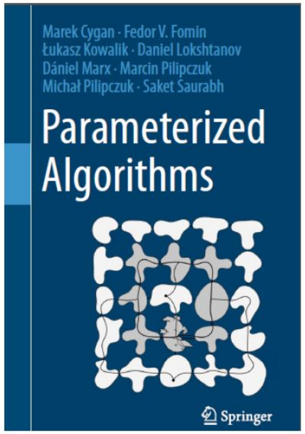
Advanced topics on Algorithms

Luciano Gualà
www.mat.uniroma2.it/~guala/

Parameterized algorithms
Episode I
(pilot)

main reference:



pick any two

We would like

1. to solve (NP-)hard problems
2. fast (polynomial time) algorithms
3. to compute exact solutions

apx algorithms

problems in P

parameterized algorithms

idea: aim for exact algorithms, but confine the exponential dependence to a parameter

goal: an algorithm whose running time is polynomial in the problem size n and exponential in the parameter

⇒ exact algorithm running fast provided k is small

parameter: k(x) nonnegative integer associated to the instance x

parameterized problem: a problem + a parameter

we say: “problem P w.r.t. parameter k”

k-Vertex Cover

Input:

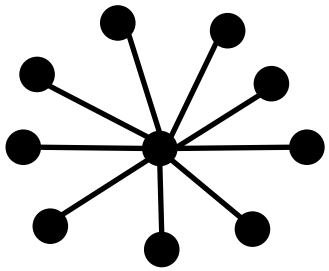
- a graph $G=(V,E)$
- a nonnegative integer k

question:

is there a vertex cover $S \subseteq V$ of size at most $|S| \leq k$

parameter: k

example: k can actually be small



star graph

Brute-force solution:

- try all $O(n^k)$ subsets of k vertices
- for each subset S:
  - check whether S is a vertex cover

running time: $O(n^k\, m)$ $\boxed{\text{BAD}}$ $n^{f(k)}$ exponent depends on k

Definition (FPT)

A parameterized problem is Fixed Parameter Tractable (FPT) if it can be solved in time

$$\underbrace{f(k)}_{\text{any function}}\; n^{\overbrace{O(1)}^{\text{independent of } n \text{ \& } k}}$$

Bounded-search tree algorithm:

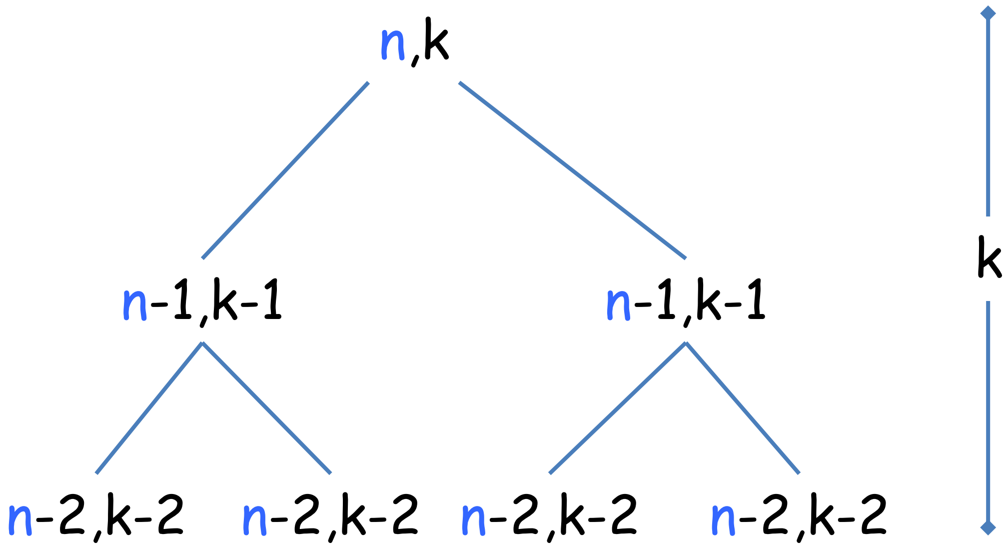
- consider any (uncovered) edge $e=(u,v)$
  (if there is no uncovered edge return TRUE)

- either $u \in S$ or $v \in S$ (or both)
- guess which one: try both possibilities

  1. – add u to S, delete u and all incident edges from G
     - recurse on G with $k'=k-1$
  2. – add v to S, delete v and all incident edges from G
     - recurse on G with $k'=k-1$
  3. - return the OR of the two outcomes

- base case: $k=0$
  if there is an (uncovered) edge in G return FALSE, return TRUE otherwise

running time: analysis of the recursion tree



- height of the tree $\leq$ k
- # of nodes $O(2^k)$
- any node costs $O(n)$

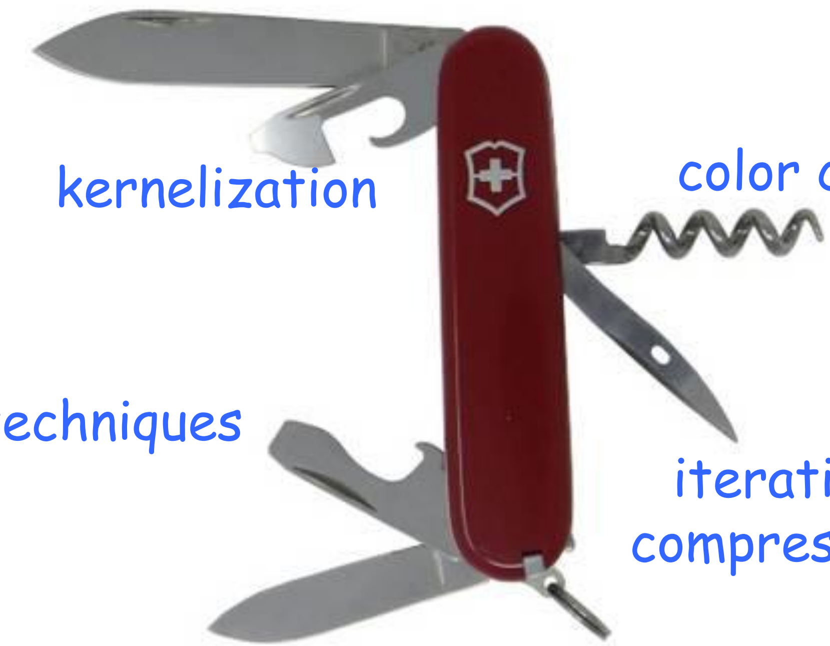
time $O(2^k n)$ $\boxed{\text{GOOD}}$ FPT algorithm

Toolbox (to show a problem is FPT)

bounded-search trees

kernelization

color coding



algebraic techniques

iterative compression

treewidth

kernelization

idea: pre-processing an instance in order to simplify it to a much smaller equivalent instance

kernelization: a polynomial-time algorithm that converts an instance (x,k) into a small and equivalent instance $\underbrace{(x',k')}_{\text{kernel}}$

equivalent: the answer of (x,k) is the same of the answer of (x',k')

small: the size of (x',k') is $\leq f(k)$

$$(x,k) \xrightarrow{\text{kernelization}} (x',k')$$

obs: after you can run every algorithm you want on the kernel

Theorem

A parameterized problem is FPT iff it admits a kernelization.

proof

($\Leftarrow$)

kernelize and obtain an instance of size $n' \le f(k)$.

run any finite algorithm with running time $g(n)$ on the kernel.

Total running time: $n^{O(1)}+g(f(k))$.

($\Rightarrow$)

Let A be an $f(k)n^c$ time algorithm

if $n \le f(k)$ the instance is already kernelized

if $f(k) \le n$

1. solve the instance by running A in time $f(k)n^c \le n^{c+1}$ (polynomial)
2. output a $O(1)$-size YES/NO instance as appropriate (to kernelize)

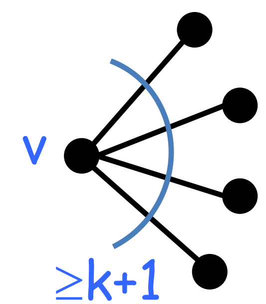
$\blacksquare$

polynomial kernel for k-Vertex Cover

based on reduction rules

rule 1: if there is a vertex v of degree $\geq$ k+1, then delete v (and all its incident edges) from G and decrement the parameter k by 1. The new instance is (G-v,k-1)

rule 2: if G contains an isolated (0-degree) vertex v, delete v from G. The new instance is (G-v,k)



polynomial kernel for k-Vertex Cover

based on reduction rules

rule 1: if there is a vertex v of degree $\geq k+1$, then delete v (and all its incident edges) from G and decrement the parameter k by 1. The new instance is $(G-v,k-1)$

rule 2: if G contains an isolated (0-degree) vertex v, delete v from G. The new instance is $(G-v,k)$

Lemma

If (G,k) is a YES-instance and none of the above rules is applicable to G, then $|E(G)| \leq k^2$ and $|V(G)| \leq 2k^2$.

proof

Since rule 1 is not applicable every vertex has degree $\leq k$

⇒ k vertices can cover $\leq k^2$ edges

Since rule 2 is not applicable, every vertex has an incident edge

⇒ $|V(G)| \leq 2k^2$ (worst case $E(G)$ is a matching of k^2 edges)

■

polynomial kernel for k-Vertex Cover

based on reduction rules

rule 1: if there is a vertex v of degree $\geq$ k+1, then delete v (and all its incident edges) from G and decrement the parameter k by 1. The new instance is (G-v,k-1)

rule 2: if G contains an isolated (0-degree) vertex v, delete v from G. The new instance is (G-v,k)

rule 3: let (G, k) be an instance such that rule 1 & 2 are not applicable. If $k<0$ or G has more than k^2 edges or more than $2k^2$ vertices, conclude that (G,k) is a NO-instance. Output a canonical NO-instance.

running time: naive implementation $O(n^2)$

a clever implementation $O(n+m)$

solving k-Vertex Cover:

kernelization + bound-search tree alg ⇒ $O(n+m+2^k k^2)$

what can be the parameter k?

- the size k of the solution we are looking for
- the maximum degree of the input graph
- the dimension of the point set in the input
- the length of the strings in the input
- the length of the clauses in the input Boolean formula
- the number of moves in a puzzle game
- the budget in an augmenting problem
- ...

Examples of NP-hard problems that are FPT:

- finding a vertex cover of size k
- finding a path of length k
- finding k disjoint triangles
- drawing a graph in the plane with k edge crossings
- finding disjoint paths that connects k pairs of vertices
- finding the maximum clique in a graph of maximum degree k
- ...

W[1]-hardness

Negative evidence similar to NP-completeness. If a problem is W[1]-hard, then the problem is not FPT unless FPT=W[1].

Examples of W[1]-hard problems:

- finding a clique/independent set of size k
- finding a dominating set of size k
- finding k pairwise disjoint sets
- given a graph G, finding k vertices that covers at least s edges (partial Vertex Cover)
- given a Boolean formula, decide if can be satisfied by assigning TRUE to at most k variables
- ...

games to play

- The FPT vs W[1]-hardness game
  - is the problem FPT?
- The $f(k)$ game for FPT problems
  - what is the best $f(k)$ dependence on the parameter?
- The exponent game for W[1]-hard problems
  - what is the best possible dependence on k in the exponent?