bounds, we would like to transfer them via reductions.
- A reduction $3\text{-SAT} \to L$ and a too fast algorithm for L would give a too fast algorithm for 3-SAT.

Reductions

- As usual with lower bounds, we would like to transfer them via reductions.
- A reduction $3\text{-SAT} \rightarrow L$ and a too fast algorithm for L would give a too fast algorithm for 3-SAT.

Vertex Cover

Input: A graph G and an integer k
Question: Is there a set $X \subseteq V(G)$ with $|X| \leq k$ such that every edge of G has at least one endpoint in X?

Vertex Cover reduction

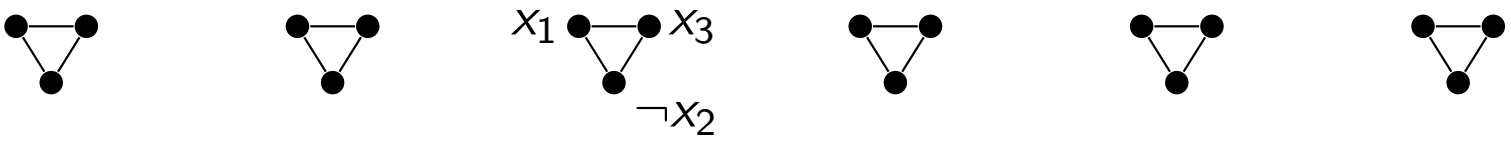
Let us inspect the standard reduction from 3-SAT to Vertex Cover.

clauses

variables

Vertex Cover reduction

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

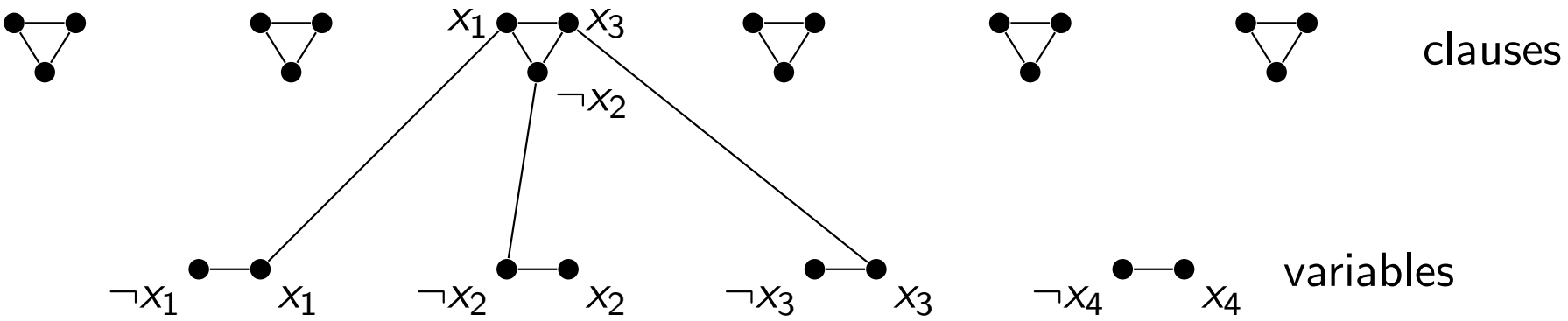


clauses

variables

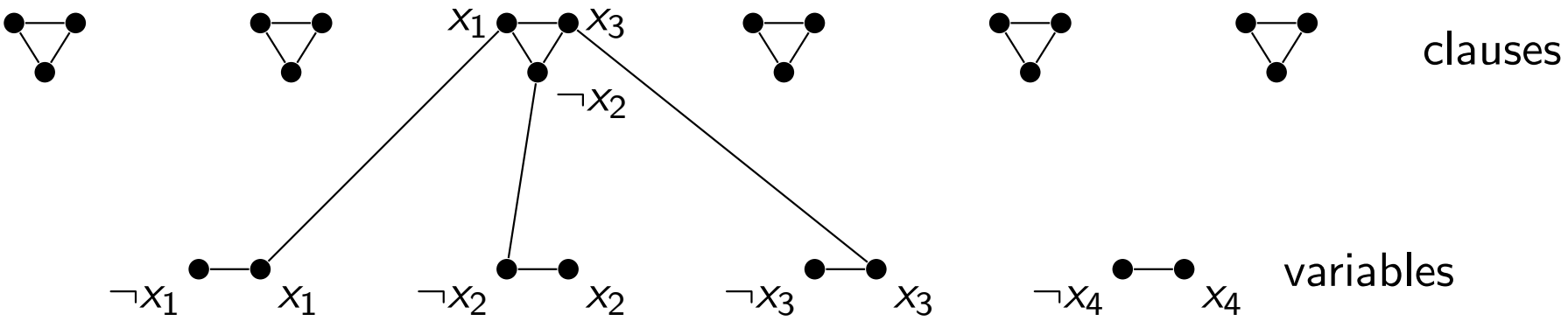
Vertex Cover reduction

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



Vertex Cover reduction

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



The formula is satisfiable iff the created graph has 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 $\mathcal{O}^\star(2^{o(N^{1/3})})$ algorithm for VC would give an $\mathcal{O}^\star(2^{o(n)})$ algorithm for $3\text{-}\mathrm{SAT}$, contradicting ETH.

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 $\mathcal{O}^\star(2^{o(N^{1/3})})$ algorithm for VC would give an $\mathcal{O}^\star(2^{o(n)})$ algorithm for $3\text{-}\mathrm{SAT}$, contradicting ETH.
- But we know no algorithm with running time $\mathcal{O}^\star(2^{o(N)})$...

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 $\mathcal{O}^\star(2^{o(N^{1/3})})$ algorithm for VC would give an $\mathcal{O}^\star(2^{o(n)})$ algorithm for $3\text{-}\mathrm{SAT}$, contradicting ETH.
- But we know no algorithm with running time $\mathcal{O}^\star(2^{o(N)})$...
- If we started with an instance of $3\text{-}\mathrm{SAT}$ that is *sparse*, i.e., $m = \mathcal{O}(n)$, then a $\mathcal{O}^\star(2^{o(N)})$ lower bound would follow.

Sparsification Lemma

Sparsification Lemma informally

There is subexponential time algorithm which reduces the number of clauses of a q-SAT formula to $\mathcal{O}(n)$, for any constant q.

Sparsification Lemma

Sparsification Lemma informally

There is subexponential time algorithm which reduces the number of clauses of a q-SAT formula to $\mathcal{O}(n)$, for any constant q.

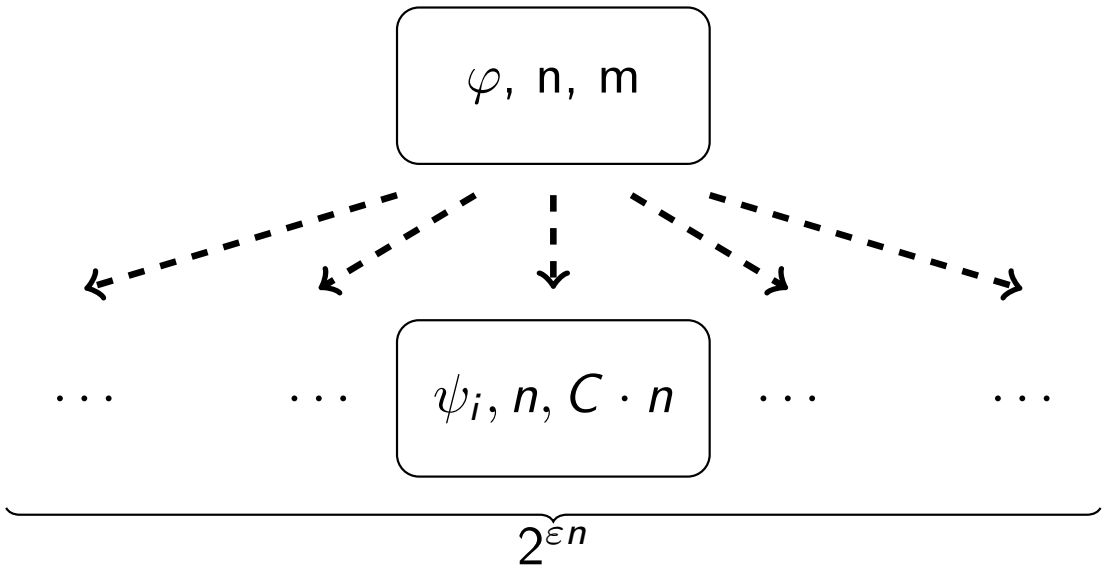
Sparsification Lemma [IPZ, 2001]

For any $q \geq 3$ and $\varepsilon > 0$ there is a constant $C = C(q, \varepsilon)$ such that any q-CNF 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 q-CNF 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 q-CNF 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 q-CNF 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

Theorem

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

Sparsification Lemma

Theorem

Unless ETH fails, there is a constant $c > 0$, such that no algorithm solves 3-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-SAT in time $\mathcal{O}^\star(2^{c(n+m)})$.
- Consider any $d > 0$. We want to solve 3-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 $\mathcal{O}^\star(2^{\varepsilon n}) + \mathcal{O}^\star(2^{\varepsilon n} \cdot 2^{\frac{d}{2(C+1)} \cdot (C+1)n}) = \mathcal{O}^\star(2^{dn})$.

Sparsification Lemma

Theorem

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

Corollary

Under ETH, there is no $\mathcal{O}^\star(2^{o(n+m)})$ time algorithm for 3-SAT.

Sparsification Lemma

Theorem

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

Corollary

Under ETH, there is no $\mathcal{O}^\star(2^{o(n+m)})$ time algorithm for 3-SAT.

Corollary

Under ETH, there is no $\mathcal{O}^\star(2^{o(N+M)})$ time algorithm for $\textsc{Vertex Cover}$.

Corollaries

- We basically needed only a *linear reduction* from 3-SAT.

Corollaries

- We basically needed only a *linear reduction* from 3-SAT.
- Other problems that also admit such reductions:
  - Feedback Vertex Set,
  - Dominating Set,
  - 3-Coloring,
  - Hamiltonian Cycle,
  - and many others.

Planar problems

- Consider Planar Vertex Cover.

Planar problems

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

Planar problems

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

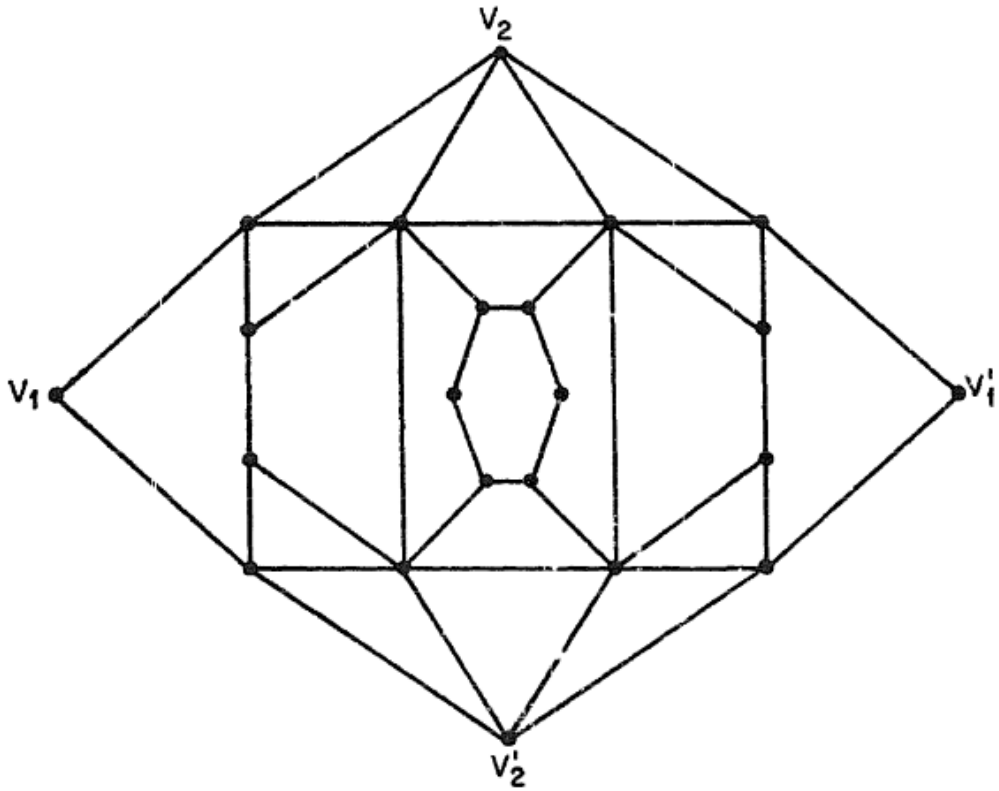


Fig. 11. Crossover H for Theorem 2.7.

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.

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 $\mathcal{O}^\star(2^{o(\sqrt{N})})$ algorithm for PlVC would give an $\mathcal{O}(2^{o(n+m)})$ algorithm for VC, contradicting ETH.

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 $\mathcal{O}^\star(2^{o(\sqrt{N})})$ algorithm for PlVC would give an $\mathcal{O}(2^{o(n+m)})$ algorithm for VC, contradicting ETH.
- PlVC actually **has** an algorithm working in $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{N})})$:

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 $\mathcal{O}^\star(2^{o(\sqrt{N})})$ algorithm for PlVC would give an $\mathcal{O}(2^{o(n+m)})$ algorithm for VC, contradicting ETH.
- PlVC actually **has** an algorithm working in $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{N})})$:
  - Divide&Conquer + Lipton-Tarjan planar separator theorem, or

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 $\mathcal{O}^\star(2^{o(\sqrt{N})})$ algorithm for PlVC would give an $\mathcal{O}(2^{o(n+m)})$ algorithm for VC, contradicting ETH.
- PlVC actually **has** an algorithm working in $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{N})})$:
  - Divide&Conquer + Lipton-Tarjan planar separator theorem, or
  - 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 $\mathcal{O}^\star(2^{o(\sqrt{N})})$ algorithm for PlVC would give an $\mathcal{O}(2^{o(n+m)})$ algorithm for VC, contradicting ETH.
- PlVC actually **has** an algorithm working in $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{N})})$:
  - Divide&Conquer + Lipton-Tarjan planar separator theorem, or
  - Treewidth DP + planar graph on N vertices has treewidth $\mathcal{O}(\sqrt{N})$.
- The $\sqrt{N}$ term in the exponent is not a coincidence!

Parameterized complexity

- An instance of a parameterized problem comes with a *parameter* k, e.g. size of the solution, treewidth, number of variables, etc.

Parameterized complexity

- An instance of a parameterized problem comes with a *parameter* k, e.g. size of the solution, treewidth, number of variables, etc.
- The parameter measures the hardness of the instance.

Parameterized complexity

- An instance of a parameterized problem comes with a *parameter* k, e.g. size of the solution, treewidth, number of variables, etc.
- The parameter measures the hardness of the instance.
- **FPT running time**: $f(k) \cdot n^{\mathcal{O}(1)} = \mathcal{O}^\star(f(k))$ for a computable function f.

Parameterized complexity

- An instance of a parameterized problem comes with a *parameter* k, e.g. size of the solution, treewidth, number of variables, etc.
- The parameter measures the hardness of the instance.
- **FPT running time**: $f(k) \cdot n^{\mathcal{O}(1)} = \mathcal{O}^\star(f(k))$ for a computable function f.
- **XP running time**: $f(k) \cdot n^{g(k)}$ for computable functions f, g.

Parameterized complexity

- An instance of a parameterized problem comes with a *parameter* k, e.g. size of the solution, treewidth, number of variables, etc.
- The parameter measures the hardness of the instance.
- **FPT running time**: $f(k) \cdot n^{\mathcal{O}(1)} = \mathcal{O}^\star(f(k))$ for a computable function f.
- **XP running time**: $f(k) \cdot n^{g(k)}$ for computable functions f, g.
- Vertex Cover is FPT because it admits a simple $\mathcal{O}^\star(2^k)$ branching algorithm.

Parameterized complexity

- An instance of a parameterized problem comes with a *parameter* k, e.g. size of the solution, treewidth, number of variables, etc.
- The parameter measures the hardness of the instance.
- **FPT running time**: $f(k) \cdot n^{\mathcal{O}(1)} = \mathcal{O}^\star(f(k))$ for a computable function f.
- **XP running time**: $f(k) \cdot n^{g(k)}$ for computable functions f, g.
- Vertex Cover is FPT because it admits a simple $\mathcal{O}^\star(2^k)$ branching algorithm.
- Clique is XP because it admits a simple $\mathcal{O}(k^2 \cdot n^k)$ algorithm, but probably is not FPT.

Parameterized complexity

- An instance of a parameterized problem comes with a *parameter* k, e.g. size of the solution, treewidth, number of variables, etc.
- The parameter measures the hardness of the instance.
- **FPT running time**: $f(k) \cdot n^{\mathcal{O}(1)} = \mathcal{O}^\star(f(k))$ for a computable function f.
- **XP running time**: $f(k) \cdot n^{g(k)}$ for computable functions f, g.
- Vertex Cover is FPT because it admits a simple $\mathcal{O}^\star(2^k)$ branching algorithm.
- Clique is XP because it admits a simple $\mathcal{O}(k^2 \cdot n^k)$ algorithm, but probably is not FPT.
  - It is W[1]-hard.

ETH and parameterized complexity

- **Goal**: lower bounds on functions f and g under ETH.

ETH and parameterized complexity

- **Goal**: lower bounds on functions f and g under ETH.
- $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $\mathcal{O}^\star(2^{o(n+m)})$ algorithm, contradiction.

ETH and parameterized complexity

- **Goal**: lower bounds on functions f and g under ETH.
- $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $\mathcal{O}^\star(2^{o(n+m)})$ algorithm, contradiction.
- $\mathcal{O}^\star(2^{o(\sqrt{k})})$ algorithm for PlVC would be also a $\mathcal{O}^\star(2^{o(\sqrt{n})})$ algorithm, contradiction.

ETH and parameterized complexity

- **Goal**: lower bounds on functions f and g under ETH.
- $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $\mathcal{O}^\star(2^{o(n+m)})$ algorithm, contradiction.
- $\mathcal{O}^\star(2^{o(\sqrt{k})})$ algorithm for PlVC would be also a $\mathcal{O}^\star(2^{o(\sqrt{n})})$ algorithm, contradiction.
  - **Remark**: $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ possible using *bidimensionality*.

ETH and parameterized complexity

- **Goal**: lower bounds on functions f and g under ETH.
- $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $\mathcal{O}^\star(2^{o(n+m)})$ algorithm, contradiction.
- $\mathcal{O}^\star(2^{o(\sqrt{k})})$ algorithm for PlVC would be also a $\mathcal{O}^\star(2^{o(\sqrt{n})})$ algorithm, contradiction.
  - **Remark**: $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ possible using *bidimensionality*.

Lower bounds for parameterized problems under ETH

If L admits a reduction from $3\text{-}\mathrm{SAT}$ where $k = f(n+m)$, then L does not admit an $\mathcal{O}^\star(2^{o(f^{-1}(k))})$ algorithm unless ETH fails.

ETH and parameterized complexity

- **Goal**: lower bounds on functions f and g under ETH.
- $\mathcal{O}^\star(2^{o(k)})$ algorithm for VC would be also a $\mathcal{O}^\star(2^{o(n+m)})$ algorithm, contradiction.
- $\mathcal{O}^\star(2^{o(\sqrt{k})})$ algorithm for PlVC would be also a $\mathcal{O}^\star(2^{o(\sqrt{n})})$ algorithm, contradiction.
  - **Remark**: $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ possible using *bidimensionality*.

Lower bounds for parameterized problems under ETH

If L admits a reduction from $3\text{-}\mathrm{SAT}$ where $k = f(n+m)$, then L does not admit an $\mathcal{O}^\star(2^{o(f^{-1}(k))})$ algorithm unless ETH fails.

- The parameter blow-up governs the strength of the lower bound.

ETH and parameterized complexity

- For many parameterized problems (VC, FVS, OCT) the situation is simple:

ETH and parameterized complexity

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

ETH and parameterized complexity

- For many parameterized problems (VC, FVS, OCT) the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(c^k)$ 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 (VC, FVS, OCT) the situation is simple:
  - An algorithm with running time $\mathcal{O}^\star(c^k)$ 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.
- Let's look at some more exotic running times.

Slightly super-exponential parameterized time

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

Slightly super-exponential parameterized time

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

Slightly super-exponential parameterized time

- 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.

Slightly super-exponential parameterized time

- 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.

Slightly super-exponential parameterized time

- 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.

Slightly super-exponential parameterized time

- 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.

Slightly super-exponential parameterized time

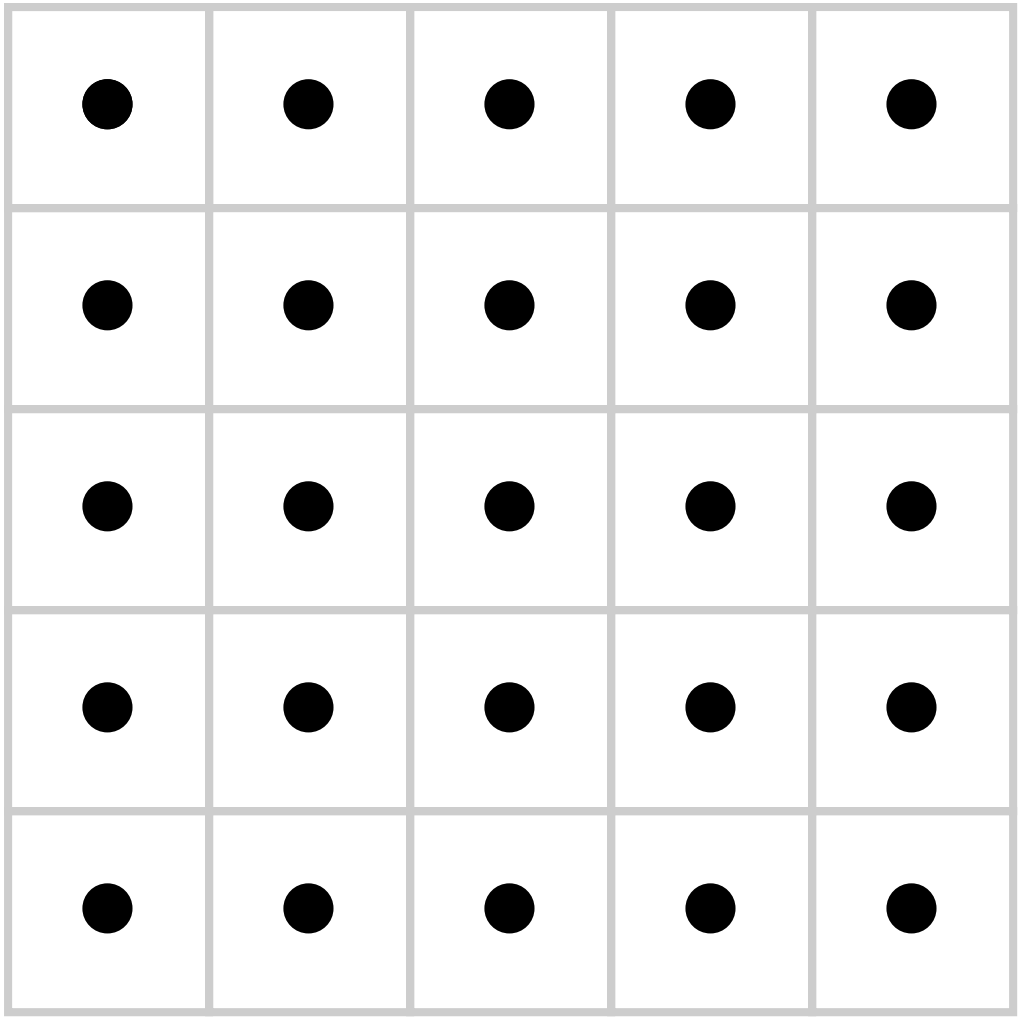
- Slightly super-exponential $= \mathcal{O}^\star(2^{\mathcal{O}(k \log k)}) = \mathcal{O}^\star(k^{\mathcal{O}(k)})$
- Appears naturally:
  (a) Iterate through $k!$ possibilities.
  (b) A branching procedure branches $\mathcal{O}(k)$ times, each time choosing one of $\mathrm{poly}(k)$ options.
  (c) 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.
- Results by Lokshtanov, Marx, and Saurabh (2011).

$k \times k$-Clique

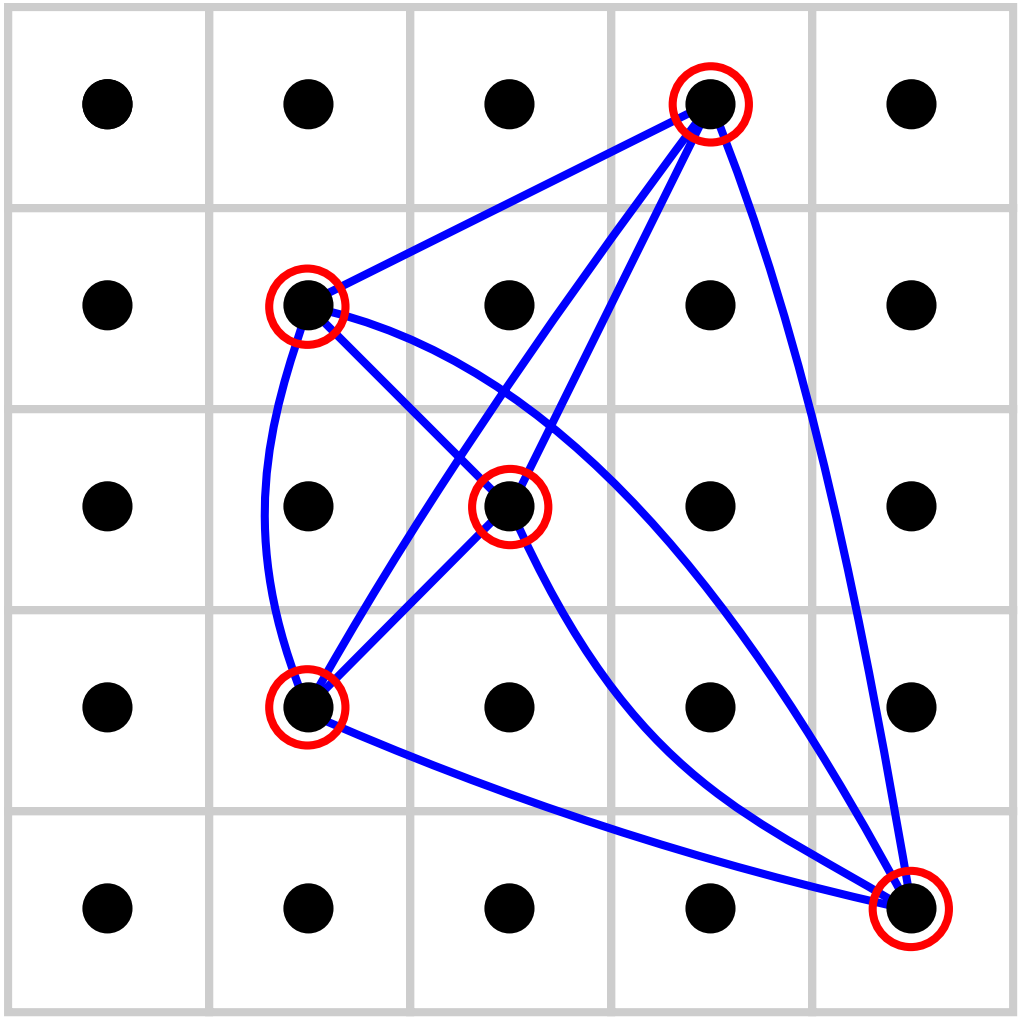
Input: A graph H on vertex set $[k] \times [k]$
Question: Is there a k-clique in H that contains exactly one vertex from each row?

- $[k] = \{1, 2, \ldots, k\}$.

On a picture



On a picture



About $k \times k$-Clique

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

About $k \times k$-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.

About $k \times k$-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.

About $k \times k$-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.
- **Now**: $k \times k$-Clique does not admit an $\mathcal{O}^\star(2^{o(k \log k)})$ algorithm unless ETH fails.

Lower bound for $k \times k$-Clique

- **Starting point**: 3-Coloring on an N-vertex graph does not admit a $2^{o(N)}$ algorithm.

Lower bound for $k \times k$-Clique

- **Starting point**: 3-Coloring on an N-vertex graph does not admit a $2^{o(N)}$ algorithm.
- Take an instance G of 3-Coloring.

Lower bound for $k \times k$-Clique

- **Starting point**: 3-Coloring on an N-vertex graph does not admit a $2^{o(N)}$ algorithm.
- Take an instance G of 3-Coloring.
- Divide the vertices into $k := \frac{2N}{\log_3 N}$ groups, each of size $\frac{\log_3 N}{2}$.

Lower bound for $k \times k$-Clique

- **Starting point**: 3-Coloring on an N-vertex graph does not admit a $2^{o(N)}$ algorithm.
- Take an instance G of 3-Coloring.
- Divide the vertices into $k := \frac{2N}{\log_3 N}$ groups, each of size $\frac{\log_3 N}{2}$.
- For each of the groups list all the 3-colorings.

Lower bound for $k \times k$-Clique

- **Starting point**: 3-Coloring on an N-vertex graph does not admit a $2^{o(N)}$ algorithm.
- Take an instance G of 3-Coloring.
- Divide the vertices into $k := \frac{2N}{\log_3 N}$ groups, each of size $\frac{\log_3 N}{2}$.
- For each of the groups list all the 3-colorings.
  - There is $3^{\frac{\log_3 N}{2}} = \sqrt{N} \leq k$ of them.

Lower bound for $k \times k$-Clique

- For $i \in [k]$ and $j \in [\sqrt{N}]$, vertex (i, j) represents the j-th coloring of the i-th group.

Lower bound for $k \times k$-Clique

- For $i \in [k]$ and $j \in [\sqrt{N}]$, vertex (i, j) represents the j-th coloring of the i-th group.
- For $i \neq i'$, put an edge between (i, j) and (i', j') if respective colorings together form a proper coloring of the union of the groups i and i'.

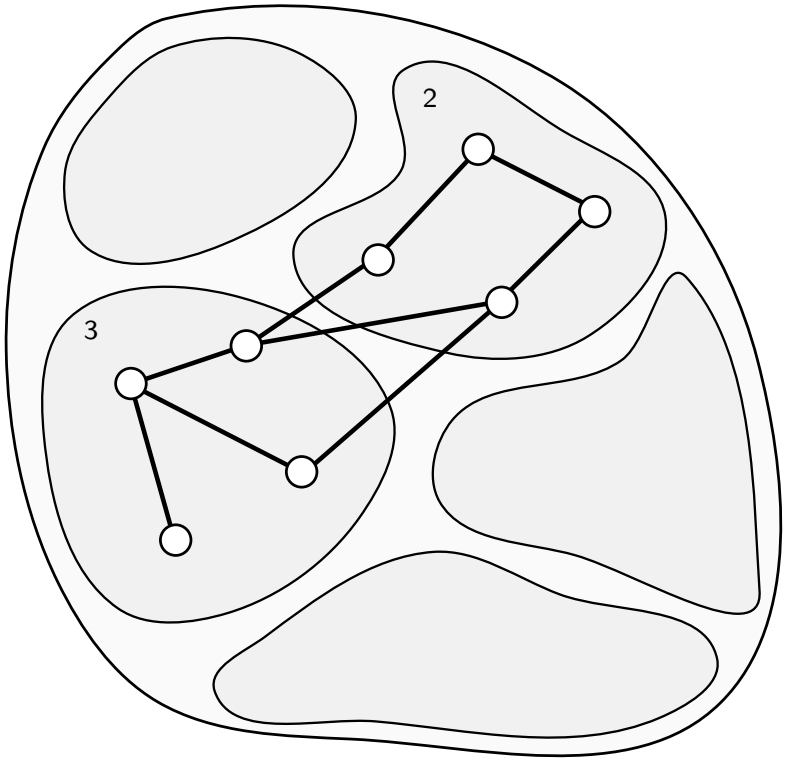
Lower bound for $k \times k$-Clique

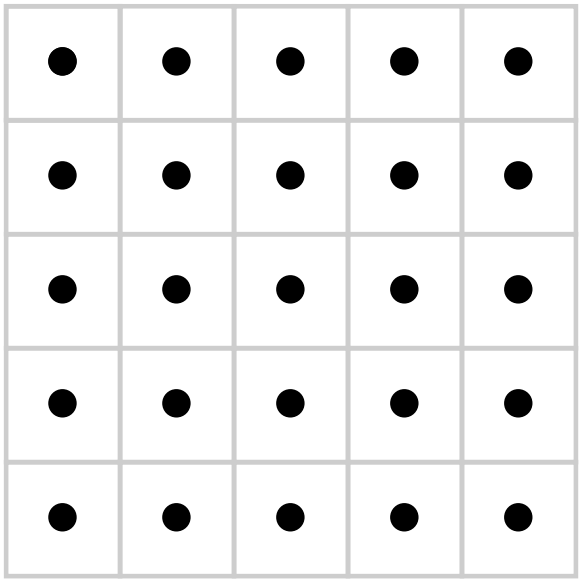
- For $i \in [k]$ and $j \in [\sqrt{N}]$, vertex (i, j) represents the j-th coloring of the i-th group.
- For $i \neq i'$, put an edge between (i, j) and (i', j') if respective colorings together form a proper coloring of the union of the groups i and i'.
- **Note**: colorings that are not proper already on their own groups will become isolated vertices.

Lower bound for $k \times k$-Clique

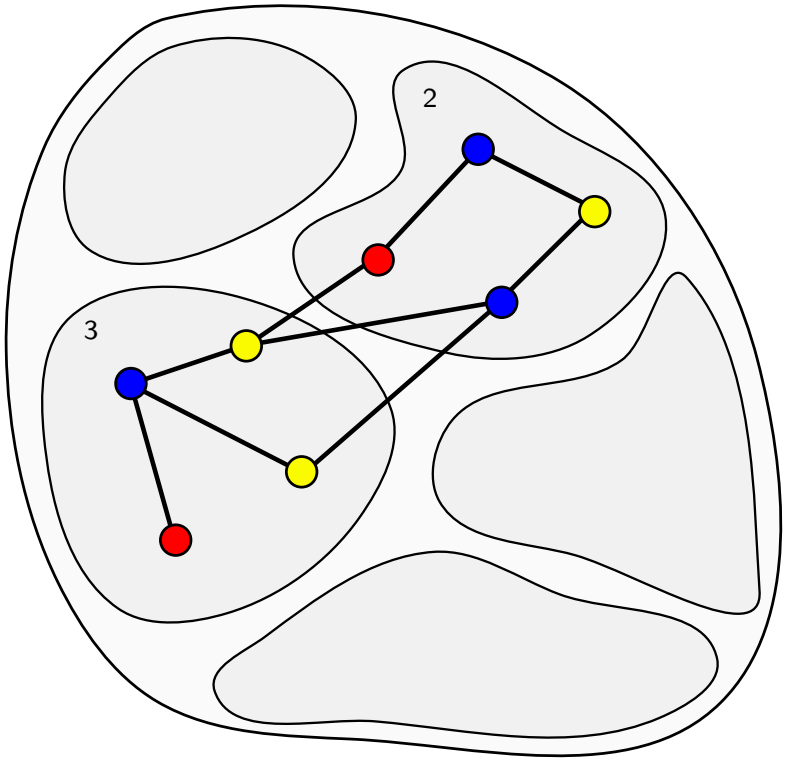
- For $i \in [k]$ and $j \in [\sqrt{N}]$, vertex (i, j) represents the j-th coloring of the i-th group.
- For $i \neq i'$, put an edge between (i, j) and (i', j') if respective colorings together form a proper coloring of the union of the groups i and i'.
- **Note**: colorings that are not proper already on their own groups will become isolated vertices.
- Finally, fill the rows with isolated vertices up to size k.

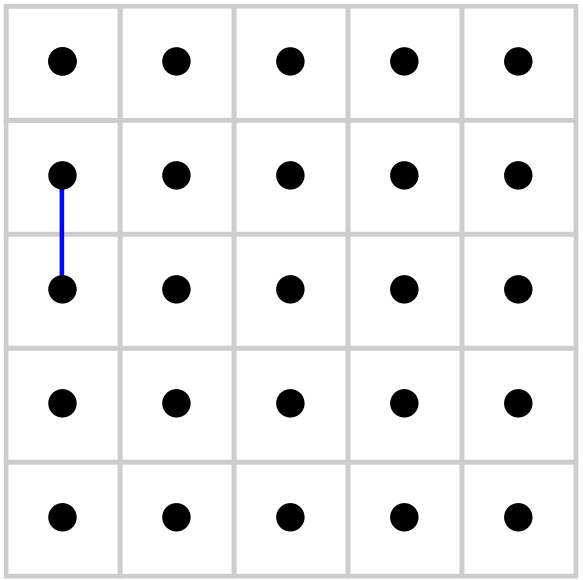
On a picture



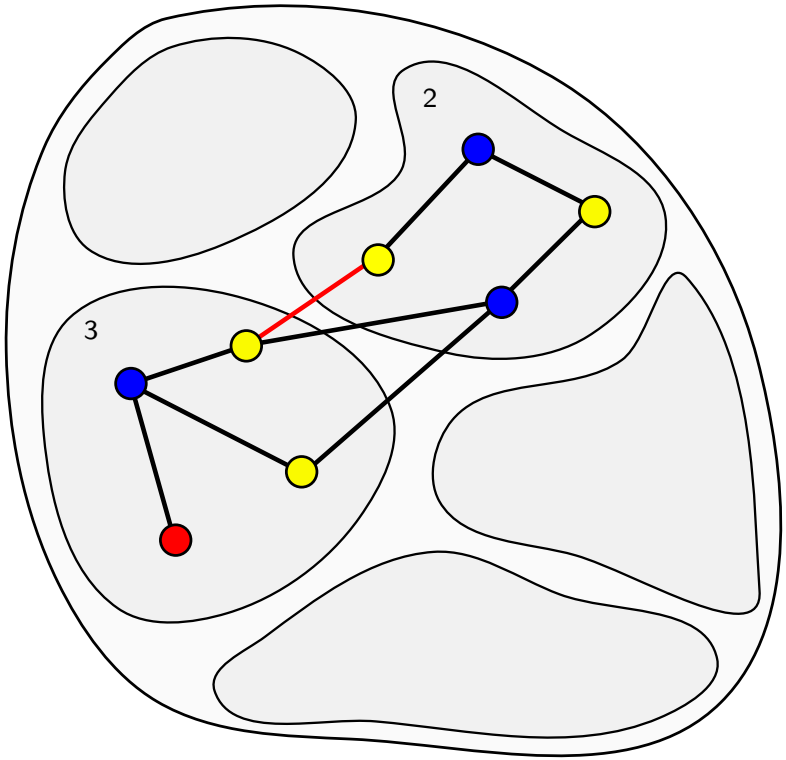


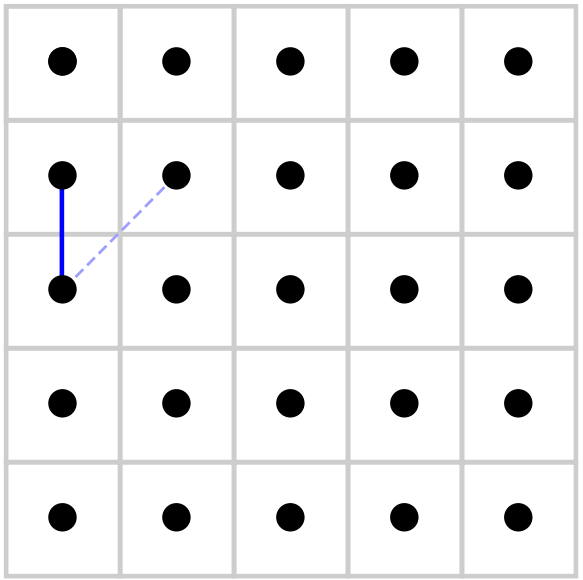
On a picture



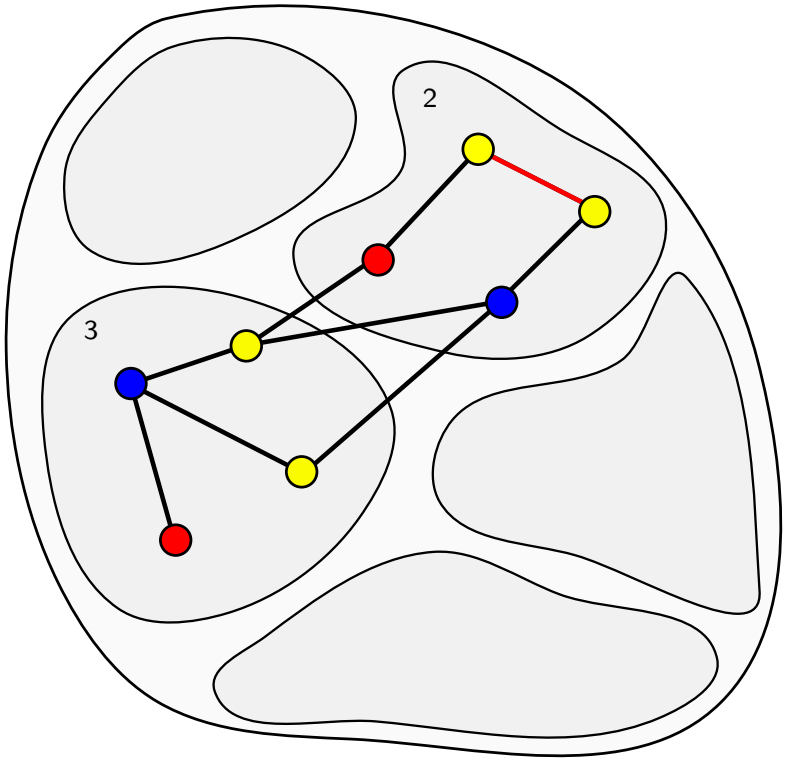


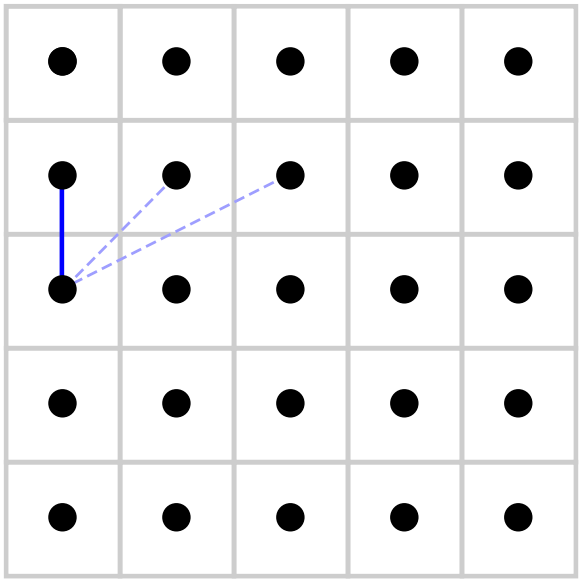
On a picture





On a picture





Equivalence

- If there is a coloring, then there is a clique: trivial.

Equivalence

- If there is a coloring, then there is a clique: trivial.
- If there is a clique, then consider the coloring imposed by it.

Equivalence

- If there is a coloring, then there is a clique: trivial.
- If there is a clique, then consider the coloring imposed by it.
- Suppose there is an edge with endpoints of the same color.

Equivalence

- If there is a coloring, then there is a clique: trivial.
- If there is a clique, then consider the coloring imposed by it.
- Suppose there is an edge with endpoints of the same color.
  - Within a group: the coloring of this group would yield an isolated vertex.

Equivalence

- If there is a coloring, then there is a clique: trivial.
- If there is a clique, then consider the coloring imposed by it.
- Suppose there is an edge with endpoints of the same color.
  - Within a group: the coloring of this group would yield an isolated vertex.
  - Between two groups: the corresponding colorings of the groups wouldn't be connected by an edge.

Equivalence

- If there is a coloring, then there is a clique: trivial.
- If there is a clique, then consider the coloring imposed by it.
- Suppose there is an edge with endpoints of the same color.
  - Within a group: the coloring of this group would yield an isolated vertex.
  - Between two groups: the corresponding colorings of the groups wouldn't be connected by an edge.
- Since $k = \mathcal{O}(N/\log N)$, an $\mathcal{O}^\star(2^{o(k \log k)})$ algorithm for $k \times k$-Clique implies a $2^{o\left(\frac{N}{\log N} \cdot \log N\right)} = 2^{o(N)}$ algorithm for 3-Coloring.

Equivalence

- If there is a coloring, then there is a clique: trivial.
- If there is a clique, then consider the coloring imposed by it.
- Suppose there is an edge with endpoints of the same color.
  - Within a group: the coloring of this group would yield an isolated vertex.
  - Between two groups: the corresponding colorings of the groups wouldn't be connected by an edge.
- Since $k = \mathcal{O}(N/\log N)$, an $\mathcal{O}^\star(2^{o(k \log k)})$ algorithm for $k \times k$-Clique implies a $2^{o\left(\frac{N}{\log N} \cdot \log N\right)} = 2^{o(N)}$ algorithm for 3-Coloring.
- And we are done.

Further corollaries

- Closest String: Given equal-length strings $x_1, x_2, \ldots, x_n$ over Σ and an integer d, decide whether there is some y within Hamming distance $\leq d$ from each x_i.

Further corollaries

- Closest String: Given equal-length strings $x_1, x_2, \ldots, x_n$ over Σ and an integer d, decide whether there is some y within Hamming distance $\leq d$ from each x_i.
  - There are algorithms with running time $\mathcal{O}^\star(2^{\mathcal{O}(d \log d)})$ and $\mathcal{O}^\star(2^{\mathcal{O}(d \log |\Sigma|)})$.

Further corollaries

- Closest String: Given equal-length strings $x_1, x_2, \ldots, x_n$ over Σ and an integer d, decide whether there is some y within Hamming distance $\leq d$ from each x_i.
  - There are algorithms with running time $\mathcal{O}^\star(2^{\mathcal{O}(d \log d)})$ and $\mathcal{O}^\star(2^{\mathcal{O}(d \log |\Sigma|)})$.
  - Existence of algorithms with running time $\mathcal{O}^\star(2^{o(d \log d)})$ or $\mathcal{O}^\star(2^{o(d \log |\Sigma|)})$ would contradict ETH.

Further corollaries

- Closest String: Given equal-length strings $x_1, x_2, \ldots, x_n$ over Σ and an integer d, decide whether there is some y within Hamming distance $\leq d$ from each x_i.
  - There are algorithms with running time $\mathcal{O}^\star(2^{\mathcal{O}(d \log d)})$ and $\mathcal{O}^\star(2^{\mathcal{O}(d \log |\Sigma|)})$.
  - Existence of algorithms with running time $\mathcal{O}^\star(2^{o(d \log d)})$ or $\mathcal{O}^\star(2^{o(d \log |\Sigma|)})$ would contradict ETH.
- Treewidth dynamic programming, e.g. Cycle Packing:

Further corollaries

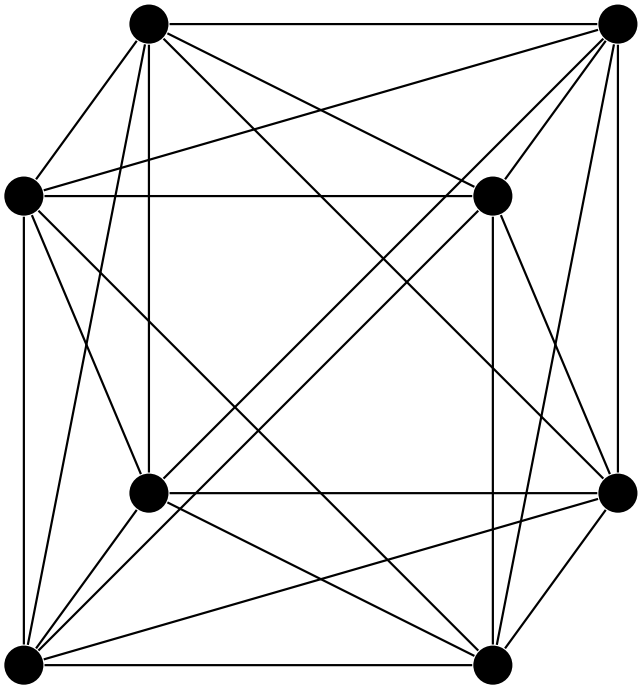
- Closest String: Given equal-length strings $x_1, x_2, \ldots, x_n$ over Σ and an integer d, decide whether there is some y within Hamming distance $\leq d$ from each x_i.
  - There are algorithms with running time $\mathcal{O}^\star(2^{\mathcal{O}(d \log d)})$ and $\mathcal{O}^\star(2^{\mathcal{O}(d \log |\Sigma|)})$.
  - Existence of algorithms with running time $\mathcal{O}^\star(2^{o(d \log d)})$ or $\mathcal{O}^\star(2^{o(d \log |\Sigma|)})$ would contradict ETH.
- Treewidth dynamic programming, e.g. Cycle Packing:
  - $\mathcal{O}^\star(2^{\mathcal{O}(t \log t)})$ algorithm by remembering a matching within the bag.

Further corollaries

- Closest String: Given equal-length strings $x_1, x_2, \ldots, x_n$ over Σ and an integer d, decide whether there is some y within Hamming distance $\leq d$ from each x_i.
  - There are algorithms with running time $\mathcal{O}^\star(2^{\mathcal{O}(d \log d)})$ and $\mathcal{O}^\star(2^{\mathcal{O}(d \log |\Sigma|)})$.
  - Existence of algorithms with running time $\mathcal{O}^\star(2^{o(d \log d)})$ or $\mathcal{O}^\star(2^{o(d \log |\Sigma|)})$ would contradict ETH.
- Treewidth dynamic programming, e.g. Cycle Packing:
  - $\mathcal{O}^\star(2^{\mathcal{O}(t \log t)})$ algorithm by remembering a matching within the bag.
  - $\mathcal{O}^\star(2^{o(t \log t)})$ would contradict ETH.

Edge Clique Cover (ECC)

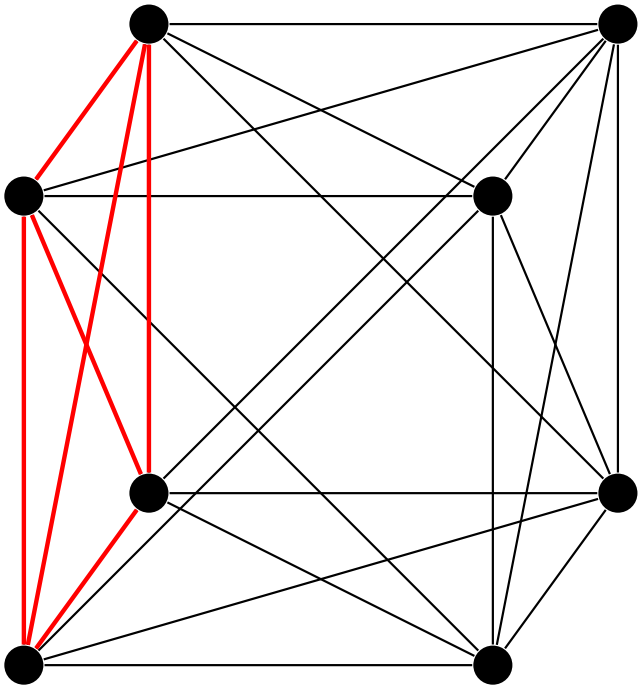
Input: Graph G, integer k
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover

Edge Clique Cover (ECC)

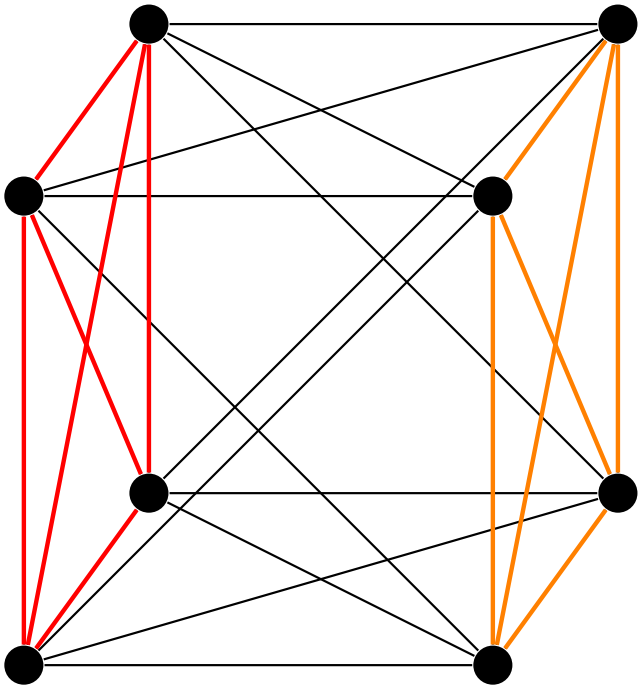
Input: Graph G, integer k
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover

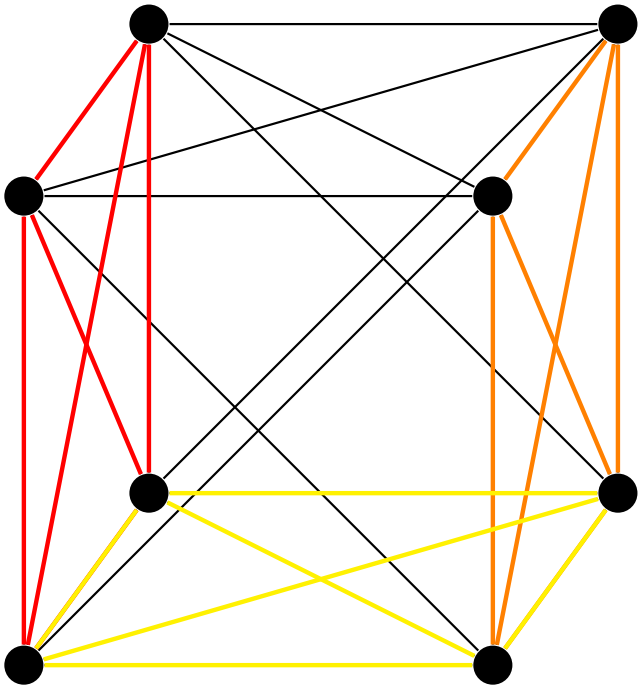
Edge Clique Cover (ECC)

Input: Graph G, integer k
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover (ECC)

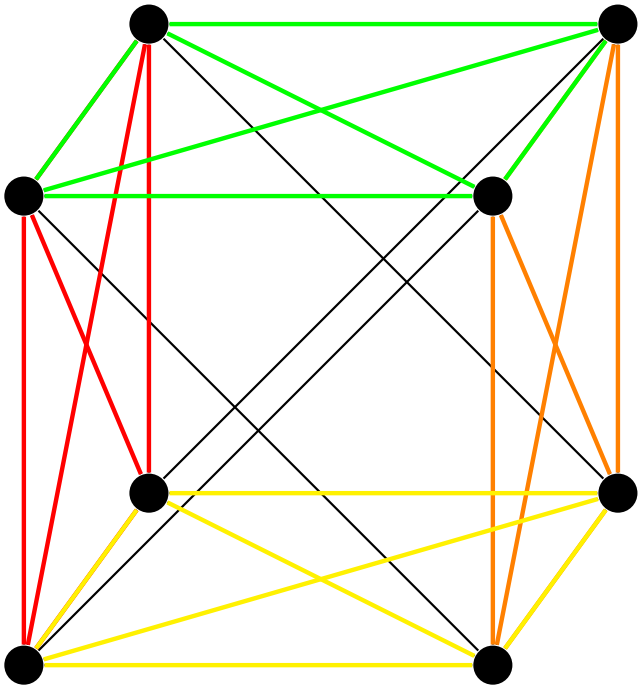
Input: Graph G, integer k
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover

Edge Clique Cover (ECC)

Input: Graph G, integer k
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?

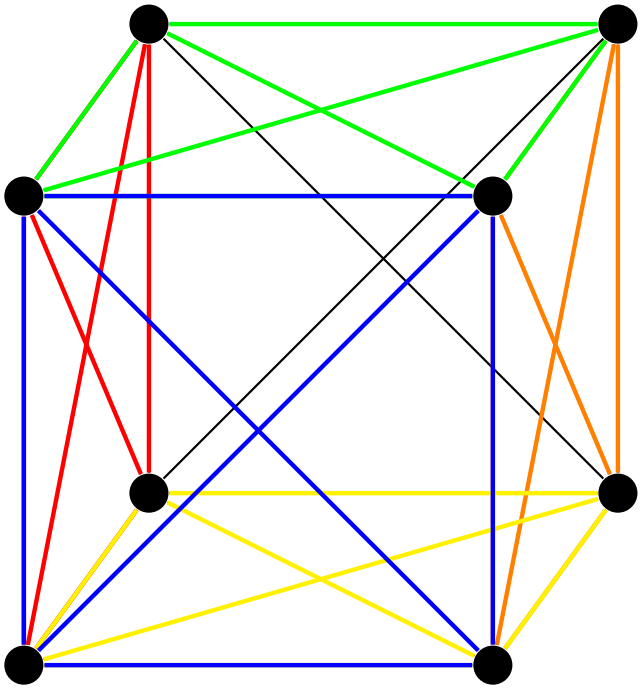


Edge Clique Cover

Edge Clique Cover (ECC)

Input: Graph G, integer k

Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?

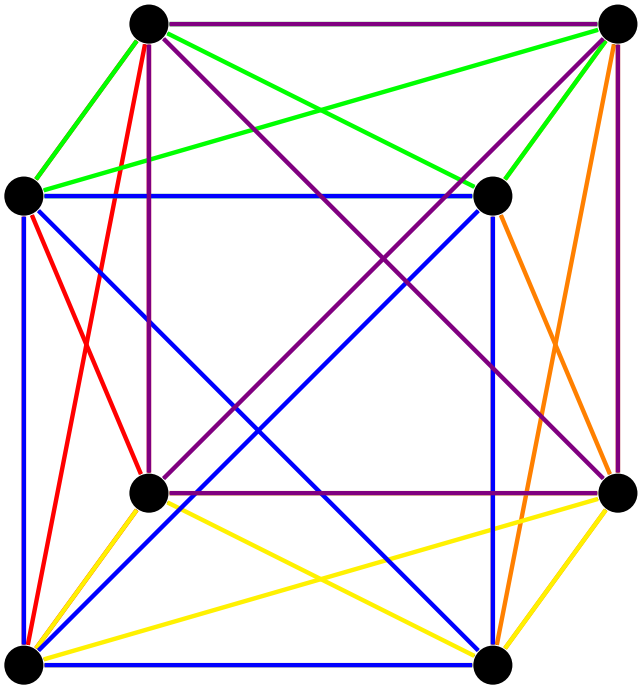


Edge Clique Cover

Edge Clique Cover (ECC)

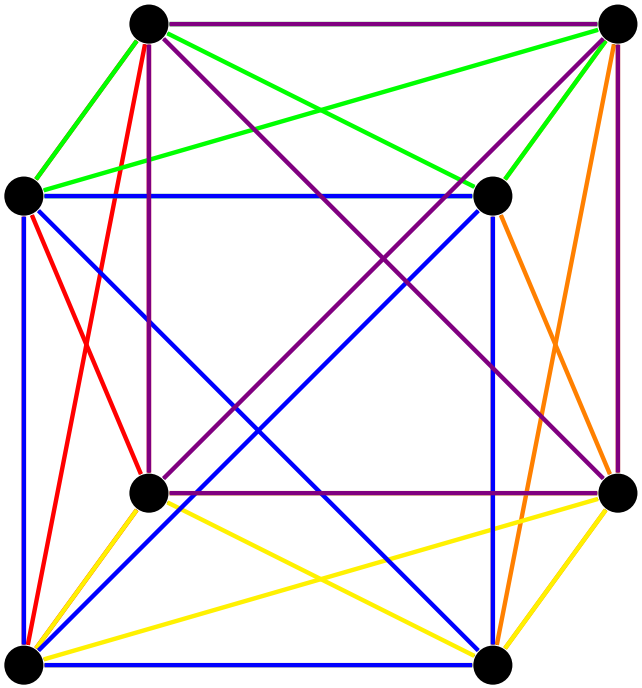
Input: Graph G, integer k

Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover (ECC)

Input: Graph G, integer k
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



6 cliques, but not optimal

puzzle: find a solution with 5 cliques

Bound for ECC

Theorem (Gramm, Guo, Hüffner, Niedermeier)

Edge Clique Cover admits a kernel with at most 2^k vertices. That is, one can preprocess the instance in polynomial time to ensure that the number of vertices in at most 2^k.

Bound for ECC

Theorem (Gramm, Guo, Hüffner, Niedermeier)

Edge Clique Cover admits a kernel with at most 2^k vertices. That is, one can preprocess the instance in polynomial time to ensure that the number of vertices in at most 2^k.

- Basically just as follows:
  If there are two twins in the graph, remove one of them.

Bound for ECC

Theorem (Gramm, Guo, Hüffner, Niedermeier)

Edge Clique Cover admits a kernel with at most 2^k vertices. That is, one can preprocess the instance in polynomial time to ensure that the number of vertices in at most 2^k.

- Basically just as follows:
  If there are two twins in the graph, remove one of them.
- Kernel plus DP on the subsets of edges $\Rightarrow$ $\mathcal{O}^\star(2^{2^{\mathcal{O}(k)}})$ algorithm.

Bound for ECC

Theorem (Gramm, Guo, Hüffner, Niedermeier)

Edge Clique Cover admits a kernel with at most 2^k vertices. That is, one can preprocess the instance in polynomial time to ensure that the number of vertices in at most 2^k.

- Basically just as follows:
  If there are two twins in the graph, remove one of them.
- Kernel plus DP on the subsets of edges $\Rightarrow$ $\mathcal{O}^\star(2^{2^{\mathcal{O}(k)}})$ algorithm.

Theorem (Cygan, Pilipczuk, P)

There is a reduction from 3-SAT to ECC that outputs an instance with $k = \mathcal{O}(\log(n+m))$. Consequently, there is no $\mathcal{O}^\star(2^{2^{o(k)}})$ time algorithm for ECC unless ETH fails.

ETH and W[1]-hardness

- Under $\mathrm{W}[1] \neq \mathrm{FPT}$, $\textsc{Clique}$ cannot be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time.

ETH and W[1]-hardness

- Under $\mathrm{W}[1] \neq \mathrm{FPT}$, Clique cannot be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time.
- Best known algorithm: $\mathcal{O}^\star(n^{0.492k})$

ETH and W[1]-hardness

- Under $\mathrm{W}[1] \neq \mathrm{FPT}$, Clique cannot be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time.
- Best known algorithm: $\mathcal{O}^\star(n^{0.492k})$
- Can we prove that the linear dependence on k is necessary?

ETH and W[1]-hardness

- Under $\mathrm{W}[1] \neq \mathrm{FPT}$, Clique cannot be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time.
- Best known algorithm: $\mathcal{O}^\star(n^{0.492k})$
- Can we prove that the linear dependence on k is necessary?

Theorem (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 and W[1]-hardness

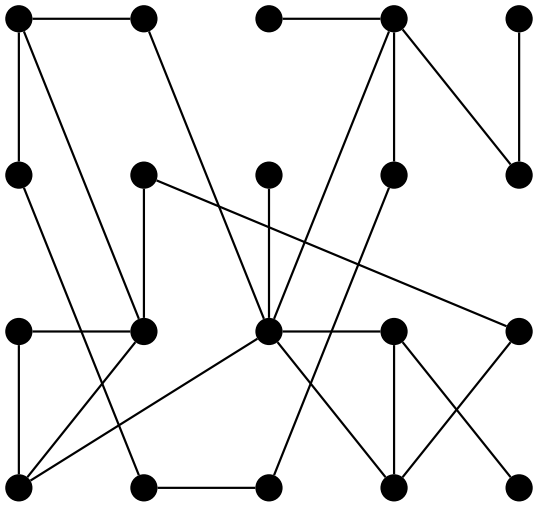
- Under $\mathrm{W}[1] \neq \mathrm{FPT}$, Clique cannot be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time.
- Best known algorithm: $\mathcal{O}^\star(n^{0.492k})$
- Can we prove that the linear dependence on k is necessary?

Theorem (Chen, Huang, Kanj, Xia)

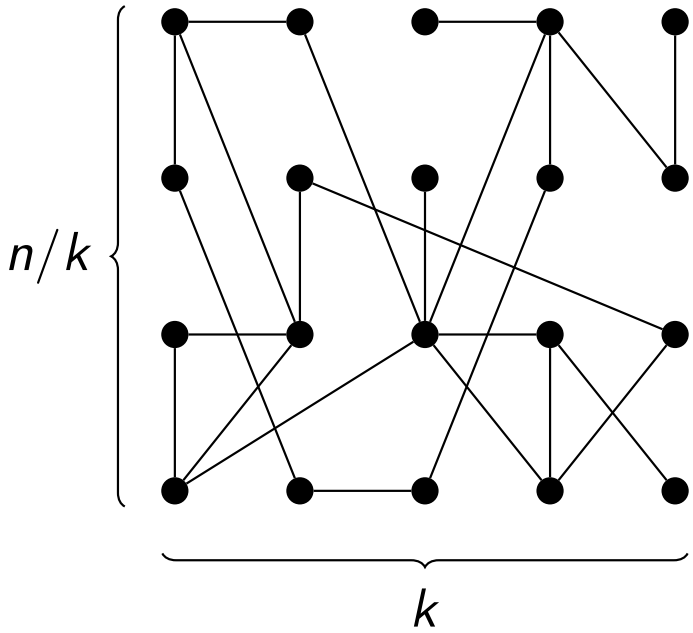
Unless ETH fails, Clique cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.

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

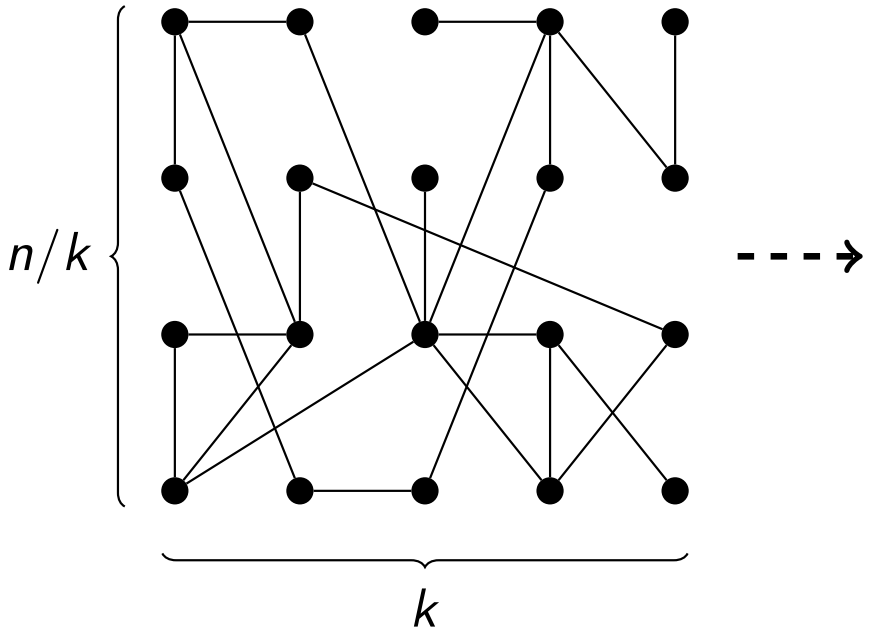
Hardness for Clique



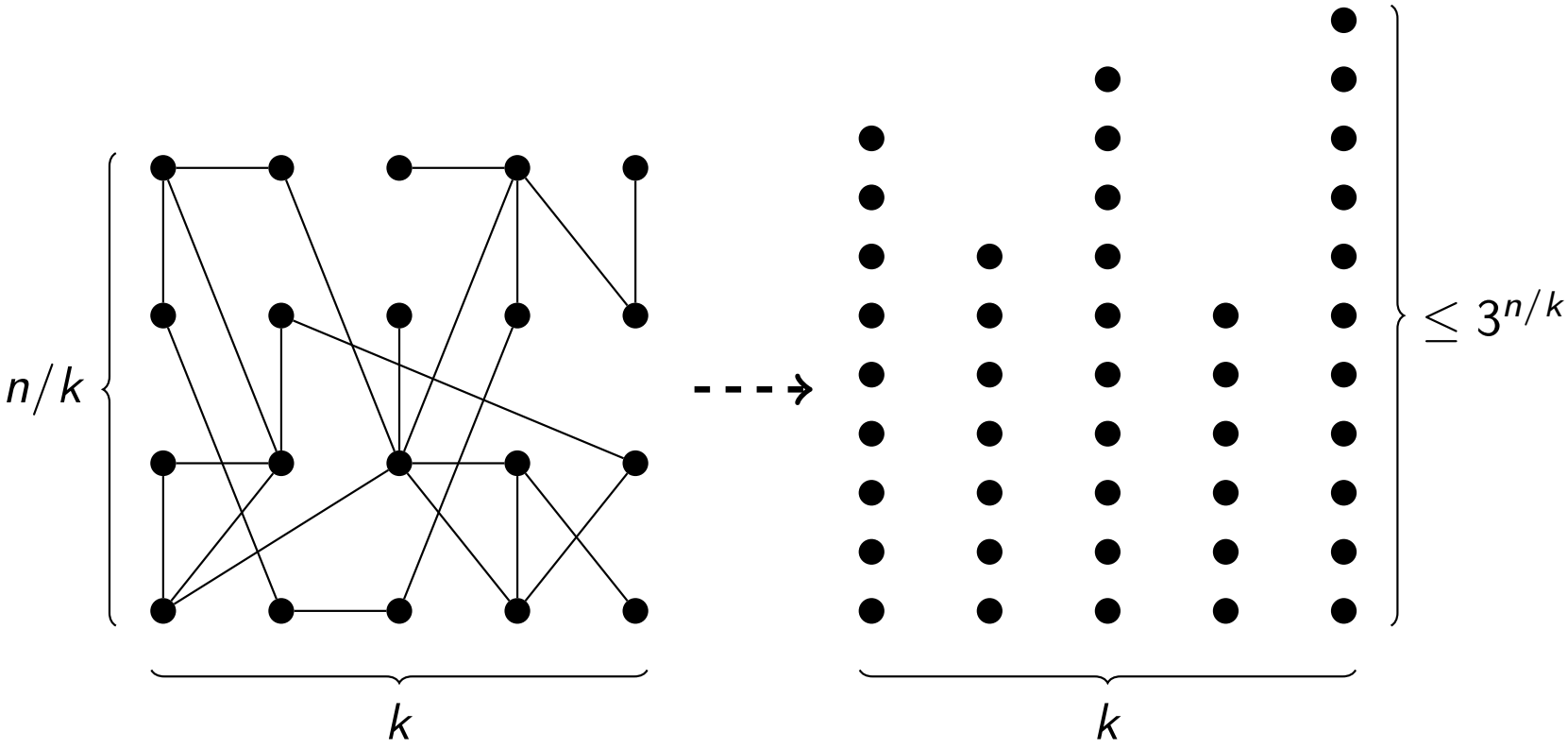
Hardness for Clique



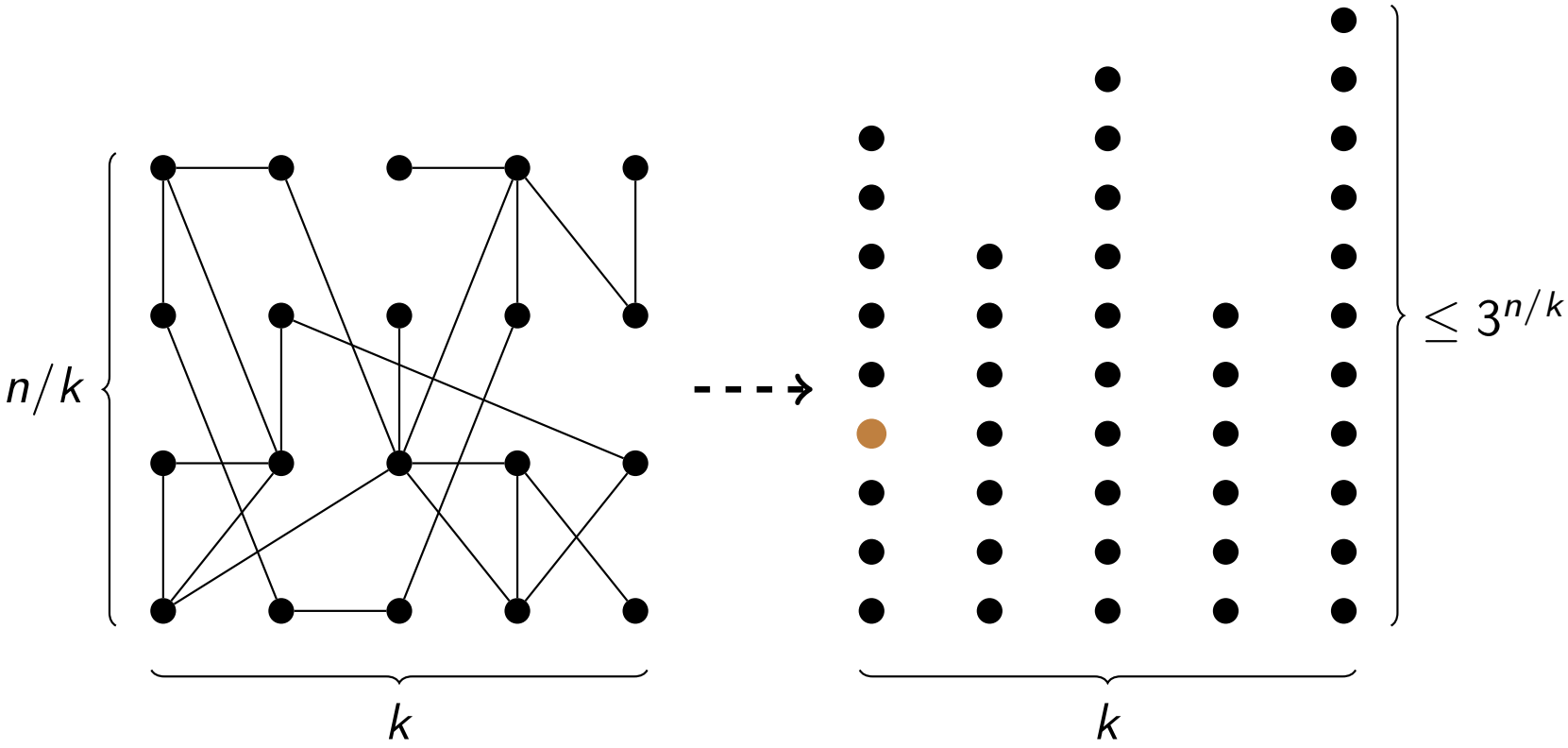
Hardness for Clique



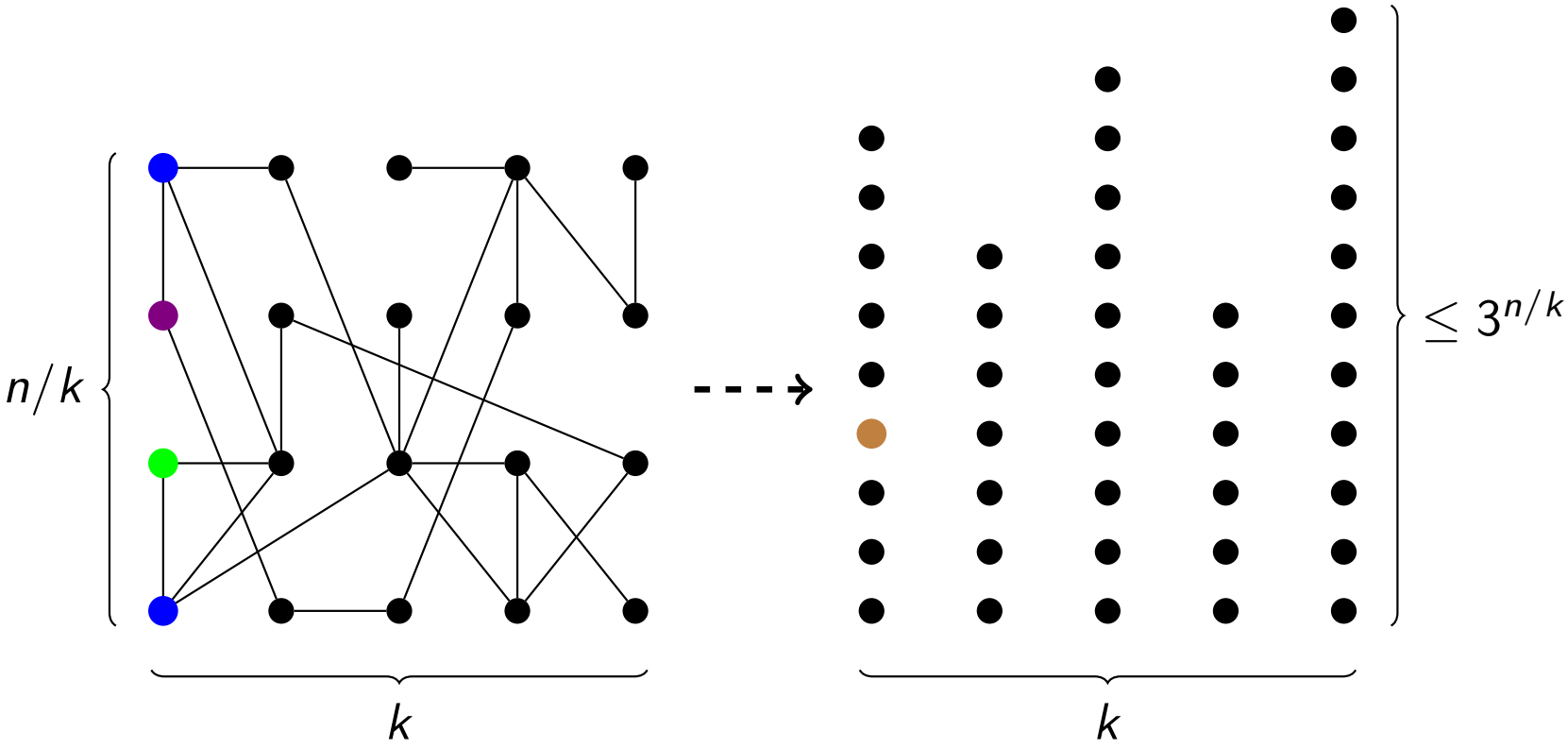
Hardness for Clique



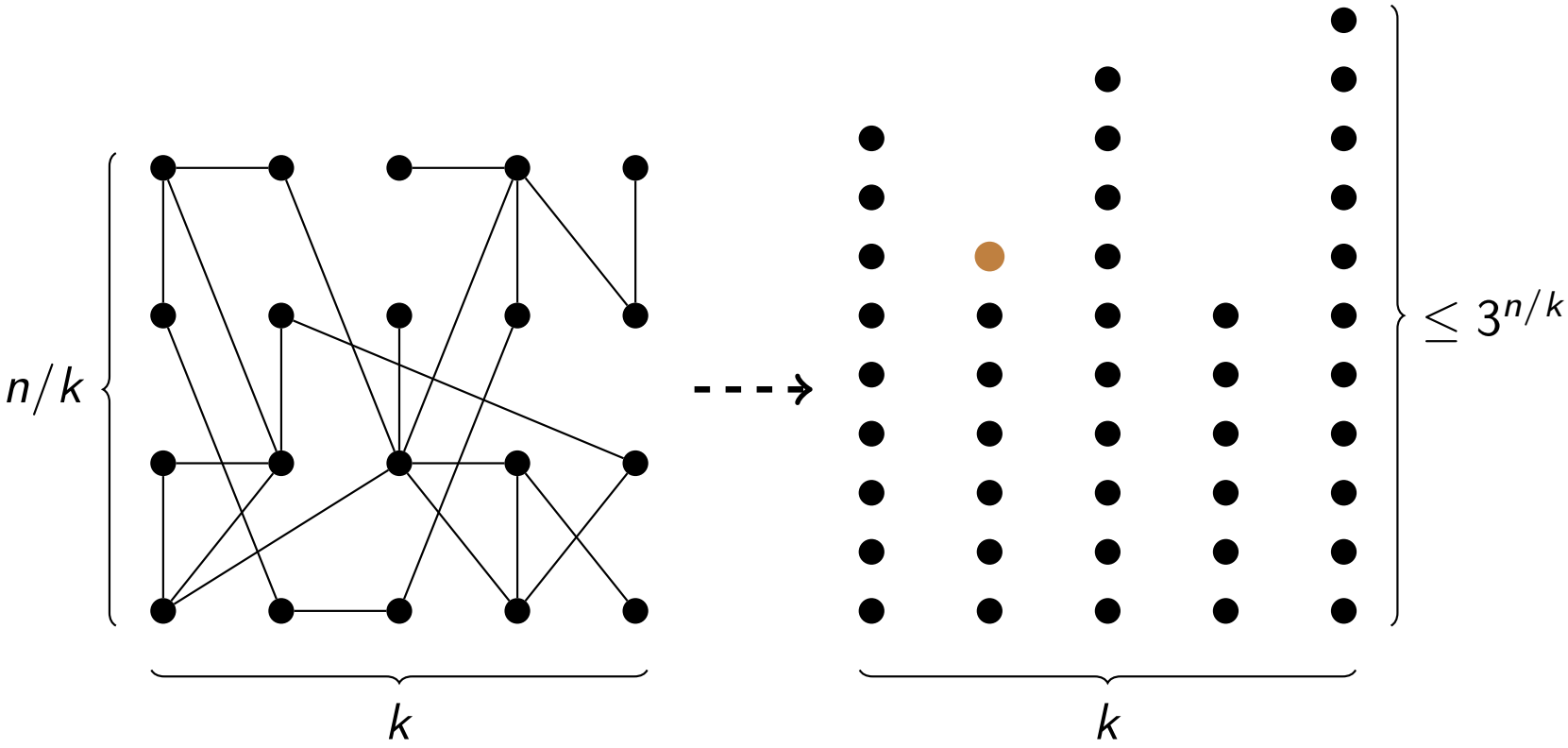
Hardness for Clique



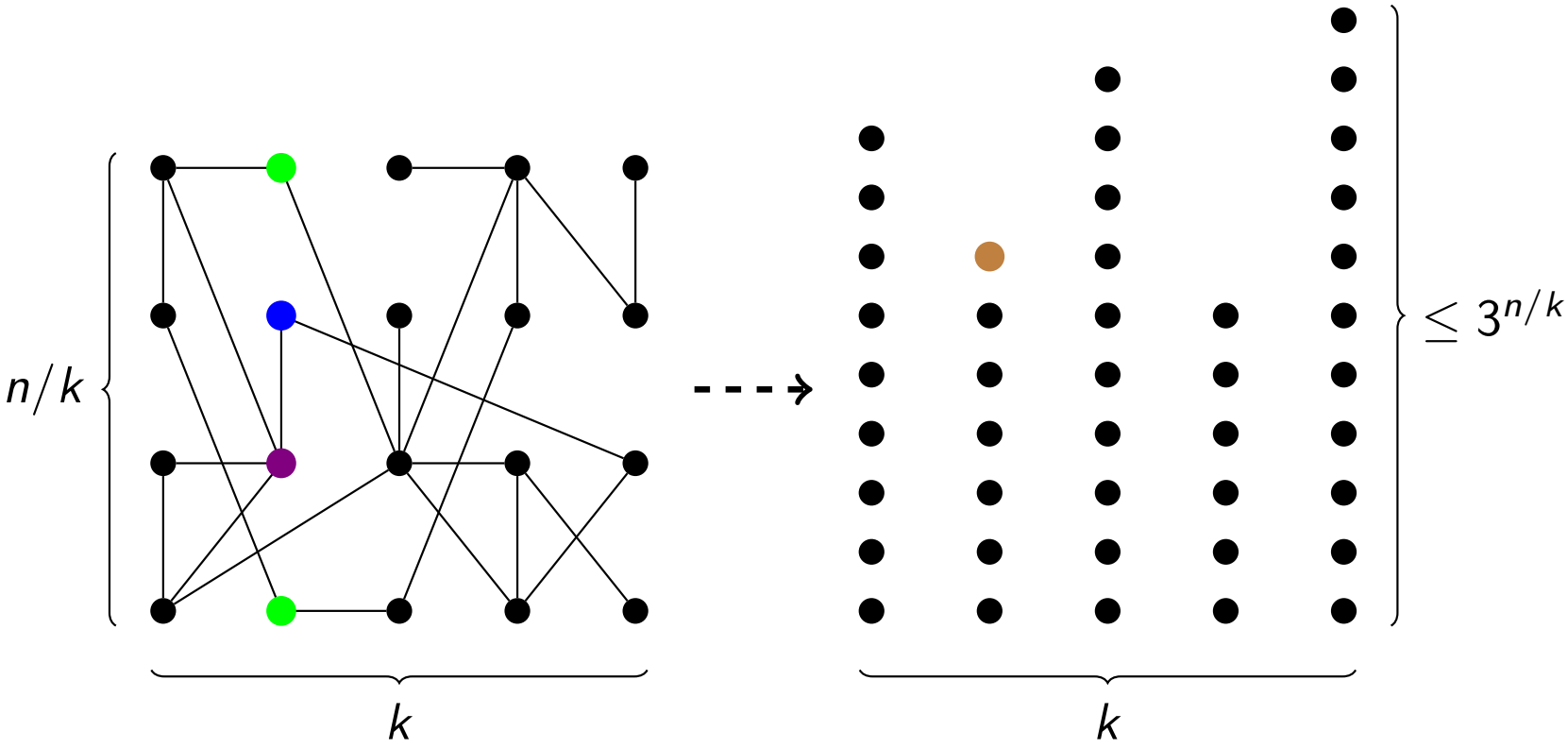
Hardness for Clique



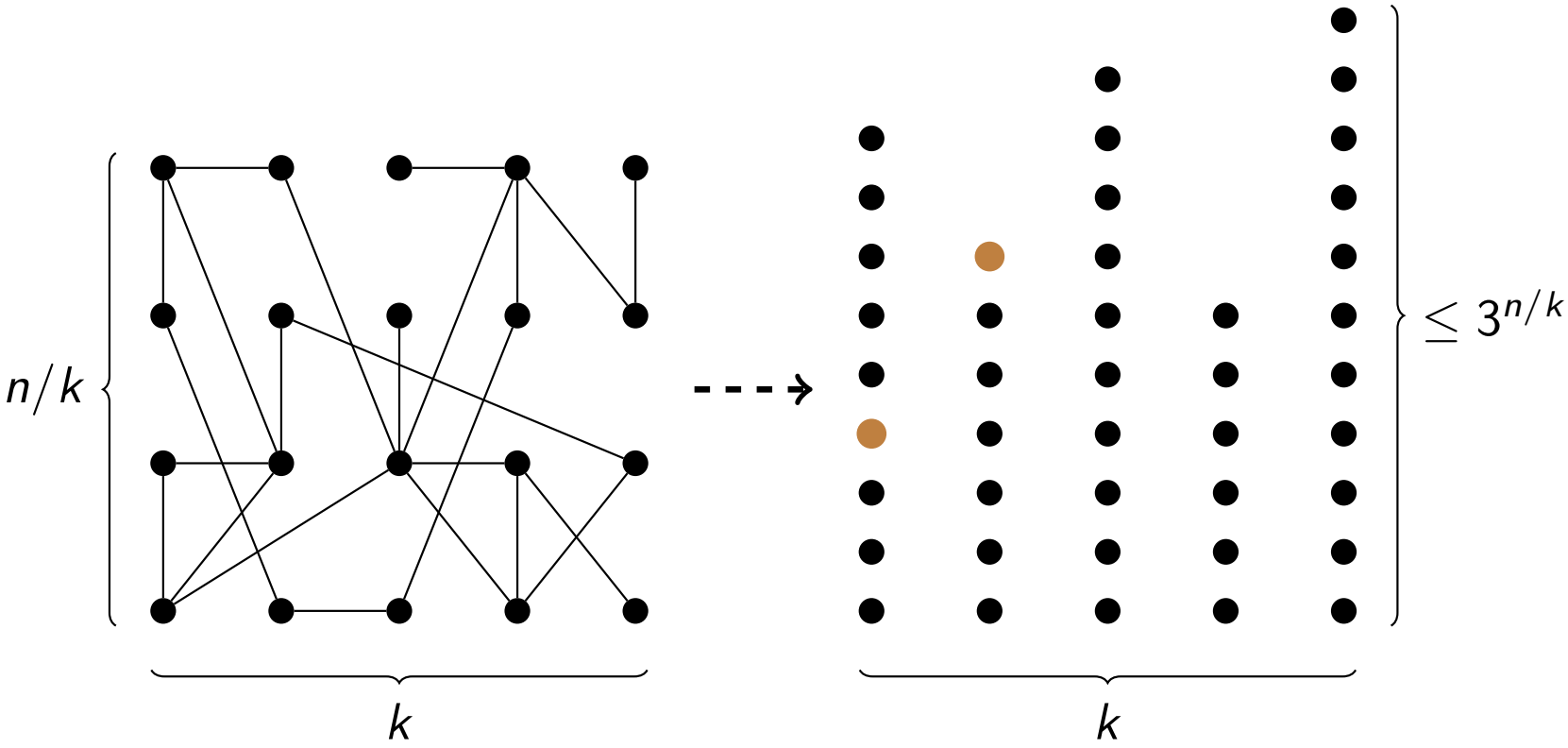
Hardness for Clique



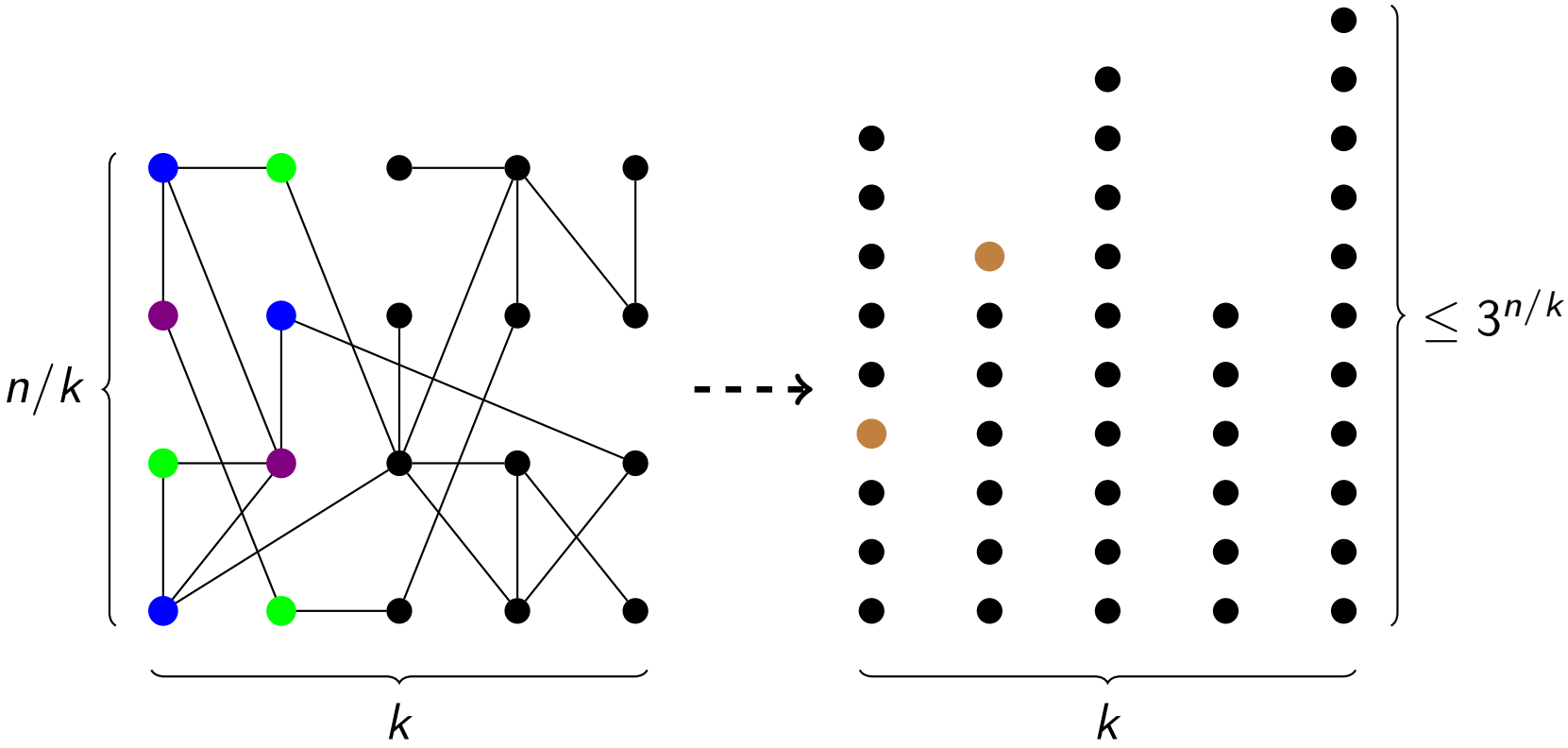
Hardness for Clique



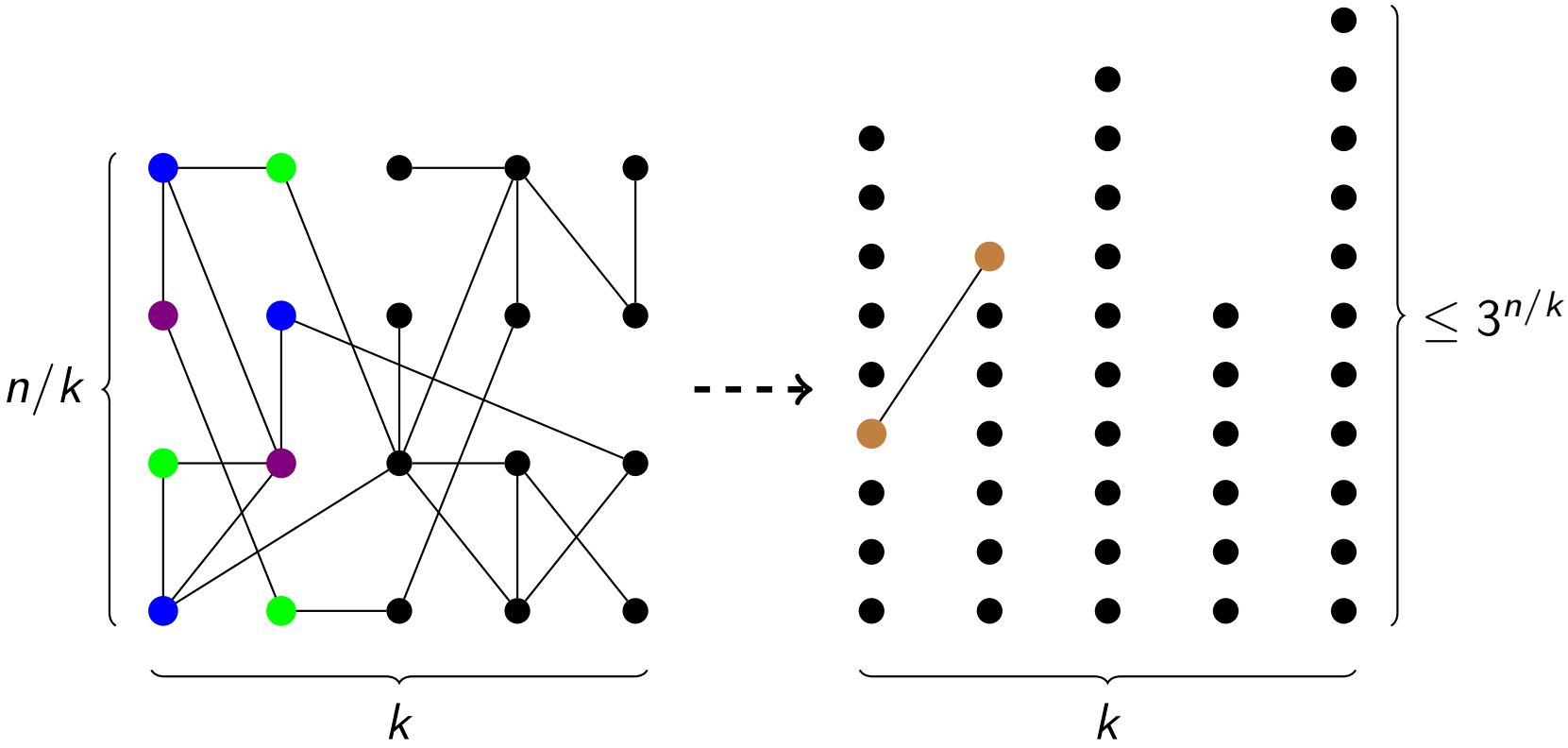
Hardness for Clique



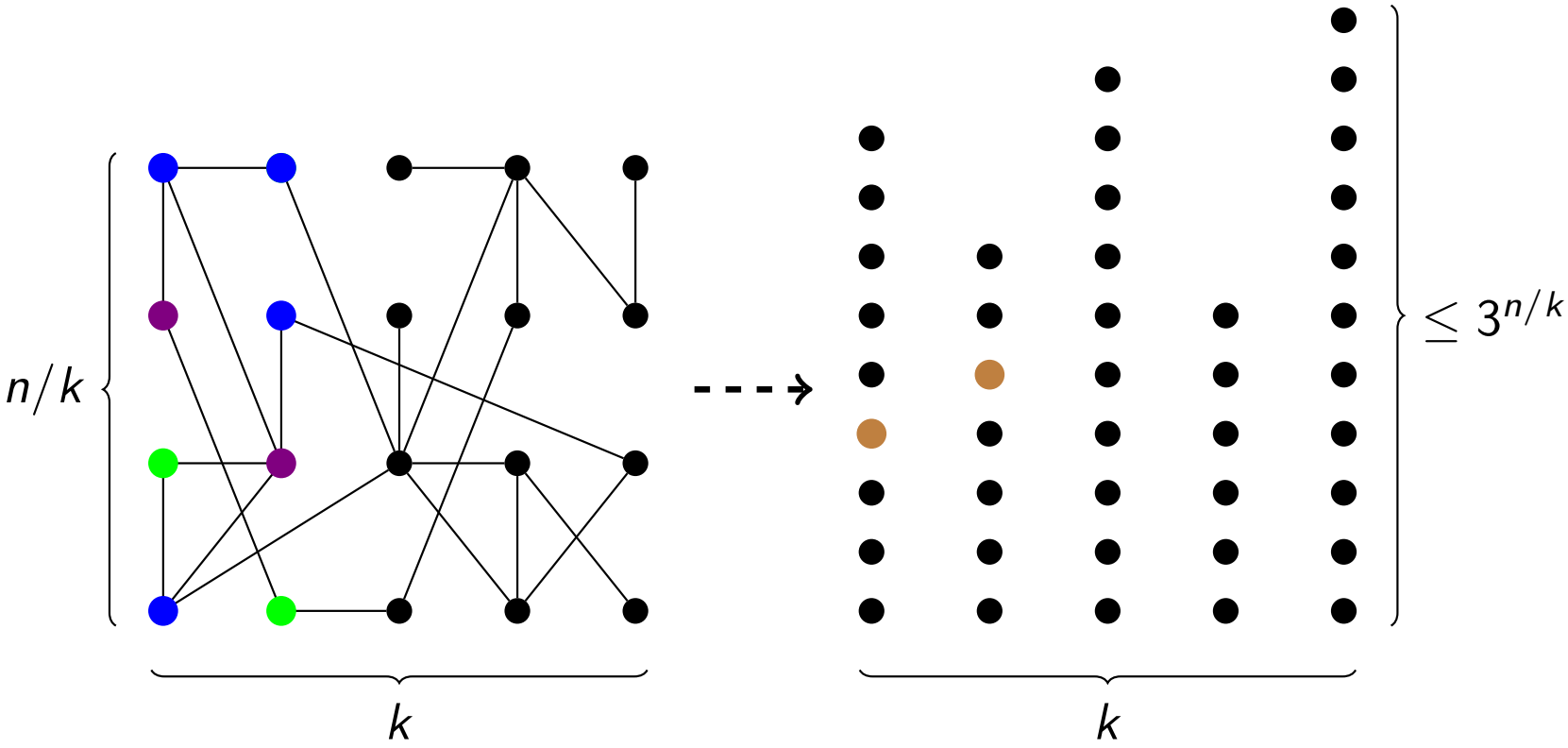
Hardness for Clique



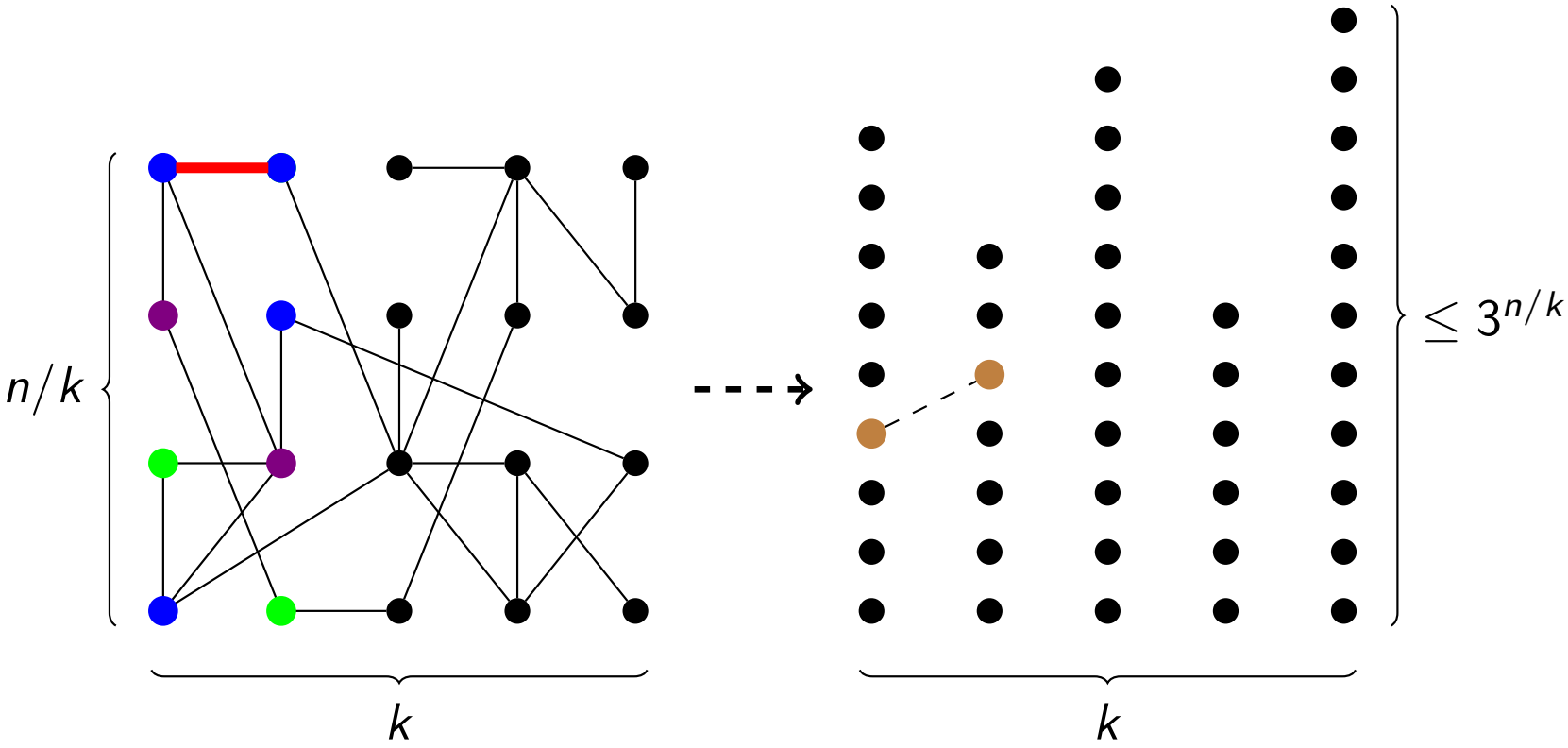
Hardness for Clique



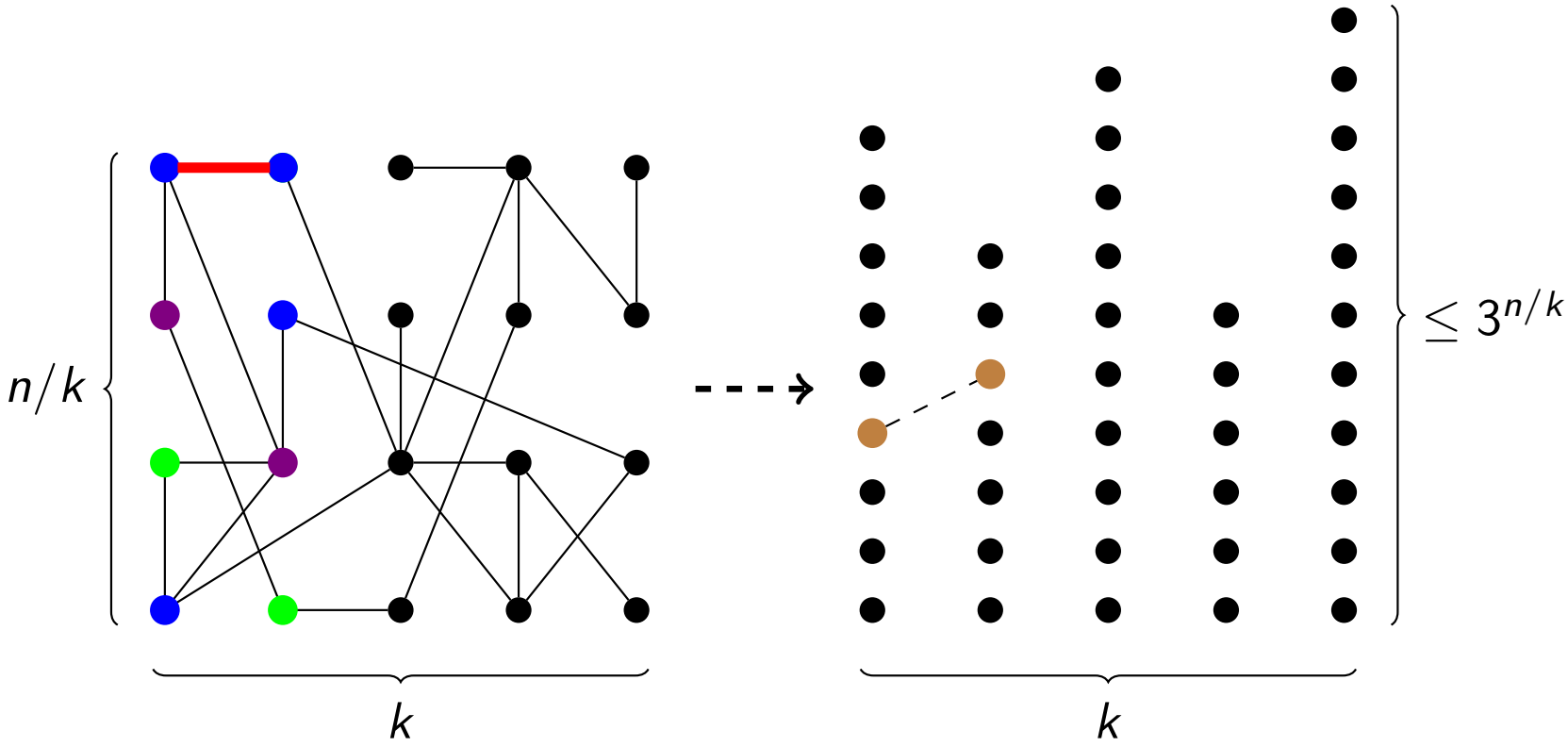
Hardness for Clique



Hardness for Clique



Hardness for Clique



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

Hardness for Clique

Theorem (Chen, Huang, Kanj, Xia)

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

Proof continued:

- Constructed graph has $N \leq k \cdot 3^{n/k}$ vertices.

Hardness for Clique

Theorem (Chen, Huang, Kanj, Xia)

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

Proof continued:

- Constructed graph has $N \leq k \cdot 3^{n/k}$ vertices.
- Lets try $k = \log n$.

Hardness for Clique

Theorem (Chen, Huang, Kanj, Xia)

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

Proof continued:

- Constructed graph has $N \leq k \cdot 3^{n/k}$ vertices.
- Lets try $k = \log n$.
- $2^k \cdot N^{o(k)} = n \cdot (\log n)^{o(\log n)} \cdot 3^{n \cdot o(\log n)/\log n} = \mathcal{O}^\star(2^{o(n)})$.

Hardness for Clique

Theorem (Chen, Huang, Kanj, Xia)

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

Proof continued:

- Constructed graph has $N \leq k \cdot 3^{n/k}$ vertices.
- Lets try $k = \log n$.
- $2^k \cdot N^{o(k)} = n \cdot (\log n)^{o(\log n)} \cdot 3^{n \cdot o(\log n)/\log n} = \mathcal{O}^\star(2^{o(n)})$.
- Similarly $k = \log \log n$ implies no $2^{2^k} \cdot N^{o(k)}$ time algorithm.

Hardness for Clique

Theorem (Chen, Huang, Kanj, Xia)

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

Proof continued:

- Constructed graph has $N \leq k \cdot 3^{n/k}$ vertices.
- Lets try $k = \log n$.
- $2^k \cdot N^{o(k)} = n \cdot (\log n)^{o(\log n)} \cdot 3^{n \cdot o(\log n)/\log n} = \mathcal{O}^\star(2^{o(n)})$.
- 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).

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.
- **Recent discovery**: For planar problems, a typical behaviour is:
  - the existence of an $\mathcal{O}(n^{\mathcal{O}(\sqrt{k})})$ algorithm, and
  - the nonexistence of an $f(k) \cdot n^{o(\sqrt{k})}$ algorithm under ETH.

Corollaries

- By reductions from Clique, one can prove lower bounds on the running times of XP algorithms.
- **Recent discovery**: For planar problems, a typical behaviour is:
  - the existence of an $\mathcal{O}(n^{\mathcal{O}(\sqrt{k})})$ algorithm, and
  - the nonexistence of an $f(k) \cdot n^{o(\sqrt{k})}$ algorithm under ETH.
- **Example**: Planar d-Scattered Set — given an edge-weighted planar graph G, a real d and an integer k, verify whether there are k points in pairwise distance d from each other.
  (Marx, P)

Strong ETH

- Under ETH we can estimate the asymptotics of the behaviour in the exponent.

Strong ETH

- Under ETH we can estimate the asymptotics of the behaviour in the exponent.
- Under SETH we can precise pinpoint the base of the exponent.

Strong ETH

- Under ETH we can estimate the asymptotics of the behaviour in the exponent.
- Under SETH we can precise pinpoint the base of the exponent.
- For lower bounds under ETH, we cared only about the *asymptotics* of the parameter blow-up.

Strong ETH

- Under ETH we can estimate the asymptotics of the behaviour in the exponent.
- Under SETH we can precise pinpoint the base of the exponent.
- For lower bounds under ETH, we cared only about the *asymptotics* of the parameter blow-up.
- Under SETH, we need to care *exactly* how the parameter is transformed.

Strong ETH

- Under ETH we can estimate the asymptotics of the behaviour in the exponent.
- Under SETH we can precise pinpoint the base of the exponent.
- For lower bounds under ETH, we cared only about the *asymptotics* of the parameter blow-up.
- Under SETH, we need to care *exactly* how the parameter is transformed.
- Therefore, lower bounds under SETH are much more delicate and much scarcer.

Strong ETH

- Under ETH we can estimate the asymptotics of the behaviour in the exponent.
- Under SETH we can precise pinpoint the base of the exponent.
- For lower bounds under ETH, we cared only about the *asymptotics* of the parameter blow-up.
- Under SETH, we need to care *exactly* how the parameter is transformed.
- Therefore, lower bounds under SETH are much more delicate and much scarcer.
- Probably more during Ryan's talk in the morning;
  now a quick parameterized perspective.

SETH and treewidth DPs

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

SETH and treewidth DPs

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

SETH and treewidth DPs

- Many standard problems can be solved in time $\mathcal{O}^\star(c^t)$, where c some constant and t is the width of a given tree decomposition of the graph.
  - VC and IS in $\mathcal{O}^\star(2^t)$, DS and OCT in $\mathcal{O}^\star(3^t)$.
  - Cut&Count (Cygan, Nederlof, Pilipczuk, P, van Rooij, Wojtaszczyk): Steiner Tree, CVC, and FVS in $\mathcal{O}^\star(3^t)$.

SETH and treewidth DPs

- Many standard problems can be solved in time $\mathcal{O}^\star(c^t)$, where c some constant and t is the width of a given tree decomposition of the graph.
  - VC and IS in $\mathcal{O}^\star(2^t)$, DS and OCT in $\mathcal{O}^\star(3^t)$.
  - Cut&Count (Cygan, Nederlof, Pilipczuk, P, van Rooij, Wojtaszczyk): Steiner Tree, CVC, and FVS in $\mathcal{O}^\star(3^t)$.
- Space of states is formed by all relevant interactions between the solution and the bag.

SETH and treewidth DPs

- Many standard problems can be solved in time $\mathcal{O}^\star(c^t)$, where c some constant and t is the width of a given tree decomposition of the graph.
  - VC and IS in $\mathcal{O}^\star(2^t)$, DS and OCT in $\mathcal{O}^\star(3^t)$.
  - Cut&Count (Cygan, Nederlof, Pilipczuk, P, van Rooij, Wojtaszczyk): Steiner Tree, CVC, and FVS in $\mathcal{O}^\star(3^t)$.
- Space of states is formed by all relevant interactions between the solution and the bag.
- These results are tight.

SETH and treewidth DPs

Theorem (LMS+CNPPvRW)

Assume that CNF-SAT cannot be solved in time $\mathcal{O}^\star(c^n)$ for any $c < 2$. Then for every $\varepsilon > 0$ the following holds (p/t is the width of a given path/tree decomposition of the input graph):

- 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 $\mathcal{O}^\star((3-\varepsilon)^p)$;
- Steiner Tree cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$;
- Feedback Vertex Set cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$;
- Connected Vertex Cover cannot be solved in time $\mathcal{O}^\star((3-\varepsilon)^p)$.

SETH and treewidth DPs

Theorem (Cygan, Kratsch, Nederlof)

Hamiltonian Path can be solved in time $\mathcal{O}^\star((2+\sqrt{2})^p)$, but the existence of an algorithm with running time $\mathcal{O}^\star((2+\sqrt{2}-\varepsilon)^p)$ would yield an algorithm for CNF-SAT with running time $\mathcal{O}^\star(c^n)$ for some $c < 2$.

SETH and treewidth DPs

Theorem (Cygan, Kratsch, Nederlof)

Hamiltonian Path can be solved in time $\mathcal{O}^\star((2+\sqrt{2})^p)$, but the existence of an algorithm with running time $\mathcal{O}^\star((2+\sqrt{2}-\varepsilon)^p)$ would yield an algorithm for CNF-SAT with running time $\mathcal{O}^\star(c^n)$ for some $c < 2$.

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

SETH and covering problems

Set Cover

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

SETH and covering problems

Set Cover

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

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

SETH and covering problems

Set Cover

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

- Denote $n = |U|$ and $m = |\mathcal{F}|$.
- Brute-force $\mathcal{O}^\star(2^m)$ algorithm.

SETH and covering problems

Set Cover

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

- Denote $n = |U|$ and $m = |\mathcal{F}|$.
- Brute-force $\mathcal{O}^\star(2^m)$ algorithm.
- Covering DP over subsets of $U \Rightarrow \mathcal{O}^\star(2^n)$ algorithm.

SETH and covering problems

Set Cover

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

- Denote $n = |U|$ and $m = |\mathcal{F}|$.
- Brute-force $\mathcal{O}^\star(2^m)$ algorithm.
- Covering DP over subsets of $U \Rightarrow \mathcal{O}^\star(2^n)$ algorithm.
- Could any of these be improved?

SETH and covering problems

- Results of Cygan, Dell, Lokshtanov, Marx, Nederlof, Okamoto, Paturi, Saurabh, and Wahlström.

SETH and covering problems

- Results of Cygan, Dell, Lokshtanov, Marx, Nederlof, Okamoto, Paturi, Saurabh, and Wahlström.
- Under SETH, there is no $\mathcal{O}^\star(c^m)$ algorithm for Set Cover for any $c < 2$ (sort-of-equivalent to SETH).

SETH and covering problems

- Results of Cygan, Dell, Lokshtanov, Marx, Nederlof, Okamoto, Paturi, Saurabh, and Wahlström.
- Under SETH, there is no $\mathcal{O}^\star(c^m)$ algorithm for Set Cover for any $c < 2$ (sort-of-equivalent to SETH).
- Breaking 2 in the base of the exponent for many covering problems is *equivalent* to breaking it for Set Cover/n.

SETH and covering problems

- Results of Cygan, Dell, Lokshtanov, Marx, Nederlof, Okamoto, Paturi, Saurabh, and Wahlström.
- Under SETH, there is no $\mathcal{O}^\star(c^m)$ algorithm for Set Cover for any $c < 2$ (sort-of-equivalent to SETH).
- Breaking 2 in the base of the exponent for many covering problems is *equivalent* to breaking it for Set Cover/n.
  - Steiner Tree, Connected Vertex Cover, Set Partitioning.

SETH and covering problems

- Results of Cygan, Dell, Lokshtanov, Marx, Nederlof, Okamoto, Paturi, Saurabh, and Wahlström.
- Under SETH, there is no $\mathcal{O}^\star(c^m)$ algorithm for Set Cover for any $c < 2$ (sort-of-equivalent to SETH).
- Breaking 2 in the base of the exponent for many covering problems is *equivalent* to breaking it for Set Cover/n.
  - Steiner Tree, Connected Vertex Cover, Set Partitioning.
- Fundamental link between Set Cover/n and Set Cover/m is still not discovered.

SETH and covering problems

- Results of Cygan, Dell, Lokshtanov, Marx, Nederlof, Okamoto, Paturi, Saurabh, and Wahlström.
- Under SETH, there is no $\mathcal{O}^\star(c^m)$ algorithm for Set Cover for any $c < 2$ (sort-of-equivalent to SETH).
- Breaking 2 in the base of the exponent for many covering problems is *equivalent* to breaking it for Set Cover/n.
  - Steiner Tree, Connected Vertex Cover, Set Partitioning.
- Fundamental link between Set Cover/n and Set Cover/m is still not discovered.

Set Cover Conjecture

Let λ_q be the infinimum 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 $\mathcal{O}^\star((2-\varepsilon)^n)$ for any $\varepsilon > 0$.

Conclusions

- ETH allows us to estimate the asymptotics of the behaviour in the exponents.

Conclusions

- ETH allows us to estimate the asymptotics of the behaviour in the exponents.
- SETH gives a precise bound on the base of the exponent, but is less plausible and less applicable.

Conclusions

- ETH allows us to estimate the asymptotics of the behaviour in the exponents.
- SETH gives a precise bound on the base of the exponent, but is less plausible and less applicable.
- ETH and SETH are robust assumptions, under which for many problems we can pinpoint the precise influence of a parameter on the complexity of the problem.

Conclusions

- ETH allows us to estimate the asymptotics of the behaviour in the exponents.
- SETH gives a precise bound on the base of the exponent, but is less plausible and less applicable.
- ETH and SETH are robust assumptions, under which for many problems we can pinpoint the precise influence of a parameter on the complexity of the problem.
- **Optimality program**: Finding matching lower and upper bounds on the parameterized complexity of various problems.

Commercial break!

This, and many more in the new book

Parameterized algorithms

by Cygan, Fomin, Kowalik,
Lokshtanov, Marx, Pilipczuk, P., and Saurabh.
Out in early 2015.