(Notes based on Spring 2015 notes)

TODAY: Fixed-parameter algorithms
- vertex cover
- fixed-parameter tractability
- kernelization
- connection to approximation

another alternative to approx. algorithms ~ dealing with NP-hardness

Pick any 2: (cf. friends, sleep, work)
1. hard problems
2. fast (poly.-time) algorithms
3. exact solutions

P

approx. algs. [L21] ↔ FPT [TODAY]

Idea: aim for exact algorithm, but isolate exponential term to a parameter
⇒ get fast solution for instances with small parameter value
- hope parameter is small in practice

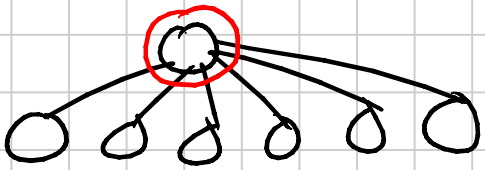
Parameter = nonnegative integer $k(x)$ (problem input)
- often a "natural" parameter (k in input)
- not necessarily efficiently computable (e.g. OPT)

Parameterized problem = problem + parameter
"problem w.r.t. parameter"
(potentially many interesting parameterizations)

Goal: polynomial in problem size n, exponential in parameter k

Example: k-Vertex Cover (NP-hard)
Given: graph $G=(V,E)$, nonnegative integer k
Q: is there a set S of $\le k$ vertices that "covers" all edges: $\forall e \in E\ \exists v \in S \cap e$
Parameter: k

Note: can have $k \ll |V|$:



Brute-force solution: (BAD)
- try all $\binom{V}{k}+\underbrace{\binom{V}{k-1}+\cdots+\binom{V}{0}}$ sets of $\le k$ vxs.
  can skip – bigger is better
- test coverage in $O(m)$ time (m = #edges)

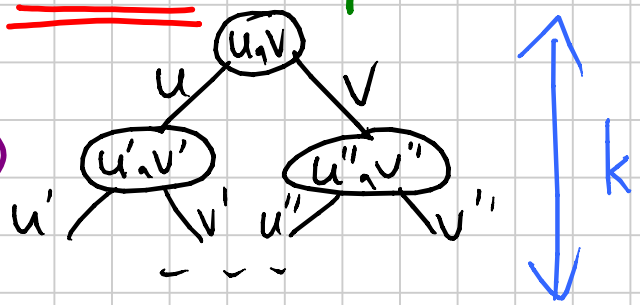
$\Rightarrow O(V^k E)$ time
- polynomial for fixed k
- but not same polynomial – e.g. not $O(V^{100})$
- inefficient in most cases

$\Rightarrow$ define $n^{f(k)}$ to be BAD
↳ here $n = |V|+|E|$

General technique: Bounded search-tree algorithm: (GOOD)

(resembles our approx. alg. for vertex cover)

- pick arbitrary edge $e=(u,v)$
- know that either $u \in S$ or $v \in S$ (or both) but don't know which
- guess: try both possibilities
  1. add u to S
     delete u & incident edges from G
     recurse with $k' = k-1$
  2. ditto with v instead of u
  - return OR of two outcomes
- like guessing in dynamic programming, but memoization doesn't help here
- recursion tree:



Total # of nodes: 2^k

- at leaf ($k=0$):
  return $|E| \stackrel{?}{=} 0$
- $O(V)$ time to delete u or v

$\Rightarrow O(2^k \cdot V)$ time

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

FPT: parameterized problem is **fixed-parameter tractable (FPT)** if there is an algorithm with running time $\leq f(k)\, n^{O(1)}$

- $f: \mathbb{N} \to \mathbb{N}$ (nonneg.)
- parameter
- indep. of k & n

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

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

Proof: ($\Leftarrow$) trivial (assuming $f'(k)$ & $n^{c'} \geq 1$)

($\Rightarrow$) if $n \leq f(k)$ then $f(k) \cdot n^c \leq f(k)^{c+1}$

if $f(k) \leq n$ then $f(k) \cdot n^c \leq n^{c+1}$

so $f(k) \cdot n^c \leq \max\{f(k)^{c+1}, n^{c+1}\}$

$\leq \underbrace{f(k)^{c+1}}_{f'(k)} + n^{\underbrace{c+1}_{c'}}$. □

OR: $xy \leq x^2 + y^2 \rightarrow f'(k) = f(k)^2$ & $c' = 2c$

Example: $O(2^k n) \leq O(4^k + n^2)$

Kernelization: a simplifying self-reduction

polynomial-time algorithm converting input (x, k) into small equivalent input (x', k')

Can have: $k' \neq k$

$|x'| \leq f(k)$ ← small

↳ answer(x) = answer(x') ← equivalent

Theorem: FPT $\Leftrightarrow$ $\exists$ kernelization

Proof: ($\Leftarrow$) kernelize $\Rightarrow$ $n' \leq f(k)$

run any finite $g(n')$ algorithm

$\Rightarrow$ $n^{O(1)} + g(f(k))$ time

($\Rightarrow$) let A be an $f(k) \cdot n^c$ algorithm

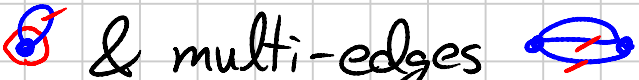
assuming k is known:

- if $n \leq f(k)$ then already kernelized
- if $f(k) \leq n$:
  - run A → $f(k) \cdot n^c \leq n^{c+1}$ time ✓
  - output $O(1)$-size YES/NO instance as appropriate (to kernelize)

if k is unknown: run A for n^{c+1} time & if not done, know already kernelized □

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

Polynomial kernel for k-vertex cover:

- make graph simple:
  - remove loops & multi-edges 
- any vertex of degree $>k$ must be in cover (else need $>k$ vertices to cover inc. edges)
- remove such vertices (& incident edges) one at a time, decreasing k accordingly

$\Rightarrow$ remaining graph has max. degree $\leq k$

$\Rightarrow$ each remaining cover vertex covers $\leq k$ edges

$\Rightarrow$ if # remaining edges $> k^2$, answer is NO: output canonical NO instance:  , $\emptyset$

- else $|E'| \leq k^2$
- remove isolated vertices

$\Rightarrow |V'| \leq 2k^2$

$\Rightarrow$ reduced to instance (V', E') of size $O(k^2)$

quadratic kernel

Can get linear kernel too!

- running time: $O(VE)$ obvious, $O(V+E)$ with more work
- if we now apply:
  - brute-force solution $\Rightarrow O(V+E+(2k^2)^k k^2)$ $= O(V+E+2^k k^{2k+2})$ time
  - bounded search-tree solution $\Rightarrow O(V+E+2^k k^2)$ time

Obs: Even naive alg. + kernel = non-trivial algs.!

Best algorithm to date: $O(kV + 1.274^k)$

[Chen, Kanj, Xia – TCS 2010]

Connection to approximation algorithms:

- take optimization problem + integral OPT
- consider associated decision problem: $OPT \leq k$?
- parameterize by k

Theorem: optimization problem has EPTAS

efficient PTAS: $f(1/\varepsilon) \cdot n^{O(1)}$

e.g. Approx-Partition [L21]

$\Rightarrow$ decision problem is FPT

Proof: (like FPTAS $\leftrightarrow$ pseudopoly. alg.)

- say maximization problem (& $\leq k$ decision)
- run EPTAS with $\varepsilon = 1/2k$ in $f(2k) \cdot n^{O(1)}$
- relative error $\leq 1/2k < 1/k$

$\Rightarrow$ absolute error < 1 if $OPT \leq k$

- so if we find solution with value $\leq k$

  then $OPT \leq (1 + 1/2k) \cdot k \leq k + 1/2$

  integral $\Rightarrow OPT \leq k \Rightarrow$ YES.
- else $OPT > k$

□

Note: The contrapositive of the above theorem often used to rule out existence of an EPTAS for some problems