Exponential Time Hypothesis and Parametrized Complexity
Part 1

Aeren

October 11, 2022

Introduction

- We strongly believe that $\mathsf{P} \neq \mathsf{NP}$.

Introduction

- We strongly believe that P $\neq$ NP.
- However, that doesn't say much about how much time will the NP-Complete problems actually take to solve.

Introduction

- We strongly believe that P $\neq$ NP.
- However, that doesn't say much about how much time will the NP-Complete problems actually take to solve.
- We'll firstly focus on the kSAT problem in order to estimate their difficulties.

Sparsification Lemma

- There are two parameters to look at:

Sparsification Lemma

- There are two parameters to look at:
    1. n, the number of variables, and

Sparsification Lemma

- There are two parameters to look at:
    1. n, the number of variables, and
    2. m, the actual length of the input.

Sparsification Lemma

- There are two parameters to look at:
    1. n, the number of variables, and
    2. m, the actual length of the input.
- It turns out that the parameter m doesn't affect our analysis much.

Sparsification Lemma

DEFINITION

A kSAT instance is called **sparse** if $m \in O(n)$

Sparsification Lemma

DEFINITION

A kSAT instance is called **sparse** if $m \in O(n)$

The sparsification lemma enable us to convert an arbitrary kSAT instance into a collection of sparse kSAT instances.

Sparsification Lemma

LEMMA (Sparsification Lemma)

For all $k \in \mathbb{N}$ and $\epsilon \in (0, 1]$, there is a constant $c(k, \epsilon)$ and an algorithm which runs in $2^{\epsilon \cdot n} \cdot poly(m)$ time such that

Sparsification Lemma

LEMMA (Sparsification Lemma)

For all $k \in \mathbb{N}$ and $\epsilon \in (0, 1]$, there is a constant $c(k, \epsilon)$ and an algorithm which runs in $2^{\epsilon \cdot n} \cdot poly(m)$ time such that

1. the input is a k-CNF formula ϕ,

Sparsification Lemma

LEMMA (Sparsification Lemma)

For all $k \in \mathbb{N}$ and $\epsilon \in (0, 1]$, there is a constant $c(k, \epsilon)$ and an algorithm which runs in $2^{\epsilon \cdot n} \cdot poly(m)$ time such that

1. the input is a k-CNF formula ϕ,
2. the output is k-CNF formulae $\phi_1, \cdots, \phi_t$ where $t \leq 2^{\epsilon \cdot n}$,

Sparsification Lemma

LEMMA (Sparsification Lemma)

For all $k \in \mathbb{N}$ and $\epsilon \in (0,1]$, there is a constant $c(k,\epsilon)$ and an algorithm which runs in $2^{\epsilon \cdot n} \cdot poly(m)$ time such that

1. the input is a k-CNF formula ϕ,
2. the output is k-CNF formulae $\phi_1, \cdots, \phi_t$ where $t \leq 2^{\epsilon \cdot n}$,
3. $\phi \in$ SAT if and only if $\phi_i \in$ SAT for some $1 \leq i \leq n$, and

Sparsification Lemma

LEMMA (Sparsification Lemma)

For all $k \in \mathbb{N}$ and $\epsilon \in (0, 1]$, there is a constant $c(k, \epsilon)$ and an algorithm which runs in $2^{\epsilon \cdot n} \cdot poly(m)$ time such that

1. the input is a k-CNF formula ϕ,
2. the output is k-CNF formulae $\phi_1, \cdots, \phi_t$ where $t \le 2^{\epsilon \cdot n}$,
3. $\phi \in$ SAT if and only if $\phi_i \in$ SAT for some $1 \le i \le n$, and
4. each ϕ_i has $\le c(k, \epsilon) \cdot n$ clauses.

Sparsification Lemma

LEMMA (Sparsification Lemma)

For all $k \in \mathbb{N}$ and $\epsilon \in (0, 1]$, there is a constant $c(k, \epsilon)$ and an algorithm which runs in $2^{\epsilon \cdot n} \cdot poly(m)$ time such that

1. the input is a k-CNF formula ϕ,
2. the output is k-CNF formulae $\phi_1, \cdots, \phi_t$ where $t \leq 2^{\epsilon \cdot n}$,
3. $\phi \in$ SAT if and only if $\phi_i \in$ SAT for some $1 \leq i \leq n$, and
4. each ϕ_i has $\leq c(k, \epsilon) \cdot n$ clauses.

- With the lemma, we're free to replace every m on the exponent with n. (HW)

Sparsification Lemma

LEMMA (Sparsification Lemma)

For all $k \in \mathbb{N}$ and $\epsilon \in (0,1]$, there is a constant $c(k,\epsilon)$ and an algorithm which runs in $2^{\epsilon \cdot n} \cdot poly(m)$ time such that

1. the input is a k-CNF formula ϕ,
2. the output is k-CNF formulae $\phi_1, \cdots, \phi_t$ where $t \leq 2^{\epsilon \cdot n}$,
3. $\phi \in \text{SAT}$ if and only if $\phi_i \in \text{SAT}$ for some $1 \leq i \leq n$, and
4. each ϕ_i has $\leq c(k,\epsilon) \cdot n$ clauses.

- With the lemma, we're free to replace every m on the exponent with n. (HW)
- As $poly(m)$ factor is redundant, we'll stop writing it explicitly from now on.

Exponential Time Hypothesis

The following bounds are known for exact algorithms for kSAT.

Exponential Time Hypothesis

The following bounds are known for exact algorithms for kSAT.

- Deterministic $O(1.3303^n)$ for 3SAT (Makino et al.)

Exponential Time Hypothesis

The following bounds are known for exact algorithms for kSAT.

- Deterministic $O(1.3303^n)$ for 3SAT (Makino et al.)
- Randomized $O(1.308^n)$ for 3SAT (Hertli)

Exponential Time Hypothesis

The following bounds are known for exact algorithms for kSAT.

- Deterministic $O(1.3303^n)$ for 3SAT (Makino et al.)
- Randomized $O(1.308^n)$ for 3SAT (Hertli)
- Determinisitc $O\left(\left(2 - \frac{1}{k+1}\right)^n\right)$ for kSAT (Dantsin et al.)

Exponential Time Hypothesis

The following bounds are known for exact algorithms for kSAT.

- Deterministic $O(1.3303^n)$ for 3SAT (Makino et al.)
- Randomized $O(1.308^n)$ for 3SAT (Hertli)
- Determinisitc $O\left(\left(2 - \frac{1}{k+1}\right)^n\right)$ for kSAT (Dantsin et al.)
- Randomized $O\left(2^{(1-1/k)\cdot n}\right)$ for kSAT (Paturi et al.)

Exponential Time Hypothesis

The following bounds are known for exact algorithms for kSAT.

- Deterministic $O(1.3303^n)$ for 3SAT (Makino et al.)
- Randomized $O(1.308^n)$ for 3SAT (Hertli)
- Determinisitc $O\left(\left(2 - \frac{1}{k+1}\right)^n\right)$ for kSAT (Dantsin et al.)
- Randomized $O\left(2^{(1-1/k)\cdot n}\right)$ for kSAT (Paturi et al.)

It seems likely that kSAT has a lowerbound of form $2^{s_k \cdot n}$ for some constant s_k.

Exponential Time Hypothesis

DEFINITION

For all $k \geq 1$, $s_k = \inf\{s \mid \text{there is an } 2^{s \cdot n} \text{ algorithm for } k\text{SAT}\}$

Exponential Time Hypothesis

DEFINITION

For all $k \geq 1$, $s_k = \inf\{s \mid \text{there is an } 2^{s \cdot n} \text{ algorithm for } k\text{SAT}\}$

We expect 3SAT to have an exponential lowerbound. Therefore, we hypothesize the following.

Exponential Time Hypothesis

DEFINITION

For all $k \geq 1$, $s_k = \inf\{s |$ there is an $2^{s \cdot n}$ algorithm for kSAT$\}$

We expect 3SAT to have an exponential lowerbound. Therefore, we hypothesize the following.

HYPOTHESIS

1. **Exponential Time Hypothesis**(ETH): For all $k \geq 3$, $s_k > 0$. As $s_3 \leq s_4 \leq \cdots$, it is equivalent to $s_3 > 0$.

Exponential Time Hypothesis

DEFINITION

For all $k \geq 1$, $s_k = \inf\{s |$ there is an $2^{s \cdot n}$ algorithm for kSAT$\}$

We expect 3SAT to have an exponential lowerbound. Therefore, we hypothesize the following.

HYPOTHESIS

1. **Exponential Time Hypothesis**(ETH): For all $k \geq 3$, $s_k > 0$. As $s_3 \leq s_4 \leq \cdots$, it is equivalent to $s_3 > 0$.
2. **Strong Exponential Time Hypothesis**(SETH): $\lim_{k \to \infty} s_k = 1$

Exponential Time Hypothesis

DEFINITION

For all $k \geq 1$, $s_k = \inf\{s \mid \text{ there is an } 2^{s \cdot n} \text{ algorithm for } k\text{SAT}\}$

We expect 3SAT to have an exponential lowerbound. Therefore, we hypothesize the following.

HYPOTHESIS

1. **Exponential Time Hypothesis**(ETH): For all $k \geq 3$, $s_k > 0$. As $s_3 \leq s_4 \leq \cdots$, it is equivalent to $s_3 > 0$.
2. **Strong Exponential Time Hypothesis**(SETH): $\lim_{k \to \infty} s_k = 1$

We'll mostly use ETH for the remaining lecture.

Exponential Time Hypothesis

THEOREM

Assume ETH holds.

Exponential Time Hypothesis

THEOREM

Assume ETH holds.

1. If 3SAT $\leq_p B_1 \leq_p \cdots \leq_p B_L$ with all of them having linear blowups, every algorithm for B_i runs in $2^{\Omega(n)}$ time for all $1 \leq i \leq L$.

Exponential Time Hypothesis

THEOREM

Assume ETH holds.

1. If 3SAT $\leq_p B_1 \leq_p \cdots \leq_p B_L$ with all of them having linear blowups, every algorithm for B_i runs in $2^{\Omega(n)}$ time for all $1 \leq i \leq L$.
2. If 3SAT $\leq_p B_1 \leq_p \cdots \leq_p B_L$ with exactly one of them having quadratic blowup and the rest having linear blowups, every algorithm for B_i runs in $2^{\Omega(\sqrt{n})}$ time for all $1 \leq i \leq L$.

Exponential Time Hypothesis

THEOREM

Assume ETH holds.

1. If 3SAT $\leq_p B_1 \leq_p \cdots \leq_p B_L$ with all of them having linear blowups, every algorithm for B_i runs in $2^{\Omega(n)}$ time for all $1 \leq i \leq L$.
2. If 3SAT $\leq_p B_1 \leq_p \cdots \leq_p B_L$ with exactly one of them having quadratic blowup and the rest having linear blowups, every algorithm for B_i runs in $2^{\Omega(\sqrt{n})}$ time for all $1 \leq i \leq L$.

- Last week, Karuna demonstrated to us a reduction 3SAT $\leq_p$ 3COL with linear blowup. Therefore, assuming ETH, every algorithm for 3COL runs in $2^{\Omega(n)}$ time.

Exponential Time Hypothesis

THEOREM

Assume ETH holds.

1. If 3SAT $\leq_p B_1 \leq_p \cdots \leq_p B_L$ with all of them having linear blowups, every algorithm for B_i runs in $2^{\Omega(n)}$ time for all $1 \leq i \leq L$.
2. If 3SAT $\leq_p B_1 \leq_p \cdots \leq_p B_L$ with exactly one of them having quadratic blowup and the rest having linear blowups, every algorithm for B_i runs in $2^{\Omega(\sqrt{n})}$ time for all $1 \leq i \leq L$.

- Last week, Karuna demonstrated to us a reduction 3SAT $\leq_p$ 3COL with linear blowup. Therefore, assuming ETH, every algorithm for 3COL runs in $2^{\Omega(n)}$ time.
- He also demonstrated to us a reduction 3COL $\leq_p$ Planar-3COL with quadratic blowup. Therefore, assuming ETH, every algorithm for Planar-3COL runs in $2^{\Omega(\sqrt{n})}$ time.

Exponential Time Hypothesis

THEOREM

Assume ETH holds.

1. If 3SAT $\leq_p B_1 \leq_p \cdots \leq_p B_L$ with all of them having linear blowups, every algorithm for B_i runs in $2^{\Omega(n)}$ time for all $1 \leq i \leq L$.
2. If 3SAT $\leq_p B_1 \leq_p \cdots \leq_p B_L$ with exactly one of them having quadratic blowup and the rest having linear blowups, every algorithm for B_i runs in $2^{\Omega(\sqrt{n})}$ time for all $1 \leq i \leq L$.

- Last week, Karuna demonstrated to us a reduction 3SAT $\leq_p$ 3COL with linear blowup. Therefore, assuming ETH, every algorithm for 3COL runs in $2^{\Omega(n)}$ time.
- He also demonstrated to us a reduction 3COL $\leq_p$ Planar-3COL with quadratic blowup. Therefore, assuming ETH, every algorithm for Planar-3COL runs in $2^{\Omega(\sqrt{n})}$ time.
- Few more examples will be in the HW.

Parameterized Complexity

Parameterized Complexity

Recall that the following problems are NP-Complete.

Parameterized Complexity

Recall that the following problems are NP-Complete.

- Vertex-Cover $= \{(G, k) | G$ has a vertex cover of size $k\}$.

Parameterized Complexity

Recall that the following problems are NP-Complete.

- Vertex-Cover $= \{(G, k) | G$ has a vertex cover of size $k\}$.
- Clique $= \{(G, k) | G$ has a clique of size $k\}$.

Parameterized Complexity

Recall that the following problems are NP-Complete.

- Vertex-Cover $= \{(G, k) | G$ has a vertex cover of size $k\}$.
- Clique $= \{(G, k) | G$ has a clique of size $k\}$.

For fixed $k \in \mathbb{N}$, we define

Parameterized Complexity

Recall that the following problems are NP-Complete.

- Vertex-Cover $= \{(G,k) | G$ has a vertex cover of size $k\}$.
- Clique $= \{(G,k) | G$ has a clique of size $k\}$.

For fixed $k \in \mathbb{N}$, we define

- Vertex-Cover$_k$ $= \{(G,k) | G$ has a vertex cover of size $k\}$.

Parameterized Complexity

Recall that the following problems are NP-Complete.

- Vertex-Cover $= \{(G, k) | G$ has a vertex cover of size $k\}$.
- Clique $= \{(G, k) | G$ has a clique of size $k\}$.

For fixed $k \in \mathbb{N}$, we define

- Vertex-Cover$_k$ $= \{(G, k) | G$ has a vertex cover of size $k\}$.
- Clique$_k$ $= \{(G, k) | G$ has a clique of size $k\}$.

Such problems are called **parameterized problems**.

Parameterized Complexity

Recall that the following problems are NP-Complete.

- Vertex-Cover $= \{(G, k) | G$ has a vertex cover of size $k\}$.
- Clique $= \{(G, k) | G$ has a clique of size $k\}$.

For fixed $k \in \mathbb{N}$, we define

- Vertex-Cover$_k$ $= \{(G, k) | G$ has a vertex cover of size $k\}$.
- Clique$_k$ $= \{(G, k) | G$ has a clique of size $k\}$.

Such problems are called **parameterized problems**.
Both of these can be solved in time $O(n^k)$ with bruteforce, i.e. they're in P.

Parameterized Complexity

Recall that the following problems are NP-Complete.

- Vertex-Cover $= \{(G, k) | G$ has a vertex cover of size $k\}$.
- Clique $= \{(G, k) | G$ has a clique of size $k\}$.

For fixed $k \in \mathbb{N}$, we define

- Vertex-Cover$_k$ $= \{(G, k) | G$ has a vertex cover of size $k\}$.
- Clique$_k$ $= \{(G, k) | G$ has a clique of size $k\}$.

Such problems are called **parameterized problems**.
Both of these can be solved in time $O(n^k)$ with bruteforce, i.e. they're in P.
However, we don't like the k in the exponent. Can we get rid of it?

Parameterized Complexity

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)
Let $G = (V, E)$ be the input graph. Consider a labelled rooted binary tree where each label is of form (S, T, e) where

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)
Let $G = (V, E)$ be the input graph. Consider a labelled rooted binary tree where each label is of form (S, T, e) where

- S is the set of unscanned vertices,

Parameterized Complexity

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)
Let $G = (V, E)$ be the input graph. Consider a labelled rooted binary tree where each label is of form (S, T, e) where

- S is the set of unscanned vertices,
- T is the set of vertices in the vertex cover of the induced subgraph of $V - S$, and

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)

Let $G = (V, E)$ be the input graph. Consider a labelled rooted binary tree where each label is of form (S, T, e) where

- S is the set of unscanned vertices,
- T is the set of vertices in the vertex cover of the induced subgraph of $V - S$, and
- e is an edge in the induced subgraph of S, if there's any.

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)

We recursively construct the tree as follows.

Parameterized Complexity

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)

We recursively construct the tree as follows.

1. The root has label $(V, \emptyset, e_0)$ where e_0 is an arbitrary edge.

Parameterized Complexity

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)

We recursively construct the tree as follows.

1. The root has label $(V, \emptyset, e_0)$ where e_0 is an arbitrary edge.
2. For a node with label (S, T, e),

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)

We recursively construct the tree as follows.

1. The root has label $(V, \emptyset, e_0)$ where e_0 is an arbitrary edge.
2. For a node with label (S, T, e),
3. if S is empty, T has to be a vertex cover of size $\leq k$, so return it.

Parameterized Complexity

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)

We recursively construct the tree as follows.

1. The root has label $(V, \emptyset, e_0)$ where e_0 is an arbitrary edge.
2. For a node with label (S, T, e),
3. if S is empty, T has to be a vertex cover of size $\leq k$, so return it.
4. Otherwise, if the depth of the node is $< k$, at least one endpoint of $e = (u, v)$ has to be in the vertex cover, so attach two childs with label $(S - \{u\}, T \cup \{u\}, e_l(\in S - \{u\}))$ and $(S - \{v\}, T \cup \{v\}, e_r(\in S - \{v\}))$. Note that at least one of u or v has to be in the final vertex cover, so this step is forced.

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)

We recursively construct the tree as follows.

1. The root has label $(V, \emptyset, e_0)$ where e_0 is an arbitrary edge.
2. For a node with label (S, T, e),
3. if S is empty, T has to be a vertex cover of size $\leq k$, so return it.
4. Otherwise, if the depth of the node is $< k$, at least one endpoint of $e = (u, v)$ has to be in the vertex cover, so attach two childs with label $(S - \{u\}, T \cup \{u\}, e_l(\in S - \{u\}))$ and $(S - \{v\}, T \cup \{v\}, e_r(\in S - \{v\}))$. Note that at least one of u or v has to be in the final vertex cover, so this step is forced.
5. If the algorithm fails to return a vertex cover after constructing the entire tree, report that there's no vertex cover of size $\leq k$.

THEOREM

Vertex-Cover$_k$ can be solved in $O(2^k \cdot n)$ time.

PROOF)
There are $O(2^k)$ nodes, each of which store $O(n)$ information. Therefore, the algorithm runs in $O(2^k \cdot n)$ time.
■

Parameterized Complexity

Is there a better result?

Parameterized Complexity

Is there a better result?
The answer is both "yes" and "no" depending on the meaning of "better".

Parameterized Complexity

Is there a better result?
The answer is both "yes" and "no" depending on the meaning of "better".
First, a "no".

Parameterized Complexity

Is there a better result?
The answer is both "yes" and "no" depending on the meaning of "better".
First, a "no".

THEOREM (Cai & Juedes)

If Vertex-Cover$_k$ can be solved in $2^{o(k)} \cdot n^L$ for some L, then 3SAT can be solved in $2^{o(k)}$ time, violating the ETH.

Parameterized Complexity

Is there a better result?
The answer is both "yes" and "no" depending on the meaning of "better".
First, a "no".

THEOREM (Cai & Juedes)

If Vertex-Cover$_k$ can be solved in $2^{o(k)} \cdot n^L$ for some L, then 3SAT can be solved in $2^{o(k)}$ time, violating the ETH.

We'll prove this in the upcoming lecture.

Kernelization

DEFINITION

- **Kernelization** is a procedure of reducing a parameterized problem $X(n, k)$ to down to a problem $X(f(k), g(k))$ for some function f bounded by a computable function in k and some function g in $n^{O(1)}$ preprocessing time.

Kernelization

DEFINITION

- **Kernelization** is a procedure of reducing a parameterized problem $X(n, k)$ to down to a problem $X(f(k), g(k))$ for some function f bounded by a computable function in k and some function g in $n^{O(1)}$ preprocessing time.
- The reduced problem is called **the kernel**, and $f(k)$ is the size of the kernel.

Kernelization

DEFINITION

- **Kernelization** is a procedure of reducing a parameterized problem $X(n, k)$ to down to a problem $X(f(k), g(k))$ for some function f bounded by a computable function in k and some function g in $n^{O(1)}$ preprocessing time.
- The reduced problem is called **the kernel**, and $f(k)$ is the size of the kernel.
- A problem is **kernelizable** if a kernelization exists.

Kernelization

DEFINITION

- **Kernelization** is a procedure of reducing a parameterized problem $X(n, k)$ to down to a problem $X(f(k), g(k))$ for some function f bounded by a computable function in k and some function g in $n^{O(1)}$ preprocessing time.
- The reduced problem is called **the kernel**, and $f(k)$ is the size of the kernel.
- A problem is **kernelizable** if a kernelization exists.

Now we'll look at a "yes".

Kernelization

THEOREM

Vertex-Cover$_k$ can be solved in $O(n + 2k^2 \cdot 2^{2k^2})$ time.

Kernelization

THEOREM

Vertex-Cover$_k$ can be solved in $O(n + 2k^2 \cdot 2^{2k^2})$ time.

PROOF)

Kernelization

THEOREM

Vertex-Cover$_k$ can be solved in $O(n + 2k^2 \cdot 2^{2k^2})$ time.

PROOF)

1. If a vertex u has degree $> k$, any vertex cover of size $\leq k$ must contain it. Remove u from the graph along with all edges incident to it, add it to the answer, then decrease k by 1. (This step takes $O(n)$ time.)

Kernelization

THEOREM

Vertex-Cover$_k$ can be solved in $O(n + 2k^2 \cdot 2^{2k^2})$ time.

PROOF)

1. If a vertex u has degree $> k$, any vertex cover of size $\leq k$ must contain it. Remove u from the graph along with all edges incident to it, add it to the answer, then decrease k by 1. (This step takes $O(n)$ time.)
2. If k becomes 0, before exhausting all edges, output that no vertex cover of size $\leq k$ exist.

Kernelization

THEOREM

Vertex-Cover$_k$ can be solved in $O(n + 2k^2 \cdot 2^{2k^2})$ time.

PROOF)

1. If a vertex u has degree $> k$, any vertex cover of size $\leq k$ must contain it. Remove u from the graph along with all edges incident to it, add it to the answer, then decrease k by 1. (This step takes $O(n)$ time.)
2. If k becomes 0, before exhausting all edges, output that no vertex cover of size $\leq k$ exist.
3. Now every vertex has degree $\leq k$. If there's a vertex cover of size $\leq k$, the number of edges in the graph is bounded by k^2. So if there are more than k^2 edges, output that no vertex cover of size $\leq k$ exist.

THEOREM

Vertex-Cover$_k$ can be solved in $O(n + 2k^2 \cdot 2^{2k^2})$ time.

PROOF)

1. If a vertex u has degree $> k$, any vertex cover of size $\leq k$ must contain it. Remove u from the graph along with all edges incident to it, add it to the answer, then decrease k by 1. (This step takes $O(n)$ time.)
2. If k becomes 0, before exhausting all edges, output that no vertex cover of size $\leq k$ exist.
3. Now every vertex has degree $\leq k$. If there's a vertex cover of size $\leq k$, the number of edges in the graph is bounded by k^2. So if there are more than k^2 edges, output that no vertex cover of size $\leq k$ exist.
4. Repeat the branching algorithm for the vertex cover. (This step takes $O(2k^2 \cdot 2^{2k^2}$ time.)

∎

Kernelization

THEOREM

Vertex-Cover$_k$ can be solved in $O(n + 2k^2 \cdot 2^{2k^2})$ time.

PROOF)

1. If a vertex u has degree $> k$, any vertex cover of size $\leq k$ must contain it. Remove u from the graph along with all edges incident to it, add it to the answer, then decrease k by 1. (This step takes $O(n)$ time.)
2. If k becomes 0, before exhausting all edges, output that no vertex cover of size $\leq k$ exist.
3. Now every vertex has degree $\leq k$. If there's a vertex cover of size $\leq k$, the number of edges in the graph is bounded by k^2. So if there are more than k^2 edges, output that no vertex cover of size $\leq k$ exist.
4. Repeat the branching algorithm for the vertex cover. (This step takes $O(2k^2 \cdot 2^{2k^2}$ time.)

■

(HW BOJ 20259)

Kernelization

Further results about vertex cover kernelizations:

Kernelization

Further results about vertex cover kernelizations:

- Lampis achieved a kernel of size $2k - c \cdot \log k$ for any constant c.

Kernelization

Further results about vertex cover kernelizations:

- Lampis achieved a kernel of size $2k - c \cdot \log k$ for any constant c.
- If there exists a kernelization of Vertex-Cover$_k$ with kernel of size $O(\log k)$, then P=NP. (HW)

Kernelization

Further results about vertex cover kernelizations:

- Lampis achieved a kernel of size $2k - c \cdot \log k$ for any constant c.
- If there exists a kernelization of Vertex-Cover$_k$ with kernel of size $O(\log k)$, then P=NP. (HW)
- If there exists $\epsilon > 0$ and a kernelization of Vertex-Cover$_k$ with kernel of size $O(2^{2-\epsilon})$, then coNP $\subseteq$ NP-P.

Fixed-Parameter Tractability

DEFINITION

A parameterized problem $X(n, k)$ is **fixed-parameter tractable** if there exists a computable function f such that $X(n, k)$ can be solved in time $f(k) \cdot n^{O(1)}$.

Fixed-Parameter Tractability

DEFINITION

A parameterized problem $X(n, k)$ is **fixed-parameter tractable** if there exists a computable function f such that $X(n, k)$ can be solved in time $f(k) \cdot n^{O(1)}$.
The set of fixed-parameter tractable problems is denoted by FPT.

Fixed-Parameter Tractability

DEFINITION

A parameterized problem $X(n,k)$ is **fixed-parameter tractable** if there exists a computable function f such that $X(n,k)$ can be solved in time $f(k) \cdot n^{O(1)}$.
The set of fixed-parameter tractable problems is denoted by FPT.

Recall that if a parameterized problem is kernelizble, it has an algorithm running in $n^{O(1)} + g(k)$ time for some function g, assuming it is decidable.

Fixed-Parameter Tractability

DEFINITION

A parameterized problem $X(n, k)$ is **fixed-parameter tractable** if there exists a computable function f such that $X(n, k)$ can be solved in time $f(k) \cdot n^{O(1)}$.
The set of fixed-parameter tractable problems is denoted by FPT.

Recall that if a parameterized problem is kernelizble, it has an algorithm running in $n^{O(1)} + g(k)$ time for some function g, assuming it is decidable.
The following theorem establishes an equivalence between fixed-parameter tractability and kernelizability.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Kernelizable & Decidable $\rightarrow$ Fixed-Parameter Tractable

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Kernelizable & Decidable $\rightarrow$ Fixed-Parameter Tractable

- First, run the kernelization algorithm on the input to obtain an input of size $f(k)$ in $n^{O(1)}$ time.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Kernelizable & Decidable $\rightarrow$ Fixed-Parameter Tractable

- First, run the kernelization algorithm on the input to obtain an input of size $f(k)$ in $n^{O(1)}$ time.
- Second, run the algorithm for the kernel problem in $g(f(k))$ time. (it exists by its decidablitiy)

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Kernelizable & Decidable $\rightarrow$ Fixed-Parameter Tractable

- First, run the kernelization algorithm on the input to obtain an input of size $f(k)$ in $n^{O(1)}$ time.
- Second, run the algorithm for the kernel problem in $g(f(k))$ time. (it exists by its decidablitiy)
- The total process runs in $n^{O(1)} + g(f(k))$ time. Since $g(f)$ is computable, every kernelizable and decidable parameterized problem is fixed-parameter tractable.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Fixed-Parameter Tractable $\rightarrow$ Kernelizable & Decidable

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Fixed-Parameter Tractable $\rightarrow$ Kernelizable & Decidable

- If the problem either accepts or rejects all inputs, then returning an empty input is a valid kernelization. Now assume neither are the case.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Fixed-Parameter Tractable $\rightarrow$ Kernelizable & Decidable

- If the problem either accepts or rejects all inputs, then returning an empty input is a valid kernelization. Now assume neither are the case.
- Let I_A be an accepting instance and I_R a rejecting instance of the problem.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Fixed-Parameter Tractable $\rightarrow$ Kernelizable & Decidable

- If the problem either accepts or rejects all inputs, then returning an empty input is a valid kernelization. Now assume neither are the case.
- Let I_A be an accepting instance and I_R a rejecting instance of the problem.
- Furthermore, let A be an algorithm for the problem that runs in $f(k) \cdot n^c$ time.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Fixed-Parameter Tractable $\rightarrow$ Kernelizable & Decidable

- If the problem either accepts or rejects all inputs, then returning an empty input is a valid kernelization. Now assume neither are the case.
- Let I_A be an accepting instance and I_R a rejecting instance of the problem.
- Furthermore, let A be an algorithm for the problem that runs in $f(k) \cdot n^c$ time.
- We can kernelize an input as follows.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Fixed-Parameter Tractable $\rightarrow$ Kernelizable & Decidable

- If the problem either accepts or rejects all inputs, then returning an empty input is a valid kernelization. Now assume neither are the case.
- Let I_A be an accepting instance and I_R a rejecting instance of the problem.
- Furthermore, let A be an algorithm for the problem that runs in $f(k) \cdot n^c$ time.
- We can kernelize an input as follows.
    1. Run A for the given input for at most n^{c+1} steps.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Fixed-Parameter Tractable $\rightarrow$ Kernelizable & Decidable

- If the problem either accepts or rejects all inputs, then returning an empty input is a valid kernelization. Now assume neither are the case.
- Let I_A be an accepting instance and I_R a rejecting instance of the problem.
- Furthermore, let A be an algorithm for the problem that runs in $f(k) \cdot n^c$ time.
- We can kernelize an input as follows.
  1. Run A for the given input for at most n^{c+1} steps.
  2. If A terminates, return I_A or I_R depending on the result.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Fixed-Parameter Tractable $\rightarrow$ Kernelizable & Decidable

- If the problem either accepts or rejects all inputs, then returning an empty input is a valid kernelization. Now assume neither are the case.
- Let I_A be an accepting instance and I_R a rejecting instance of the problem.
- Furthermore, let A be an algorithm for the problem that runs in $f(k) \cdot n^c$ time.
- We can kernelize an input as follows.
  1. Run A for the given input for at most n^{c+1} steps.
  2. If A terminates, return I_A or I_R depending on the result.
  3. Otherwise, return the input itself.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Fixed-Parameter Tractable $\rightarrow$ Kernelizable & Decidable

- The input returns itself if and only if $f(k) \cdot n^c > n^{c+1} \leftrightarrow f(k) > n$.

Fixed-Parameter Tractability

THEOREM

A parameterized problem is fixed-parameter tractable if and only if it is kernelizable and decidable.

PROOF) Fixed-Parameter Tractable $\rightarrow$ Kernelizable & Decidable

- The input returns itself if and only if $f(k) \cdot n^c > n^{c+1} \leftrightarrow f(k) > n$.
- Therefore, the size of the kernel is bounded by $\max(|I_A|, |I_R|, f(k))$, which is clearly computable.

■

Parameterized Reduction

Parameterized Reduction

DEFINITION

Let A and B be parameterized problems. A **parameterized reduction** of A onto B maps an instance (x, k) of A to an instance (x', k') of B such that

Parameterized Reduction

DEFINITION

Let A and B be parameterized problems. A **parameterized reduction** of A onto B maps an instance (x, k) of A to an instance (x', k') of B such that

1. x' depends only on x

Parameterized Reduction

DEFINITION

Let A and B be parameterized problems. A **parameterized reduction** of A onto B maps an instance (x, k) of A to an instance (x', k') of B such that

1. x' depends only on x
2. k' depends only on k

Parameterized Reduction

DEFINITION

Let A and B be parameterized problems. A **parameterized reduction** of A onto B maps an instance (x, k) of A to an instance (x', k') of B such that

1. x' depends only on x
2. k' depends only on k
3. The function that maps x to x' can be computed in polynomial time.

Parameterized Reduction

DEFINITION

Let A and B be parameterized problems. A **parameterized reduction** of A onto B maps an instance (x, k) of A to an instance (x', k') of B such that

1. x' depends only on x
2. k' depends only on k
3. The function that maps x to x' can be computed in polynomial time.
4. k' is bounded by a computable function of k.

Parameterized Reduction

DEFINITION

Let A and B be parameterized problems. A **parameterized reduction** of A onto B maps an instance (x, k) of A to an instance (x', k') of B such that

1. x' depends only on x
2. k' depends only on k
3. The function that maps x to x' can be computed in polynomial time.
4. k' is bounded by a computable function of k.
5. $(x, k) \in A$ if and only if $(x', k') \in B$

Parameterized Reduction

DEFINITION

Let A and B be parameterized problems. A **parameterized reduction** of A onto B maps an instance (x, k) of A to an instance (x', k') of B such that

1. x' depends only on x
2. k' depends only on k
3. The function that maps x to x' can be computed in polynomial time.
4. k' is bounded by a computable function of k.
5. $(x, k) \in A$ if and only if $(x', k') \in B$

Clearly, if $B \in$ FPT and A is parameterized reducible to B, then $A \in$ FPT.

Parameterized Reduction

- The standard reduction from Independent-Set$_k$ to Vertex-Cover$_k$ maps an instance (G, k) to $(G, n-k)$. As $n-k$ does not just depend on k, this is not a parameterized reduction.

Parameterized Reduction

- The standard reduction from Independent-Set$_k$ to Vertex-Cover$_k$ maps an instance (G, k) to $(G, n-k)$. As $n-k$ does not just depend on k, this is not a parameterized reduction.
- The standard reduction of Independent-Set$_k$ to Clique$_k$ maps (G, k) to (G, k), which is clearly a parameterized reduction.

Parameterized Reduction

- The standard reduction from Independent-Set$_k$ to Vertex-Cover$_k$ maps an instance (G, k) to $(G, n-k)$. As $n-k$ does not just depend on k, this is not a parameterized reduction.
- The standard reduction of Independent-Set$_k$ to Clique$_k$ maps (G, k) to (G, k), which is clearly a parameterized reduction.
- It is unlikely that there's a parameterized reduction from Independent-Set$_k$ to Vertex-Cover$_k$ since

Parameterized Reduction

- The standard reduction from Independent-Set$_k$ to Vertex-Cover$_k$ maps an instance (G, k) to $(G, n - k)$. As $n - k$ does not just depend on k, this is not a parameterized reduction.
- The standard reduction of Independent-Set$_k$ to Clique$_k$ maps (G, k) to (G, k), which is clearly a parameterized reduction.
- It is unlikely that there's a parameterized reduction from Independent-Set$_k$ to Vertex-Cover$_k$ since
  1. Vertex-Cover$_k$ $\in$ FPT and

Parameterized Reduction

- The standard reduction from Independent-Set$_k$ to Vertex-Cover$_k$ maps an instance (G, k) to $(G, n - k)$. As $n - k$ does not just depend on k, this is not a parameterized reduction.
- The standard reduction of Independent-Set$_k$ to Clique$_k$ maps (G, k) to (G, k), which is clearly a parameterized reduction.
- It is unlikely that there's a parameterized reduction from Independent-Set$_k$ to Vertex-Cover$_k$ since
  1. Vertex-Cover$_k$ $\in$ FPT and
  2. Chen et al. showed that Clique$_k$ $\in$ FPT implies ETH being false.

Preview Of The Next Lecture

Preview Of The Next Lecture

- Complexity Class W[1], W[1]-Complete, and W[1]-Hard

Preview Of The Next Lecture

- Complexity Class W[1], W[1]-Complete, and W[1]-Hard
- W-Hierarchy

Preview Of The Next Lecture

- Complexity Class W[1], W[1]-Complete, and W[1]-Hard
- W-Hierarchy
- Proof of the lowerbound for parameterized problems with ETH

Preview Of The Next Lecture

- Complexity Class W[1], W[1]-Complete, and W[1]-Hard
- W-Hierarchy
- Proof of the lowerbound for parameterized problems with ETH
- And more :)

The End