Fixed Parameter Tractability

Algorithmics, 186.814, VU 6.0

Jan Dreier

Some Preliminaries (Reminder)

Complexity Classes: $\mathbf{P}$ vs. $\mathbf{NP}$

Problem is in $\mathbf{P}$ – can be solved in polynomial time
for instance: $5n$, n^3, and similar running times

Problem is $\mathbf{NP}$-hard – polynomial-time algorithm unlikely to exist
for instance: $2^n \cdot n$, $4n^n$, and similar running times

IT companies deal with huge instances of $\mathbf{NP}$-hard problems on an everyday basis.
How?

IT companies deal with huge instances of $\mathbf{NP}$-hard problems on an everyday basis.
How?

- Heuristics
    - An algorithm without any guarantees on the optimality of the solution.

IT companies deal with huge instances of $\mathbf{NP}$-hard problems on an everyday basis.

How?

- Heuristics
  - An algorithm without any guarantees on the optimality of the solution.
- Approximation
  - A (polynomial) algorithm that outputs a solution which is guaranteed to be “close” to optimal.
  - Some $\mathbf{NP}$-hard problems cannot be approximated

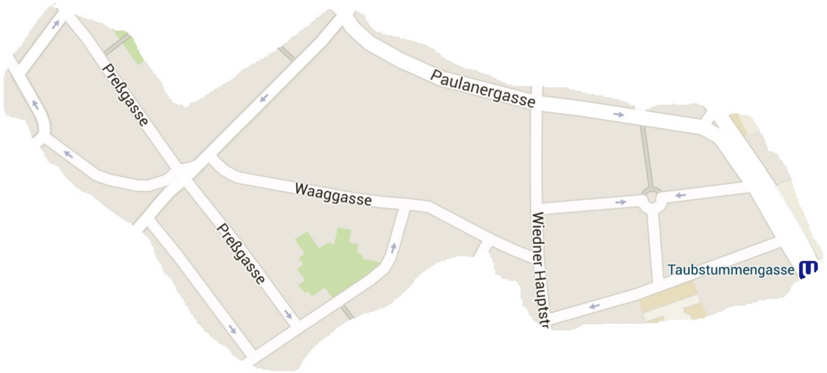
IT companies deal with huge instances of $\mathbf{NP}$-hard problems on an everyday basis.

How?

- Heuristics
  - An algorithm without any guarantees on the optimality of the solution.
- Approximation
  - A (polynomial) algorithm that outputs a solution which is guaranteed to be “close” to optimal.
  - Some $\mathbf{NP}$-hard problems cannot be approximated
- Parameterized algorithms
  - Exact and Fast
  - By exploiting properties of the problem
  - **Topic of this Block**

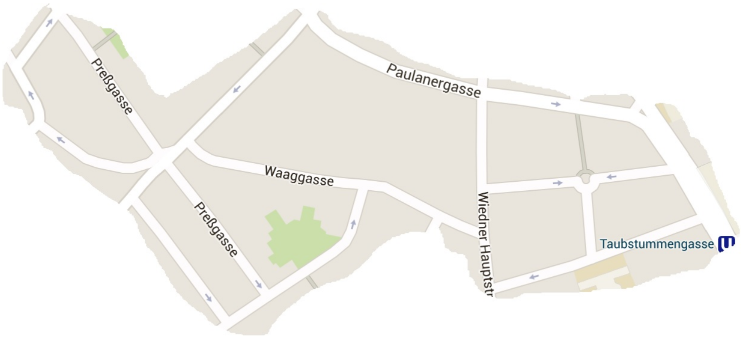
Example

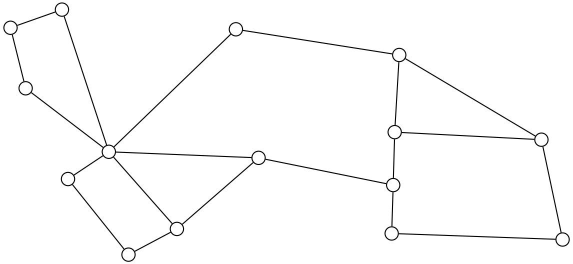
Task:
Place camera systems on crossroads so that each road is monitored.



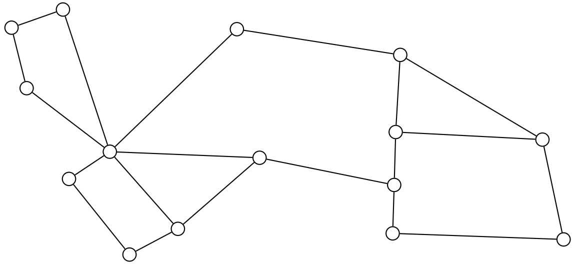
How many camera systems do we need? Can we do it with $6, 7, 8$ camera systems?

Graph Abstraction





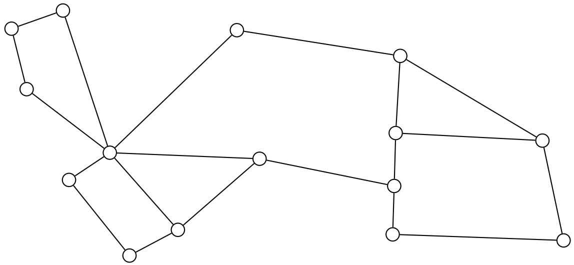
Problem Abstraction



Instance: A graph G and an integer k.

Task: Choose at most k vertices in G such that each edge is *covered* by at least one chosen vertex.

Example

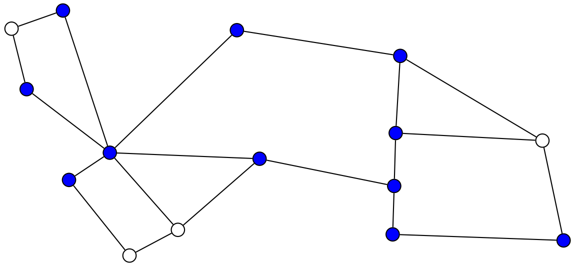


Instance: A graph G and an integer k.

Task: Choose at most k vertices in G such that each edge is covered by at least one chosen vertex.

$k = 11$?

Example

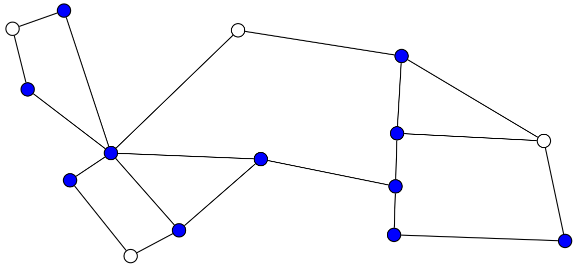


Instance: A graph G and an integer k.

Task: Choose at most k vertices in G such that each edge is covered by at least one chosen vertex.

$k = 11$?

Example



Instance: A graph G and an integer k.

Task: Choose at most k vertices in G such that each edge is covered by at least one chosen vertex.

$k = 11$? Yes, there is a solution.

Vertex Cover

VERTEX COVER

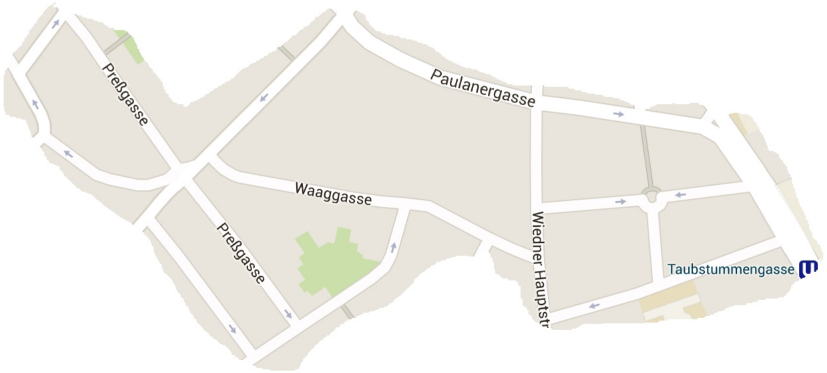
Instance: A graph G and an integer k.

Task: Choose at most k vertices in G such that each edge is covered by at least one chosen vertex.

- Famous $\mathbf{NP}$-hard problem.
  - One of **Karp's 21 $\mathbf{NP}$-hard problems** (1972).
- So, it is likely that no polynomial algorithm exists.

Back to the Motivation

Task:
Place camera systems on crossroads so that each road is monitored.



How many camera systems do we need? Can we do it with $6, 7, 8$ camera systems?

- Small numbers $\rightarrow$ treat them as constants?

Vertex Cover

Instance: A graph G and an integer k.

Task: Choose at most k vertices in G such that each edge is covered by at least one chosen vertex.

vs.

k-Vertex Cover

Instance: A graph G.

Task: Choose at most k vertices in G such that each edge is covered by at least one chosen vertex.

- small change makes **huge** difference in complexity: k-Vertex Cover is in $\mathbf{P}$!
- Runtime: $\mathcal{O}(n^k)$

Bounded Search Tree Algorithms

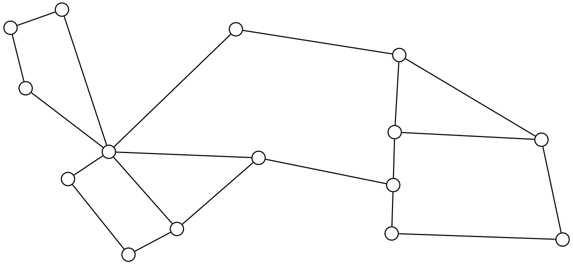
Solving k-Vertex Cover

Branching algorithm:

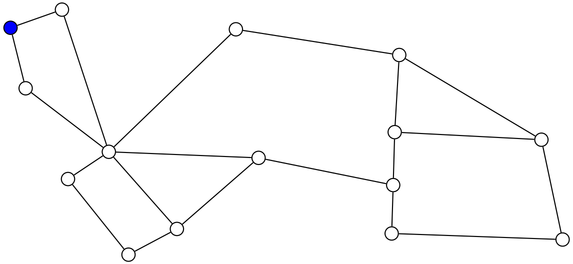
1. test $k \neq 0$; if $k = 0$ and G contains an edge, output false (for this branch).
2. branch over all vertices v (n options).
3. v is in the vertex cover $\rightarrow$ edges incident to v can be ignored.
4. Delete v and restart on the new smaller graph with $k := k - 1$.

If at least one branch succeeds, then we have a solution.

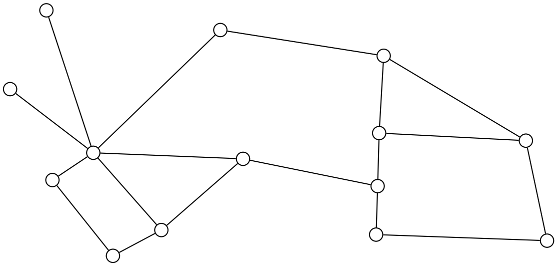
$k = 6$



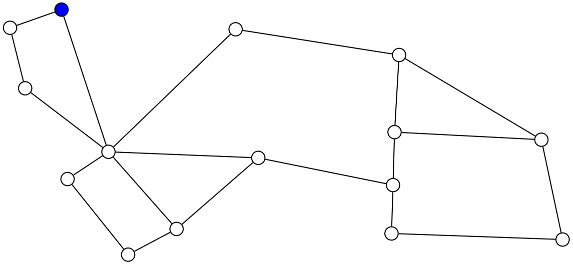
$k = 6$



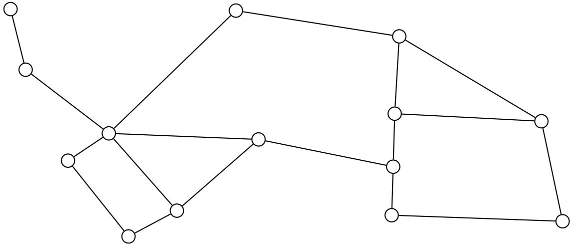
$k = 5$, Branch 1



$k = 6$



$k = 5$, Branch 2



Branching tree with $\mathcal{O}(n)$ children and depth k.

Running time: $n \cdot (n-1) \cdot (n-2) \ldots (n-(k-1)) = \mathcal{O}(n^k)$.

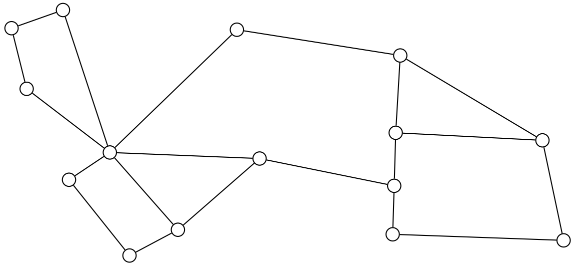
- it's polynomial!
- the running time is not practical for bigger values of k.

So, can we do *better*?

Bounded Search Tree Algorithm

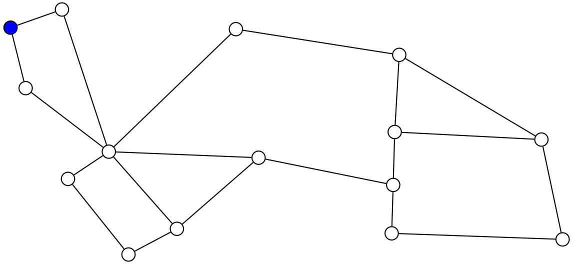
Observation: if a vertex is not picked, then *all its neighbors must be picked*.

$k = 8$



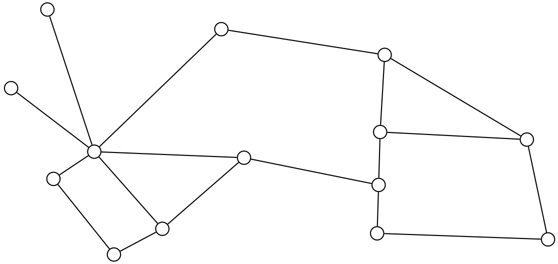
Bounded Search Tree Algorithm

Left branch (in cover), $k = 8$



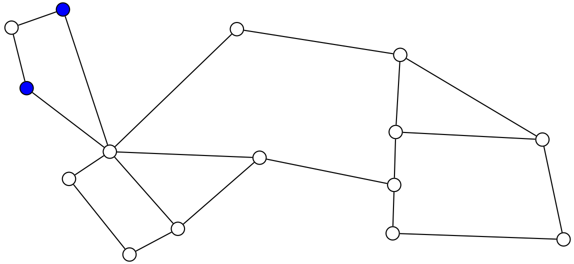
Bounded Search Tree Algorithm

Left branch (in cover), $k = 7$



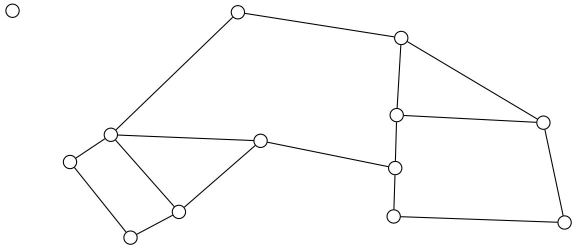
Bounded Search Tree Algorithm

Right branch (not in cover), $k = 8$



Bounded Search Tree Algorithm

Right branch (not in cover), $k = 6$



Bounded Search Tree Algorithm

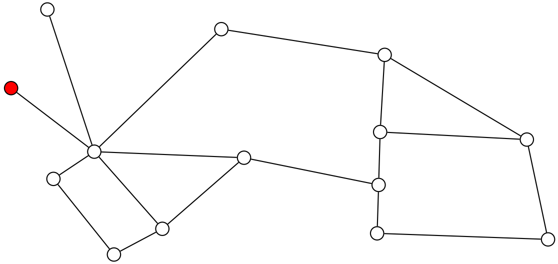
1. test $k \neq 0$; if $k = 0$ and G contains an edge, output false (for this branch).
2. pick a vertex v incident to some edge (1 option, no branching!).
3. either v is in the vertex cover, or all its neighbors are $\rightarrow$ branching.

Branching tree with 2 children and depth at most k.

Running time: $\mathcal{O}(2^k \cdot n)$.

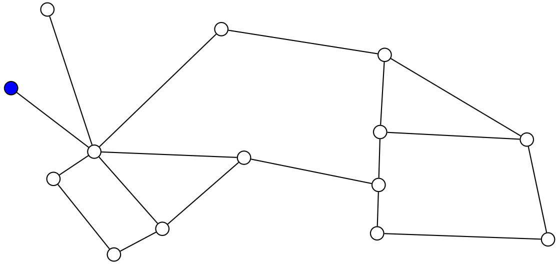
Bounded Search Tree for Vertex Cover

After left branch (in cover), $k = 7$



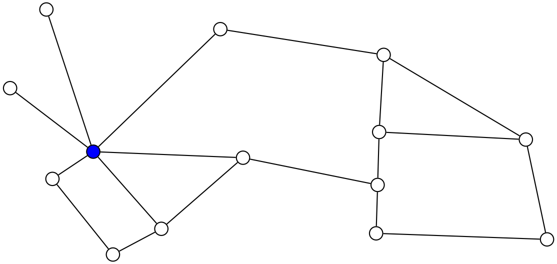
Bounded Search Tree for Vertex Cover

Left branch (in cover), $k = 7$



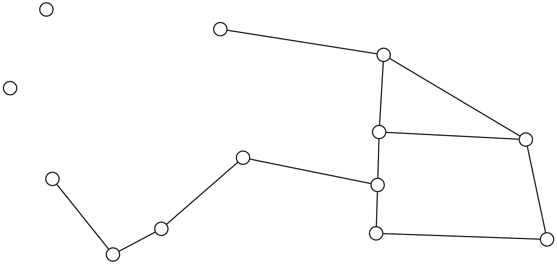
Bounded Search Tree for Vertex Cover

Right branch (not in cover), $k = 7$



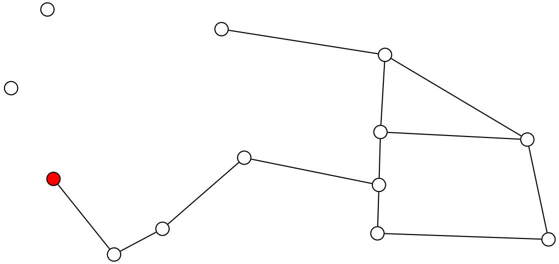
Bounded Search Tree for Vertex Cover

Next step (left branch, right branch), $k = 6$



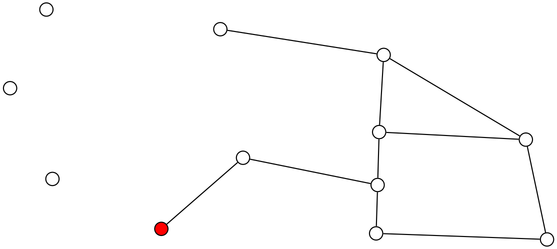
Bounded Search Tree for Vertex Cover

Next step (left branch, right branch), $k = 6$



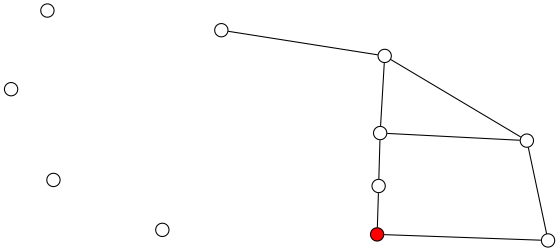
Bounded Search Tree for Vertex Cover

Next step (left branch, right branch, right branch), $k = 5$



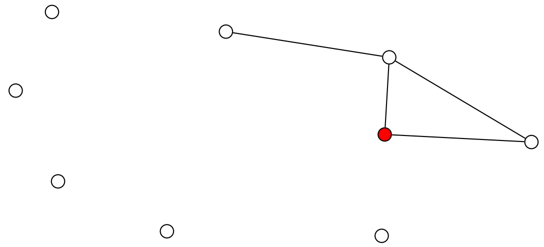
Bounded Search Tree for Vertex Cover

Next step (left, right, right, right), $k = 4$



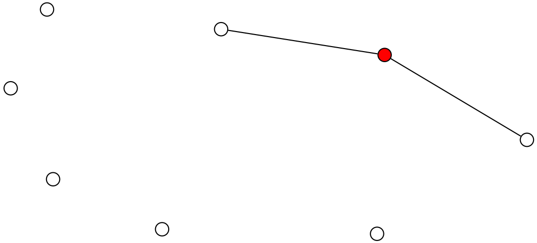
Bounded Search Tree for Vertex Cover

Next step (left, right, right, right, right), $k = 2$



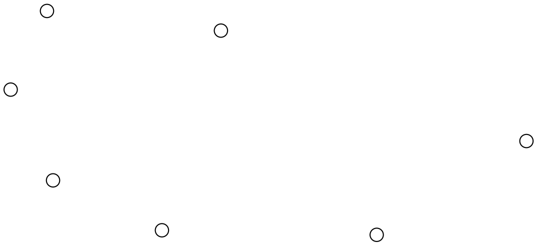
Bounded Search Tree for Vertex Cover

Next step (left, right, right, right, right, left), $k = 1$



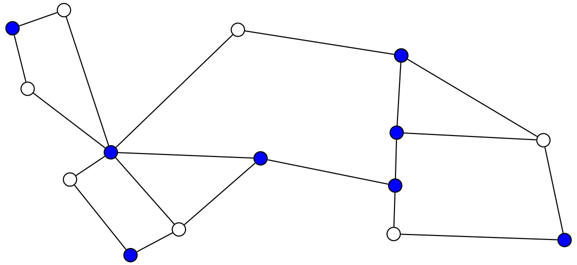
Bounded Search Tree for Vertex Cover

Next step (left, right, right, right, right, left, left), $k = 0$



Bounded Search Tree for Vertex Cover

Solution:



How much better is this?

$$\mathcal{O}(2^k \cdot n) \text{ vs. } \mathcal{O}(n^k).$$

Say $n = 1000$ and $k = 10$.
Then the left algorithm would take about 10^6 steps – relatively practical.
The right algorithm would take about 10^{30} steps – more than the number of stars in the universe!

In classical complexity, both algorithms are "polynomial". To clearly distinguish them, we use *Parameterized Complexity*.

Parameterized Complexity

A **parameterized problem** contains a **parameter** as part of the input

- the parameter will most often be a number
- in *your* application, you expect the parameter to be *small*

PARAMETERIZED VERTEX COVER

Instance: A graph G and an integer k.

Parameter: k.

Task: Choose at most k vertices in G such that each edge is covered by at least one chosen vertex.

Fixed Parameter Tractability

A **Fixed-Parameter Algorithm** (or **FPT Algorithm**) for a parameterized problem is an algorithm that runs in time:

$$f(k) \cdot \mathsf{poly}(n),$$

where f is an *arbitrary computable function* and poly is a *polynomial function independent of* k.

For example, the $\mathcal{O}(n^k)$ branching algorithm is not fixed-parameter, but the $\mathcal{O}(2^k \cdot n)$ one is fixed-parameter.

Fixed Parameter Tractability

A **Fixed-Parameter Algorithm** (or **FPT Algorithm**) for a parameterized problem is an algorithm that runs in time:

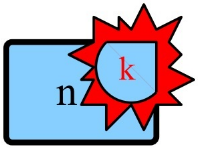
$$f(k) \cdot \text{poly}(n),$$

where f is an *arbitrary computable function* and poly is a *polynomial function independent of* k.

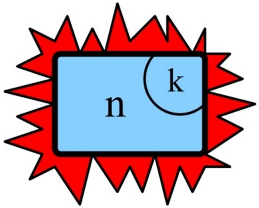
For example, the $\mathcal{O}(n^k)$ branching algorithm is not fixed-parameter, but the $\mathcal{O}(2^k \cdot n)$ one is fixed-parameter.

If an FPT algorithm exists for a parameterized problem $\mathcal{P}$, then $\mathcal{P}$ is in the complexity class **FPT (fixed parameter tractable)**.

Parameterized Vertex Cover is fixed parameter tractable / is FPT / is in FPT.



instead of



$\mathcal{O}(2^k \cdot n)$ $\mathcal{O}(n^k)$

Intuition:

- k is small enough such that $f(k)$ is still small
- but k is so large that we do not want it to interact much with n.

Remarks

Very active research field.

Books:

- R. G. Downey, M. R. Fellows: Parameterized Complexity (1999)
- R. Niedermeier: Invitation to Fixed-Parameter Algorithms (2006)
- R. G. Downey, M. R. Fellows: Fundamentals of Parameterized Complexity (2012)
- J. Flum, M. Grohe: Parameterized Complexity Theory (2006)
- Cygan et al.: Parameterized Algorithms (2015)
- ...

Course on Fixed-Parameter Algorithms and Complexity
Taught by Robert Ganian, this semester, 9-30 January:

- **Mondays (13:00-16:30) and**
- **Thursdays (13:00-16:30)**.

Examples of FPT running times

Say $\mathcal{P}$ can be solved in time:

- $\mathcal{O}(k^k \cdot n^5)$
- $20k + 3n^k$
- $\mathcal{O}(2^{2^k} + n)$
- $\mathcal{O}(k^n)$
- $\mathcal{O}(5^{k^2} \cdot n^{\log k})$
- $\mathcal{O}(2^n)$

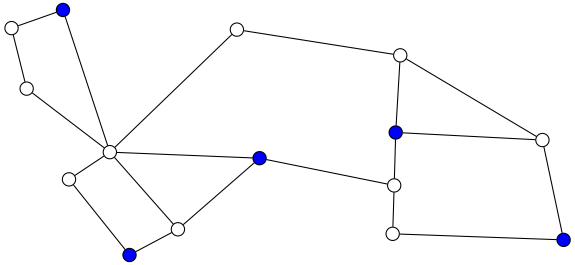
Does this imply that $\mathcal{P}$ is FPT?

Independent Set

Another famous $\mathbf{NP}$-hard problem:

INDEPENDENT SET
Instance: A graph G and an integer k.
Task: Choose at least k vertices in G which are pairwise *independent*.

Independent means that there are no edges between them.



Parameterized Independent Set

Parameterized Independent Set

Instance: A graph G and an integer k.

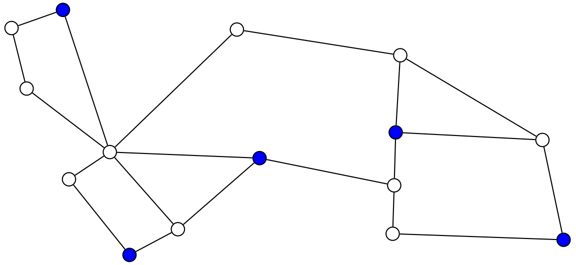
Parameter: k.

Task: Choose at least k vertices in G which are pairwise *independent*.

Is this problem FPT?

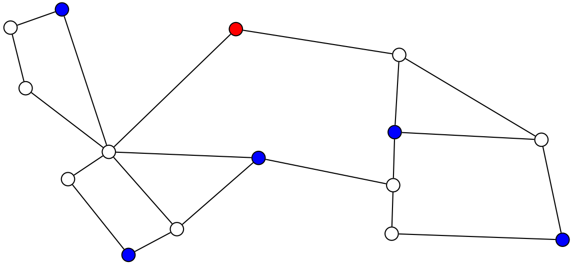
Bounded Search Tree Algorithm?

Optimality rule: if there exists an independent set which does not contain a vertex v and any neighbor of v, then there also exists an independent set containing v.



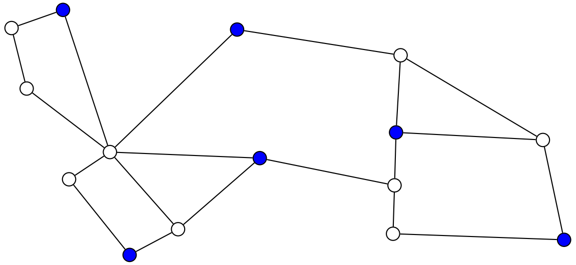
Bounded Search Tree Algorithm?

Optimality rule: if there exists an independent set which does not contain a vertex v and any neighbor of v, then there also exists an independent set containing v.



Bounded Search Tree Algorithm?

Optimality rule: if there exists an independent set which does not contain a vertex v and any neighbor of v, then there also exists an independent set containing v.



Bounded Search Tree Algorithm?

Optimality rule: if there exists a solution which does not pick a vertex v and any neighbor of v, then there also exists a solution which picks v.
Branching rule:

1. Choose a vertex v
2. Branching rule: either v is in the solution, or at least one of its neighbors is in the solution

Bounded Search Tree Algorithm?

Optimality rule: if there exists a solution which does not pick a vertex v and any neighbor of v, then there also exists a solution which picks v.
Branching rule:

1. Choose a vertex v
2. Branching rule: either v is in the solution, or at least one of its neighbors is in the solution

Problem: number of neighbors could be $\mathcal{O}(n) \rightarrow$ too many children!
Can get $\mathcal{O}(n^k)$ algorithm, but not an FPT algorithm in this way.

In fact, there are strong reasons to believe that $\textsc{Parameterized Independent Set}$ is **not FPT**.

Parameterized Complexity Classes

There is a whole ecosystem of parameterized complexity classes for NP-complete problems

$$\begin{tikzcd}
\text{para-}\mathbf{NP} \\
\text{XP} \arrow[u] \\
\cdots \\
\text{W[2]} \\
\text{W[1]} \arrow[u] \\
\text{FPT} \arrow[u]
\end{tikzcd}$$

Exponential Algorithms: 2^n etc.

Parameter does not help at all

XP Algorithms: $n^{f(k)}$

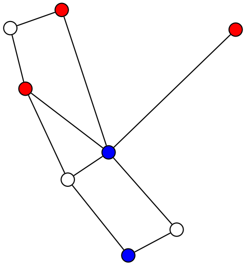
Parameter helps very little

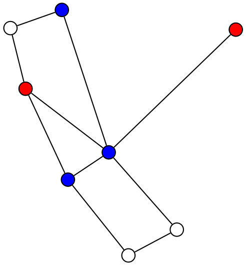
FPT Algorithms: $f(k) \cdot \text{poly}(n)$

Parameter helps

Another problem

A *proper c-coloring* of a graph is a coloring of the vertices by c colors such that no two neighbors have the same color.
Example with 3 colors:





k-Coloring

3-Coloring

Instance: A graph G.

Task: Find a proper 3-coloring of G.

3-Coloring is another famous $\mathbf{NP}$-complete problem.
What about the following parameterized version?

k-Coloring

Instance: A graph G and an integer k.

Parameter: k.

Task: Find a proper k-coloring of G.

k-Coloring

3-Coloring

Instance: A graph G.

Task: Find a proper 3-coloring of G.

3-Coloring is another famous $\mathbf{NP}$-complete problem.
What about the following parameterized version?

k-Coloring

Instance: A graph G and an integer k.

Parameter: k.

Task: Find a proper k-coloring of G.

Even an $n^{f(k)}$ algorithm would result in a **polynomial algorithm** for 3-Coloring
$\rightarrow$ k-Coloring is para$\mathbf{NP}$-complete.

Independent Set Revisited

Let's consider Independent Set on a special graph class – graphs of bounded degree.

- a graph has *degree* d if each vertex has at most d neighbors.

Independent Set is known to remain $\mathbf{NP}$-hard also on graphs of degree at most 3.

Independent Set Revisited

Let's consider Independent Set on a special graph class – graphs of bounded degree.

- a graph has *degree* d if each vertex has at most d neighbors.

Independent Set is known to remain **NP**-hard also on graphs of degree at most 3.

Parameterized Independent Set'

Instance: A graph G of degree at most 3 and an integer k.

Parameter: k.

Task: Choose at least k vertices in G which are pairwise *independent*.

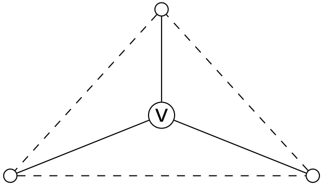
Is this problem FPT?

Solving Parameterized Independent Set'

Let's try the bounded search tree approach again.

Recall the **Optimality rule:**
if there exists an independent set which does not contain a vertex v and any neighbor of v, then adding v results in a new independent set which is larger.

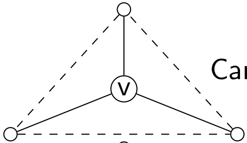
How to branch? Let's pick a vertex v:



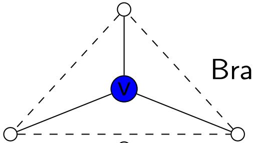
Two (equivalent) views:

- a maximum independent set will always contain either v or at least one neighbor of v.
- if neither v nor its neighbors are in an independent set, then a larger independent set can be obtained by adding v.

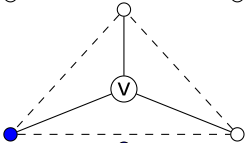
Branching for Parameterized Independent Set'



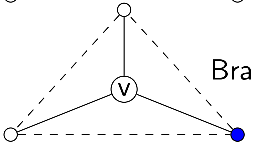
Can be ignored



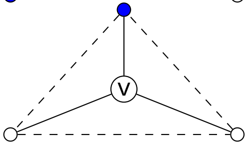
Branch 1



Branch 2



Branch 3



Branch 4

Solving Parameterized Independent Set'

Bounded search tree algorithm:

1. if $k = 0$, output true.

Solving Parameterized Independent Set'

Bounded search tree algorithm:

1. if $k = 0$, output true.
2. if $k > 0$ and the graph is empty, output false (for this branch).

Solving Parameterized Independent Set'

Bounded search tree algorithm:

1. if $k = 0$, output true.
2. if $k > 0$ and the graph is empty, output false (for this branch).
3. pick an arbitrary vertex v.
4. Branch into 4 options as per the previous slide to choose at least one vertex into the independent set.

Solving PARAMETERIZED INDEPENDENT SET'

Bounded search tree algorithm:

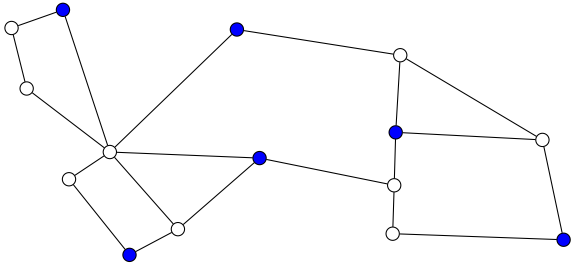
1. if $k = 0$, output true.
2. if $k > 0$ and the graph is empty, output false (for this branch).
3. pick an arbitrary vertex v.
4. Branch into 4 options as per the previous slide to choose at least one vertex into the independent set.
5. Delete the chosen vertex and all of its neighbors.
6. Restart on the remaining graph with $k := k - 1$.

Fixed-parameter algorithm, running time: $\mathcal{O}(4^k \cdot n)$.

Dominating Set

The Dominating Set problem is another classical **NP**-hard graph problem.
A vertex-set X is a *dominating set* in a graph G iff each vertex of G is either in X or has a neighbor in X.

Is the blue set a dominating set?



The Dominating Set problem

PARAMETERIZED DOMINATING SET

Instance: A graph G and an integer k.

Parameter: k.

Task: Find a dominating set in G of cardinality at most k.

The Dominating Set problem

PARAMETERIZED DOMINATING SET

Instance: A graph G and an integer k.

Parameter: k.

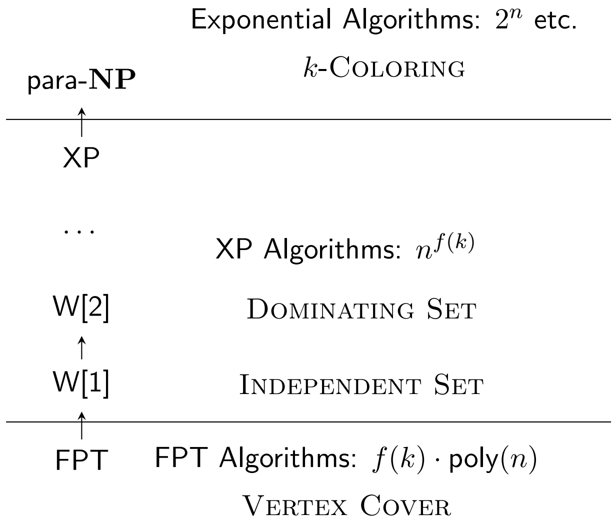
Task: Find a dominating set in G of cardinality at most k.

This is another case where the bounded search tree approach fails.

- Similar reason as for PARAMETERIZED INDEPENDENT SET: if v is not in the dominating set, then all we can say is that some neighbor of v is in the dominating set.

Like for PARAMETERIZED INDEPENDENT SET, there is strong evidence that PARAMETERIZED DOMINATING SET is not FPT.

Parameterized Complexity Classes and Algorithms


Exponential Algorithms: 2^n etc.
k-Coloring
para-**NP**
XP
...
XP Algorithms: $n^{f(k)}$
W[2]
Dominating Set
W[1]
Independent Set
FPT
FPT Algorithms: $f(k) \cdot \text{poly}(n)$
Vertex Cover

Dominating Set on Bounded Degree Graphs

Note: the (unparameterized) Dominating Set problem remains **NP**-hard even on graphs of degree at most 3.

Parameterized Dominating Set'

Instance: A graph G of maximum degree 3 and an integer k.

Parameter: k.

Task: Find a dominating set in G of cardinality at most k.

Dominating Set on Bounded Degree Graphs

Note: the (unparameterized) DOMINATING SET problem remains $\mathbf{NP}$-hard even on graphs of degree at most 3.

PARAMETERIZED DOMINATING SET'

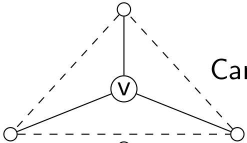
Instance: A graph G of maximum degree 3 and an integer k.

Parameter: k.

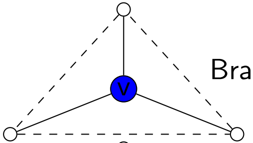
Task: Find a dominating set in G of cardinality at most k.

branching idea: either v or at least one of its neighbors must be in the dominating set.

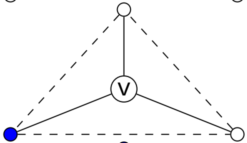
Branching for Parameterized Dominating Set'



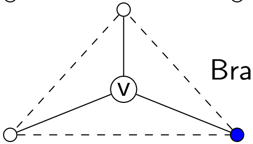
Cannot happen



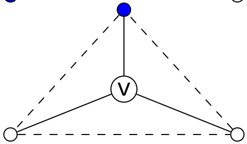
Branch 1



Branch 2



Branch 3



Branch 4

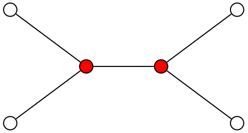
Branching for Parameterized Dominating Set'

But what happens if we put a vertex into the dominating set? What do we do with its neighbors?

Branching for Parameterized Dominating Set'

But what happens if we put a vertex into the dominating set? What do we do with its neighbors?

- Well, the're already dominated, so they're different from other vertices that still need to be dominated.
- Can we just delete them?
  - No, because they might have to go into the dominating set!



- We should mark them as "already dominated" (green).

Solving Parameterized Dominating Set'

Bounded search tree algorithm:

1. if $k = 0$ and there is an *undominated vertex*, output false (for this branch).

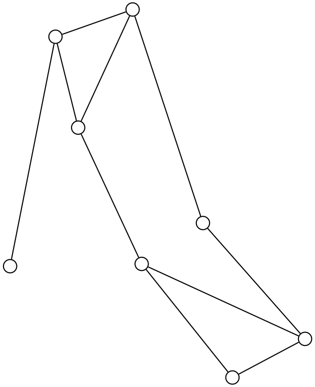
Solving Parameterized Dominating Set'

Bounded search tree algorithm:

1. if $k = 0$ and there is an *undominated vertex*, output false (for this branch).
2. if $k \geq 0$ and there are no *undominated vertices*, output true.
3. pick an arbitrary *undominated* vertex v.
4. Branch into 4 options (v or a neighbor of v) to put at least one vertex into the dominating set.
5. Delete the chosen vertex and mark all of its neighbors as dominated.
6. Restart on the remaining graph with $k := k - 1$.

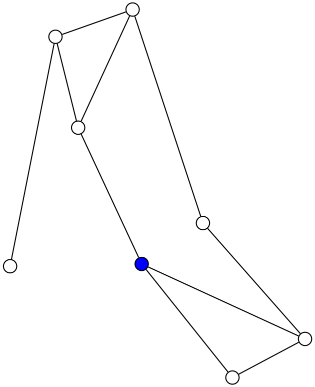
FPT algorithm, running time: $\mathcal{O}(4^k \cdot n)$.

Example run



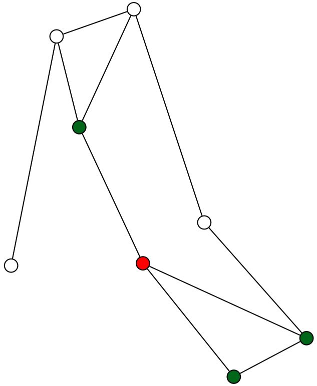
$k = 2$?

Example run

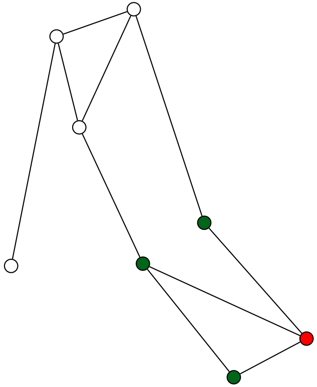


$k = 2$, some random choice of v

Example run

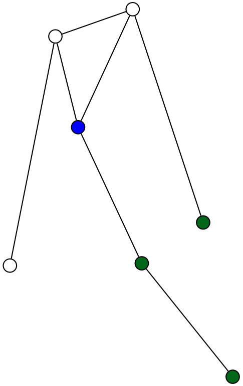

$k = 2$, if v goes into dominating set

Example run

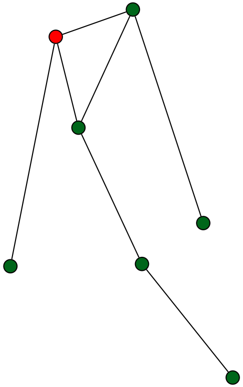


$k = 2$, if the “right” neighbor of v goes into dominating set

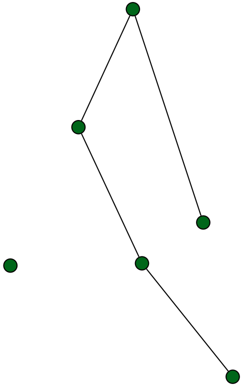
Example run


$k = 1$, next arbitrary choice of v

Example run


$k = 1$, a solution branch

Example run



$k = 0$, output true

Many other (more advanced) algorithmic techniques exist for designing FPT algorithms

Next lecture: Kernelization