Parameterized Algorithms

Lecture 2: Bounded Search Trees
A parameterized algorithm has running time of the form \( f(k)n^{O(1)} \) where \( k \) is the parameter.

A parameterized problem is fixed-parameter tractable (FPT) if it admits a parameterized algorithm.
**Vertex Cover: $O^*(2^k)$-Time Algorithm**

**Vertex Cover.**

**Input:** Graph $G$; non-negative integer $k$.

**Question:** Does $G$ have a vertex cover of size at most $k$?

![Graph G with $n=m=40$ and $k=5$](image)
Vertex Cover: $O^*(2^k)$-Time Algorithm

**Vertex Cover.**

*Input:* Graph $G$; non-negative integer $k$.

*Question:* Does $G$ have a vertex cover of size at most $k$?

$G$ is a graph with $n = m = 40$ vertices. The vertex cover is represented by the red vertices. The algorithm determines that $k = 5$ is not sufficient to cover all edges, hence the answer is **No**.
**Vertex Cover: $O^*(2^k)$-Time Algorithm**

**Vertex Cover.**

**Input:** Graph $G$; non-negative integer $k$.

**Question:** Does $G$ have a vertex cover of size at most $k$?

$G$ has a vertex cover of size at most $k$. The number of nodes $n = m = 40$ and $k = 6$.

Yes
Vertex Cover: $O^*(2^k)$-Time Algorithm

We have a list of rules of the form $[\text{Condition}]$: Action. We execute the first rule whose condition is satisfied.
We have a list of rules of the form [Condition]: Action. We execute the **first** rule whose condition is satisfied.

After we execute Rule X, the condition of Rule Y<X might be true. Thus, we always scan the list of rules from its beginning.
We have a list of rules of the form [Condition]: Action. We execute the first rule whose condition is satisfied.

After we execute Rule X, the condition of Rule Y<X might be true. Thus, we always scan the list of rules from its beginning.

The order can be essential for correctness.
We have a list of rules of the form [Condition]: Action. We execute the first rule whose condition is satisfied.

After we execute Rule X, the condition of Rule Y<X might be true. Thus, we always scan the list of rules from its beginning.

A rule where the algorithm calls itself recursively at most once is a reduction rule, and a rule where it calls itself recursively at least twice is a branching rule.
Rule 1. If $G$ has no edges, return Yes.
Rule 1. If $G$ has no edges, return Yes.

Rule 2. If $k=0$, return No.
Rule 1. If $G$ has no edges, return Yes.

Rule 2. If $k=0$, return No.

$G$ has at least one edge, and $k$ is positive. What should we do next?
**Vertex Cover: \(O^*(2^k)\)-Time Algorithm**

**Rule 1.** If \( G \) has no edges, return Yes.

**Rule 2.** If \( k=0 \), return No.

\( G \) has at least one edge, and \( k \) is positive. What should we do next?
**Vertex Cover: $O^*(2^k)$-Time Algorithm**

**Rule 1.** If $G$ has no edges, return Yes.

**Rule 2.** If $k=0$, return No.

$G$ has at least one edge, and $k$ is positive. What should we do next?

**Rule 3.** Pick an edge $\{u,v\}$.

i. Call ALG($G-u,k-1$).

ii. Call ALG($G-v,k-1$).

Return Yes iff at least one of the calls returns Yes.
Vertex Cover: $O^*(2^k)$-Time Algorithm

Correctness. Induction.
Vertex Cover: $O^*(2^k)$-Time Algorithm

**Correctness.** Induction.

**Rules 1+2.**
Suppose correctness for graphs with $n-1$ vertices, and prove correctness for graphs with $n$ vertices.
Suppose correctness for graphs with $n-1$ vertices, and prove correctness for graphs with $n$ vertices.

**Forward direction.** Suppose that $(G,k)$ has a solution $S$. 
Suppose correctness for graphs with $n-1$ vertices, and prove correctness for graphs with $n$ vertices.

**Forward direction.** Suppose that $(G,k)$ has a solution $S$. Then, $S$ must contain at least one vertex among $u$ and $v$. If it contains $u$, then $S\setminus\{u\}$ is a solution to $(G-u,k-1)$, and otherwise $S\setminus\{v\}$ is a solution to $(G-v,k-1)$. 

**Vertex Cover: $O^*(2^k)$-Time Algorithm**

Suppose correctness for graphs with $n-1$ vertices, and prove correctness for graphs with $n$ vertices.

**Forward direction.** Suppose that $(G,k)$ has a solution $S$. Then, $S$ must contain at least one vertex among $u$ and $v$. If it contains $u$, then $S\setminus\{u\}$ is a solution to $(G-u,k-1)$, and otherwise $S\setminus\{v\}$ is a solution to $(G-v,k-1)$.
**Vertex Cover: \(O^*(2^k)\)-Time Algorithm**

Suppose correctness for graphs with \(n-1\) vertices, and prove correctness for graphs with \(n\) vertices.

**Forward direction.** Suppose that \((G,k)\) has a solution \(S\). Then, \(S\) must contain at least one vertex among \(u\) and \(v\). If it contains \(u\), then \(S\{u\}\) is a solution to \((G-u,k-1)\), and otherwise \(S\{v\}\) is a solution to \((G-v,k-1)\). In either case, at least one of the recursive calls will return Yes (by the inductive hypothesis).
Suppose correctness for graphs with $n-1$ vertices, and prove correctness for graphs with $n$ vertices.

**Reverse direction.** Suppose that the algorithm returns Yes. W.l.o.g., suppose that the recursive call on $(G-u,k-1)$ returned Yes.
**Vertex Cover: \(O^*(2^k)\)-Time Algorithm**

Suppose correctness for graphs with \(n-1\) vertices, and prove correctness for graphs with \(n\) vertices.

**Reverse direction.** Suppose that the algorithm returns Yes. W.l.o.g., suppose that the recursive call on \((G-u,k-1)\) returned Yes.

By the inductive hypothesis, \((G-u,k-1)\) has a solution \(S\).
Suppose correctness for graphs with \( n-1 \) vertices, and prove correctness for graphs with \( n \) vertices.

**Reverse direction.** Suppose that the algorithm returns Yes. W.l.o.g., suppose that the recursive call on \((G-u,k-1)\) returned Yes.
By the inductive hypothesis, \((G-u,k-1)\) has a solution \(S\).
Then, \(SU\{u\}\) is a solution to \((G,k)\).
Running time. Number of recursive calls at the leaves: 
\[ N(k) = 2N(k-1); \ N(0)=1. \]
Running time. Number of recursive calls at the leaves:
N(k) = 2N(k-1); N(0)=1.
→ N(k) = 2^k.
Running time. Number of recursive calls at the leaves:

\[ N(k) = 2N(k-1); \quad N(0) = 1. \]

\[ \rightarrow N(k) = 2^k. \]
Running time. Number of recursive calls at the leaves: \( N(k) = 2N(k-1); \ N(0)=1. \)

\[ \rightarrow N(k) = 2^k. \]

Total number of recursive call (including internal nodes): \( O(2^k) \).

Each recursive call is executed in linear time.

Time complexity: \( O(2^k(n+m)) \).
**d-Hitting Set: \( O^*(d^k) \)-Time Algorithm**

**d-Hitting Set.** Given a family \( F=\{S_1, S_2, \ldots, S_m\} \) of sets of size \( d \) over a universe \( U \) and a parameter \( k \), determine whether there exists a subset \( X \) of \( U \) of size at most \( k \) that has a nonempty intersection with every set in \( F \).
**d-Hitting Set: \( O^*(d^k) \)-Time Algorithm**

**d-Hitting Set.** Given a family \( F=\{S_1,S_2,\ldots,S_m\} \) of sets of size \( d \) over a universe \( U \) and a parameter \( k \), determine whether there exists a subset \( X \) of \( U \) of size at most \( k \) that has a nonempty intersection with every set in \( F \).

Vertex Cover is 2-Hitting Set.
**Rule 1.** If $F$ is empty, return Yes.

**Rule 2.** If $k=0$, return No.
**d-Hitting Set: O*(d^k)-Time Algorithm**

**Rule 1.** If $F$ is empty, return Yes.

**Rule 2.** If $k=0$, return No.

$F$ has at least one set, and $k$ is positive. What should we do next?

**Rule 3.** Pick a set $S$ in $F$.
- For every element $e$ in $S$, call $\text{ALG}(U\{e\},F',k-1)$, where $F' = \{T \in F : e \notin T\}$.

Return Yes iff at least one of the calls returns Yes.
**d-Hitting Set: \(O^*(d^k)\)-Time Algorithm**

**Correctness.** Very similar to the case of Vertex Cover (left as exercise).
Running time. Number of recursive calls:
\[ N(k) = dN(k-1); \quad N(0) = 1. \]
\[ \rightarrow N(k) = d^k. \]
Running time. Number of recursive calls:

\[ N(k) = dN(k-1); \quad N(0) = 1. \]

\[ \rightarrow N(k) = d^k. \]
**Rule 1.** If $G$ has no edges, return Yes.

**Rule 2.** If $k=0$, return No.

**Rule 3.** Pick an edge $\{u,v\}$.

i. Call $\text{ALG}(G-u,k-1)$.

ii. Call $\text{ALG}(G-v,k-1)$.

Return Yes iff at least one of the calls returns Yes.
Vertex Cover: $O^*(1.62^k)$-Time Algorithm

**Rule 1.** If $G$ has no edges, return Yes.

**Rule 2.** If $k=0$, return No.

**Rule 3.** Pick an edge $\{u,v\}$ a non-isolated vertex $v$.
   i. Call $\text{ALG}(G-u,k-1)$ $\text{ALG}(G-N(v),k-|N(v)|)$.
   ii. Call $\text{ALG}(G-v,k-1)$.

Return Yes iff at least one of the calls returns Yes.
**Vertex Cover: $O^{*}(1.62^k)$-Time Algorithm**

**Rule 1.** If $G$ has no edges, return Yes.

**Rule 2.** If $k=0$, return No.

**Rule 3.** Pick an edge $\{u,v\}$ a non-isolated vertex $v$.

i. Call $\text{ALG}(G-u,k-1) \text{ALG}(G-N(v),k-|N(v)|)$.

ii. Call $\text{ALG}(G-v,k-1)$.

Return Yes iff at least one of the calls returns Yes.

**How can we ensure that $|N(v)|$ is at least 2?**
Rule 3. If there exists an isolated vertex $v$, call $\text{ALG}(G-v, k)$. 
Rule 3. If there exists an isolated vertex \( v \), call \( \text{ALG}(G-v,k) \).

Rule 4. If there exists a degree-1 vertex \( v \), call \( \text{ALG}(G-u,k-1) \) where \( u \) is the neighbor of \( v \).
Rule 3. If there exists an isolated vertex $v$, call $\text{ALG}(G-v,k)$.

Rule 4. If there exists a degree-1 vertex $v$, call $\text{ALG}(G-u,k-1)$ where $u$ is the neighbor of $v$.

**Correctness of Rule 4.** Forward direction: Suppose that $(G,k)$ has a solution $S$. Then, $S$ must contain at least one vertex among $u$ and $v$. Note that $S\setminus\{u,v\}$ is a vertex cover of $G-u$. Thus, $S\setminus\{u,v\}$ is a solution to $(G-u,k-1)$. 
Vertex Cover: $O^*(1.62^k)$-Time Algorithm

**Rule 3.** If there exists an isolated vertex $v$, call $\text{ALG}(G-v,k)$.

**Rule 4.** If there exists a degree-1 vertex $v$, call $\text{ALG}(G-u,k-1)$ where $u$ is the neighbor of $v$.

**Correctness of Rule 4.** **Forward direction:** Suppose that $(G,k)$ has a solution $S$. Then, $S$ must contain at least one vertex among $u$ and $v$. Note that $S\backslash\{u,v\}$ is a vertex cover of $G-u$. Thus, $S\backslash\{u,v\}$ is a solution to $(G-u,k-1)$. [True because the only neighbor of $v$ is $u$.]
**Rule 3.** If there exists an isolated vertex \( v \), call \( \text{ALG}(G-v,k) \).

**Rule 4.** If there exists a degree-1 vertex \( v \), call \( \text{ALG}(G-u,k-1) \) where \( u \) is the neighbor of \( v \).

**Rule 5.** Pick a vertex \( v \).
   i. Call \( \text{ALG}(G-N(v),k-|N(v)|) \).
   ii. Call \( \text{ALG}(G-v,k-1) \).

Return Yes iff at least one of the calls returns Yes.
**Rule 3.** If there exists an isolated vertex $v$, call $\text{ALG}(G-v,k)$.

**Rule 4.** If there exists a degree-1 vertex $v$, call $\text{ALG}(G-u,k-1)$ where $u$ is the neighbor of $v$.

**Rule 5.** Pick a vertex $v$.

i. Call $\text{ALG}(G-N(v),k-|N(v)|)$. [Note: $|N(v)| \geq 2$.]

ii. Call $\text{ALG}(G-v,k-1)$.

Return Yes iff at least one of the calls returns Yes.
Running time. Number of recursive calls:
\[ N(k) = N(k-1) + N(k-2); \quad N(0)=1. \]
**Running time. Number of recursive calls:**

\[ N(k) = N(k-1) + N(k-2); \quad N(0) = 1. \quad \text{Worse case.} \]
Vertex Cover: $O^*(1.62^k)$-Time Algorithm

**Running time.** Number of recursive calls:

$N(k) = N(k-1) + N(k-2); N(0) = 1.$
Running time. Number of recursive calls:

\[ N(k) = N(k-1) + N(k-2); \quad N(0) = 1. \]

Guess \( N(k) = c^k \).

\[ \rightarrow c^k = c^{k-1} + c^{k-2}. \]

\[ \rightarrow c^2 = c + 1. \]

\[ \rightarrow c = (1 + \sqrt{5})/2 < 1.6181. \]
**Vertex Cover: $O^*(1.62^k)$-Time Algorithm**

**Running time.** Number of recursive calls:

$N(k) = N(k-1) + N(k-2); \ N(0) = 1.$  

Guess $N(k) = c^k.$  

$\rightarrow c^k = c^{k-1} + c^{k-2}.$  

$\rightarrow c^2 = c + 1.$  

$\rightarrow c = (1 + \sqrt{5})/2 < 1.6181.$

To get an upper bound, it is suffices to find $c$ such that  

$c^k \geq c^{k-1} + c^{k-2}.$
More generally. Number of recursive calls:

\[ N(k) = N(k-t_1) + N(k-t_2) + \ldots + N(k-t_r); \quad (0)=1. \]

Guess \( N(k) = c^k \).

\[ c^k = c^{k-t_1} + c^{k-t_2} + \ldots + c^{k-t_r}. \]

\[ c_{\text{max}}(t_1, \ldots, t_r) = c_{\text{max}}(t_1, \ldots, t_r)-t_1 + c_{\text{max}}(t_1, \ldots, t_r)-t_2 + \ldots + c_{\text{max}}(t_1, \ldots, t_r)-t_r. \]

\[ [c_{\text{max}}(t_1, \ldots, t_r) \geq c_{\text{max}}(t_1, \ldots, t_r)-t_1 + c_{\text{max}}(t_1, \ldots, t_r)-t_2 + \ldots + c_{\text{max}}(t_1, \ldots, t_r)-t_r. \]
**Rule 3.** If there exists an isolated vertex $v$, call $\text{ALG}(G-v,k)$.

**Rule 4.** If there exists a degree-1 vertex $v$, call $\text{ALG}(G-u,k-1)$ where $u$ is the neighbor of $v$.

**Rule 5.** Pick a vertex $v$.

i. Call $\text{ALG}(G-N(v),k-|N(v)|)$.

ii. Call $\text{ALG}(G-v,k-1)$.

Return Yes iff at least one of the calls returns Yes.
**Vertex Cover: \(O^*(1.47^k)\)-Time Algorithm**

**Rule 3.** If there exists an isolated vertex \(v\), call \(\text{ALG}(G-v,k)\).

**Rule 4.** If there exists a degree-1 vertex \(v\), call \(\text{ALG}(G-u,k-1)\) where \(u\) is the neighbor of \(v\).

**Rule 5.** Pick a vertex \(v\).
   
i. Call \(\text{ALG}(G-N(v),k-\mid N(v)\mid)\). How to ensure \(\mid N(v)\mid \geq 3\)?
   
ii. Call \(\text{ALG}(G-v,k-1)\).

Return Yes iff at least one of the calls returns Yes.
Rule 5. If the maximum degree of a vertex in the graph is 2, then solve the problem in polynomial time.
Rule 5. If the maximum degree of a vertex in the graph is 2, then solve the problem in polynomial time.

Collection of cycles:
Rule 5. If the maximum degree of a vertex in the graph is 2, then solve the problem in polynomial time.

Rule 5. Pick a vertex \( v \) of degree at least 3.

i. Call \( \text{ALG}(G-N(v),k-|N(v)|) \).

ii. Call \( \text{ALG}(G-v,k-1) \).

Return Yes iff at least one of the calls returns Yes.
**Vertex Cover: O*(1.47^k)-Time Algorithm**

**Rule 5.** If the maximum degree of a vertex in the graph is 2, then solve the problem in polynomial time.

**Rule 5.** Pick a vertex \( v \) of degree at least 3.

i. Call \( \text{ALG}(G-N(v),k-|N(v)|) \).

ii. Call \( \text{ALG}(G-v,k-1) \).

Return Yes iff at least one of the calls returns Yes.
Vertex Cover: $O^{*}(1.47^k)$-Time Algorithm

**Running time.** Similar to the analysis of the previous algorithm (left as an exercise).
Cluster Vertex Deletion: $O^*(3^k)$-Time Algo.

Cluster Vertex Deletion. Given a graph $G$ and a parameter $k$, determine whether there exists a subset $S$ of $V(G)$ of size $\leq k$ such that $G-S$ is a cluster graph, that is, every connected component of $G-S$ is a clique.
Cluster Vertex Deletion. Given a graph $G$ and a parameter $k$, determine whether there exists a subset $S$ of $V(G)$ of size $\leq k$ such that $G-S$ is a cluster graph, that is, every connected component of $G-S$ is a clique.
Cluster Vertex Deletion: $O^*(3^k)$-Time Algo.

**Cluster Vertex Deletion.** Given a graph $G$ and a parameter $k$, determine whether there exists a subset $S$ of $V(G)$ of size $\leq k$ such that $G - S$ is a cluster graph, that is, every connected component of $G - S$ is a clique.
**Observation.** A graph $G$ is a cluster graph if and only if it has no induced $P_3$ (i.e. there do not exist three vertices $u,v,w$ such that $\{u,v\}$ and $\{v,w\}$ are edges in $G$, but $\{u,w\}$ is not).
Observation. A graph $G$ is a cluster graph if and only if it has no induced $P_3$ (i.e. there do not exist three vertices $u,v,w$ such that $\{u,v\}$ and $\{v,w\}$ are edges in $G$, but $\{u,w\}$ is not).

$\rightarrow$ Objective. Determine whether there exists a subset $S$ of $V(G)$ such that $S$ intersects the vertex set of every induced $P_3$ in $G$. 
**Observation.** A graph $G$ is a cluster graph if and only if it has no induced $P_3$ (i.e. there do not exist three vertices $u, v, w$ such that $\{u, v\}$ and $\{v, w\}$ are edges in $G$, but $\{u, w\}$ is not).

→ **Objective.** Determine whether there exists a subset $S$ of $V(G)$ such that $S$ intersects the vertex set of every induced $P_3$ in $G$.

1. A special case of 3-Hitting Set and therefore solvable in time $O^*(3^k)$. 
Observation. A graph $G$ is a cluster graph if and only if it has no induced $P_3$ (i.e. there do not exist three vertices $u,v,w$ such that $\{u,v\}$ and $\{v,w\}$ are edges in $G$, but $\{u,w\}$ is not).

→ Objective. Determine whether there exists a subset $S$ of $V(G)$ such that $S$ intersects the vertex set of every induced $P_3$ in $G$.

2. Design an $O^*(3^k)$-time algorithm directly (left as an exercise).