Algorithmen / Algorithms II

Peter Sanders

Exercise:

Daniel Seemaier, Tobias Heuer

Institute of Theoretical Informatics

Web:

`http://algo2.iti.kit.edu/AlgorithmenII_WS20.php`

9 Fixed-Parameter-Algorithms

Practical observation: Even for NP-hard problems we are in some cases able to find exact solutions:

. . . for easy instances

How does one characterize “easy”?

By an additional Parameter k (next to the input size)

Example: $k =$ output size

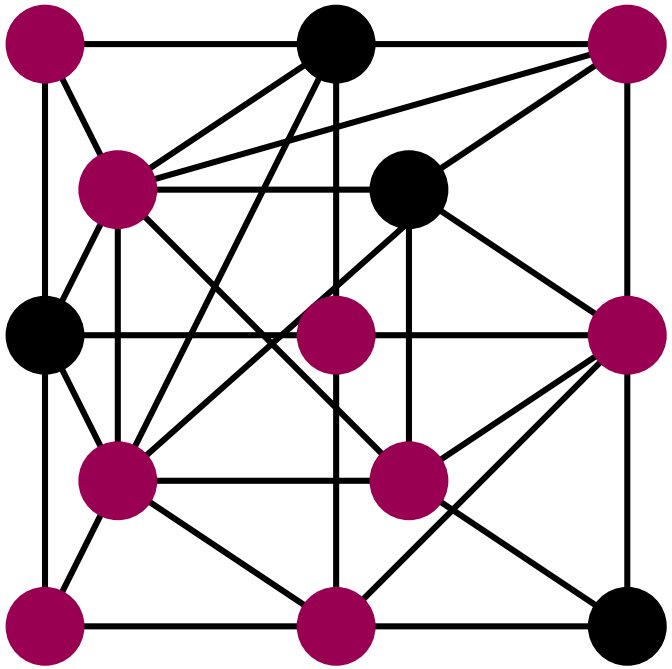
[Niedermeier, Invitation to Fixed Parameter Algorithms, Oxford U. Press, 2006]

Example: VERTEX COVER

Given: undirected graph $G = (V, E)$,

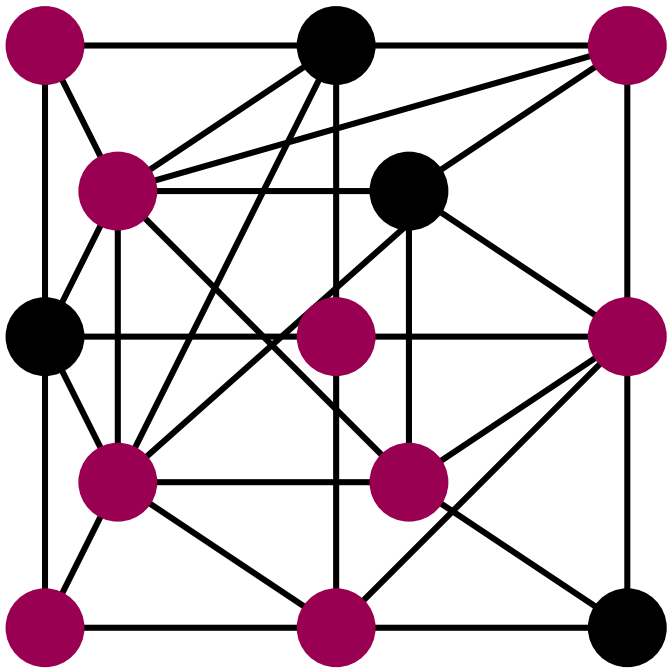
Parameter $k \in \mathbb{N}$.

Question: $\exists V' \subseteq V : |V'| = k \wedge \forall \{u, v\} \in E : u \in V' \vee v \in V'$



VERTEX COVER Basics

- One of the 21 classical NP-hard problems
- Trivial $\mathrm{O}\left(n^{k+1}\right)$ brute-force algorithm
- Equivalent to max. independent set



Fixed parameter tractable

A formal language $L \in$ FPT respective parameter $k \Leftrightarrow$

$\exists$ algorithm with runtime $\mathrm{O}(f(k) \cdot p(n))$,

f computable function, not depending on n

p polynomial, not depending on k.

Examples: $2^k n^2$, $k^{k!} n^{333}$, $n + 1.1^k$

Counter examples: n^k, $n^{\log \log k}$

Example: VERTEX COVER

Theorem: Vertex Cover in in FPT regarding the parameter output complexity

We develop algorithms using two design techniques which are also relevant in practice:

1. Kernelization: Reduction rules reduce the problem to size $\mathrm{O}(f(k))$
2. Systematic search with bounded depth.

Naive depth-bounded search

Function vertexCover$(G = (V, E), k)$: Boolean

  if $|E| = 0$ **then return** true

  if $k = 0$ **then return** false

  pick any edge $\{u, v\} \in E$

  return vertexCover$(G - v, k - 1) \vee$

  vertexCover$(G - u, k - 1)$

Operation $G - v$ removes node v and its incident edges

Naive depth-bounded search – correctness

Function vertexCover$(G = (V,E),k)$: Boolean

  if $|E| = 0$ **then return** true // trivial problem

  if $k = 0$ **then return** false // impossible problem

  pick any edge $\{u,v\} \in E$ // u or v have to be in the cover !

  //case distinction:

  return vertexCover$(G - v, k-1) \vee$ // case v in cover

  vertexCover$(G - u, k-1)$ // case u in cover

Naive depth-bounded search – runtime

Function vertexCover$(G=(V,E),k)$: Boolean

if $|E|=0$ **then return** true // $\mathrm{O}(1)$

if $k=0$ **then return** false // $\mathrm{O}(1)$

pick any edge $\{u,v\}\in E$ // $\mathrm{O}(1)$

return vertexCover$(G-v,k-1)\vee$ // $\mathrm{O}(n+m)+T(k-1)$

vertexCover$(G-u,k-1)$ // $T(k-1)$

recursion depth $k \rightsquigarrow \mathrm{O}\left(2^k\right)$ recursive calls

therefore runtime $\mathrm{O}\left(2^k(n+m)\right)$.

Formally: Solution to the recurrence $T(k)=(n+m)+2T(k-1)$

Kernelization for Vertex Cover

[Buss 1993]

Observation: $\forall v \in V$: degree$(v) > k \Longrightarrow v \in$ solution $\vee$ non-solvable

Function kernelVertexCover$(G = (V, E), k)$: Boolean

  if $|E| = 0$ **then return** true

  while $\exists v \in V$: degree$(v) > k$ **do**

    if $k = 0$ **then return** false

    $G := G - v$

    $k := k - 1$

  remove isolated nodes

  if $|E| > k^2$ **then return** false

  return vertexCover(G, k)

Kernelization for Vertex Cover – correctness

Observation: $\forall v \in V$: degree$(v) > k \Longrightarrow v \in$ solution $\vee$ non-solvable

Function kernelVertexCover$(G = (V, E), k)$: Boolean

  if $|E| = 0$ **then return** true

  while $\exists v \in V$: degree$(v) > k$ **do** // see observation

    if $k = 0$ **then return** false // $m > k = 0$!

    $G := G - v$ // v has to be part of the solution!

    $k := k - 1$

  remove isolated nodes // useless nodes

  if $|E| > k^2$ **then return** false // $\leq k$ nodes $\times \leq k$ neighbours

  return vertexCover(G, k)

Kernelization for Vertex Cover – runtime

Function kernelVertexCover$(G=(V,E),k)$: Boolean

  if $|E|=0$ **then return** true

  while $\exists v \in V$: degree$(v) > k$ **do** // $\leq k\times$

    if $k=0$ **then return** false // $m \geq k > 0$!

    G:= $G-v$// v has to be part of the solution! // $\mathrm{O}(n+m)$

    k:= $k-1$

  remove isolated nodes // useless nodes

  //Overall $\mathrm{O}((n+m)k)$

  if $|E| > k^2$ **then return** false // $\leq k$ nodes $\times \leq k$ neighbours

  return vertexCover(G,k) // $\mathrm{O}(2^k k^2)$

Overall $\mathrm{O}\big((n+m)k+2^k k^2\big)$ Exercise: $\mathrm{O}\big(n+m+2^k k^2\big)$

Kernelization for Vertex Cover – example

Function kernelVertexCover$(G = (V, E), k)$: Boolean

  if $|E| = 0$ **then return** true

  while $\exists v \in V$: degree$(v) > k$ **do**

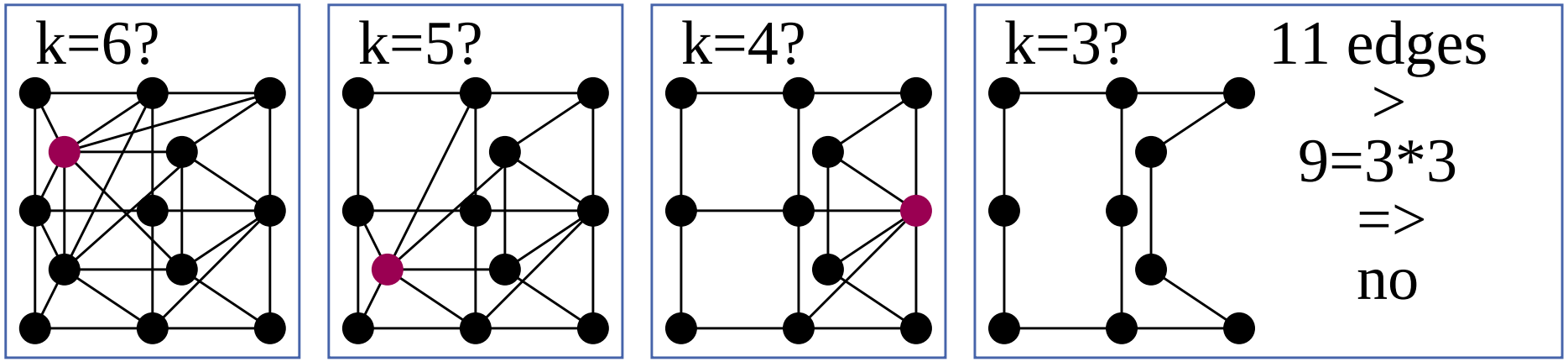
    if $k = 0$ **return** false

    $G := G - v; \quad k := k - 1$

  remove isolated nodes

  if $|E| > k^2$ **then return** false

  return vertexCover(G, k)



Reduction rule

0: not part of the cover

1: w.l.o.g. neighbour part of the cover Exercise: vertex cover for trees?

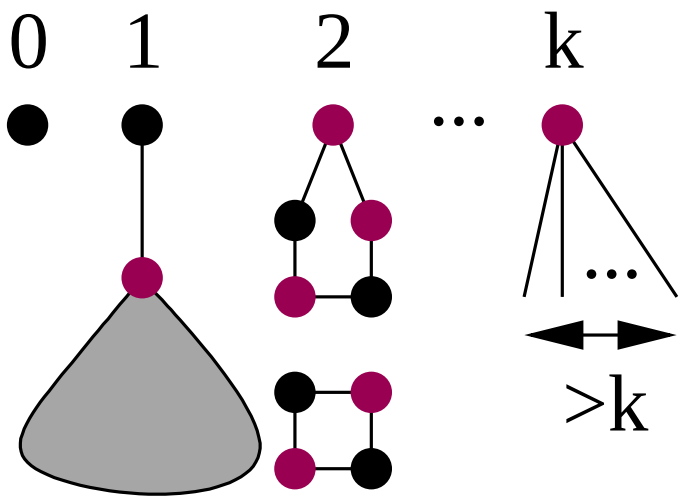
2: possible too, but more complicated. But,

trivial if all nodes have a degree of two

"Take every second node"

$> k$: has to be part of the cover

More rules ?



Improved depth-bound search

Function vertexCover2$(G=(V,E),k)$: Boolean

  if $|E|=0$ **then return** true // $\mathrm{O}(1)$

  if $k=0$ **then return** false // $\mathrm{O}(1)$

  if $\exists v\in V$: degree$(v)=1$ **then**

    return vertexCover2$(G-$neighbor$(v),k-1)$

  if $\exists v\in V$: degree$(v)\geq 3$ **then**

    return vertexCover2$(G-v,k-1)\vee$

      vertexCover2$(G-\mathcal{N}(v),k-|\mathcal{N}(v)|)$

  assert all nodes have degree 2

  return vertexCoverCollectionOfCycles(G,k)

Analysis: Solution to the recurrence

$$T(k)=(n+m)+T(k-1)+T(k-3)=\mathrm{O}\big((n+m)1.4656^k\big)$$

$\rightsquigarrow$ use generating functions

Further improvements

- kernels of size $2k$ (using Matching-Algorithms)
- reduce the time per recursive call to $\mathrm{O}(1)$
- detailed case distinction $\rightsquigarrow$

  smaller constant in the exponential part

$$\rightsquigarrow \mathrm{O}\left(1.2738^k + kn\right)$$

[Chen Kanj Xia 2006]

Summary

- important subclasses of NP-hard problems can be solvable in polynomial time
- kernelization is an important strategy in preprocessing or during the algorithm for optimization problems – including polynomially solvable ones. For example Max-Cardinality matching.
- Making brute-force algorithms less brute-force sometimes leads to provable exponential speedup. $\rightsquigarrow$ non-trivial analysis and design techniques