

Algorithmen II

Peter Sanders, Timo Bingmann

Übungen:

Tobias Heuer, Sebastian Lamm

Institut für Theoretische Informatik

Web:

http://algo2.itl.kit.edu/AlgorithmenII_WS19.php

9 Fixed-Parameter-Algorithmen

Praktische Beobachtung: Auch bei NP-harten Problemen können wir u.U. exakte Lösungen finden:
... für **einfache** Instanzen.

Wie charakterisiert man Einfachheit ?

Durch einen weiteren Parameter k (neben der Eingabegröße)

Beispiel: $k = \text{Ausgabegröße}$

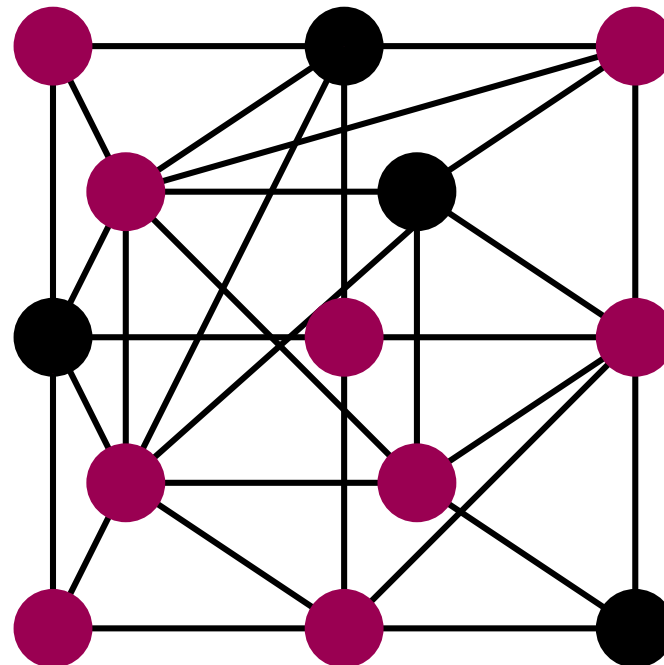
[Niedermeier, Invitation to Fixed Parameter Algorithms, Oxford U. Press, 2006]

Beispiel: VERTEX COVER (Knotenüberdeckung)

Gegeben: ungerichteter Graph $G = (V, E)$,

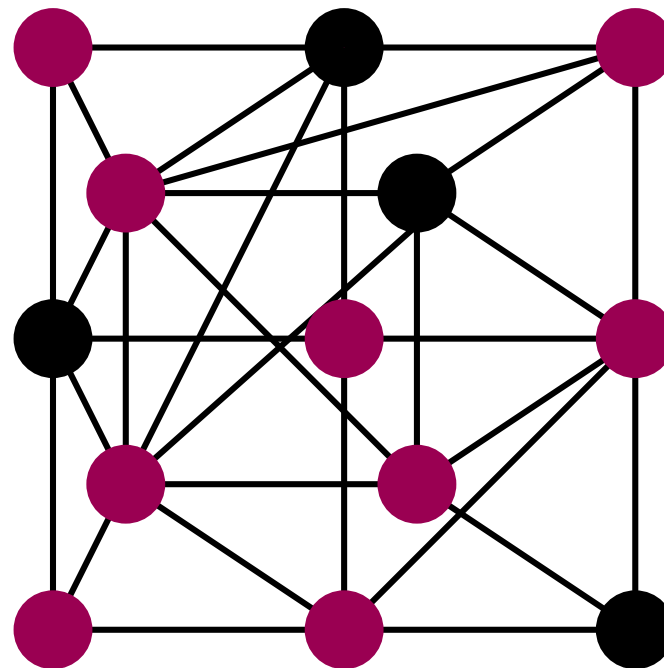
Parameter $k \in \mathbb{N}$.

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



VERTEX COVER Grundlegendes

- Eines der (21) klassischen **NP-harten** Probleme
- Trivialer $O(n^{k+1})$ Brute-Force-Algorithmus
- Äquivalent zu max. independent set (Komplement)



Fixed parameter tractable

Eine formale Sprache $L \in \text{FPT}$ bzgl. Parameter $k \Leftrightarrow$

\exists Algorithmus mit Laufzeit $\mathcal{O}(f(k) \cdot p(n))$,

f berechenbare Funktion, nicht von n abhängig,

p Polynom, nicht von k abhängig.

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

Gegenbeispiele: n^k , $n^{\log \log k}$

Beispiel: VERTEX COVER

Satz: Vertex Cover ist in FPT bzgl. des Parameters
Ausgabekomplexität

Wir entwickeln Algorithmen mittels zweier auch praktisch wichtiger
Entwurfstechniken:

1. **Kernbildung:** (Kernelization) Reduktionsregeln reduzieren Problem
auf Größe $O(f(k))$
2. Systematische **Suche** mit **beschränkter Tiefe**.

Naive tiefenbeschränkte Suche

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 tiefenbeschränkte Suche – Korrektheit

```
Function vertexCover( $G = (V, E), k$ ) : Boolean
  if  $|E| = 0$  then return true           // triviales Problem
  if  $k = 0$  then return false          // unmögliches Problem
  pick any edge  $\{u, v\} \in E$  //  $u$  oder  $v$  müssen im cover sein !
  //Fallunterscheidung:
  return vertexCover( $G - v, k - 1$ )  $\vee$  // Fall  $v$  in cover
         vertexCover( $G - u, k - 1$ ) // Fall  $u$  in cover
```


Naive tiefenbeschränkte Suche – Laufzeit

```
Function vertexCover( $G = (V, E), k$ ) : Boolean
  if  $|E| = 0$  then return true //  $O(1)$ 
  if  $k = 0$  then return false //  $O(1)$ 
  pick any edge  $\{u, v\} \in E$  //  $O(1)$ 
  return vertexCover( $G - v, k - 1$ )  $\vee$  //  $O(n + m) + T(k - 1)$ 
           vertexCover( $G - u, k - 1$ ) //  $T(k - 1)$ 
```

Rekursionstiefe $k \rightsquigarrow O(2^k)$ rekursive Aufrufe also

Laufzeit $O(2^k(n + m))$.

Formaler: Lösung der **Rekurrenz** $T(k) = (n + m) + 2T(k - 1)$

Kernbildung für Vertex Cover

Beobachtung: $\forall v \in V : \text{degree}(v) > k \implies v \in \text{Lösung} \vee \text{unlösbar}$

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

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

while $\exists v \in V : \text{degree}(v) > k$ **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)

Kernbildung für Vertex Cover – Korrektheit

Beobachtung: $\forall v \in V : \text{degree}(v) > k \implies v \in \text{Lösung} \vee \text{unlösbar}$

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

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

while $\exists v \in V : \text{degree}(v) > k$ **do** // siehe Beobachtung

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

$G := G - v$ // v muss in die Lösung!

$k := k - 1$

remove isolated nodes // nutzlose Knoten

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

return vertexCover(G, k)

Kernbildung für Vertex Cover – Laufzeit

```

Function kernelVertexCover( $G = (V, E), k$ ) : Boolean
  if  $|E| = 0$  then return true
  while  $\exists v \in V : \text{degree}(v) > k$  do                                //  $\leq k \times$ 
    if  $k = 0$  then return false                                       //  $m \geq k > 0 !$ 
     $G := G - v$  //  $v$  muss in die Lösung!                               //  $O(n + m)$ 
     $k := k - 1$ 
  remove isolated nodes                                                // nutzlose Knoten
  // Insgesamt  $O((n + m)k)$ 
  if  $|E| > k^2$  then return false //  $\leq k$  nodes  $\times \leq k$  neighbors
  return vertexCover( $G, k$ )                                           //  $O(2^k k^2)$ 

Insgesamt  $O((n + m)k + 2^k k^2)$            Aufgabe:  $O(n + m + 2^k k^2)$ 
  
```

Kernbildung für Vertex Cover – Beispiel

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

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

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

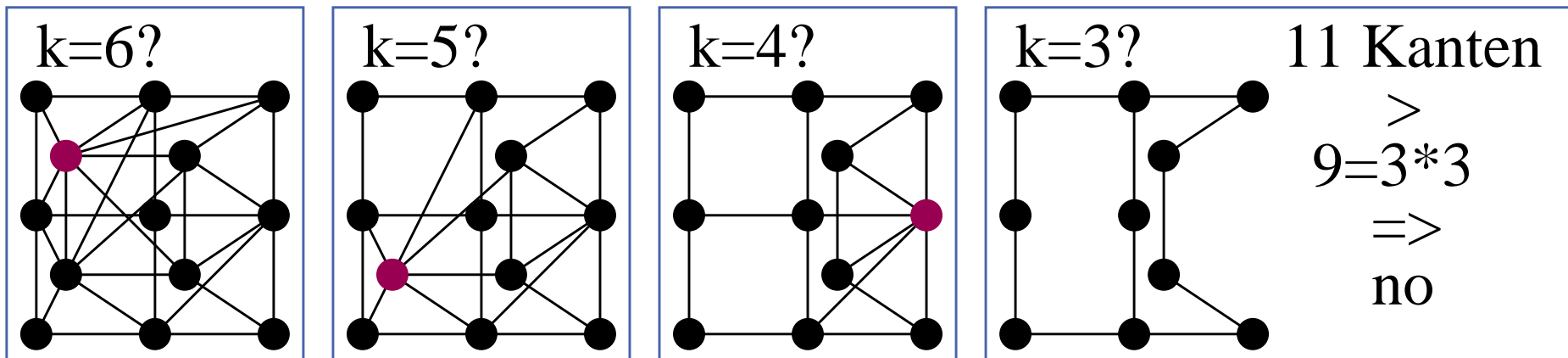
if $k = 0$ **return** false

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

remove isolated nodes

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

return vertexCover(G, k)



Reduktionsregeln

0: nicht im cover

1: OBdA Nachbar im cover

Aufgabe: vertex cover für Bäume?

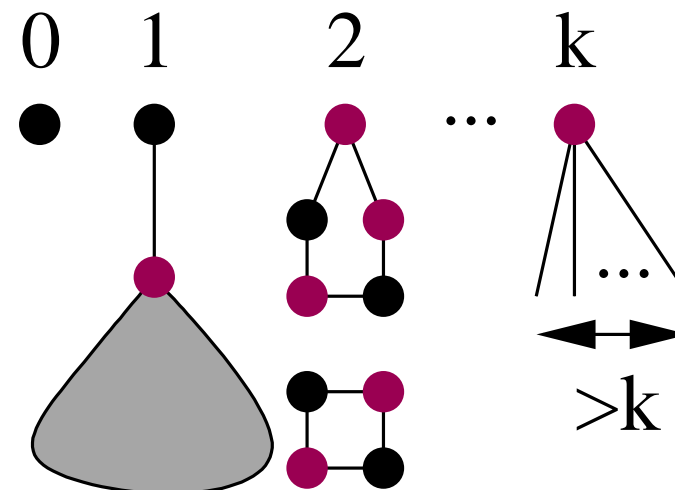
2: geht auch aber komplizierter. Aber,

trivial wenn alle Knoten Grad zwei haben

“Nimm jeden zweiten Knoten”

$> k$: muss ins cover

Mehr Regeln ?



Verbesserte tiefenbeschränkte Suche

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

if $|E| = 0$ **then return** true // $O(1)$

if $k = 0$ **then return** false // $O(1)$

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

return vertexCover2($G - \text{neighbor}(v), k - 1$)

if $\exists v \in V : \text{degree}(v) \geq 3$ **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)

Analyse: Lösung der **Rekurrenz**

$$T(k) = (n + m) + T(k - 1) + T(k - 3) = O((n + m)1.4656^k)$$

\rightsquigarrow benutze **erzeugende Funktionen**

Weitere Verbesserungen

- Kerne der Größe $2k$ (mittels Matching-Algorithmen)
- Reduziere Zeit pro rekursivem Aufruf auf $O(1)$
- Detaillierte Fallunterscheidungen \rightsquigarrow
kleinere Konstante im exponentiellen Teil

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

[Chen Kanj Xia 2006]

Zusammenfassung

- Wichtige Teilklassen NP-harter Probleme können polynomiell lösbar sein
- Kernbildung is wichtige Vor/Zwischenverarbeitungsstrategie für Optimierungsproblem – auch polynomiell lösbare. Zum Beispiel Max-Cardinality matching.
- Brute-force Algorithmen weniger brute-force machen bringt beweisbar exponentielle Beschleunigung. \rightsquigarrow nichttriviale Analyse- und Entwurfs-Techniken