**15-750: Graduate Algorithms** May 1, 2017

# Lecture 38: Fixed-Parameter Tractability

*Lecturer: Jason Li* *Scribe(s): Mu-Chu Lee*

## 1 NP-hard Problems

Recall that in the last lecture, we mentioned $NP$-hard problems.

$$L \text{ is } NP\text{-hard if } L' \leq_p L \quad \forall L' \in NP.$$

It is presumed that $NP$-hard problems are hard to solve, or intractable. Therefore, we are seeking methods to solve $NP$-hard problems by relaxing some conditions to make them tractable. We will be talking about:

1. Fixed-Parameter Tractability (FPT), **which is the focus for this lecture.**
2. Approximation algorithms

## 2 Vertex Cover Problem

### 2.1 Introduction

Consider a real world problem:

> We have a set of roads and junctions. We want to place security cameras at specific junctions such that all roads are monitored.

Now, our goal is to minimize the total number of cameras placed. We can see that this is a **Vertex Cover** problem, which is $NP$-complete!

### 2.2 Solutions

- Approximation algorithms
  - We know that we could find a 2-approximate vertex cover. However, cameras might be expensive, so a factor of 2 might still be unsatisfactory.
  - If $P \neq NP$, it is hard to approximate vertex cover to within a factor of 1.3, so we cannot expect to do better than this in general.
- FPT algorithms
  - If we know that the minimum number of cameras that we need to place is small, we can do better.
  - We **fix parameter** $k$, and solve the following problem:
    - If $OPT \leq k$ return $OPT$
    - Else return *Failure*

# 3 Fixed-Parameter Tractability

## 3.1 Definition

A fixed parameter tractable (FPT) algorithm with fixed parameter $k$ and input size $n$ is an algorithm with $O(f(k) \times poly(n))$ time complexity.
**Note that:**

1. $f(k)$ does not depend on $n$
2. $f(k)$ must not be polynomial of $k$, otherwise we could let $k = n$ and conclude $P = NP$!

## 3.2 Motivation

The motivation behind the definition could be illustrated by the vertex cover example.
Suppose $k = 20$ and $n = 10000$, where $k$ is the fixed parameter, and $n$ is the number of roads and junctions. Consider the computational time for the follow algorithms:

- Brute force: $2^n = 2^{10000} \approx 10^{3000}$
  - This is definitely intractable, and doesn't take advantage of $k$.
- Try all set of $k$ junctions: $\binom{n}{k} = \binom{10000}{20} \approx 10^{80}$
  - Poly-time for fixed $k$, and a great improvement compared to brute force.
  - Still doesn't work in practice.
- FPT algorithms: (we will give how they work later)
  - We want algorithms such as $2^k \times n$, $k! \times n^2$, ...
  - This brings the algorithm to $2^{20} \times 10000 \approx 10^{10}$, which might be solvable.

# 4 FPT for Vertex Cover with Bounded Search Tree

## 4.1 Main idea

- Pick one edge.
- One of its endpoint must be covered.
- Try both and do the recursion.

## 4.2 Algorithm

Given a graph $G = (V, E)$ and a vertex $u \in V$, let's define the notation

$$G - u = (V - u, E - e), \text{ where } e = \{(\bar{u}, \bar{v}) | (\bar{u}, \bar{v}) \in E, \bar{u} = u \text{ or } \bar{v} = u\}.$$

The bounded search tree for vertex cover is shown in Alg. 1. It is similar to running through the tree, but limiting the height to at most $k$. It is demonstrated in Fig. 1, where each node corresponds to removing a vertex in the graph from the parent's node.

**Algorithm 1** Bounded Search Tree

**procedure** BOUNDEDSEARCH($G, k$)
    Let $e = \{u, v\} \in G$
        Try $B_1$ = BoundedSearch($G - u, k - 1$) // Remove all edge connected with $u$
        Try $B_2$ = BoundedSearch($G - v, k - 1$) // Remove all edge connected with $v$
        Return $min(B_1, B_2) + 1$
**Base case 1**: If $k = 0$, $\begin{cases} \text{If } G \text{ not empty,} & \text{return FAIL} \\ \text{If } G \text{ empty,} & \text{return 0} \end{cases}$
**Base case 2**: If $G$ empty, return 0.

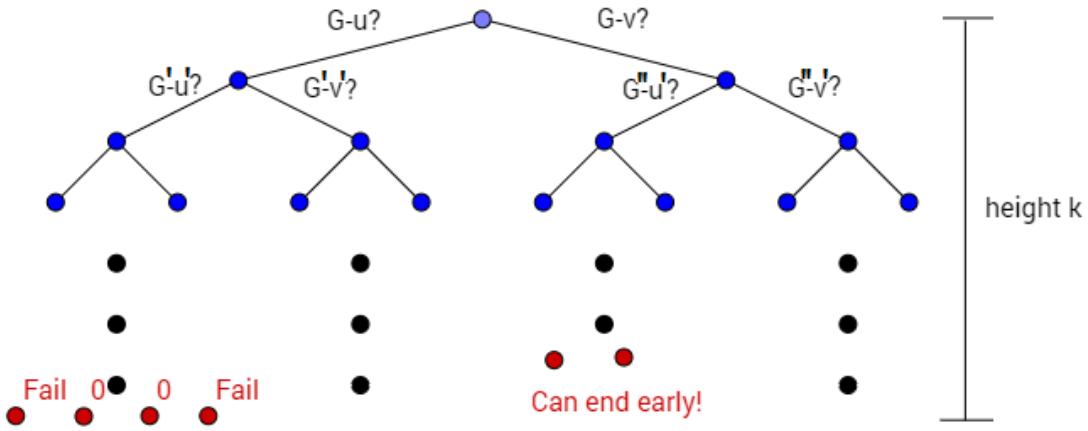


Figure 1: A fixed parameter-$k$ bounded search tree. $G'$ is $G - u$ and $G''$ is $G - v$.

### 4.3 Analysis of Bounded Search Tree

$$\begin{aligned} T(n,k) &= \text{running time on } G \text{ of size } n, \text{ parameter } k. \\ T(n,k) &= 2 \cdot T(n, k-1) + O(n) \text{ (since removing edges take } O(n) \text{ time)} \\ &= O(2^k n) \end{aligned}$$

We have $f(k) = 2^k$, which gives us a FPT algorithm.

## 5 FPT for Vertex Cover with kernelization

### 5.1 Main idea

- "Most" of the graph $G$ is easy to solve. Only a small "kernel" is hard (requires exponential time). For example, in Fig. 2, the center of the graph is the "kernel", for the part outside of the kernel is easy to solve.
- Chip away at the graph $G$ until the kernel is left (in poly-time).
- Solve the kernel (in exp-time).
- A kernelization algorithm requires $size(kernel) = f(k)$, which does not depend on $n$.

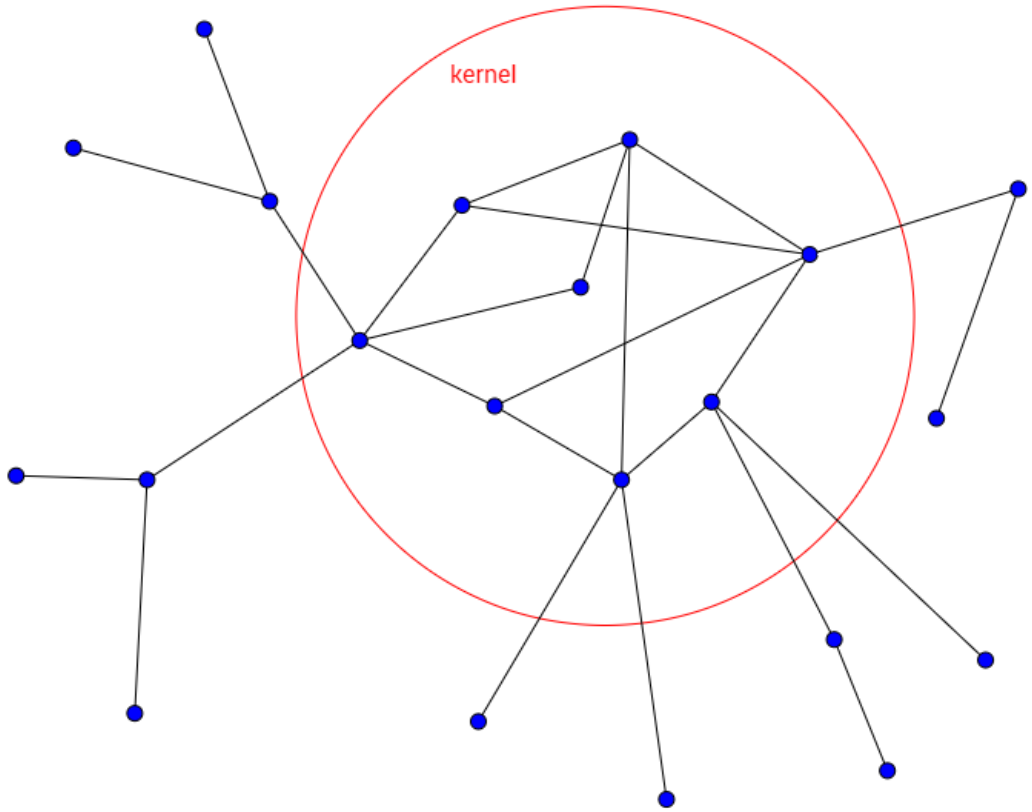


Figure 2: The center part of the graph is the kernel, which is "harder" than the rest of the graph.

### 5.2 Algorithm

**Algorithm 2** Kernelization

1: **procedure** KERNELIZATION($G, k$)
2: Apply 2 rules to $G$ iteratively until we can't anymore
3: **Rule 1:** If vertex $v$ has degree 0, remove $v$.
4: **Rule 2:** If vertex $v$ has degree $> k$, select $v$ to the vertex cover. ($v$ must be in the cover, otherwise to cover the edge $v$ connects to would require $> k$ vertices.)

### 5.3 Analysis

**Claim 5.1.** *If no rules in Alg. 2 can be applied further and $OPT \leq k$, then $|V| \leq k(k+1)$ and $|E| \leq k^2$.*

*Proof.* We can take a look at Fig. 3, where $OPT$ is the optimal vertex cover for the graph obtained by Alg. 2. There are some interesting properties that we could observe.

**Algorithm 3** Solving Vertex Cover by Kernelization

**procedure** VERTEXCOVERKERNELIZATION($G, k$)
    Apply Alg. 2 on $(G, k)$, obtain kernel.
    If $|V| > k(k+1)$ or $|E| > k^2$
        return Infeasible
    Else Run brute force on kernel.

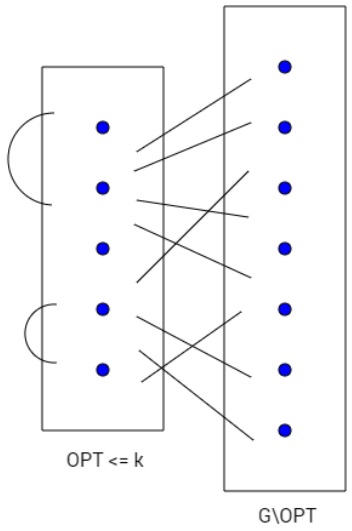


Figure 3: The graph can be split to two parts, the optimal set and the residual.

1. For vertices in $OPT$, the degree must be $\leq k$. This follows by Rule 2.
2. There are no edges between vertices in $G \backslash OPT$. If there exists an edge between vertices in $G \backslash OPT$, that edge could not be covered.

Now we can prove our claim:

1. We have at most $|OPT| \times k \leq k^2$ edges, given by Rule 2.
2. We have $|\#\text{vertices in } G \backslash OPT| \leq |OPT| \times k$ since all vertices in $G \backslash OPT$ must connect to some vertex in $OPT$ by Rule 1. Therfore, we have $|V| \leq (|OPT| + 1) \times k \leq k(k+1)$

Therefore, if $|V| > k(k+1)$ or $|E| > k^2$, then there must be no vertex cover of size $k$. □

In Alg. 3, we use the kernelization algorithm to solve vertex cover. Note that Claim 5.1 implies that if $|V| > k(k+1)$ or $|E| > k^2$, then there must be no vertex cover of size $k$.

The total runtime for Alg. 3 is:

1. $O(n^2)$ for applying rules. (Since the number of vertices in $G$ is decreases in each iteration and for every vertex we run through $O(n)$ vertices to apply Rule 2.)

2. $2^k k^2$ for solving kernel (if we use the bounded search method).

Which yields $O(n^2 + 2^k k^2)$ time complexity.