Exponential Time Hypothesis
part 1

Marek Cygan

Institute of Informatics
University of Warsaw

19th August 2014, Będlewo

Introduction

- We already know that W[1]-hard problems are different than the ones which are FPT.
- However, within the FPT class we have different dependency on the parameter in best known algorithms: $2^{\mathcal{O}(\sqrt{k})}, 2^{\mathcal{O}(k)}, 2^{\mathcal{O}(k \log k)}, 2^{2^{\mathcal{O}(k)}}$.
- Similarly for W[1]-hard problems: $f(k)n^{\mathcal{O}(k)}, f(k)n^{\mathcal{O}(\sqrt{k})}, f(k)n^{\mathcal{O}(\log \log k)}$.

Introduction

- It seems that there are some inner hierarchies of subclasses, corresponding to different classes of running times.
- To separate these classes, we need a methodology for proving lower bounds.
- Ideally, if $2^{f(k)}n^{\mathcal{O}(1)}$ is known, we want $2^{o(f(k))}n^{\mathcal{O}(1)}$ lower bound.
- Dream, for $2^{f(k)}n^{\mathcal{O}(1)}$ algorithm, $2^{(1-\epsilon)f(k)}n^{\mathcal{O}(1)}$ lower bound.
- $P \neq NP$ or even $FPT \neq W[1]$ is (probably) not strong enough to serve as a source of hardness.

Introduction

- It seems that there are some inner hierarchies of subclasses, corresponding to different classes of running times.
- To separate these classes, we need a methodology for proving lower bounds.
- Ideally, if $2^{f(k)}n^{\mathcal{O}(1)}$ is known, we want $2^{o(f(k))}n^{\mathcal{O}(1)}$ lower bound.
- Dream, for $2^{f(k)}n^{\mathcal{O}(1)}$ algorithm, $2^{(1-\epsilon)f(k)}n^{\mathcal{O}(1)}$ lower bound.
- $P \neq NP$ or even $FPT \neq W[1]$ is (probably) not strong enough to serve as a source of hardness.

If we are pessimistic enough, sometimes dreams come true.

What this lecture is about?

Exponential Time Hypothesis (ETH)

Informally: 3-CNF SAT cannot be solved in time subexponential in the number of variables.

What this lecture is about?

Exponential Time Hypothesis (ETH)

Informally: 3-CNF SAT cannot be solved in time subexponential in the number of variables.

Outline:

1. Formal definition of ETH.
2. Sparsification lemma.
3. ETH and classic NP-hardness reductions: $2^{o(n)}$, $2^{o(k)}n^{O(1)}$.
4. ETH and planar problems: $2^{o(\sqrt{n})}$, $2^{o(\sqrt{k})}n^{\mathcal{O}(1)}$.
5. ETH and W[1]-hard problems: $f(k)n^{o(k)}$.
6. Glimpse on less common lower bounds.

CNF SAT

A formula on variables $x_1, \ldots, x_n$ in *conjunctive normal form* (CNF):
$\varphi = C_1 \wedge C_2 \wedge \ldots C_m$,
where $C_i = \ell_1^i \vee \ell_2^i \vee \ldots \vee \ell_r^i$
and each ℓ_j^i is a literal, i.e., variable or its negation.

- Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4) \wedge (\neg x_2)$
- If each clause has at most q literals, we have q-CNF formula.
- CNF-SAT: given a CNF formula decide if it is satisfiable.
- q-SAT: same for q-CNF.

CNF SAT

- 3-SAT is NP-complete, hence no poly time algorithm,
- exhaustive search gives $\mathcal{O}^*(2^n)$ time algorithm for CNF-SAT,

  $\mathcal{O}^*()$ notation suppresses polynomial factors
- q-SAT is solvable in $\mathcal{O}^*(2^{\gamma_q n})$ where $\gamma_q = 1 - \Theta(1/q)$,
- current best algorithm for 3-SAT: $\mathcal{O}(2^{0.386n})$.

Current status exposes two barriers hard to break:

1. $2^{o(n)}$ algorithm for 3-SAT,
2. $\mathcal{O}^*((2-\epsilon)^n)$ algorithm for general CNF-SAT for some $\epsilon > 0$.

ETH and SETH

Definition

For $q \geqslant 3$ let δ_q be infimum of the set
$\{c : \text{there is } \mathcal{O}^*(2^{cn}) \text{ algorithm for q-SAT}\}$

ETH and SETH

Definition

For $q \geqslant 3$ let δ_q be infimum of the set
$\{c : \text{there is } \mathcal{O}^*(2^{cn}) \text{ algorithm for q-SAT}\}$

Conjectures

- Exponential Time Hypothesis (ETH): $\delta_3 > 0$.
- Strong Exponential Time Hypothesis (SETH): $\lim_{q\to\infty} \delta_q = 1$.

ETH and SETH

Definition

For $q \geqslant 3$ let δ_q be infimum of the set
$\{c : \text{there is } \mathcal{O}^*(2^{cn}) \text{ algorithm for q-SAT}\}$

Conjectures

- Exponential Time Hypothesis (ETH): $\delta_3 > 0$.
- Strong Exponential Time Hypothesis (SETH): $\lim_{q\to\infty} \delta_q = 1$.

Why not the following?

- ETH: no $2^{o(n)}$ algorithm for 3-SAT,
- SETH: no $\mathcal{O}^*((2-\epsilon)^n)$ algorithm for CNF-SAT for any $\epsilon > 0$.

Randomized ETH

- The default version of ETH/SETH refutes existence of deterministic algorithms.

Randomized ETH

- The default version of ETH/SETH refutes existence of deterministic algorithms.
- One could define $\delta_q^* \leqslant \delta_q$ as infimum over randomized two-sided error algorithms.

Two-sided error algorithms

A randomized algorithm with two-sided error is an algorithm which for any YES-instance and any NO-instance returns an incorrect answer with prob. at most constant $p < 1/2$.

Randomized ETH

- The default version of ETH/SETH refutes existence of deterministic algorithms.
- One could define $\delta_q^* \leqslant \delta_q$ as infimum over randomized two-sided error algorithms.

Randomized versions

- ETH: $\delta_3^* > 0$.
- SETH: $\lim_{q\to\infty} \delta_q^* = 1$.

Randomized ETH

Randomized versions

- ETH: $\delta_3^* > 0$.
- SETH: $\lim_{q \to \infty} \delta_q^* = 1$.

- Reasonable from the point of view of research on satisfiability.
- But inconvenient from pragmatic point of view: dealing with error probability in reductions.
- Usually for a chain of reductions excluding some deterministic algorithm, the same chain works for randomized algorithms.
- During this school we assume deterministic versions (discussion on randomized versions in the book).

Plausibility

- ETH is generally considered as plausible.
- Many believe that SETH can be refuted anytime.
- Still, SETH based lower bounds make sense: existence of better algorithm would constitute a major breakthrough in the complexity of satisfiability.
- In other words, if you do not believe in a lower bound X which is SETH based, then refute SETH first before trying to disprove X.

Take-home message

SETH $\implies$ ETH $\implies$ W[1] $\neq$ FPT $\implies$ P $\neq$ NP

FPT $\neq$ W[1] $\implies$ no $\mathcal{O}^*(f(k))$
ETH $\implies$ no $\mathcal{O}^*(2^{o(f(k))})$
SETH $\implies$ no $\mathcal{O}^*(2^{(1-\epsilon)f(k)})$.

Sparsification Lemma

Consider a reduction

Given a 3-SAT formula φ with n vars and m clauses, produces equivalent instance of problem X of size $\mathcal{O}(n+m)$.

Sparsification Lemma

Consider a reduction

Given a 3-SAT formula φ with n vars and m clauses, produces equivalent instance of problem X of size $\mathcal{O}(n+m)$.

Lemma

Under ETH there is no $\mathcal{O}^*(2^{o(|I|^{1/3})})$ time algorithm for X.

Sparsification Lemma

Consider a reduction

Given a 3-SAT formula φ with n vars and m clauses, produces equivalent instance of problem X of size $\mathcal{O}(n+m)$.

Lemma

Under ETH there is no $\mathcal{O}^*(2^{o(|I|^{1/3})})$ time algorithm for X.

Proof by contradiction:

- Assume there is such an algorithm $\mathcal{A}$.
- For a given 3-SAT formula φ: (i) apply the reduction, (ii) use $\mathcal{A}$.
- Pipelined algorithm works in $2^{o(|I|^{1/3})} = 2^{o((n+m)^{1/3})} = 2^{o(n)}$.

Sparsification Lemma

Consider a reduction

Given a 3-SAT formula φ with n vars and m clauses, produces equivalent instance of problem X of size $\mathcal{O}(n+m)$.

Lemma

Under ETH there is no $\mathcal{O}^*(2^{o(|I|^{1/3})})$ time algorithm for X.

Proof by contradiction:

- Assume there is such an algorithm $\mathcal{A}$.
- For a given 3-SAT formula φ: (i) apply the reduction, (ii) use $\mathcal{A}$.
- Pipelined algorithm works in $2^{o(|I|^{1/3})} = 2^{o((n+m)^{1/3})} = 2^{o(n)}$.

If we knew that $m = \mathcal{O}(n)$, we would get a much stronger lower bound of $\mathcal{O}^*(2^{o(|I|)})$.

Sparsification Lemma

Sparsification Lemma informally

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

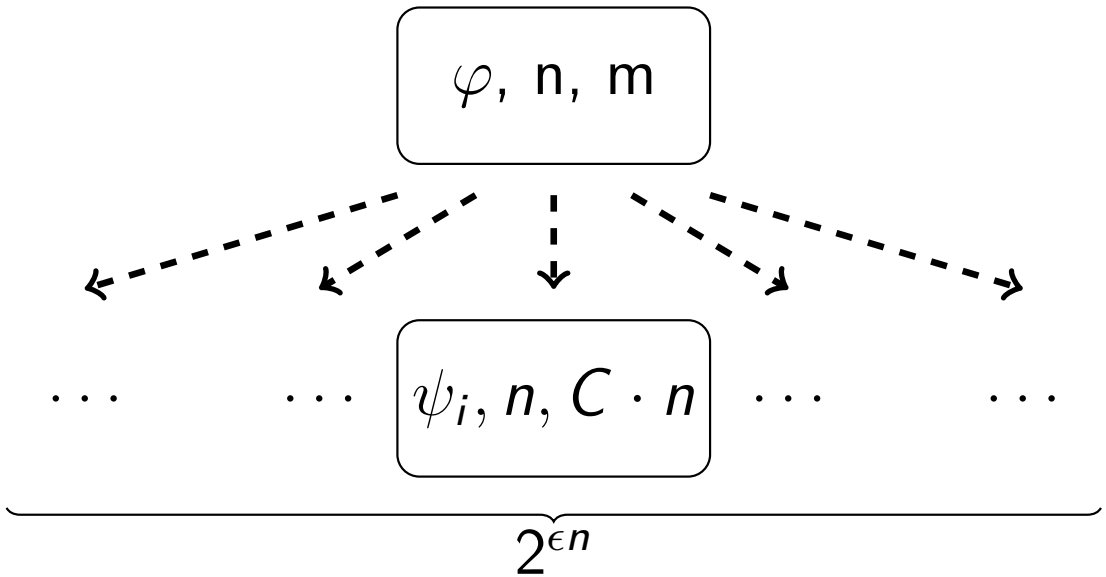
Sparsification Lemma

For any $q \geqslant 3$ and $\epsilon > 0$ there is a constant $C = C(q, \epsilon)$ such that any q-CNF formula φ can be expressed as $\varphi = \bigvee_{i=1}^{t} \psi_i$, where $t \leqslant 2^{\epsilon n}$ 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 $\mathcal{O}^*(2^{\epsilon n})$.

Sparsification Lemma

Sparsification Lemma

For any $q \geqslant 3$ and $\epsilon > 0$ there is a constant $C = C(q, \epsilon)$ such that any q-CNF formula φ can be expressed as $\varphi = \bigvee_{i=1}^{t} \psi_i$, where $t \leqslant 2^{\epsilon n}$ 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 $\mathcal{O}^*(2^{\epsilon n})$.



Sparsification Lemma

Theorem

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

Sparsification Lemma

Theorem

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

Proof by contradiction:

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

Sparsification Lemma

Theorem

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

Corollary

Unless ETH fails, there is no $2^{o(n+m)}$ time algorithm for 3-SAT.

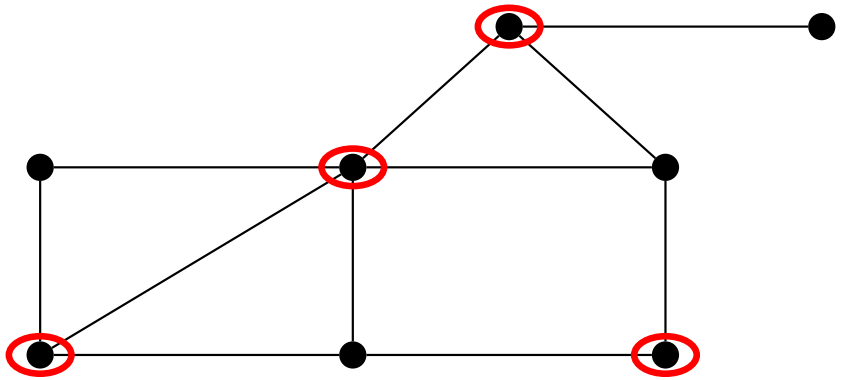
Combining ETH with classic NP-hardness reductions

Vertex Cover

Vertex Cover

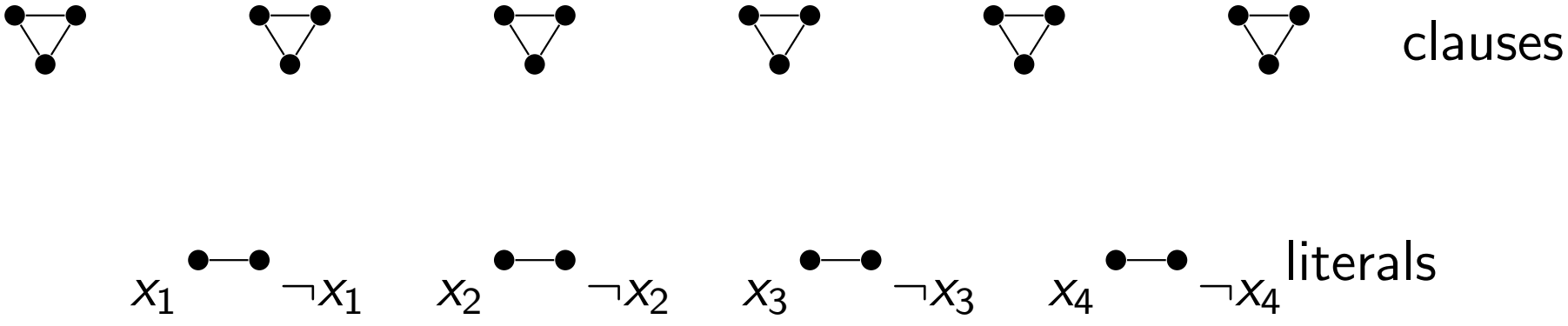
Input: undirected G, integer k

Question: is there a subset $X \subseteq V(G)$ of size at most k, such that for each $uv \in E(G)$ we have $\{u, v\} \cap X \neq \emptyset$.



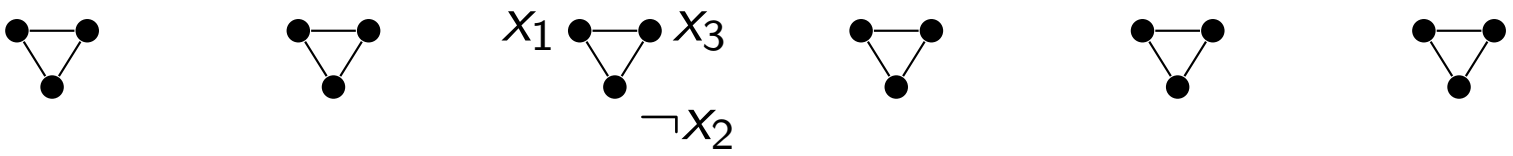
Vertex Cover

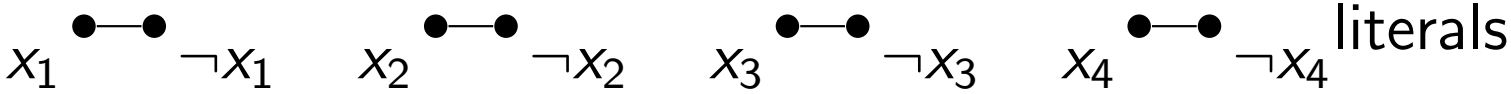
Let us inspect the standard reduction from 3-SAT to Vertex Cover.



Vertex Cover

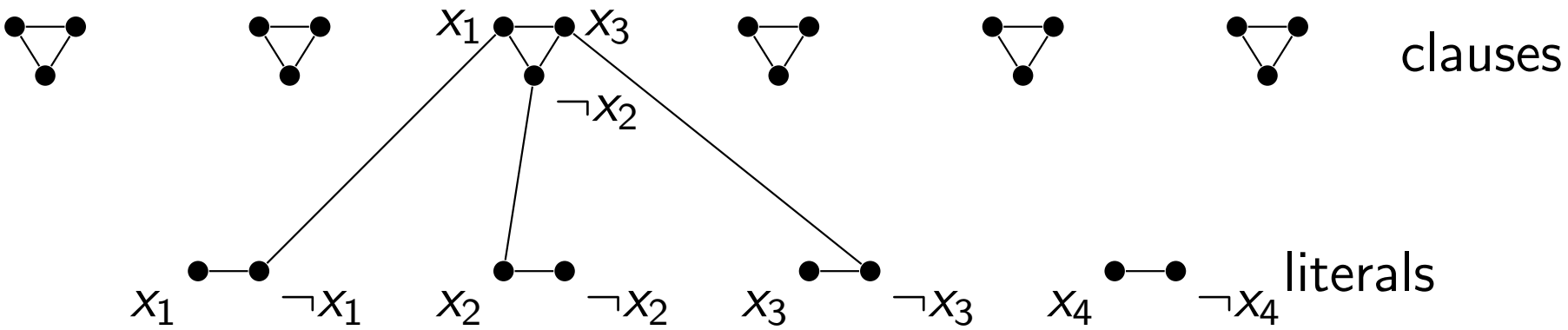
Let us inspect the standard reduction from 3-SAT to Vertex Cover.





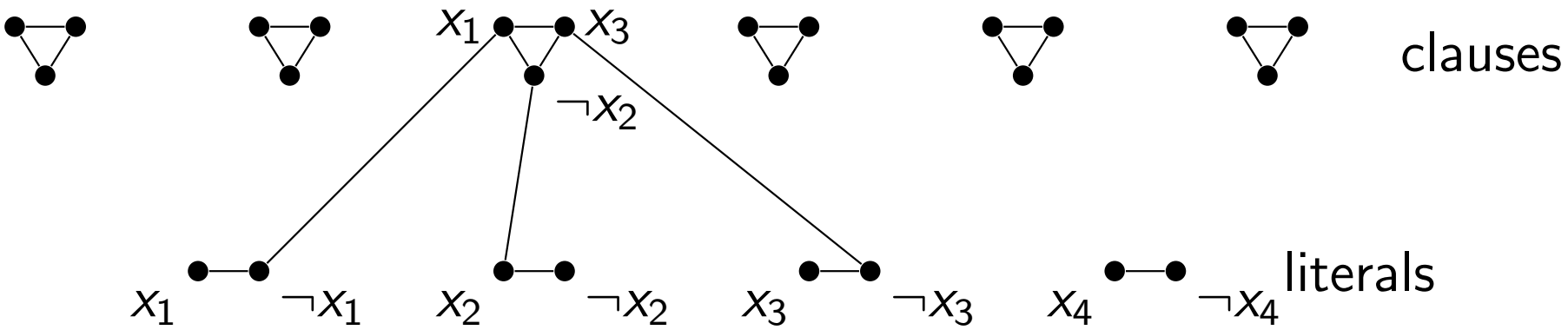
Vertex Cover

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



Vertex Cover

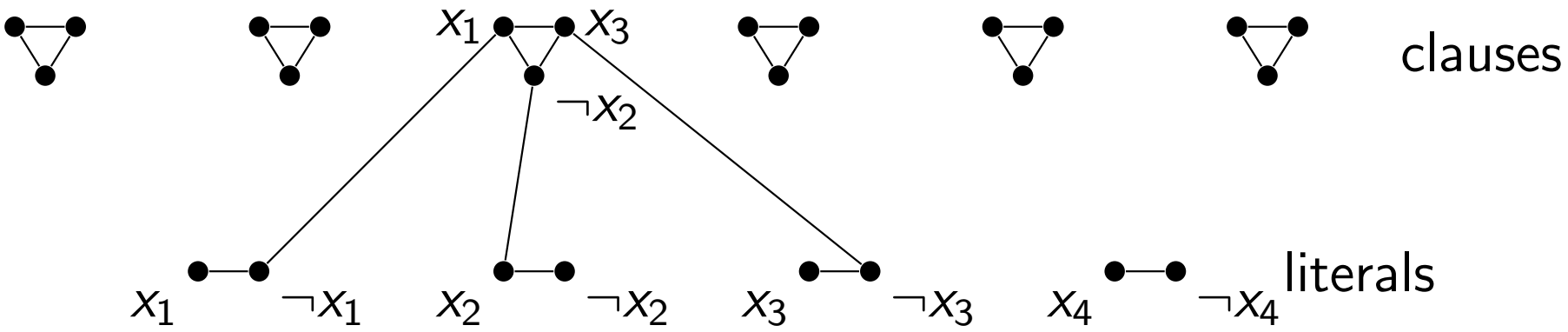
Let us inspect the standard reduction from 3-SAT to Vertex Cover.



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

Vertex Cover

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 $n + 2m$.

Created instance has $O(n + m)$ vertices and edges.

Lemma

Unless ETH fails, there is no $2^{o(n+m)}$ time algorithm for 3-SAT.

Reduction

Given a 3-SAT formula φ with n variables and m clauses one can in polynomial time create a graph with $O(n+m)$ vertices, which has a vertex cover of size $2n+m$ iff φ is satisfiable.

Lemma

Unless ETH fails, there is no $2^{o(n+m)}$ time algorithm for 3-SAT.

Reduction

Given a 3-SAT formula φ with n variables and m clauses one can in polynomial time create a graph with $O(n+m)$ vertices, which has a vertex cover of size $2n+m$ iff φ is satisfiable.

Corollary

Unless ETH fails, there is no $2^{o(|V(G)|+|E(G)|)}$ time algorithm VC.

Lemma

Unless ETH fails, there is no $2^{o(n+m)}$ time algorithm for 3-SAT.

Reduction

Given a 3-SAT formula φ with n variables and m clauses one can in polynomial time create a graph with $O(n+m)$ vertices, which has a vertex cover of size $2n+m$ iff φ is satisfiable.

Corollary

Unless ETH fails, there is no $2^{o(|V(G)|+|E(G)|)}$ time algorithm VC.

Corollary 2

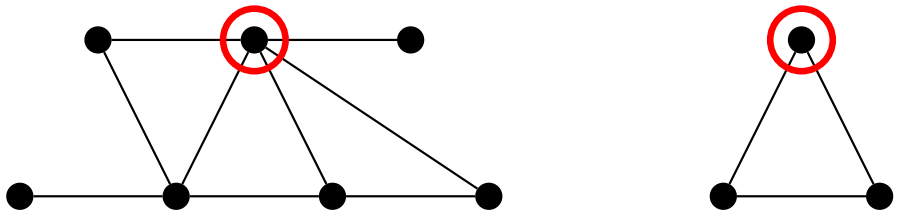
Unless ETH fails, there is no $\mathcal{O}^*(2^{o(k)})$ time FPT algorithm for VC.

FVS

Feedback Vertex Set (FVS)

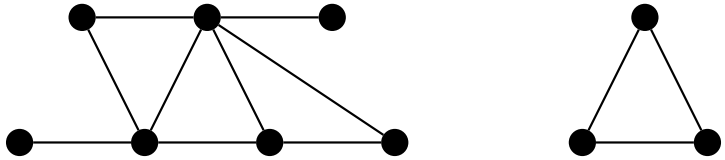
Input: undirected G, integer k

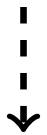
Question: is there a subset $X \subseteq V(G)$ of size at most k, such that $G \setminus X$ is a forest



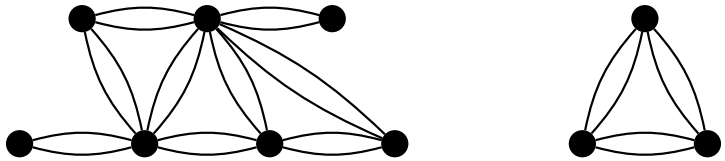
FVS

VC instance



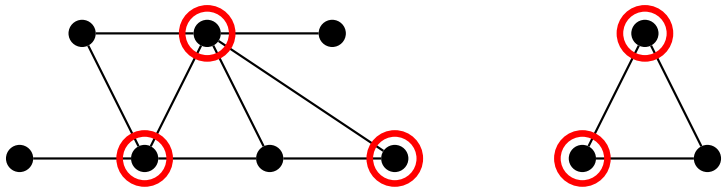


FVS instance

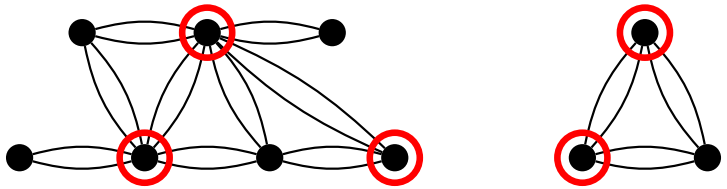


FVS

VC instance

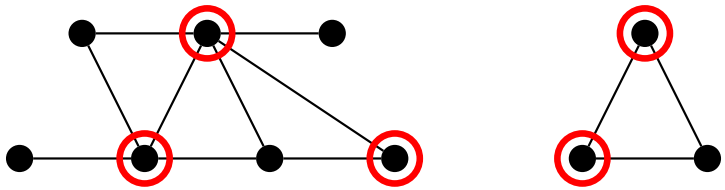


FVS instance

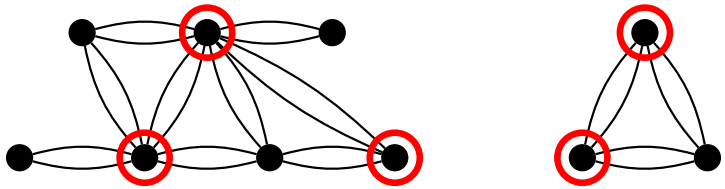


FVS

VC instance



FVS instance



Lemma

Unless ETH fails, there is neither $2^{o(n+m)}$ nor $\mathcal{O}^*(2^{o(k)})$ time algorithm for FVS.

Theorem

Unless ETH fails, there is no $2^{o(n+m)}$ time algorithm for:

- Vertex Cover,
- Feedback Vertex Set,
- Odd Cycle Transversal (exercise),
- Hamiltonian Cycle,
- 3-Coloring (exercise).

Planar lower bounds

Planar Vertex Cover

Input: undirected planar G, integer k

Question: is there a VC of size at most k.

Planar lower bounds

Planar Vertex Cover

Input: undirected planar G, integer k
Question: is there a VC of size at most k.

Planar graphs have treewidth $\mathcal{O}(\sqrt{n})$ (algorithmic proof, Wed).

Planar lower bounds

Planar Vertex Cover

Input: undirected planar G, integer k
Question: is there a VC of size at most k.

Planar graphs have treewidth $\mathcal{O}(\sqrt{n})$ (algorithmic proof, Wed).

VC can be solved in time $\mathcal{O}^*(2^{\mathrm{tw}})$,
hence Planar VC can be solved in time $2^{\mathcal{O}(\sqrt{n})}$.

Planar lower bounds

Planar Vertex Cover

Input: undirected planar G, integer k
Question: is there a VC of size at most k.

Planar graphs have treewidth $\mathcal{O}(\sqrt{n})$ (algorithmic proof, Wed).

VC can be solved in time $\mathcal{O}^*(2^{\mathrm{tw}})$,
hence Planar VC can be solved in time $2^{\mathcal{O}(\sqrt{n})}$.

Bidimensionality (Wed): planar VC can be solved in time $\mathcal{O}^*(2^{\sqrt{k}})$.

Planar lower bounds

How can we prove $2^{o(\sqrt{n})}$ lower bound for Planar VC?

Planar lower bounds

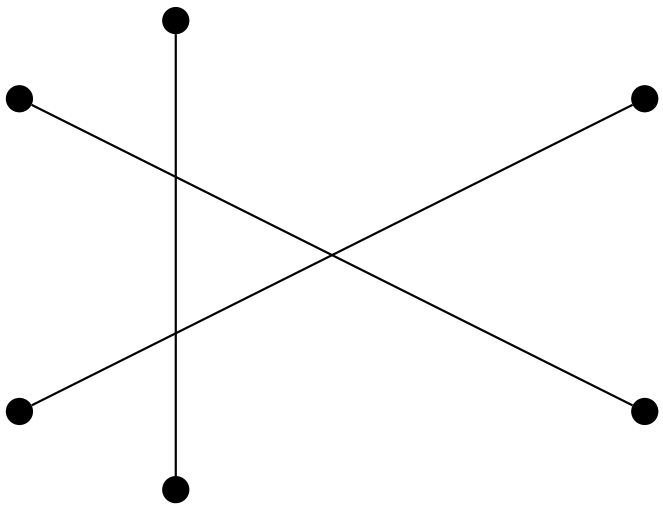
How can we prove $2^{o(\sqrt{n})}$ lower bound for Planar VC?

Idea: transform a Vertex Cover instance into a planar one, increasing the size to at most $\mathcal{O}((n+m)^2)$.

Planar lower bounds

How can we prove $2^{o(\sqrt{n})}$ lower bound for Planar VC?

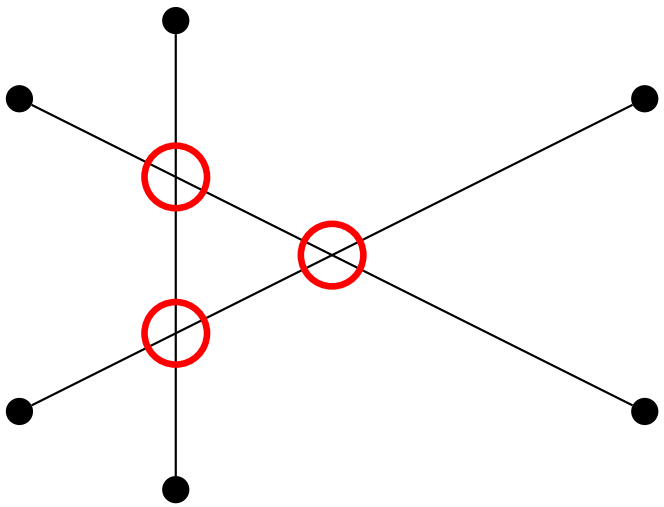
Idea: transform a Vertex Cover instance into a planar one, increasing the size to at most $\mathcal{O}((n+m)^2)$.



Planar lower bounds

How can we prove $2^{o(\sqrt{n})}$ lower bound for Planar VC?

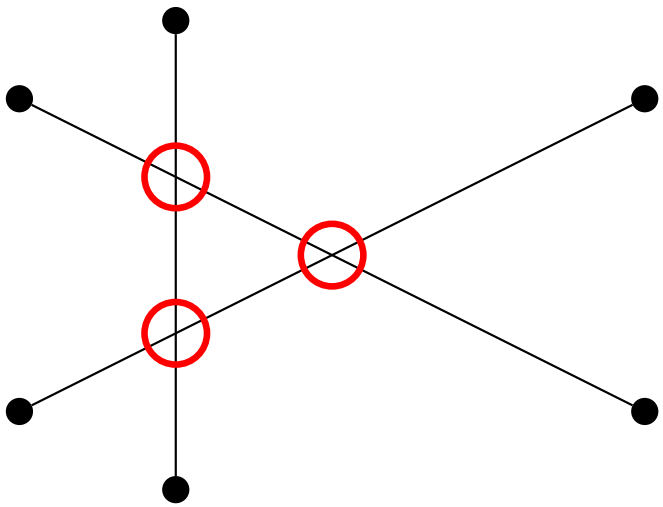
Idea: transform a Vertex Cover instance into a planar one, increasing the size to at most $\mathcal{O}((n+m)^2)$.

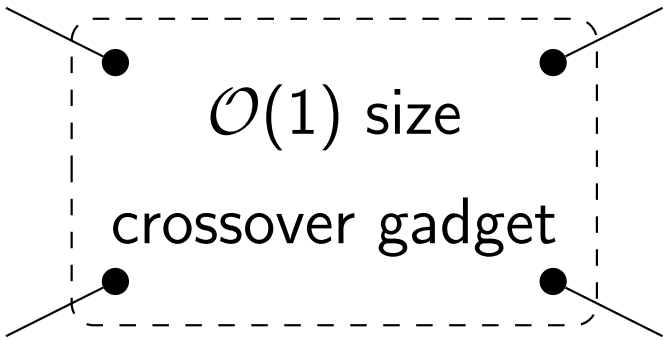


Planar lower bounds

How can we prove $2^{o(\sqrt{n})}$ lower bound for Planar VC?

Idea: transform a Vertex Cover instance into a planar one, increasing the size to at most $\mathcal{O}((n+m)^2)$.





ETH lower bounds methodology

Observation

In order to exclude an algorithm with running time $\mathcal{O}^*(2^{o(f(k))})$, we need to provide a reduction from 3-SAT to B that outputs instances with parameter k bounded by $g(n+m)$, where g is the inverse of f.

ETH lower bounds methodology

Observation

In order to exclude an algorithm with running time $\mathcal{O}^*(2^{o(f(k))})$, we need to provide a reduction from 3-SAT to B that outputs instances with parameter k bounded by $g(n+m)$, where g is the inverse of f.

- for $\mathcal{O}^*(2^{o(k)})$ we need $k = \mathcal{O}(n+m)$,
- for $\mathcal{O}^*(2^{o(\sqrt{k})})$ we need $k = \mathcal{O}((n+m)^2)$,
- for $\mathcal{O}^*(2^{o(k^2)})$ we need $k = \mathcal{O}(\sqrt{n+m})$,
- for $\mathcal{O}^*(2^{2^{o(k)}})$ we need $k = \mathcal{O}(\log(n+m))$.

ETH for W[1]-hard problems

Theorem (implies W[1] $\neq$ FPT)

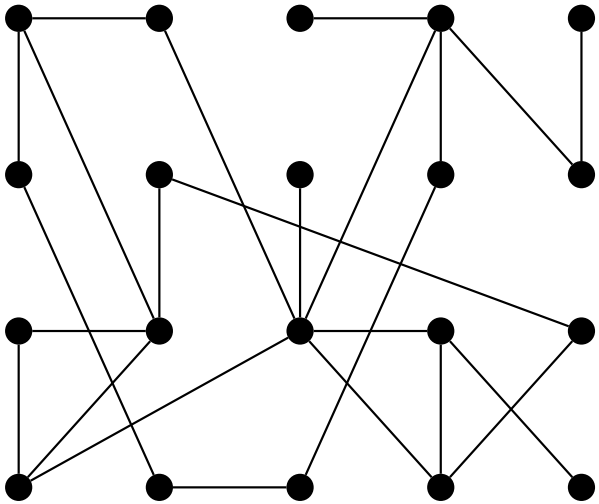
Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

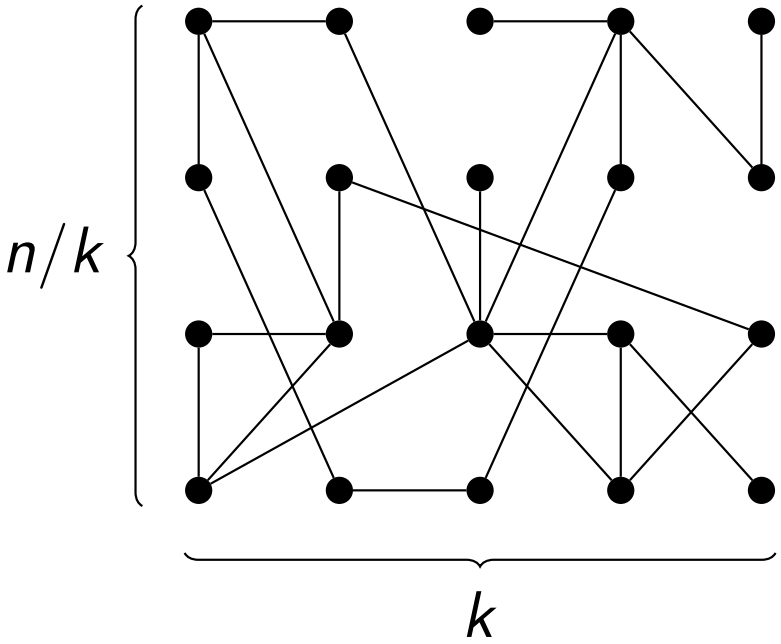
Theorem (implies W[1] $\neq$ FPT)

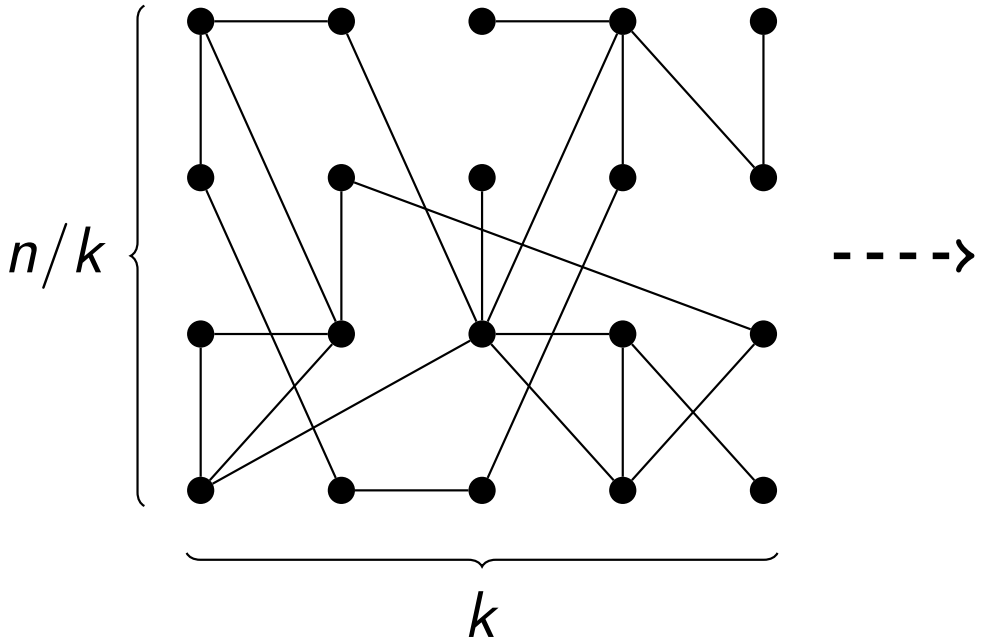
Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

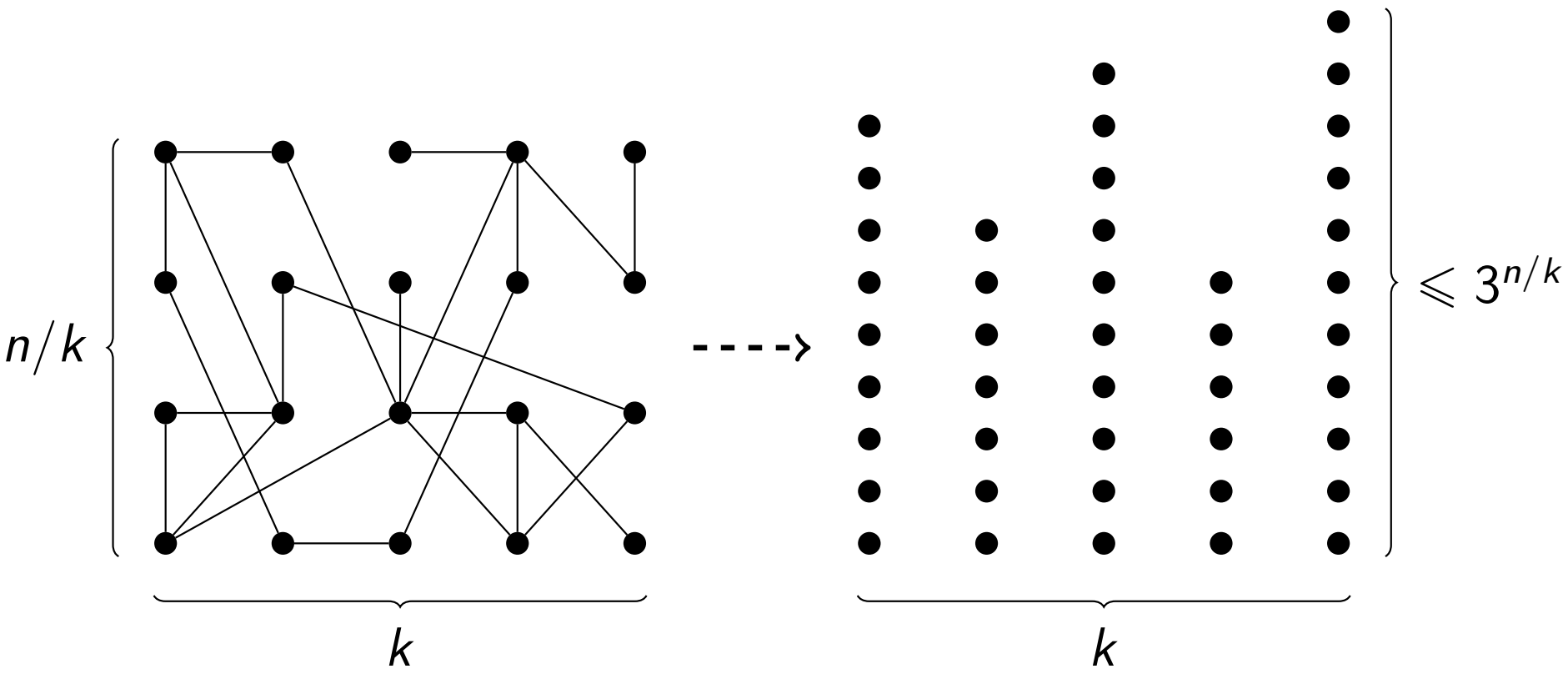
Proof:

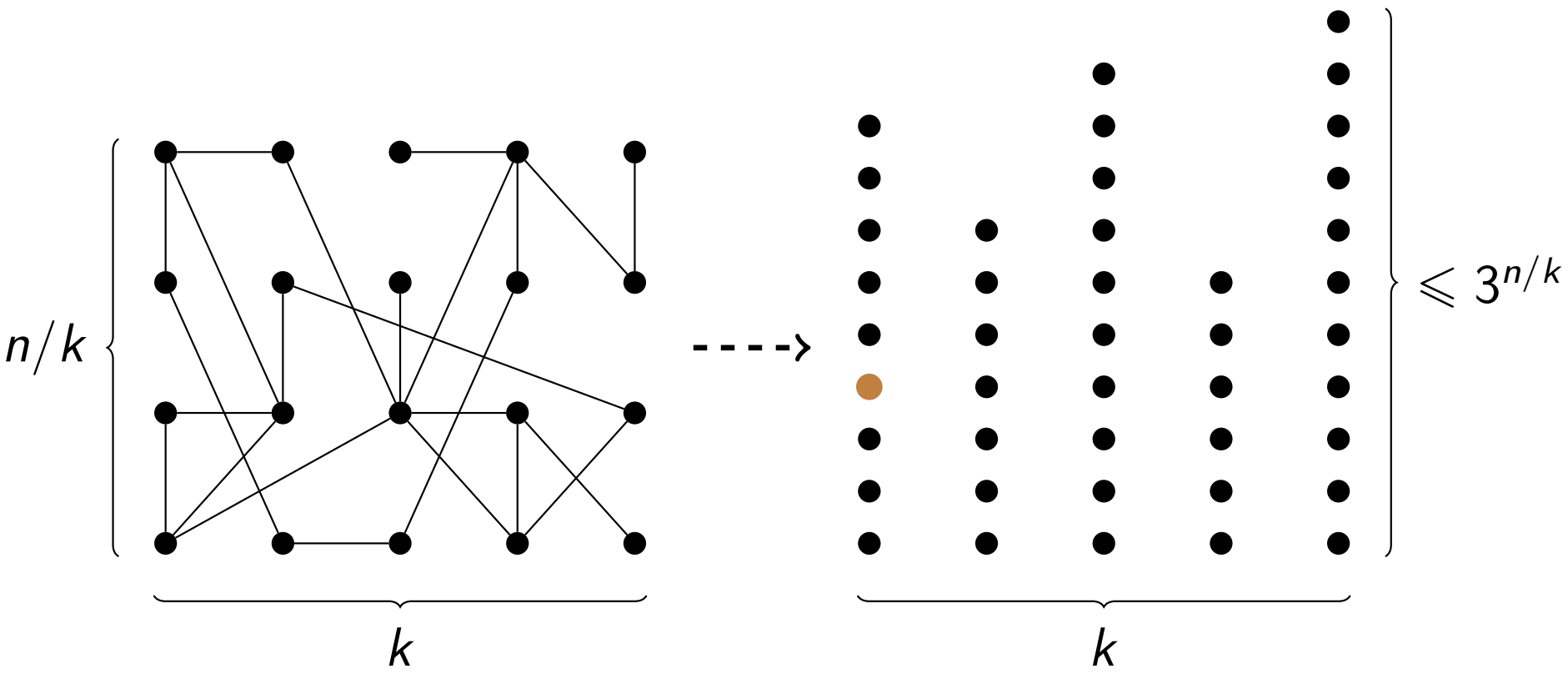
- A reduction from 3-Coloring (for which there is no $2^{\mathcal{O}(n+m)}$ time algorithm under ETH).
- In 3-Coloring we have n variables with 3 possible choices each.
- In Clique we have k variables with N possible choices each.
- We want $3^n \tilde{=} N^k$, so $N \tilde{=} 3^{n/k}$.
- But how to choose k? (decide later, but it has to be $\omega(1)$)

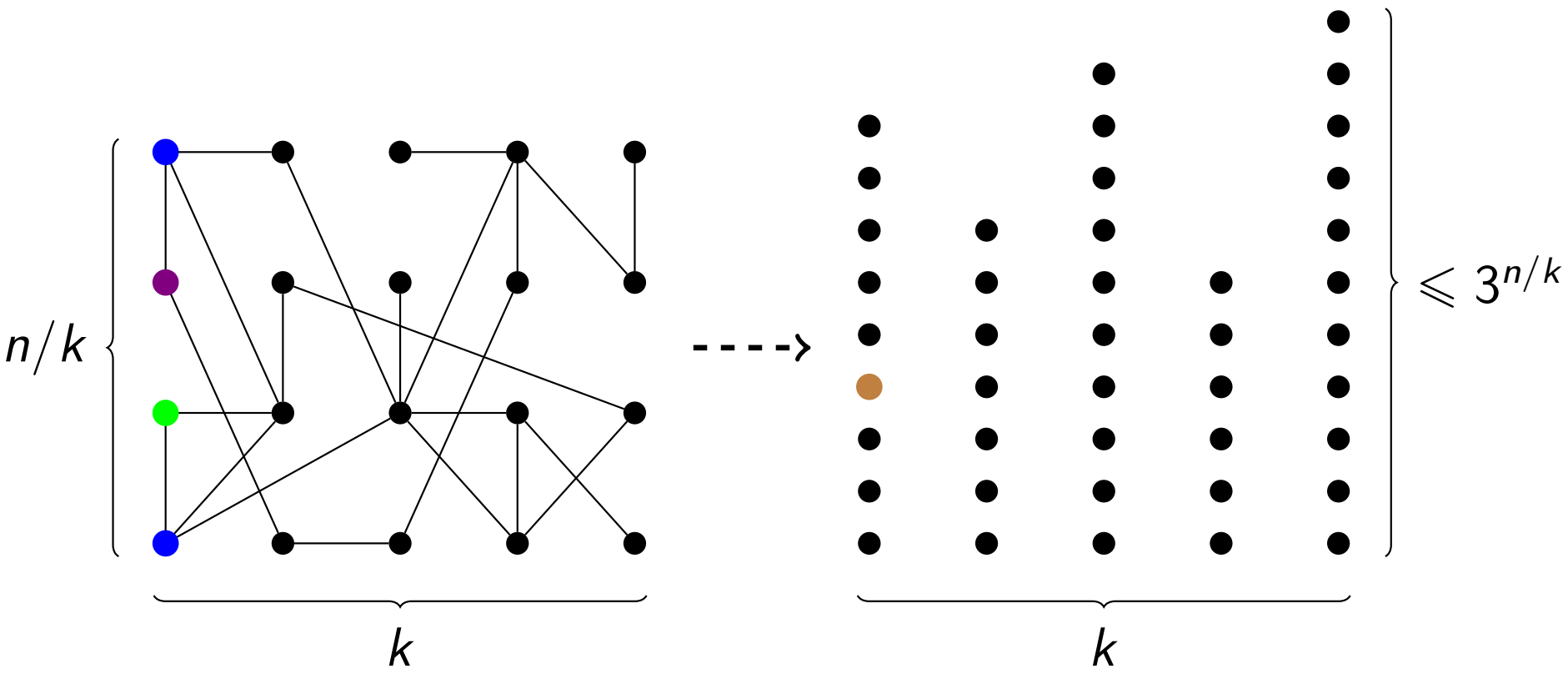


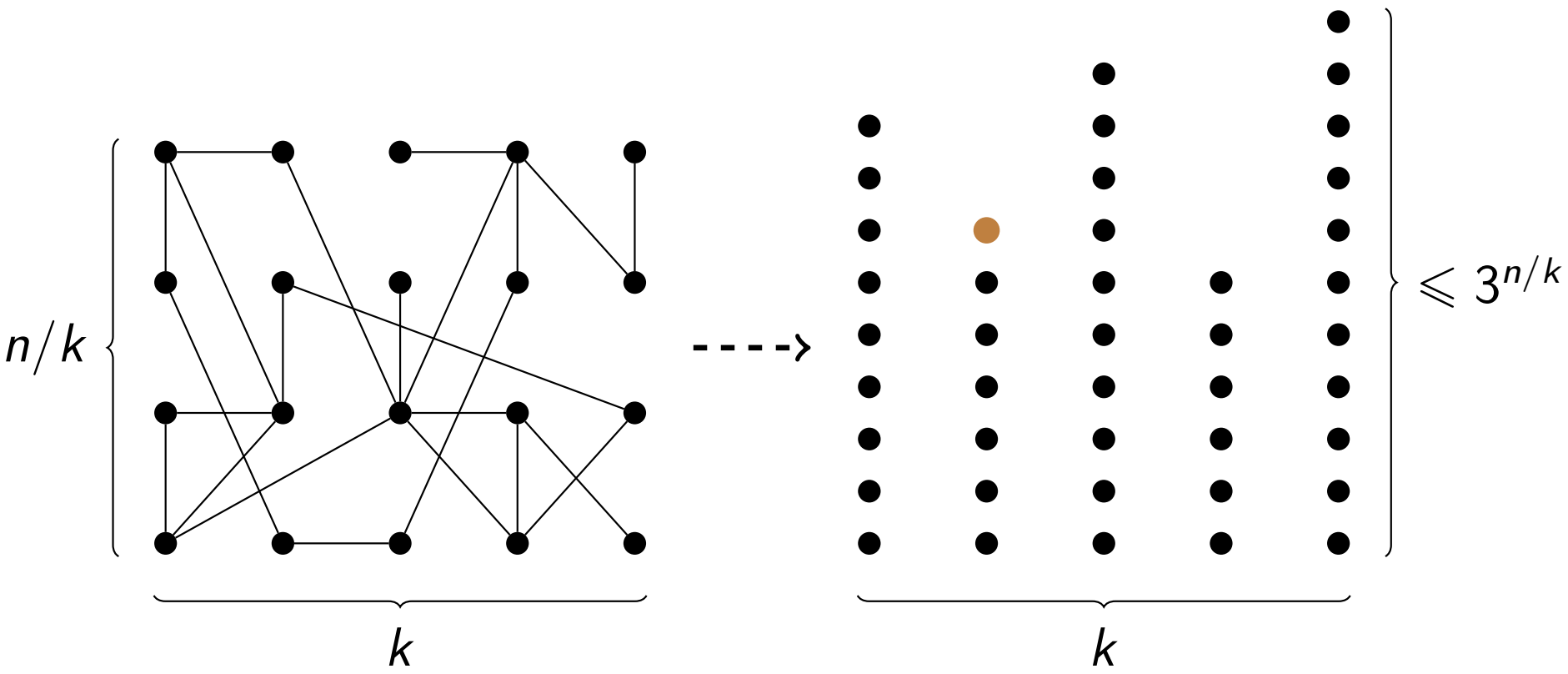

n/k
k

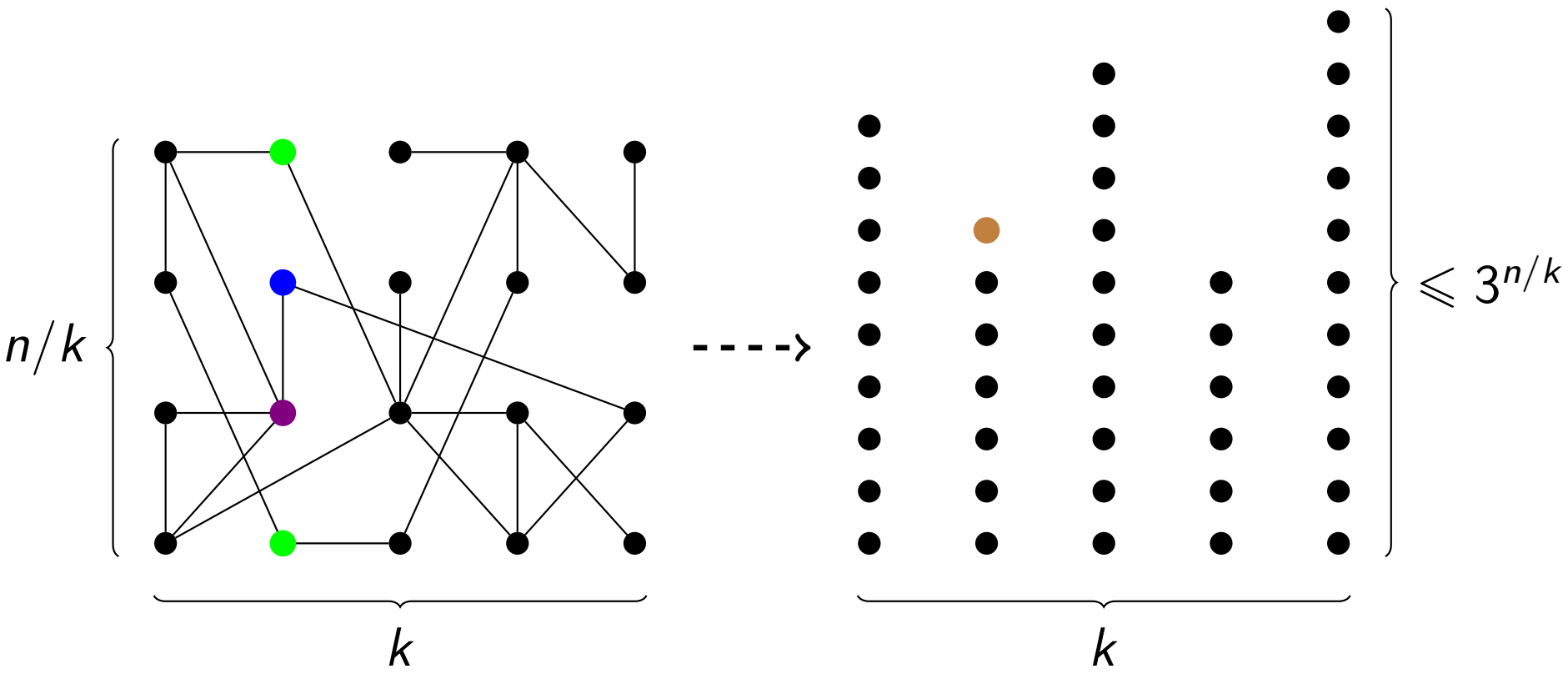

n/k
k

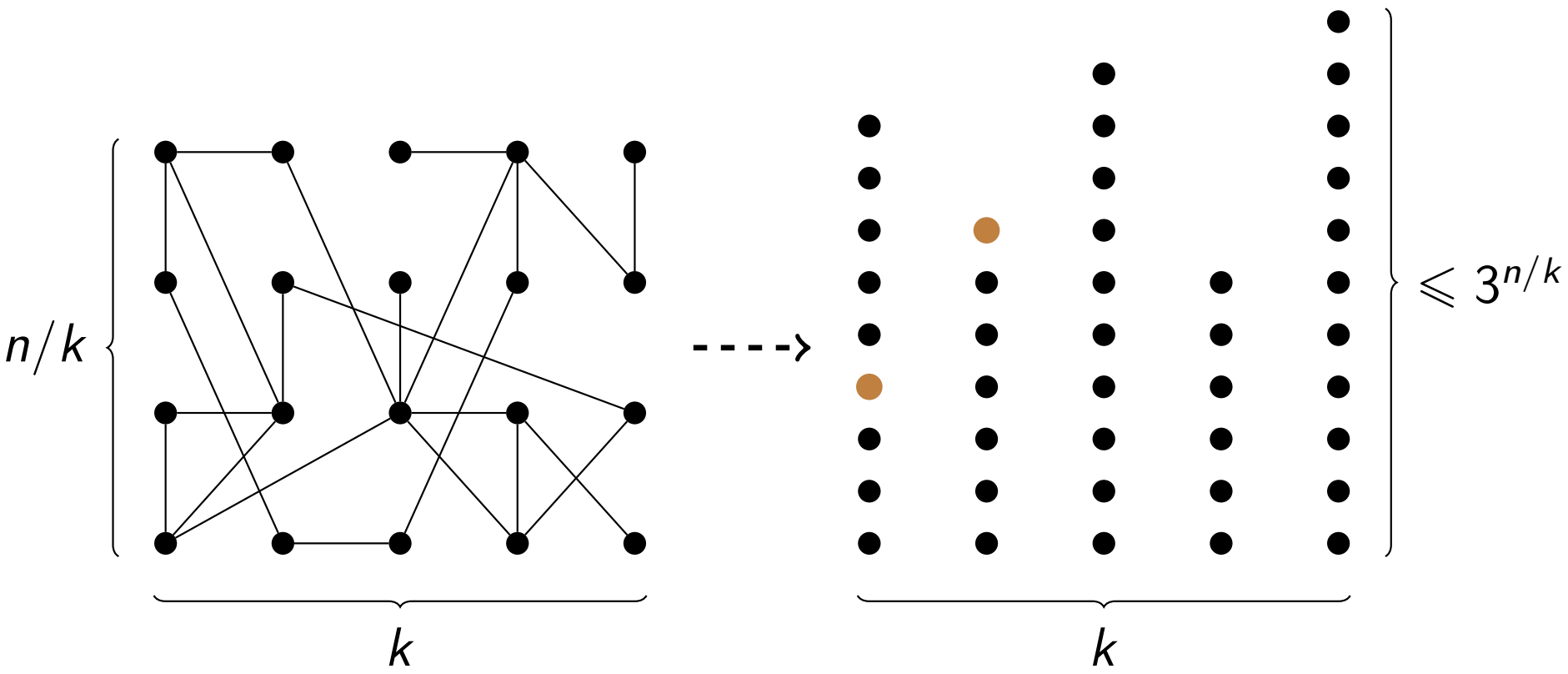

n/k
k
k
$\leqslant 3^{n/k}$

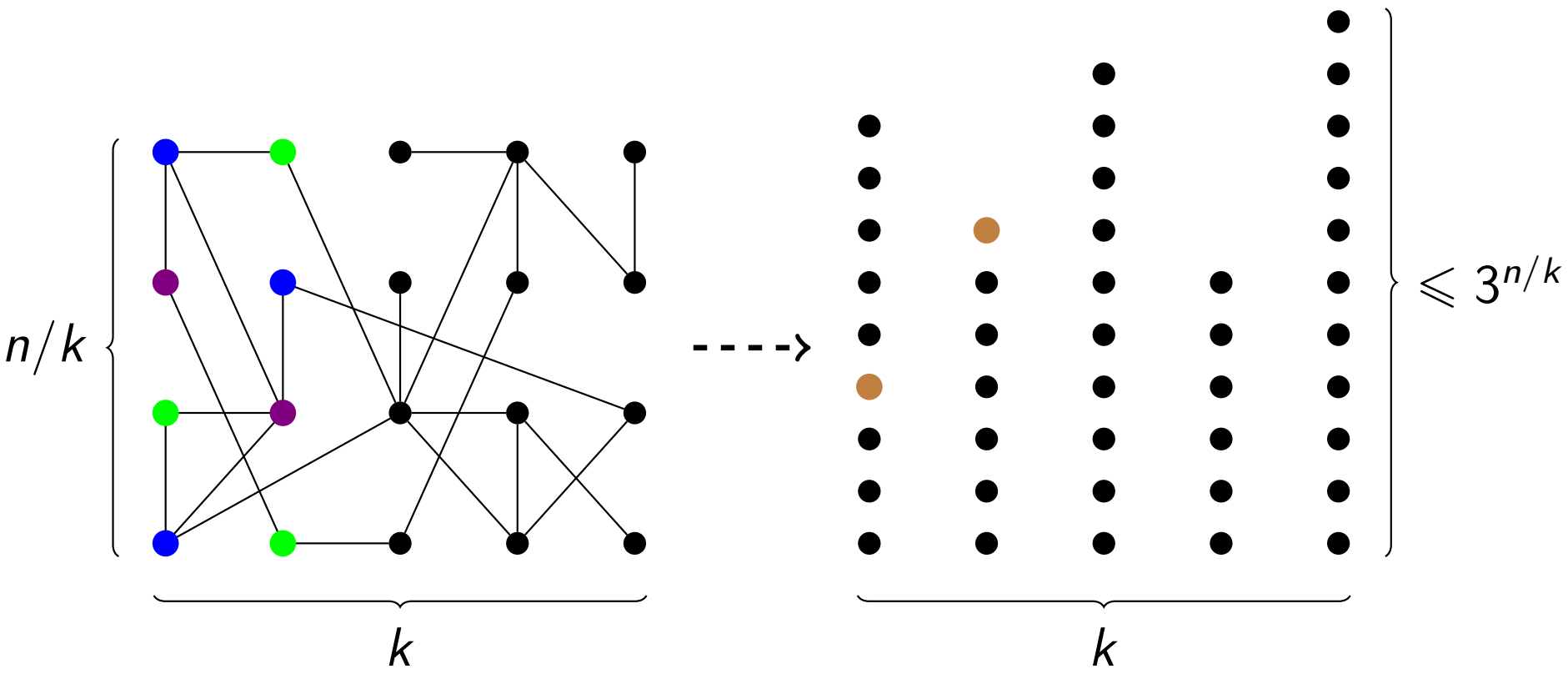

n/k
k
k
$\leqslant 3^{n/k}$

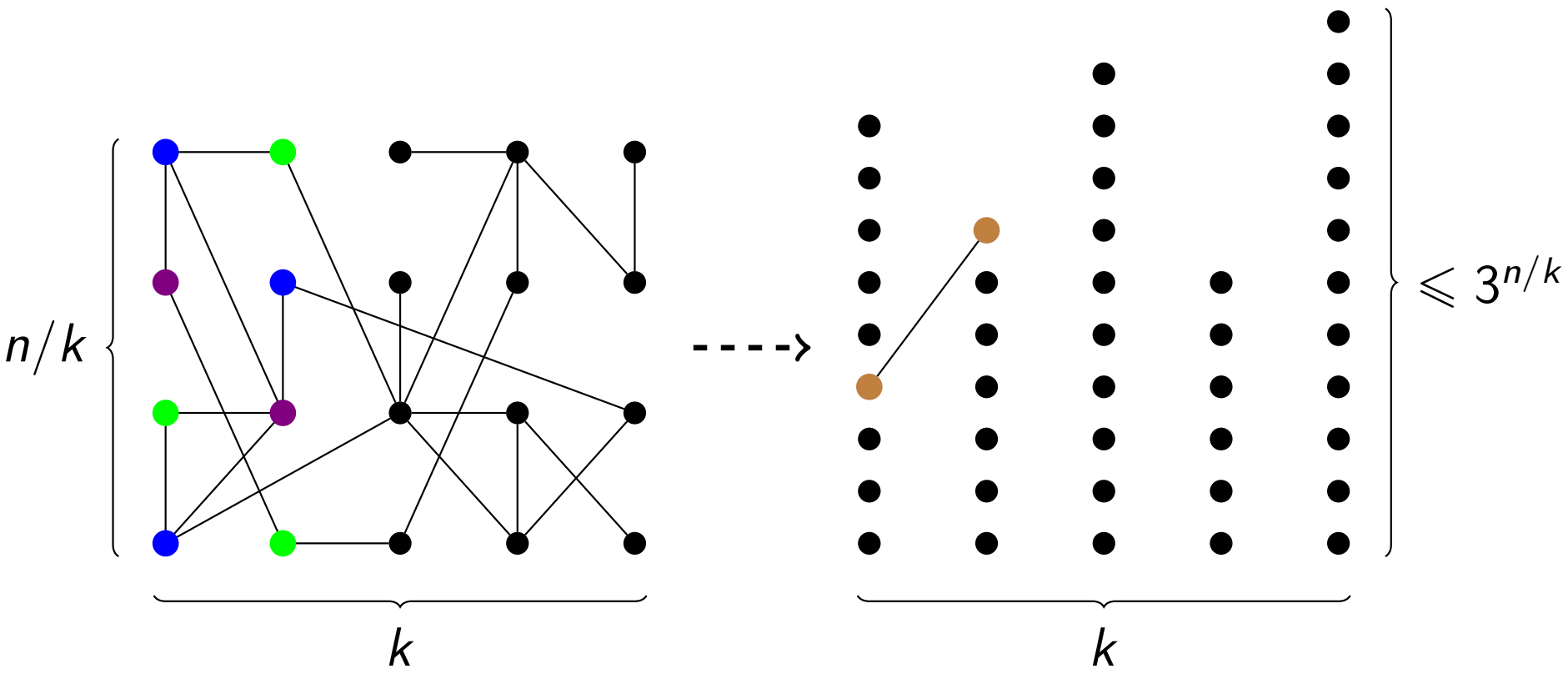

n/k
k
k
$\leqslant 3^{n/k}$

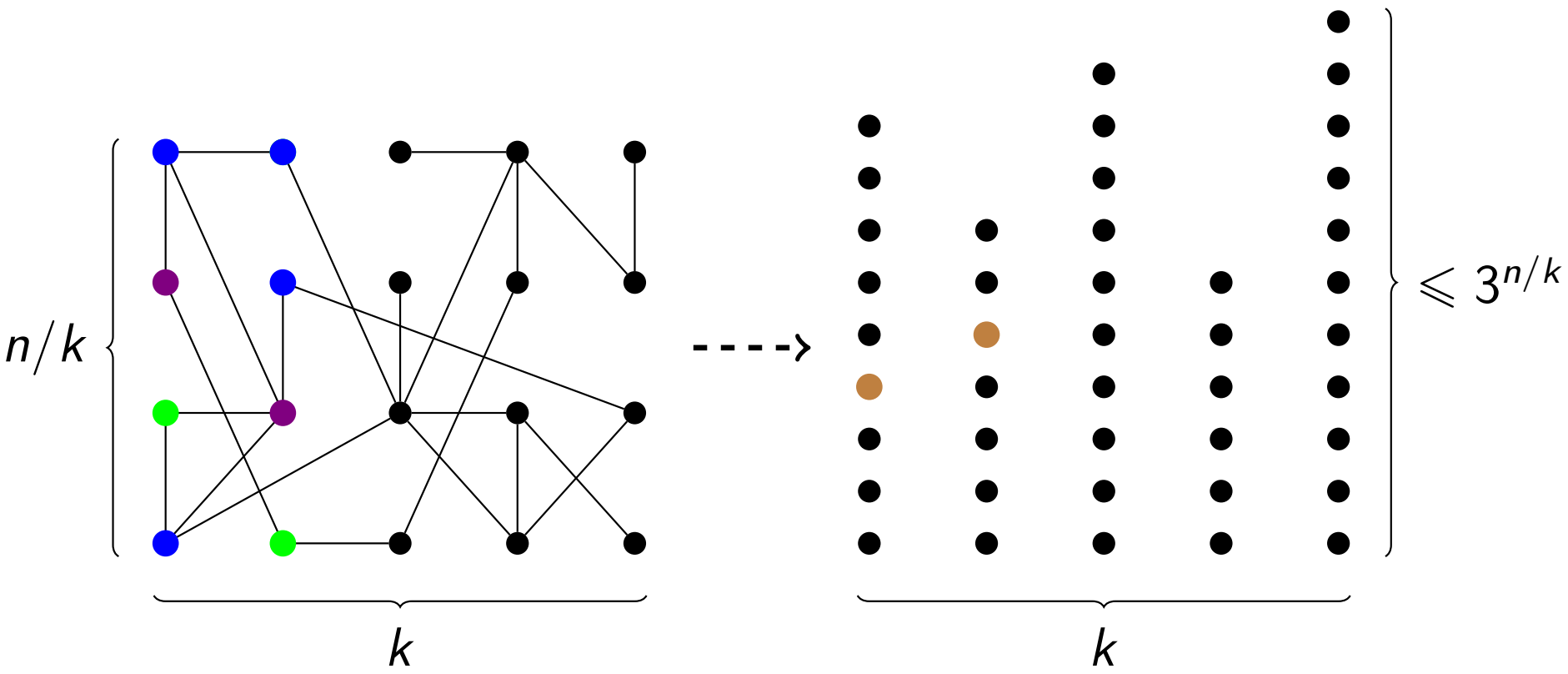

n/k
k
k
$\leqslant 3^{n/k}$

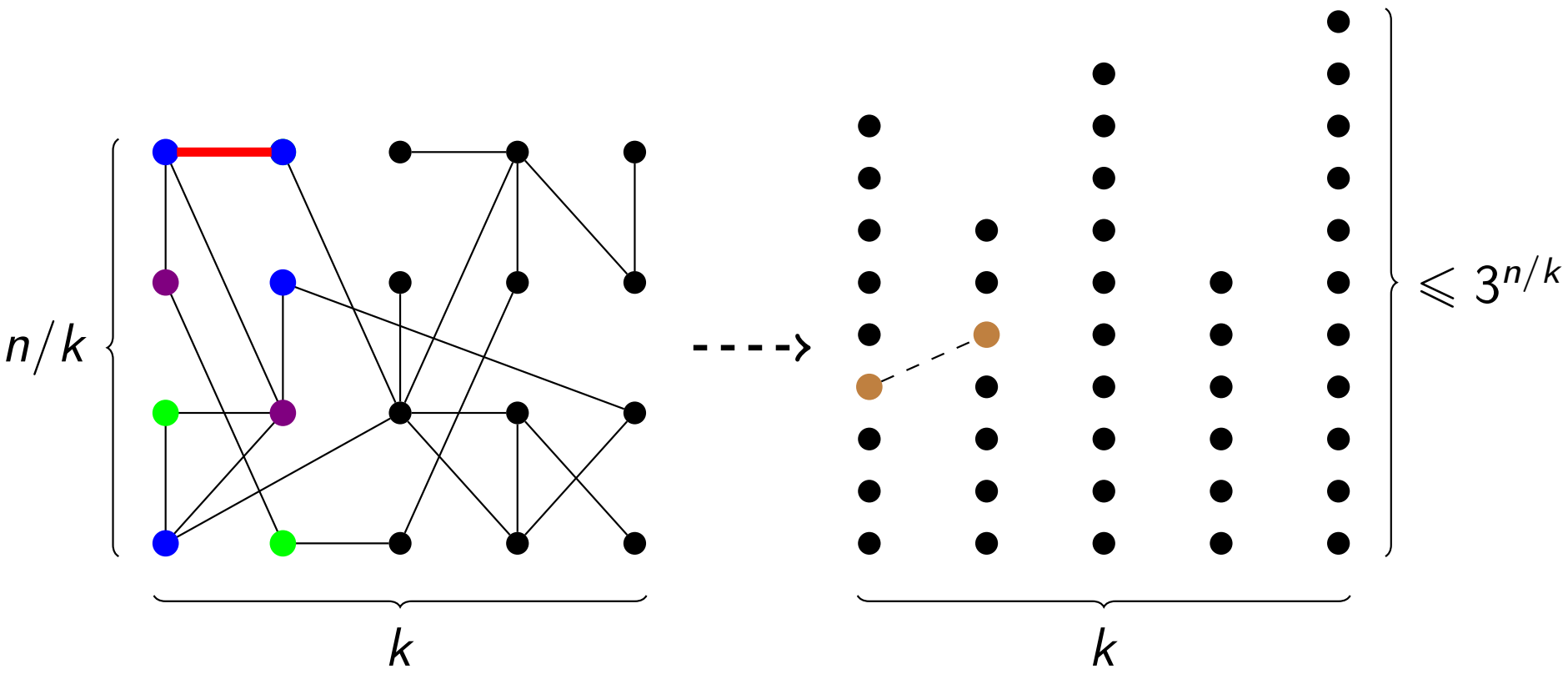

n/k
k
k
$\leqslant 3^{n/k}$

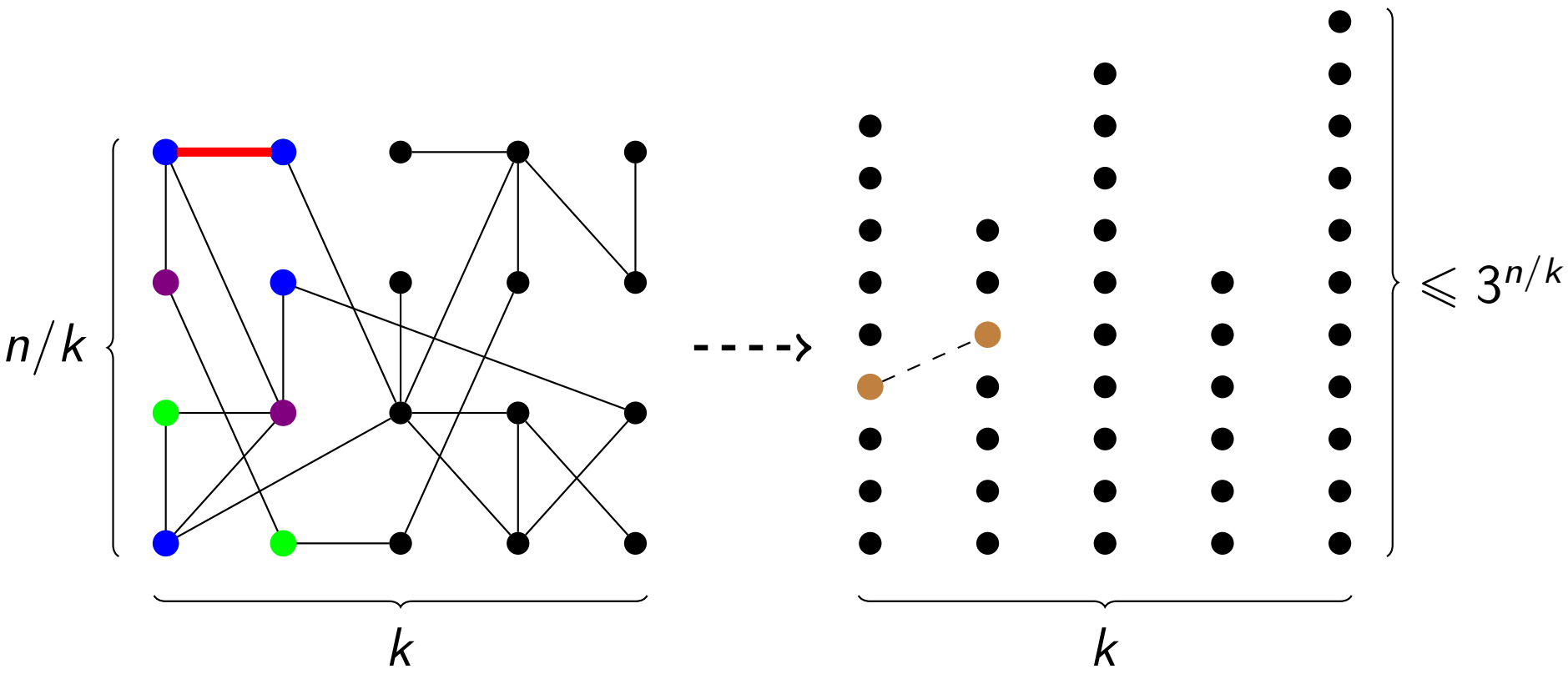

n/k
k
k
$\leqslant 3^{n/k}$


n/k
k
k
$\leqslant 3^{n/k}$


n/k
k
k
$\leqslant 3^{n/k}$


n/k
k
k
$\leqslant 3^{n/k}$


n/k
k
k
$\leqslant 3^{n/k}$



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

Theorem (implies W[1] $\neq$ FPT)

Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

Proof continued:

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

Theorem (implies W[1] $\neq$ FPT)

Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

Proof continued:

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

Theorem (implies W[1] $\neq$ FPT)

Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

Proof continued:

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

Theorem (implies W[1] $\neq$ FPT)

Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

Proof continued:

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

Theorem (implies W[1] $\neq$ FPT)

Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

Proof continued:

- Created graph has $N = k \cdot 3^{n/k}$ vertices.
- Lets try $k = \log n$.
- $2^k \cdot N^{o(k)} = n \cdot (\log n)^{\log n} \cdot 3^{n \cdot o(\log n)/\log n} = \mathcal{O}^*(2^{o(n)})$.
- Similarly $k = \log \log n$ implies no $2^{2^k} N^{o(k)}$ time algorithm.
- To exclude all computable $f(k)$, one needs roughly $f^{-1}(n)$ (technical difficulties omitted).

Theorem (implies W[1] $\neq$ FPT)

Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

- A reduction from Clique to X which outputs parameter $g(k)$ implies lower bound of $f(k)n^{o(g^{-1}(k))}$.

Theorem (implies W[1] $\neq$ FPT)

Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

- A reduction from Clique to X which outputs parameter $g(k)$ implies lower bound of $f(k)n^{o(g^{-1}(k))}$.
- Reductions from Clique to Multicolored Clique, Independent Set, and Dominating Set from lecture 4 result in instances with parameters $\mathcal{O}(k)$.

Theorem (implies W[1] $\neq$ FPT)

Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

- A reduction from Clique to X which outputs parameter $g(k)$ implies lower bound of $f(k)n^{o(g^{-1}(k))}$.
- Reductions from Clique to Multicolored Clique, Independent Set, and Dominating Set from lecture 4 result in instances with parameters $\mathcal{O}(k)$.
- Consequently (under ETH) there is no $f(k)n^{o(k)}$ algorithm for those problems.

Theorem (implies W[1] $\neq$ FPT)

Unless ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for Clique.

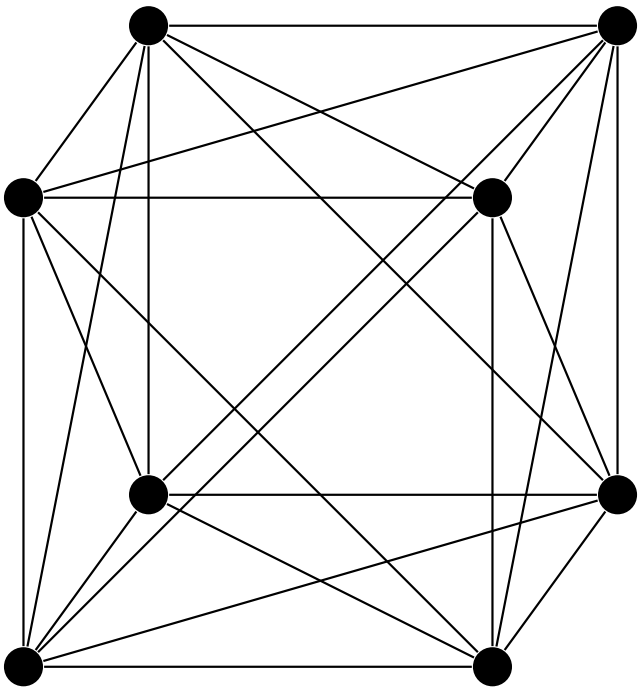
- A reduction from Clique to X which outputs parameter $g(k)$ implies lower bound of $f(k)n^{o(g^{-1}(k))}$.
- Reductions from Clique to Multicolored Clique, Independent Set, and Dominating Set from lecture 4 result in instances with parameters $\mathcal{O}(k)$.
- Consequently (under ETH) there is no $f(k)n^{o(k)}$ algorithm for those problems.
- For Odd Set the reduction does not give so strong lower bound (Ex: check what is implied by the reduction from lecture 4).

ETH used for somewhat exotic lower bounds

Edge Clique Cover (ECC)

Input: Graph G, integer k

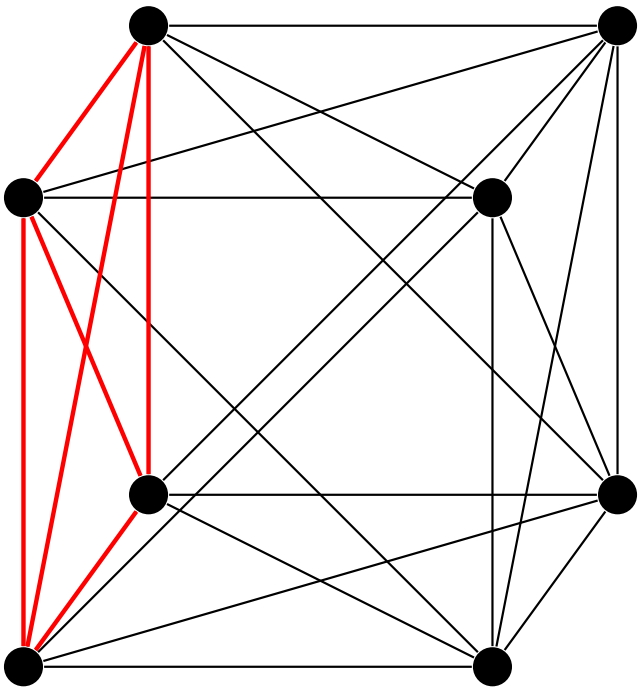
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover (ECC)

Input: Graph G, integer k

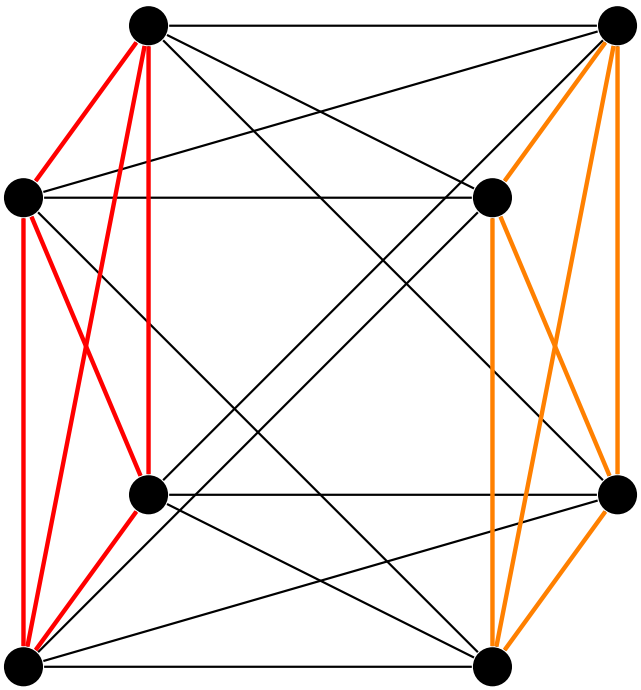
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover (ECC)

Input: Graph G, integer k

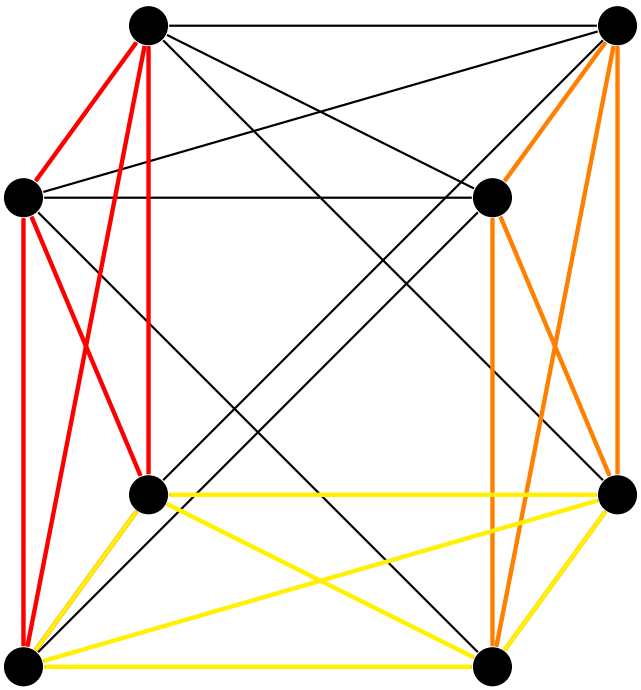
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover (ECC)

Input: Graph G, integer k

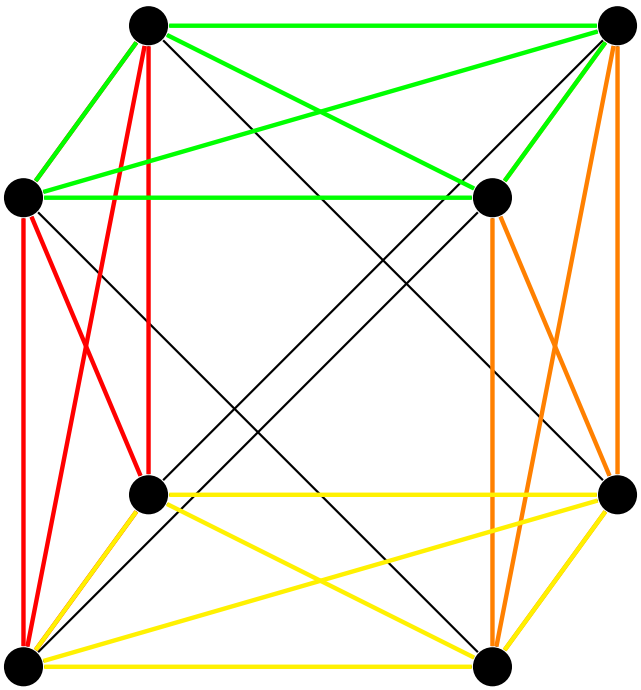
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover (ECC)

Input: Graph G, integer k

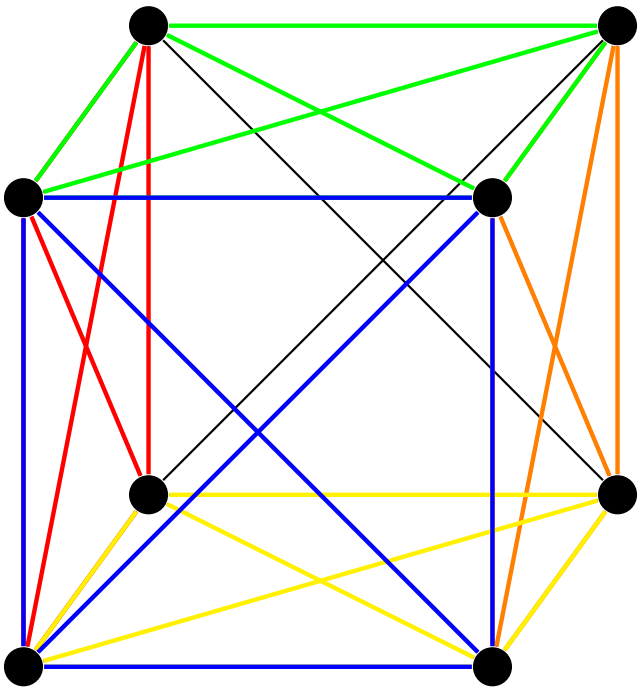
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover (ECC)

Input: Graph G, integer k

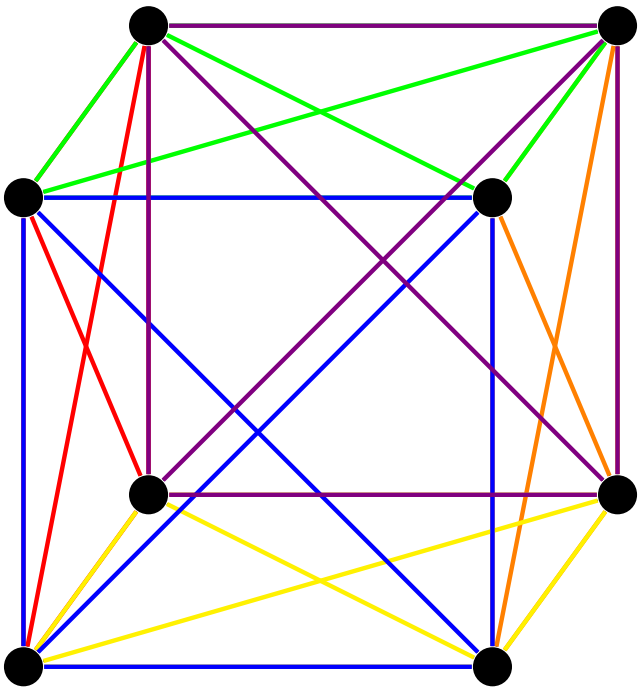
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



Edge Clique Cover (ECC)

Input: Graph G, integer k

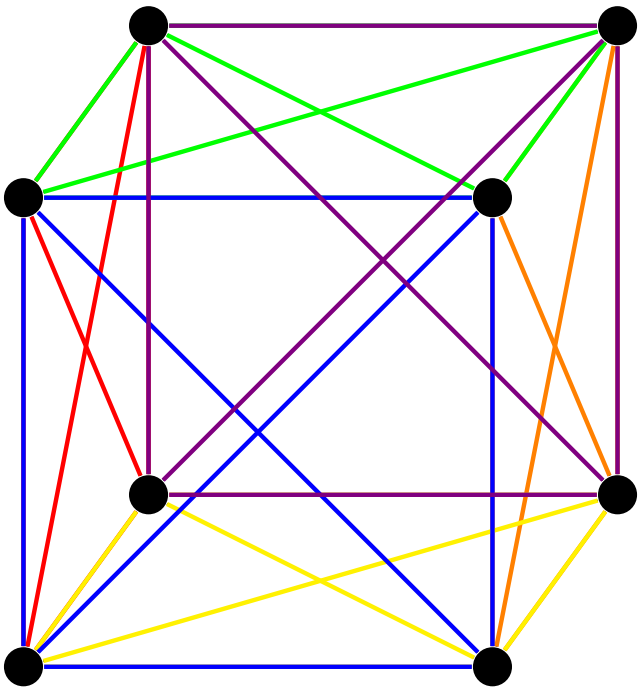
Question: Does there exists a collection of k cliques $C_1, C_2, \ldots, C_k$ in G, such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$?



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

ECC

Book, section 2.3.3

Edge Clique Cover admits a kernel with at most 2^k vertices.

ECC

Book, section 2.3.3

Edge Clique Cover admits a kernel with at most 2^k vertices.

Exercise

By casting the problem as Set Cover, we obtain $\mathcal{O}^*(2^{2^{\mathcal{O}(k)}})$ algorithm.

ECC

Book, section 2.3.3

Edge Clique Cover admits a kernel with at most 2^k vertices.

Exercise

By casting the problem as Set Cover, we obtain $\mathcal{O}^*(2^{2^{\mathcal{O}(k)}})$ algorithm.

Theorem

Unless ETH fails, there is no $\mathcal{O}^*(2^{2^{o(k)}})$ time algorithm for ECC.
Proof by reducing formula with n vars, m clauses to poly size ECC instance with $k = \mathcal{O}(\log(n+m))$.

Closest Substring

Closest Substring

Input: strings $s_1, \ldots, s_t \in \Sigma^*$, integers L, d.
Question: Is there $s \in \Sigma^L$, such that for every $1 \leqslant i \leqslant t$ there is a substring s_i' of s_i of length L with $d_H(s, s_i) \leqslant d$.

Closest Substring

Closest Substring - important parts

Input: t strings of total size n

Question: fairly natural question about Hamming distance

Closest Substring

Closest Substring - important parts

Input: t strings of total size n
Question: fairly natural question about Hamming distance

Lemma

Closest Substring over constant size alphabet can be solved in time $f(d,t)n^{\mathcal{O}(\log\log t)}$. Mover under ETH there is no $f(d,t)n^{o(\log\log t)}$ time algorithm for any f.

Summary

Take-home message 1

$$\text{SETH} \implies \text{ETH} \implies \text{W[1]} \neq \text{FPT} \implies \text{P} \neq \text{NP}$$

$$\text{FPT} \neq \text{W[1]} \implies \text{no } \mathcal{O}^*(f(k))$$
$$\text{ETH} \implies \text{no } \mathcal{O}^*(2^{o(f(k))})$$
$$\text{SETH} \implies \text{no } \mathcal{O}^*(2^{(1-\epsilon)f(k)}).$$

Take-home message 2

When showing lower bounds, we have to come up with a reduction and track what happens with the parameter in the new instance.

Summary

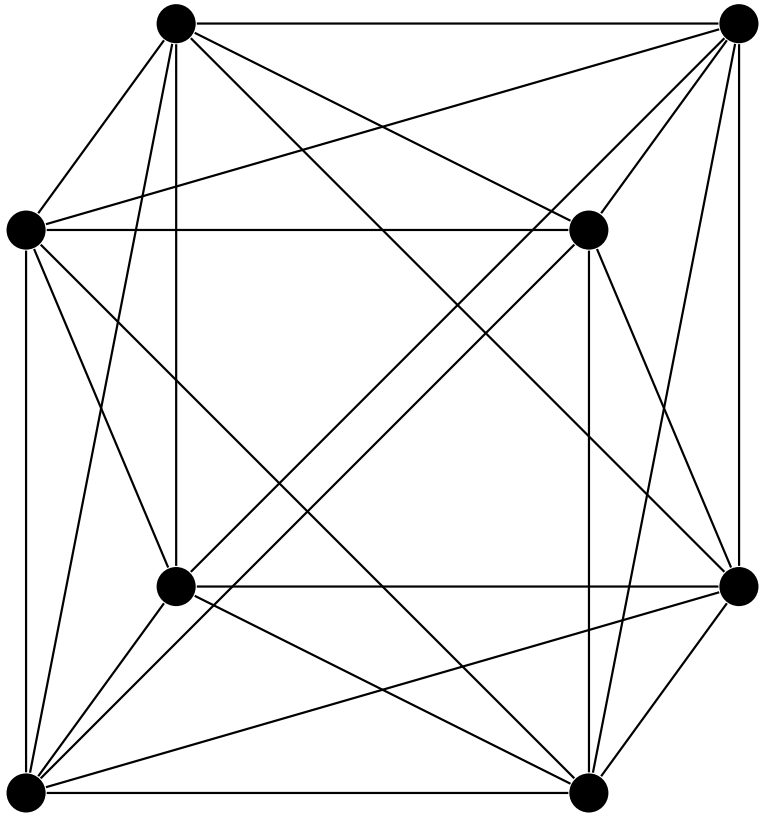
- We have learned that ETH is a tool that allows proving exciting lower bounds.
- More on this subject on Thursday (lectures by Michał and Daniel L.).
- Exercises: 14.1, 14.2.(d) (+OCT), show $\mathcal{O}^*(2^{2^{O(k)}})$ algorithm for ECC (after kernelization), check what lower bound (under ETH) is implied for Odd Set by reduction from lecture 4.

Summary

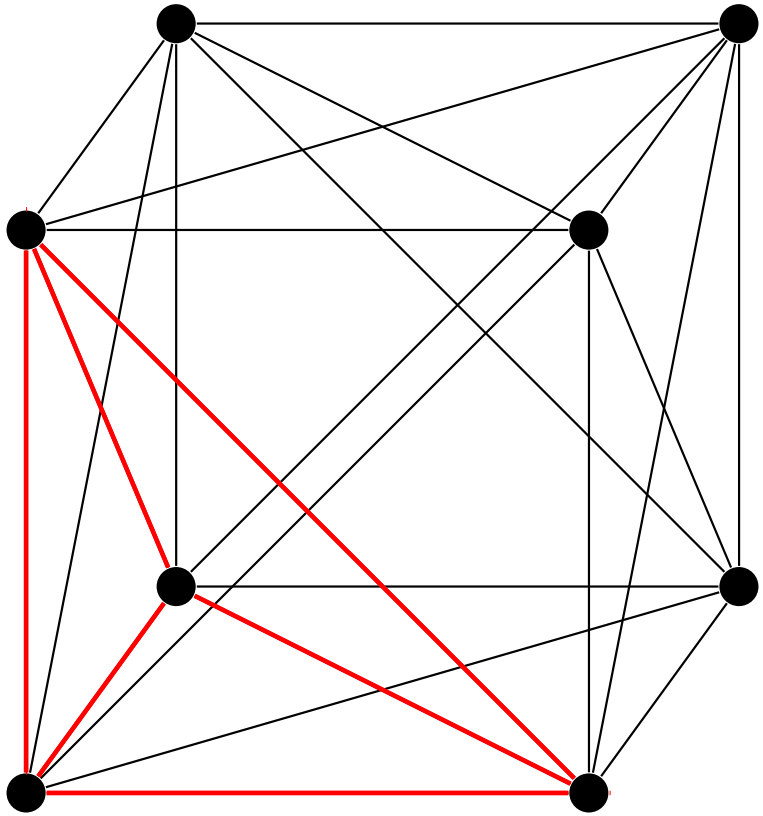
- We have learned that ETH is a tool that allows proving exciting lower bounds.
- More on this subject on Thursday (lectures by Michał and Daniel L.).
- Exercises: 14.1, 14.2.(d) (+OCT), show $\mathcal{O}^*(2^{2^{O(k)}})$ algorithm for ECC (after kernelization), check what lower bound (under ETH) is implied for Odd Set by reduction from lecture 4.

Thank you for your attention

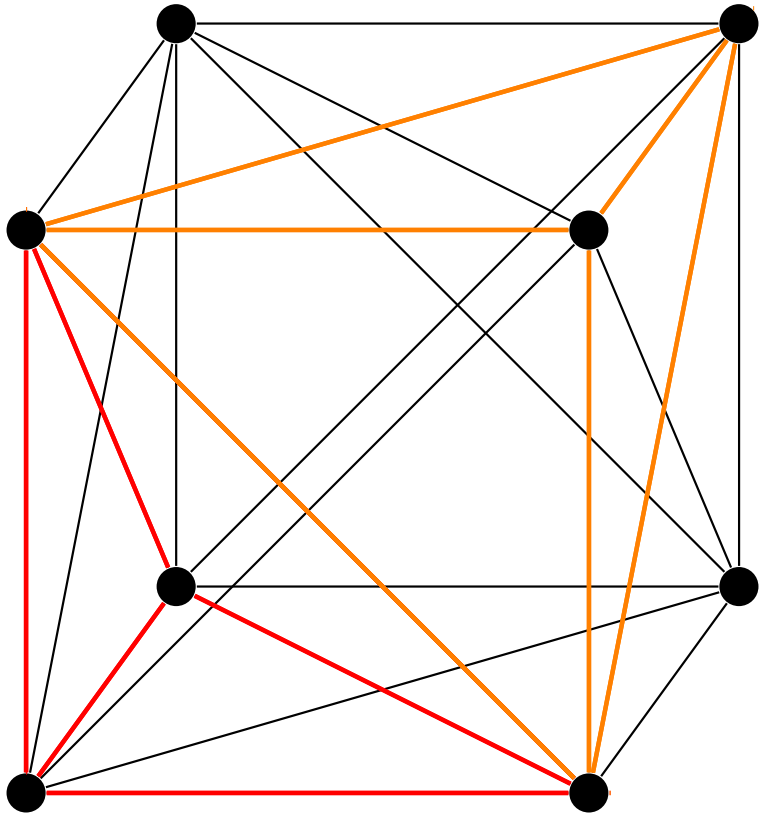
Hidden slide



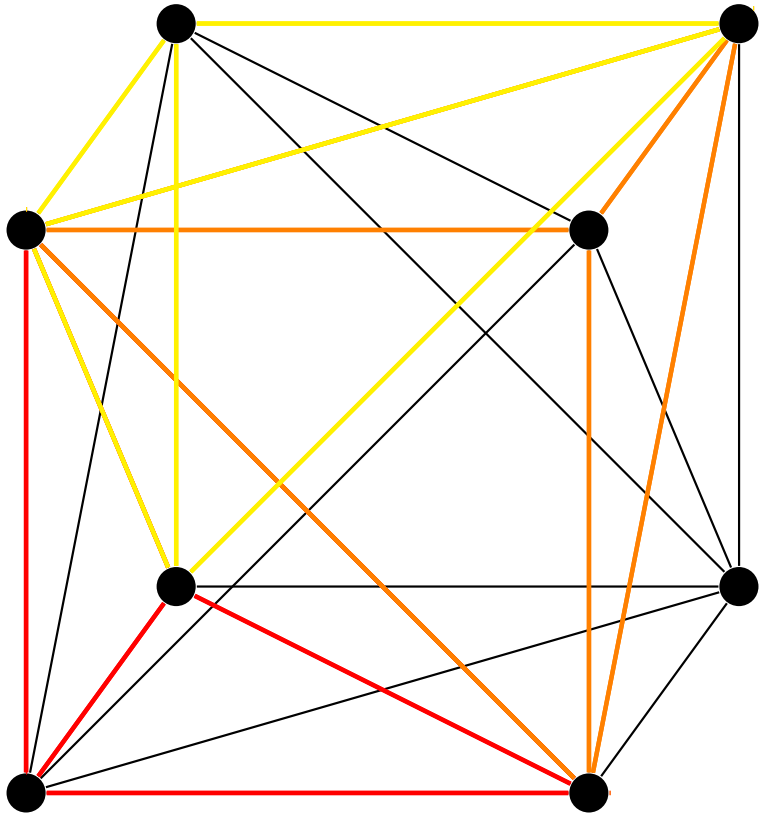
Hidden slide



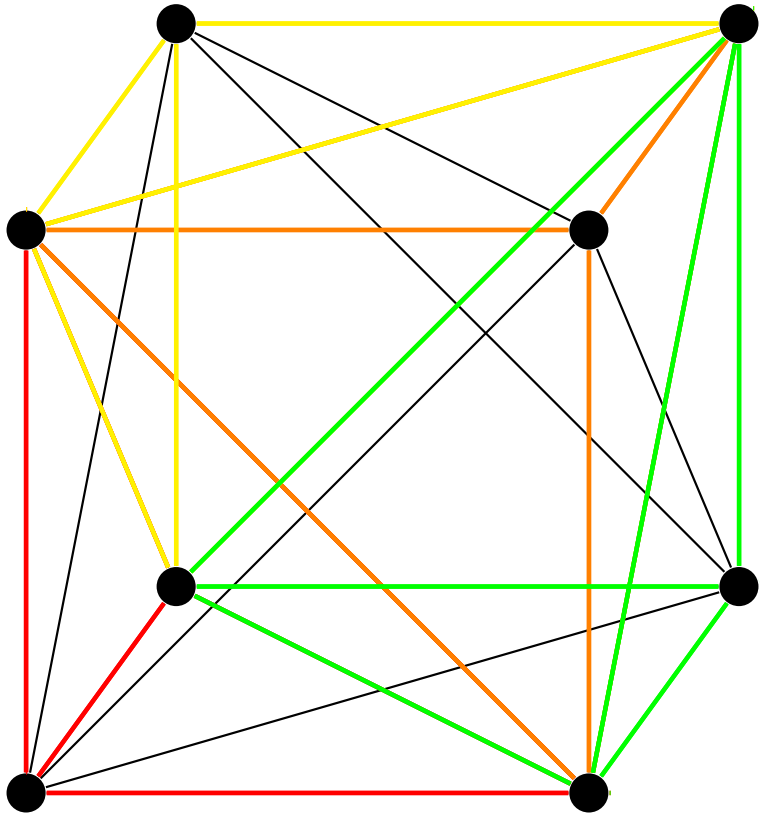
Hidden slide



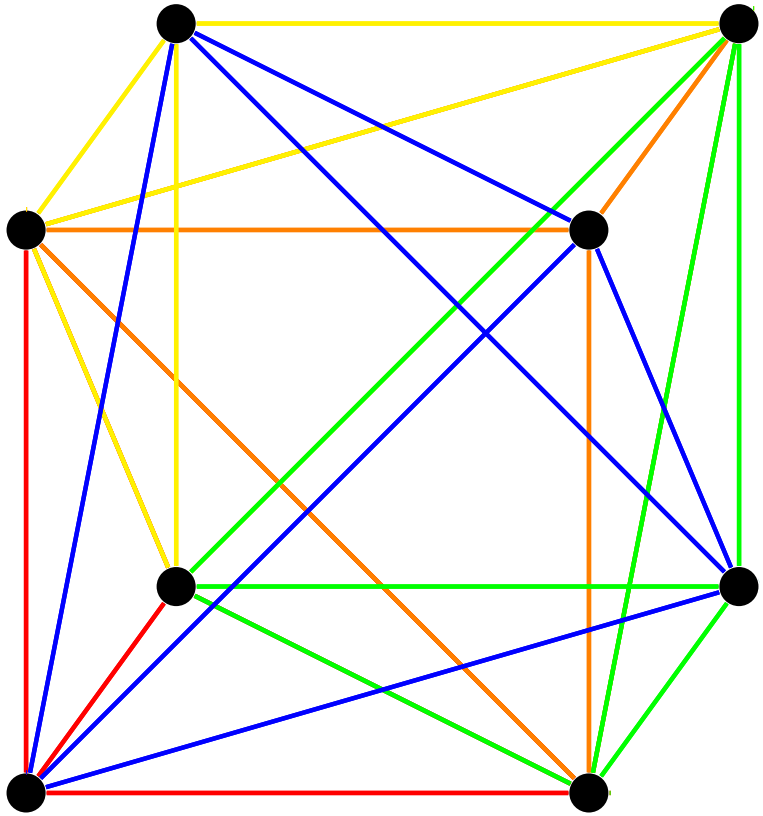
Hidden slide



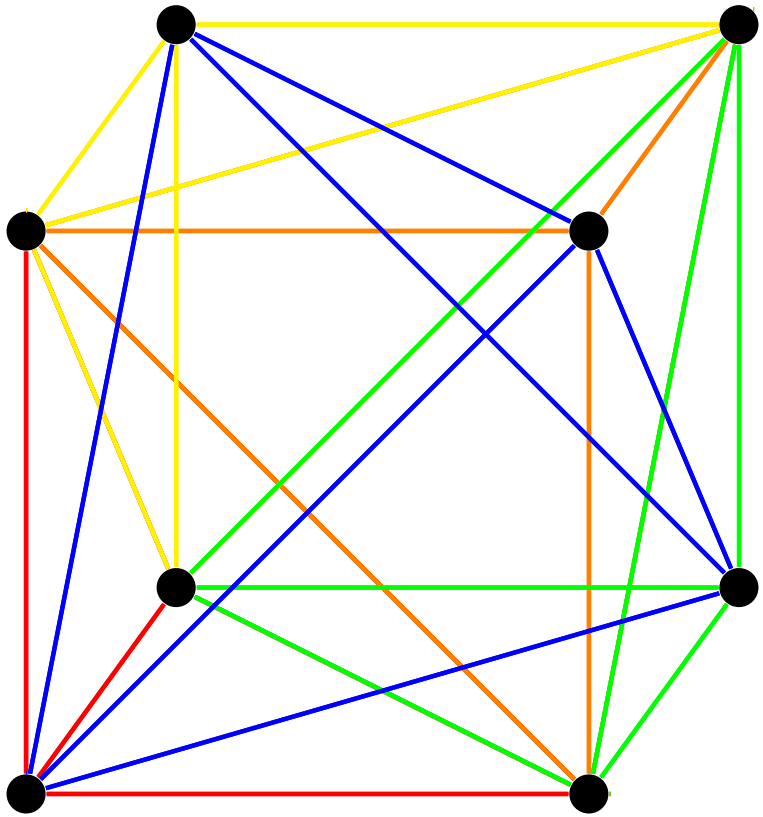
Hidden slide



Hidden slide



Hidden slide



Magical solution with 5 cliques