Technische Universität Dresden

International Center for Computational Logic

COMPLEXITY THEORY

Lecture 29: Parameterized Complexity

Sergei Obiedkov

Knowledge-Based Systems

TU Dresden, 2 Feb 2026

Vertex Cover

Vertex Cover

Input:	An undirected graph $G = (V, E)$ and a natural number k
Problem:	Does G contain k vertices that touch all edges (vertex cover)?

- A solution is a subset $V' \subseteq V$ of size k.

Vertex Cover

Vertex Cover

Input: An undirected graph $G = (V, E)$ and a natural number k

Problem: Does G contain k vertices that touch all edges (vertex cover)?

- A solution is a subset $V' \subseteq V$ of size k.
- Brute-force search: $\binom{n}{k}$ possible solutions to check, where $n = |V|$.

Vertex Cover

Vertex Cover

Input: An undirected graph $G = (V, E)$ and a natural number k

Problem: Does G contain k vertices that touch all edges (vertex cover)?

- A solution is a subset $V' \subseteq V$ of size k.
- Brute-force search: $\binom{n}{k}$ possible solutions to check, where $n = |V|$.
- For fixed k,

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k!} = \Theta(n^k).$$

Vertex Cover

Input:	An undirected graph $G = (V, E)$ and a natural number k
Problem:	Does G contain k vertices that touch all edges (vertex cover)?

- A solution is a subset $V' \subseteq V$ of size k.
- Brute-force search: $\binom{n}{k}$ possible solutions to check, where $n = |V|$.
- For fixed k,

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k!} = \Theta(n^k).$$

- For $k = n/2$, this is exponential in n:

$$\binom{n}{k} = \binom{n}{n/2} \geq \frac{2^n}{n+1}.$$

Kernelization

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?
  - Remove isolated vertices from G.

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?
  - Remove isolated vertices from G.
- What vertices must be in every vertex cover of size k?

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?
  - Remove isolated vertices from G.
- What vertices must be in every vertex cover of size k?
  - Include a vertex with degree $> k$ into a vertex cover, remove it from G, and decrement k.

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?
  - Remove isolated vertices from G.
- What vertices must be in every vertex cover of size k?
  - Include a vertex with degree $> k$ into a vertex cover, remove it from G, and decrement k.
- Apply these reduction rules until $1 \leq degree(v) \leq k$ for every $v \in V$.

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?
  - Remove isolated vertices from G.
- What vertices must be in every vertex cover of size k?
  - Include a vertex with degree $> k$ into a vertex cover, remove it from G, and decrement k.
- Apply these reduction rules until $1 \leq degree(v) \leq k$ for every $v \in V$.
- How many edges can be covered by k vertices in the resulting graph?

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?
  - Remove isolated vertices from G.
- What vertices must be in every vertex cover of size k?
  - Include a vertex with degree $> k$ into a vertex cover, remove it from G, and decrement k.
- Apply these reduction rules until $1 \leq degree(v) \leq k$ for every $v \in V$.
- How many edges can be covered by k vertices in the resulting graph?
  - At most k^2. So, reject if $|E| > k^2$.

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?
  - Remove isolated vertices from G.
- What vertices must be in every vertex cover of size k?
  - Include a vertex with degree $> k$ into a vertex cover, remove it from G, and decrement k.
- Apply these reduction rules until $1 \leq degree(v) \leq k$ for every $v \in V$.
- How many edges can be covered by k vertices in the resulting graph?
  - At most k^2. So, reject if $|E| > k^2$.
- If this graph has a vertex cover $S \subseteq V$ of size k, how many vertices can V contain?

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?
  - Remove isolated vertices from G.
- What vertices must be in every vertex cover of size k?
  - Include a vertex with degree $> k$ into a vertex cover, remove it from G, and decrement k.
- Apply these reduction rules until $1 \leq degree(v) \leq k$ for every $v \in V$.
- How many edges can be covered by k vertices in the resulting graph?
  - At most k^2. So, reject if $|E| > k^2$.
- If this graph has a vertex cover $S \subseteq V$ of size k, how many vertices can V contain?
  - $|V \setminus S| \leq k|S| = k^2 \quad \Rightarrow \quad |V| \leq k^2 + k$. So, reject if $|V| > k^2 + k$.

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?
  - Remove isolated vertices from G.
- What vertices must be in every vertex cover of size k?
  - Include a vertex with degree $> k$ into a vertex cover, remove it from G, and decrement k.
- Apply these reduction rules until $1 \leq degree(v) \leq k$ for every $v \in V$.
- How many edges can be covered by k vertices in the resulting graph?
  - At most k^2. So, reject if $|E| > k^2$.
- If this graph has a vertex cover $S \subseteq V$ of size k, how many vertices can V contain?
  - $|V \setminus S| \leq k|S| = k^2 \quad \Rightarrow \quad |V| \leq k^2 + k$. So, reject if $|V| > k^2 + k$.
- We have obtained a kernel with $O(k^2)$ vertices and $O(k^2)$ edges.

Simplify by Preprocessing

Idea: Simplify the problem by making G smaller.

- What vertices are useless in a vertex cover?
  - Remove isolated vertices from G.
- What vertices must be in every vertex cover of size k?
  - Include a vertex with degree $> k$ into a vertex cover, remove it from G, and decrement k.
- Apply these reduction rules until $1 \le degree(v) \le k$ for every $v \in V$.
- How many edges can be covered by k vertices in the resulting graph?
  - At most k^2. So, reject if $|E| > k^2$.
- If this graph has a vertex cover $S \subseteq V$ of size k, how many vertices can V contain?
  - $|V \setminus S| \le k|S| = k^2 \quad \Rightarrow \quad |V| \le k^2 + k$. So, reject if $|V| > k^2 + k$.
- We have obtained a kernel with $O(k^2)$ vertices and $O(k^2)$ edges.
- Brute-force search needs to consider only $\binom{k^2+k}{k} = 2^{O(k \log k)}$ possible solutions.

Bounded Search Trees

Edge-Based Recursion

For $G = (V, E)$ and $u \in V$:

$$V_u = V \setminus \{u\} \qquad E_u = E \cap V_u^2 \qquad G_u = (V_u, E_u).$$

For any $(u, v) \in E$, graph G has a vertex cover of size k if and only if there is a vertex cover of size $k - 1$ for graph G_u or graph G_v.

Edge-Based Recursion

For $G = (V, E)$ and $u \in V$:

$$V_u = V \setminus \{u\} \qquad E_u = E \cap V_u^2 \qquad G_u = (V_u, E_u).$$

For any $(u, v) \in E$, graph G has a vertex cover of size k if and only if there is a vertex cover of size $k - 1$ for graph G_u or graph G_v.

Proof:

$\Rightarrow$ Let S be a vertex cover of G and $|S| = k$. Then $u \in S$ or $v \in S$. Assume $u \in S$. There are no edges incident to u in $E_u \subseteq E$. Hence, $S \setminus \{u\}$ is a vertex cover of G_u.

Edge-Based Recursion

For $G = (V, E)$ and $u \in V$:

$$V_u = V \setminus \{u\} \qquad E_u = E \cap V_u^2 \qquad G_u = (V_u, E_u).$$

For any $(u, v) \in E$, graph G has a vertex cover of size k if and only if there is a vertex cover of size $k - 1$ for graph G_u or graph G_v.

Proof:

$\Rightarrow$ Let S be a vertex cover of G and $|S| = k$. Then $u \in S$ or $v \in S$. Assume $u \in S$. There are no edges incident to u in $E_u \subseteq E$. Hence, $S \setminus \{u\}$ is a vertex cover of G_u.

$\Leftarrow$ Let S_u be a vertex cover of G_u and $|S_u| = k - 1$. Then, for every edge $(u', v') \in E$:

- $(u', v') \in E_u \quad \Rightarrow \quad u' \in S_u$ or $v' \in S_u$
- $(u', v') \notin E_u \quad \Rightarrow \quad u \in \{u', v'\}$

Hence, $S_u \cup \{u\}$ is a vertex cover of G.

Edge-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- If $E = \emptyset$, return $\emptyset$.
- If $k = 0$, report that there is no cover of size $\leq k$.
- Select an edge $(u, v) \in E$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - 1$ for G_v.
  If found, return $S \cup \{v\}$.
- Report that there is no cover of size $\leq k$.

Edge-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- If $E = \varnothing$, return $\varnothing$.
- If $k = 0$, report that there is no cover of size $\leq k$.
- Select an edge $(u, v) \in E$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.

  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - 1$ for G_v.

  If found, return $S \cup \{v\}$.
- Report that there is no cover of size $\leq k$.

- The execution of the algorithm follows a complete binary tree of height k

Edge-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- If $E = \emptyset$, return $\emptyset$.
- If $k = 0$, report that there is no cover of size $\leq k$.
- Select an edge $(u, v) \in E$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - 1$ for G_v.
  If found, return $S \cup \{v\}$.
- Report that there is no cover of size $\leq k$.

- The execution of the algorithm follows a complete binary tree of height k
- Running time: $O(2^k|E|)$

Edge-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- If $E = \emptyset$, return $\emptyset$.
- If $k = 0$, report that there is no cover of size $\leq k$.
- Select an edge $(u, v) \in E$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - 1$ for G_v.
  If found, return $S \cup \{v\}$.
- Report that there is no cover of size $\leq k$.

- The execution of the algorithm follows a complete binary tree of height k
- Running time: $O(2^k|E|)$, or $O(2^k k^2)$ if we have already applied kernelization

Bounded Search Trees

- Let μ be a function associating an instance of an optimization problem with an integer indicating how hard the instance is.
- Let I be an instance of such a problem.
- In a branching step, generate instances $I_1, \ldots, I_\ell$ such that
  1. For all i, a feasible solution S of I_i corresponds to a feasible solution $h_i(S)$ of I;
  2. For some i and some feasible solution S of I_i, a solution $h_i(S)$ is optimal for I;
  3. The number $\ell > 1$ is small, e.g., bounded by a function of $\mu(I)$ alone;
  4. For all i, we have $\mu(I_i) \leq \mu(I) - c$ for some constant $c > 0$.
- We obtain a bounded search tree whose branching is controlled by condition 3 and depth is controlled by condition 4.

Edge-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- If $E = \varnothing$, return $\varnothing$.
- If $k = 0$, report that there is no cover of size $\leq k$.
- Select an edge $(u, v) \in E$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - 1$ for G_v.
  If found, return $S \cup \{v\}$.
- Report that there is no cover of size $\leq k$.

- Running time: $O(2^k|E|)$, or $O(2^k k^2)$ if we have already applied kernelization

Edge-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- If $E = \varnothing$, return $\varnothing$.
- If $k = 0$, report that there is no cover of size $\leq k$.
- Select an edge $(u, v) \in E$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - 1$ for G_v.
  If found, return $S \cup \{v\}$.
- Report that there is no cover of size $\leq k$.

- Running time: $O(2^k|E|)$, or $O(2^k k^2)$ if we have already applied kernelization

Can we use simpler subproblems?

Edge-Based vs Vertex-Based Recursion

For $G = (V, E)$ and $u \in V$:

$$V_u = V \setminus \{u\} \qquad E_u = E \cap V_u^2 \qquad G_u = (V_u, E_u).$$

For any $(u, v) \in E$, graph G has a vertex cover of size k if and only if there is a vertex cover of size $k - 1$ for graph G_u or graph G_v.

Edge-Based vs Vertex-Based Recursion

For $G = (V, E)$ and $u \in V$:

$$V_u = V \setminus \{u\} \qquad E_u = E \cap V_u^2 \qquad G_u = (V_u, E_u).$$

For any $(u, v) \in E$, graph G has a vertex cover of size k if and only if there is a vertex cover of size $k - 1$ for graph G_u or graph G_v.

For $G = (V, E)$ and $U \subseteq V$:

$$V_U = V \setminus U \qquad E_U = E \cap V_U^2 \qquad G_U = (V_U, E_U).$$

For any $u \in V$, graph G has a vertex cover of size k if and only if there is a vertex cover of size $k - 1$ for graph G_u or a vertex cover of size $k - |N(u)|$ in graph $G_{N(u)}$, where $N(u) = \{v \in V \mid (u, v) \in E\}$.

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} degree(v)$
- If $degree(u) < 2$, solve in linear time.
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.

  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.

  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} degree(v)$
- If $degree(u) < 2$, solve in linear time.
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.

  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.

  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

- Running time: the number of nodes in the tree $\times\ O(|E|)$
- How many nodes are there in this tree?

Vertex-Based Recursion

- Running time: the number of nodes in the tree $\times$ $O(|E|)$

Vertex-Based Recursion

- Running time: the number of nodes in the tree $\times\ O(|E|)$
- Such a tree with ℓ leaves contains $\leq 2\ell - 1$ nodes.

Vertex-Based Recursion

- Running time: the number of nodes in the tree $\times$ $O(|E|)$
- Such a tree with ℓ leaves contains $\leq 2\ell - 1$ nodes.
- The number of leaves in a tree obtained with the parameter k is at most

$$T(k) = \begin{cases} T(k-1) + T(k-2) & \text{if } k > 1; \\ 2 & \text{otherwise.} \end{cases}$$

Vertex-Based Recursion

- Running time: the number of nodes in the tree $\times\ O(|E|)$
- Such a tree with ℓ leaves contains $\leq 2\ell - 1$ nodes.
- The number of leaves in a tree obtained with the parameter k is at most

$$T(k) = \begin{cases} T(k-1) + T(k-2) & \text{if } k > 1; \\ 2 & \text{otherwise.} \end{cases}$$

- To have $T(k) \leq c\lambda^k$ for some constants $c > 0$ and $\lambda > 1$, it suffices that, for $k > 1$,

$$T(k) = T(k-1) + T(k-2) \leq c\lambda^{k-1} + c\lambda^{k-2} \leq c\lambda^k.$$

Vertex-Based Recursion

- Running time: the number of nodes in the tree $\times\ O(|E|)$
- Such a tree with ℓ leaves contains $\leq 2\ell - 1$ nodes.
- The number of leaves in a tree obtained with the parameter k is at most

$$T(k) = \begin{cases} T(k-1) + T(k-2) & \text{if } k > 1; \\ 2 & \text{otherwise.} \end{cases}$$

- To have $T(k) \leq c\lambda^k$ for some constants $c > 0$ and $\lambda > 1$, it suffices that, for $k > 1$,

$$T(k) = T(k-1) + T(k-2) \leq c\lambda^{k-1} + c\lambda^{k-2} \leq c\lambda^k.$$

- This holds when $\lambda + 1 \leq \lambda^2$.

Vertex-Based Recursion

- Running time: the number of nodes in the tree $\times$ $O(|E|)$
- Such a tree with ℓ leaves contains $\leq 2\ell - 1$ nodes.
- The number of leaves in a tree obtained with the parameter k is at most

$$T(k) = \begin{cases} T(k-1) + T(k-2) & \text{if } k > 1; \\ 2 & \text{otherwise.} \end{cases}$$

- To have $T(k) \leq c\lambda^k$ for some constants $c > 0$ and $\lambda > 1$, it suffices that, for $k > 1$,

$$T(k) = T(k-1) + T(k-2) \leq c\lambda^{k-1} + c\lambda^{k-2} \leq c\lambda^k.$$

- This holds when $\lambda + 1 \leq \lambda^2$.
- The smallest λ satisfying this is $\dfrac{1 + \sqrt{5}}{2} < 1.6181$.

Vertex-Based Recursion

- Running time: the number of nodes in the tree $\times\ O(|E|)$
- Such a tree with ℓ leaves contains $\leq 2\ell - 1$ nodes.
- The number of leaves in a tree obtained with the parameter k is at most

$$T(k) = \begin{cases} T(k-1) + T(k-2) & \text{if } k > 1; \\ 2 & \text{otherwise.} \end{cases}$$

- To have $T(k) \leq c\lambda^k$ for some constants $c > 0$ and $\lambda > 1$, it suffices that, for $k > 1$,

$$T(k) = T(k-1) + T(k-2) \leq c\lambda^{k-1} + c\lambda^{k-2} \leq c\lambda^k.$$

- This holds when $\lambda + 1 \leq \lambda^2$.
- The smallest λ satisfying this is $\dfrac{1+\sqrt{5}}{2} < 1.6181$.
- This works if we set $c = 2$; then, $T(0) = 2 = 2 \cdot 1.6181^0$ and $T(1) = 2 \leq 2 \cdot 1.6181^1$.

Vertex-Based Recursion

- Running time: the number of nodes in the tree $\times\, O(|E|)$
- Such a tree with ℓ leaves contains $\leq 2\ell - 1$ nodes.
- The number of leaves in a tree obtained with the parameter k is at most

$$T(k) = \begin{cases} T(k-1) + T(k-2) & \text{if } k > 1; \\ 2 & \text{otherwise.} \end{cases}$$

- To have $T(k) \leq c\lambda^k$ for some constants $c > 0$ and $\lambda > 1$, it suffices that, for $k > 1$,

$$T(k) = T(k-1) + T(k-2) \leq c\lambda^{k-1} + c\lambda^{k-2} \leq c\lambda^k.$$

- This holds when $\lambda + 1 \leq \lambda^2$.
- The smallest λ satisfying this is $\dfrac{1+\sqrt{5}}{2} < 1.6181$.
- This works if we set $c = 2$; then, $T(0) = 2 = 2 \cdot 1.6181^0$ and $T(1) = 2 \leq 2 \cdot 1.6181^1$.
- Runtime: $O(1.6181^k|E|)$, or $O(1.6181^k k^2)$ if we have already applied kernelization.

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} \mathit{degree}(v)$
- If $\mathit{degree}(u) < 2$, solve in linear time.
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.

  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.

  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

Can we use simpler subproblems?

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} \mathit{degree}(v)$
- If $\mathit{degree}(u) < 3$, solve in linear time. How?
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.
  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} degree(v)$
- If $degree(u) < 3$, solve in linear time. How?
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.
  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

- $T(k) = T(k-1) + T(k-3)$

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} degree(v)$
- If $degree(u) < 3$, solve in linear time. How?
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.
  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

- $T(k) = T(k-1) + T(k-3)$
- Runtime: $O(1.4656^k |E|)$, or $O(1.4656^k k^2)$ if we have already applied kernelization.

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} degree(v)$
- If $degree(u) < 3$, solve in linear time. How?
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.
  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

- $T(k) = T(k-1) + T(k-3)$
- Runtime: $O(1.4656^k |E|)$, or $O(1.4656^k k^2)$ if we have already applied kernelization.

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} degree(v)$
- If $degree(u) < 3$, solve in linear time. How?
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.
  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

- $T(k) = T(k-1) + T(k-3)$
- Runtime: $O(1.4656^k|E|)$, or $O(1.4656^k k^2)$ if we have already applied kernelization.

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} degree(v)$
- If $degree(u) < 3$, solve in linear time. How?
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.
  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

- $T(k) = T(k-1) + T(k-3)$
- Runtime: $O(1.4656^k|E|)$, or $O(1.4656^k k^2)$ if we have already applied kernelization.

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} degree(v)$
- If $degree(u) < 3$, solve in linear time. How?
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.
  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

- $T(k) = T(k-1) + T(k-3)$
- Runtime: $O(1.4656^k |E|)$, or $O(1.4656^k k^2)$ if we have already applied kernelization.

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} degree(v)$
- If $degree(u) < 3$, solve in linear time. How?
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.
  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.
  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

- $T(k) = T(k-1) + T(k-3)$
- Runtime: $O(1.4656^k |E|)$, or $O(1.4656^k k^2)$ if we have already applied kernelization.

Vertex-Based Recursion

Branching Algorithm

Input: $G = (V, E), k \in \mathbb{N}$.

Output: A vertex cover of graph G of size $\leq k$ if exists.

- $u := \arg\max_{v \in V} degree(v)$
- If $degree(u) < 3$, solve in linear time. How?
- If $k \leq 0$, report that there is no cover of size $\leq k$.
- Recursively find a cover S of size $\leq k - 1$ for G_u.

  If found, return $S \cup \{u\}$.
- Recursively find a cover S of size $\leq k - |N(u)|$ for $G_{N(u)}$.

  If found, return $S \cup N(u)$.
- Report that there is no cover of size $\leq k$.

- $T(k) = T(k-1) + T(k-3)$
- Runtime: $O(1.4656^k|E|)$, or $O(1.4656^k k^2)$ if we have already applied kernelization.

Kernels and Fixed-Parameter Tractability

Kernel

Definition 29.1: A parameterized problem is a language $\mathbf{L} \subseteq \Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. For $(x, k) \in \Sigma^* \times \mathbb{N}$, the number k is the parameter.

Kernel

Definition 29.1: A parameterized problem is a language $\mathbf{L} \subseteq \Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. For $(x, k) \in \Sigma^* \times \mathbb{N}$, the number k is the parameter.

Definition 29.2: A kernel for a parameterized problem $\mathbf{L} \subseteq \Sigma^* \times \mathbb{N}$ is a function K computable in polynomial time that maps an instance (x, k) to an equivalent instance (x', k')

$$(x, k) \in \mathbf{L} \iff K(x, k) \in \mathbf{L}$$

such that $k' \leq k$ and $|x'| \leq s(k)$, where s is some computable function.

Kernel

Definition 29.1: A parameterized problem is a language $\mathbf{L} \subseteq \Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. For $(x, k) \in \Sigma^* \times \mathbb{N}$, the number k is the parameter.

Definition 29.2: A kernel for a parameterized problem $\mathbf{L} \subseteq \Sigma^* \times \mathbb{N}$ is a function K computable in polynomial time that maps an instance (x, k) to an equivalent instance (x', k')

$$(x, k) \in \mathbf{L} \iff K(x, k) \in \mathbf{L}$$

such that $k' \leq k$ and $|x'| \leq s(k)$, where s is some computable function.

Vertex Cover has a kernel with at most $k(k+1)$ vertices and at most k^2 edges.

Kernel for Independent Set

Independent Set

Input: An undirected graph G and a natural number k

Problem: Does G contain k vertices that share no edges (independent set)?

We'll use an additional parameter: the maximum degree d of a vertex.

Kernel for Independent Set

Independent Set

Input:	An undirected graph G and a natural number k
Problem:	Does G contain k vertices that share no edges (independent set)?

We'll use an additional parameter: the maximum degree d of a vertex.

Any graph with $\geq k(d+1)$ vertices has an independent set of size k.

Kernel for Independent Set

Independent Set

Input: An undirected graph G and a natural number k

Problem: Does G contain k vertices that share no edges (independent set)?

We'll use an additional parameter: the maximum degree d of a vertex.

Any graph with $\geq k(d+1)$ vertices has an independent set of size k.

Accept if $n \geq k(d+1)$; otherwise, solve by brute-force search.

Kernel for Independent Set

Independent Set

Input: An undirected graph G and a natural number k

Problem: Does G contain k vertices that share no edges (independent set)?

We'll use an additional parameter: the maximum degree d of a vertex.

Any graph with $\geq k(d + 1)$ vertices has an independent set of size k.

Accept if $n \geq k(d + 1)$; otherwise, solve by brute-force search.

- Kernel: a fixed yes-instance or the (small) graph itself

Kernel for INDEPENDENT SET

INDEPENDENT SET

Input:	An undirected graph G and a natural number k
Problem:	Does G contain k vertices that share no edges (independent set)?

We'll use an additional parameter: the maximum degree d of a vertex.

Any graph with $\geq k(d+1)$ vertices has an independent set of size k.

Accept if $n \geq k(d+1)$; otherwise, solve by brute-force search.

- Kernel: a fixed yes-instance or the (small) graph itself
- Running time: $O(n)$ for counting vertices $+ f(k,d)$ for brute-force search

Kernel for Independent Set

Independent Set

Input:	An undirected graph G and a natural number k
Problem:	Does G contain k vertices that share no edges (independent set)?

We'll use an additional parameter: the maximum degree d of a vertex.

Any graph with $\geq k(d+1)$ vertices has an independent set of size k.

Accept if $n \geq k(d+1)$; otherwise, solve by brute-force search.

- Kernel: a fixed yes-instance or the (small) graph itself
- Running time: $O(n)$ for counting vertices + $f(k, d)$ for brute-force search FPT

The class FPT

Definition 29.3: A parameterized problem $\mathbf{L} \subseteq \Sigma^* \times \mathbb{N}$ is fixed-parameter tractable if there exist a constant c, a computable function $f : \mathbb{N} \to \mathbb{N}$, and an algorithm that correctly decides whether $(x, k) \in \mathbf{L}$ in time bounded by

$$f(k) \cdot |(x, k)|^c.$$

FPT is the class of all fixed-parameter tractable problems.

The class FPT

Definition 29.3: A parameterized problem $\mathbf{L} \subseteq \Sigma^* \times \mathbb{N}$ is fixed-parameter tractable if there exist a constant c, a computable function $f : \mathbb{N} \to \mathbb{N}$, and an algorithm that correctly decides whether $(x, k) \in \mathbf{L}$ in time bounded by

$$f(k) \cdot |(x, k)|^c.$$

FPT is the class of all fixed-parameter tractable problems.

P $\subseteq$ FPT

The class FPT

Definition 29.3: A parameterized problem $\mathbf{L} \subseteq \Sigma^* \times \mathbb{N}$ is fixed-parameter tractable if there exist a constant c, a computable function $f: \mathbb{N} \to \mathbb{N}$, and an algorithm that correctly decides whether $(x, k) \in \mathbf{L}$ in time bounded by

$$f(k) \cdot |(x, k)|^c.$$

FPT is the class of all fixed-parameter tractable problems.

P $\subseteq$ FPT

If a decidable problem $\mathbf{L}$ has a kernel, then $\mathbf{L} \in$ FPT.

FPT and Kernels

Theorem 29.4: Every problem in FPT has a kernel.

FPT and Kernels

Theorem 29.4: Every problem in FPT has a kernel.

Proof: Let **L** $\in$ FPT, and let A be an algorithm for **L** with running time $\leq f(k) \cdot |(x,k)|^c$.

FPT and Kernels

Theorem 29.4: Every problem in FPT has a kernel.

Proof: Let **L** $\in$ FPT, and let A be an algorithm for **L** with running time $\leq f(k) \cdot |(x,k)|^c$.

Kernel for (x,k)

- Let $A(x,k)$ run for time $|(x,k)|^{c+1}$
- If it terminates and accepts, return some $x \in$ **L**.
- If it terminates and rejects, return some $x \notin$ **L**.
- Otherwise, return (x,k).

FPT and Kernels

Theorem 29.4: Every problem in FPT has a kernel.

Proof: Let **L** $\in$ FPT, and let A be an algorithm for **L** with running time $\leq f(k) \cdot |(x,k)|^c$.

Kernel for (x,k)

- Let $A(x,k)$ run for time $|(x,k)|^{c+1}$
- If it terminates and accepts, return some $x \in$ **L**.
- If it terminates and rejects, return some $x \notin$ **L**.
- Otherwise, return (x,k).

- The output instance is computed in polynomial time and is equivalent to (x,k).

FPT and Kernels

Theorem 29.4: Every problem in FPT has a kernel.

Proof: Let **L** $\in$ FPT, and let A be an algorithm for **L** with running time $\leq f(k) \cdot |(x,k)|^c$.

Kernel for (x,k)

- Let $A(x,k)$ run for time $|(x,k)|^{c+1}$
- If it terminates and accepts, return some $x \in$ **L**.
- If it terminates and rejects, return some $x \notin$ **L**.
- Otherwise, return (x,k).

- The output instance is computed in polynomial time and is equivalent to (x,k).
- If the algorithm terminates, the size of the output is constant.

FPT and Kernels

Theorem 29.4: Every problem in FPT has a kernel.

Proof: Let **L** $\in$ FPT, and let A be an algorithm for **L** with running time $\leq f(k) \cdot |(x,k)|^c$.

Kernel for (x,k)

- Let $A(x,k)$ run for time $|(x,k)|^{c+1}$
- If it terminates and accepts, return some $x \in$ **L**.
- If it terminates and rejects, return some $x \notin$ **L**.
- Otherwise, return (x,k).

- The output instance is computed in polynomial time and is equivalent to (x,k).
- If the algorithm terminates, the size of the output is constant.
- If not:

$$\begin{aligned} |(x,k)|^{c+1} &< f(k) \cdot |(x,k)|^c \\ |(x,k)| &< f(k) \end{aligned}$$

Slice-wise Polynomial Problems

The class XP

Definition 29.5: A parameterized problem **L** $\subseteq \Sigma^* \times \mathbb{N}$ is slice-wise polynomial if there exist two computable functions $f, g\colon \mathbb{N} \to \mathbb{N}$, and an algorithm that correctly decides whether $(x, k) \in$ **L** in time bounded by

$$f(k) \cdot |(x, k)|^{g(k)}.$$

XP is the class of all slice-wise polynomial problems.

The class XP

Definition 29.5: A parameterized problem $\mathbf{L} \subseteq \Sigma^* \times \mathbb{N}$ is slice-wise polynomial if there exist two computable functions $f, g \colon \mathbb{N} \to \mathbb{N}$, and an algorithm that correctly decides whether $(x, k) \in \mathbf{L}$ in time bounded by

$$f(k) \cdot |(x, k)|^{g(k)}.$$

XP is the class of all slice-wise polynomial problems.

- Polynomial for each fixed k
- Degree depends on k

The class XP

Definition 29.5: A parameterized problem **L** $\subseteq \Sigma^* \times \mathbb{N}$ is slice-wise polynomial if there exist two computable functions $f, g\colon \mathbb{N} \to \mathbb{N}$, and an algorithm that correctly decides whether $(x, k) \in$ **L** in time bounded by

$$f(k) \cdot |(x, k)|^{g(k)}.$$

XP is the class of all slice-wise polynomial problems.

- Polynomial for each fixed k
- Degree depends on k

P $\subseteq$ FPT $\subseteq$ XP

The class XP

Definition 29.5: A parameterized problem **L** $\subseteq \Sigma^* \times \mathbb{N}$ is slice-wise polynomial if there exist two computable functions $f, g \colon \mathbb{N} \to \mathbb{N}$, and an algorithm that correctly decides whether $(x, k) \in$ **L** in time bounded by

$$f(k) \cdot |(x, k)|^{g(k)}.$$

XP is the class of all slice-wise polynomial problems.

- Polynomial for each fixed k
- Degree depends on k

P $\subseteq$ FPT $\subseteq$ XP

Example 29.6:

- **Clique**: Given G, k, does G contain a clique of size k?
- Brute force: $O(n^k) \Rightarrow$ in XP
- Believed not to be in FPT

LP-Based Kernel for Vertex Cover

VERTEX COVER as an Integer Linear Program

VERTEX COVER

Input:	An undirected graph $G = (V, E)$ and a natural number k
Problem:	Does G contain k vertices that touch all edges (vertex cover)?

VERTEX COVER as an Integer Linear Program

VERTEX COVER

Input:	An undirected graph $G = (V, E)$ and a natural number k
Problem:	Does G contain k vertices that touch all edges (vertex cover)?

- Introduce a variable x_v for every $v \in V$
- Minimize $\sum_{v \in V} x_v$ subject to
  1. $x_u + x_v \geq 1$ for every $(u, v) \in E$
  2. $0 \leq x_v \leq 1$ for every $v \in V$
  3. $x_v \in \mathbb{Z}$ for every $v \in V$

Vertex Cover as ~~an Integer~~ Linear Program

Vertex Cover

Input:	An undirected graph $G = (V, E)$ and a natural number k
Problem:	Does G contain k vertices that touch all edges (vertex cover)?

- Introduce a variable x_v for every $v \in V$
- Minimize $\sum_{v \in V} x_v$ subject to
  1. $x_u + x_v \geq 1$ for every $(u, v) \in E$
  2. $0 \leq x_v \leq 1$ for every $v \in V$
  3. ~~$x_v \in \mathbb{Z}$ for every $v \in V$~~
- Can be solved in polynomial time

VERTEX COVER as a Linear Program

- Minimize $\sum_{v\in V} x_v$ subject to
  1. $x_u + x_v \geq 1$ for every $(u, v) \in E$
  2. $0 \leq x_v \leq 1$ for every $v \in V$

Vertex Cover as a Linear Program

- Minimize $\sum_{v \in V} x_v$ subject to
  1. $x_u + x_v \geq 1$ for every $(u, v) \in E$
  2. $0 \leq x_v \leq 1$ for every $v \in V$
- Consider a solution to this problem. Denote

$$V_0 = \left\{ v \in V \mid x_v < \frac{1}{2} \right\} \qquad V_{\frac{1}{2}} = \left\{ v \in V \mid x_v = \frac{1}{2} \right\} \qquad V_1 = \left\{ v \in V \mid x_v > \frac{1}{2} \right\}$$

Theorem 29.7: G has a minimum vertex cover S such that $V_1 \subseteq S \subseteq V_1 \cup V_{\frac{1}{2}}$.

Proof: See blackboard.

Reduction rule: If $\sum_{v \in V} x_v > k$, return a no-instance. Otherwise, include V_1 in the vertex cover, remove V_0 and V_1 from G, and decrease k by $|V_1|$.

This gives a kernel with $\leq 2k$ vertices.

Outlook

What's next?

- Summary and consultation
- Examinations