Lecture 18: Fixed-Parameter Algorithms

- Vertex Cover
- Fixed-Parameter Tractability
- Kernelization
- Connection to Approximation

Fixed Parameter Algorithms

Fixed Parameter Algorithms are an alternative way to deal with NP-hard problems instead of approximation algorithms. There are three general desired features of an algorithm:

1. Solve (NP-)hard problems
2. Run in polynomial time (fast)
3. Get exact solutions

In general, unless P = NP, an algorithm can have two of these three features, but not all three. An algorithm that has Features 2 and 3 is an algorithm in P (poly-time exact). An approximation algorithm has Features 1 and 2. It solves hard problems, and it runs fast, but it does not give exact solutions. Fixed-parameter algorithms will have Features 1 and 3. They will solve hard problems and give exact solutions, but they will not run very fast.

Idea: The idea is to aim for an exact algorithm but isolate exponential terms to a specific *parameter*. When the value of this parameter is small, the algorithm gets fast instances. Hopefully, this parameter will be small in practice.

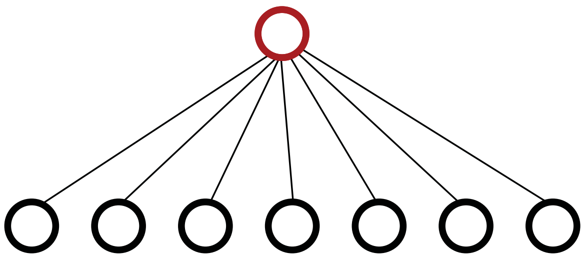
Parameter: A parameter is a nonnegative integer $k(x)$ where x is the problem input. Typically, the parameter is a natural property of the problem (some k in input). It may not necessarily be efficiently computable (e.g., OPT).

Parameterized Problem: A parameterized problem is simply the problem plus the parameter or the problem as seen with respect to the parameter. There are potentially many interesting parameterizations for any given problem.

Goal: The goal of fixed-parameter algorithms is to have an algorithm that is polynomial in the problem size n but possibly exponential in the parameter k, and still get an exact solution.

k-Vertex Cover

Given a graph $G = (V, E)$ and a nonnegative integer k, is there a set $S \subseteq V$ of vertices of size at most k, $|S| \leq k$, that covers all edges. This is a decision problem for Vertex Cover and is also NP-hard. We will use k as the parameter to develop a fixed-parameter algorithm for k-Vertex Cover. Note that we can have $k << |V|$ as the figure below shows:



Brute-force solution (bad)

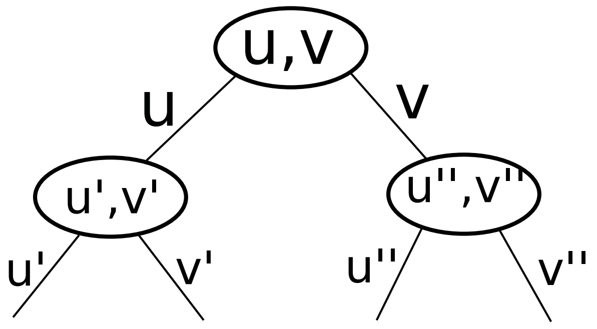
Try all $\binom{v}{k} + \binom{v}{k-1} + \cdots + \binom{v}{0}$ sets of $\leq k$ vertices. Can skip all terms smaller than $\binom{v}{k}$ because bigger sets have more coverage. Testing coverage takes $O(m)$ time where m is the number of edges. Therefore, the total runtime is $O(V^k|E|)$. It is polynomial for fixed k but not the same polynomial for all k's. It is inefficient in most cases. Hence we define $n^{f(k)}$ to be **bad**, where $n = |V| + |E|$ is the input size.

Bounded search-tree algorithm (good)

This is a general technique used to improve brute force searches. It works as follows:

- pick arbitrary edge $e = (u, v)$
- we know that either $u \in S$ or $v \in S$ (or both) but don't know which
- guess which one: try both possibilities
    1. add u to S, delete u and incident edges from G, and recurse with $k' = k - 1$.
    2. do the same but with v instead of u
    3. return the OR of the two outcomes

This is like guessing in dynamic programming but memoization doesn't help here. The recursion tree looks like the following:



At a leaf ($k = 0$), return YES if $|E| = 0$ (all edges covered). It takes $O(V)$ time to delete u or v. Therefore this has a total runtime of $(2^k|V|)$.

- $O(V)$ for fixed k
- degree of polynomial is independent of k
- also polynomial for $k = O(\lg|V|)$
- practical for e.g. $k \leq 32$
- Hence we define $f(k) \cdot n^{O(1)}$ to be **good**

Fixed Parameter Tractability

A parameterized problem is fixed-parameter tractable (FPT) if there is an algorithm with running time $\leq f(k) \cdot n^{O(1)}$, such that $f : \mathbb{N} \to \mathbb{N}$ (non negative) and k is the parameter, and the $O(1)$ degree of the polynomial is independent of k and n.

Question: Why $f(k) \cdot n^{O(1)}$ and not $f(k) + n^{O(1)}$?

Theorem: $\exists f(k) \cdot n^c$ algorithm $\iff \exists f'(k) + n^{c'}$

Proof:
$(\Leftarrow)$
 Trivial (assuming $f'(k)$ and $n^{c'}$ are ≥ 1)
$(\Rightarrow)$
 if $n \leq f(k)$, then $f(k) \cdot n^c \leq f(k)^{c+1}$

if $f(k) \leq n$ then $f(k) \cdot n^c \leq n^{c+1}$
Therefore $f(k) \cdot n^c \leq \max(f(k)^{c+1}, n^{c+1}) \leq f(k)^{c+1} + n^{c+1} = f'(k) + n^{c'}$ $\square$

Alternatively, since $xy \leq x^2 + y^2$, can just make $f'(k) = (f(k))^2$ and $c' = 2c$.
Example: $O(2^k \cdot n) \leq O(4^k + n^2)$

Kernelization

Kernelization is a simplifying self-reduction. It is a polynomial time algorithm that converts an input (x, k) into a *small* and *equivalent* input (x', k'). Here, *small* means $|x'| \leq f(k)$ and *equivalent* means the answer to x is the same as the answer to x'.

Theorem: a problem is FPT $\iff$ $\exists$ a kernelization

Proof:
$(\Leftarrow)$
Kernelize $\Rightarrow n' \leq f(k)$
Run any finite $g(n')$ algorithm
Totals to $n^{O(1)} + g(f(k))$ time
$(\Rightarrow)$
let A be an $f(k) \cdot n^c$ algorithm, then assuming k is known:
if $n \leq f(k)$, it's already kernelized.
if $f(k) \leq n$, then

1. run $A \to f(k) \cdot n^c \leq n^{c+1}$ time
2. output $O(1)$-sized YES/NO instance as appropriate (to kernelize)

if k is unknown: run A for n^{c+1} time and if it is still not done, we know it is already kernelized.

So we know (exponential) kernel exists. Recent work aims to find polynomial (even linear) kernels when possible.

Polynomial kernel for k-Vertex Cover

To create a kernel for k-Vertex Cover, the algorithm follows the following steps:

- Make graph simple by removing all self loops and multi-edges
- Any vertex of degree $> k$ must be in the cover (else would need to add $> k$ vertices to cover incident edges)

- Remove such vertices (and incident edges) one at a time, decreasing k accordingly
- Remaining graph has maximum degree $\leq k$
- Each remaining vertex covers $\leq k$ edges
- If the number of remaining edges is $> k$, answer NO and output canonical NO instance.
- Else, $|E'| \leq k^2$
- Remove all isolated vertices (degree 0 vertices)
- Now $|V'| \leq 2k^2$
- The input has been reduced to instance (V', E') of size $O(k^2)$

The runtime of the kernelization algorithm is naively $O(VE)$. ($O(V+E)$ with more work.) After this, we can apply either a brute-force algorithm on the kernel, which yields an overall runtime $O(V+E+(2k^2)^k k^2) = O(V+E+2^k k^{2k+2})$. Or we can apply a bounded search-tree solution, which yields a runtime of $O(V+E+2^k k^2)$.

The best algorithm to date: $O(kV + 1.274^k)$ by [Chen, Kanj, Xia - TCS 2010].

Connection to Approximation Algorithms

Take an optimization problem, integral OPT and consider its associated decision problem: "OPT $\leq k$?" and parameterize by k.

Theorem: optimization problem has EPTAS
(EPTAS: efficient PTAS, $f(\frac{1}{\epsilon}) \cdot n^{O(1)}$ e.g. $Approx_Partition[L17]$)
$\Rightarrow$ decision problem is FPT

Proof: (like FPTAS, pseudopolynomial algorithm)

- Say maximization problem (and $\leq k$ decision)
- run EPTAS with $\epsilon = \frac{1}{2k}$ in $f(2k) \cdot n^{O(1)}$ time.
- relative error $\leq \frac{1}{2k} < \frac{1}{k}$
- $\Rightarrow$ absolute error < 1 if OPT $\leq k$

- So if we find a solution with value $\leq k$, then OPT $\leq (1 + \frac{1}{2k}) \cdot k \leq k + \frac{1}{2}$
  Integral $\Rightarrow$ OPT $\leq k \Rightarrow$ YES
- else OPT $> k$

$\square$

Also: $=, \leq, \geq$ decision problems are equivalent with respect to FPT.

(Can use this relation to prove that EPTASs don't exists in some cases)