Algorithmen für NP-schwere Probleme

Holger Dell

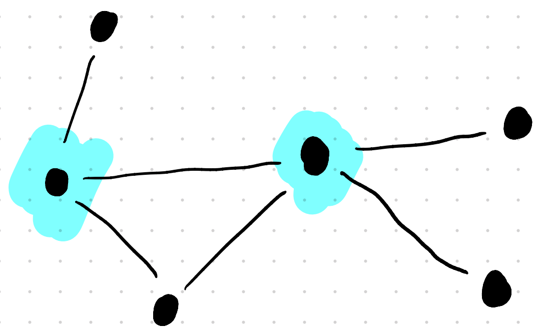
1. Approximationsalgorithmen

→ Approximativ optimale Lösung reicht

2. Parametrisierte Algorithmen

→ Laufzeit muss nicht polynomiell sein

Erinnerung: Minimum Vertex-Cover



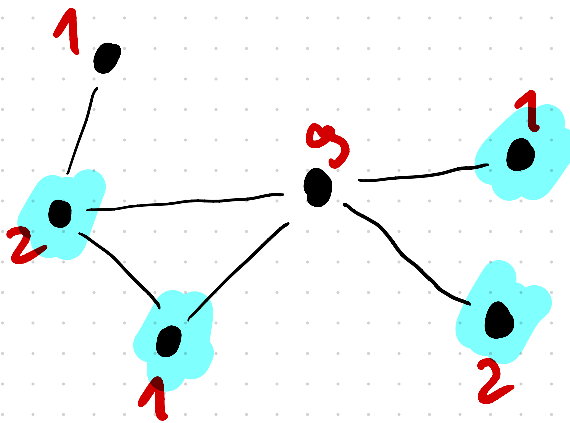
Kleinste Menge $C \subseteq V(G)$,
die alle Kanten trifft.

Minimum Vertex-Cover ist NP-schwer.

Allgemeiner:

Minimum Vertex-Cover mit Gewichten

$w: V(G) \to \mathbb{N}$



~~Kleinste~~ Menge $C \subseteq V(G)$, die alle Kanten trifft und

$$w(C) = \sum_{v \in C} w(v)$$

minimiert.

Approximationsalgorithmus durch LP-Rundung

Erickson J.6

Reduziere Vertex-Cover auf ILPs

minimiere $\sum_{v} w(v) \cdot x(v)$

s.d. $x(u) + x(v) \geq 1$ für alle Kanten $\{u, v\}$

$x(v) \in \{0, 1\}$ für alle Knoten v

OPT = Gewicht des leichtesten Vertex-Covers

LP-Relaxierung

minimiere $\sum_{v} w(v) \cdot x(v)$

s.d. $x(u) + x(v) \geq 1$ für alle Kanten $\{u, v\}$

$0 \leq x(v) \leq 1$ für alle Knoten v

$\rightsquigarrow$ Löse LP in Polynomialzeit und erhalte optimale Lösung x^*

Begriffe

OPT = Wert der optimalen integralen Lösung

= Kleinstes Gewicht eines VC's

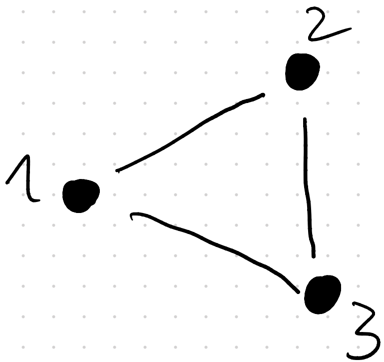
NP-schwer

x^* = optimale fraktionale Lösung

in Polyzeit

$\widetilde{OPT}$ = Wert von x^* = $\sum_V w(v) \cdot x^*(v)$

Beispiel



(Gewichte = 1)

$$\min \quad x_1 + x_2 + x_3$$
$$\text{s.d.} \quad x_1 + x_2 \geq 1$$
$$x_2 + x_3 \geq 1$$
$$x_3 + x_1 \geq 1$$
$$0 \leq x_v \leq 1 \quad \forall v \in \{1,2,3\}$$

$$\rightsquigarrow \quad x_1^* = x_2^* = x_3^* = 1/2$$

$OPT = 2$ $\qquad \widetilde{OPT} = 3/2$

Idee: Runde die optimale fraktionale Lsg

Beobachtung: Für jede Kante $\{u,v\}$ gilt
$x^*(u) \geq \frac{1}{2}$ oder $x^*(v) \geq \frac{1}{2}$

Daher ist

$$x'(v) = \begin{cases} 1 & \text{falls } x^*(v) \geq \frac{1}{2} \\ 0 & \text{sonst} \end{cases}$$

eine Lösung des ILPs.

Wie gut ist die gerundete Lösung x'?

$$(*) \qquad x'(v) \leq 2 \cdot x^*(v) \quad \forall v \in V$$

$$\Rightarrow \sum_v w(v) \cdot x'(v) \overset{(*)}{\leq} 2 \sum_v w(v) \cdot x^*(v)$$

$$= 2 \cdot \widetilde{OPT} \leq 2 \cdot OPT$$

Übersicht: Algorithmus LP-Rundung

$A \left\{ \begin{array}{l} \text{1. Gegeben } G \\ \text{2. Formuliere als ILP} \\ \text{3. Löse die LP-Relaxierung } \rightsquigarrow x^* \\ \text{4. Runde } x^* \text{ zu } x' \end{array} \right.$

$$\mathrm{OPT} \leq \text{Wert von } x' \leq 2 \cdot \mathrm{OPT}$$

Approximationsfaktor

E: J.2 A ist ein 2-Approximationsalgorithmus

Randomized LP-Rounding

E: J.7

Idee: Nutze x^* als Wahrscheinlichkeit

$$x'(v) = \begin{cases} 1 & \text{mit WSK } x^*(v) \\ 0 & \text{sonst} \end{cases}$$

Erwarteter Wert von x':

$$\mathbb{E}\left[\sum_v w(v) \cdot x'(v)\right] = \sum w(v) \cdot \mathbb{E}[x'(v)]$$

$$= \sum w(v) \cdot x^*(v) = \widetilde{OPT} \leq OPT$$

__Aber:__ x' ist nicht immer zulässig!!

Für jede Kante $\{u,v\}$ gilt:

$$\Pr(x'(u)=0 \wedge x'(v)=0) = \Pr(x'(u)=0) \cdot \Pr(x'(v)=0)$$

$$= \underbrace{(1-x^*(u))}_{\leq \frac{1}{2} \text{ oder}} \cdot \underbrace{(1-x^*(v))}_{\leq \frac{1}{2}}$$

$$\leq \frac{1}{2}$$

$\rightsquigarrow$ Im Erwartungswert wird mindestens die Hälfte aller Kanten abgedeckt.

$O(\log n)$ - Approximation von VC durch Randomisiertes Runden

1. Berechne x^* für G
2. Runde x^* randomisiert zu x'
3. Nimm alle Knoten v mit $x'(v)=1$ in C
4. Setze $G = G - C$ und gehe zu 1.
5. Nach $O(\log n)$ Schritten ist C ein VC

$\rightsquigarrow$ $\text{OPT} \leq w(C) \leq O(\log n) \cdot \text{OPT}$

Zwei kombinatorische Approximationsalgorithmen:

EJ.3 → Gieriges Vertex-Cover

EJ.5 → Dummes Vertex-Cover

$\underline{\textsc{GreedyVertexCover}(G)\text{:}}$

$C \leftarrow \varnothing$

while G has at least one edge

  $v \leftarrow$ vertex in G with maximum degree

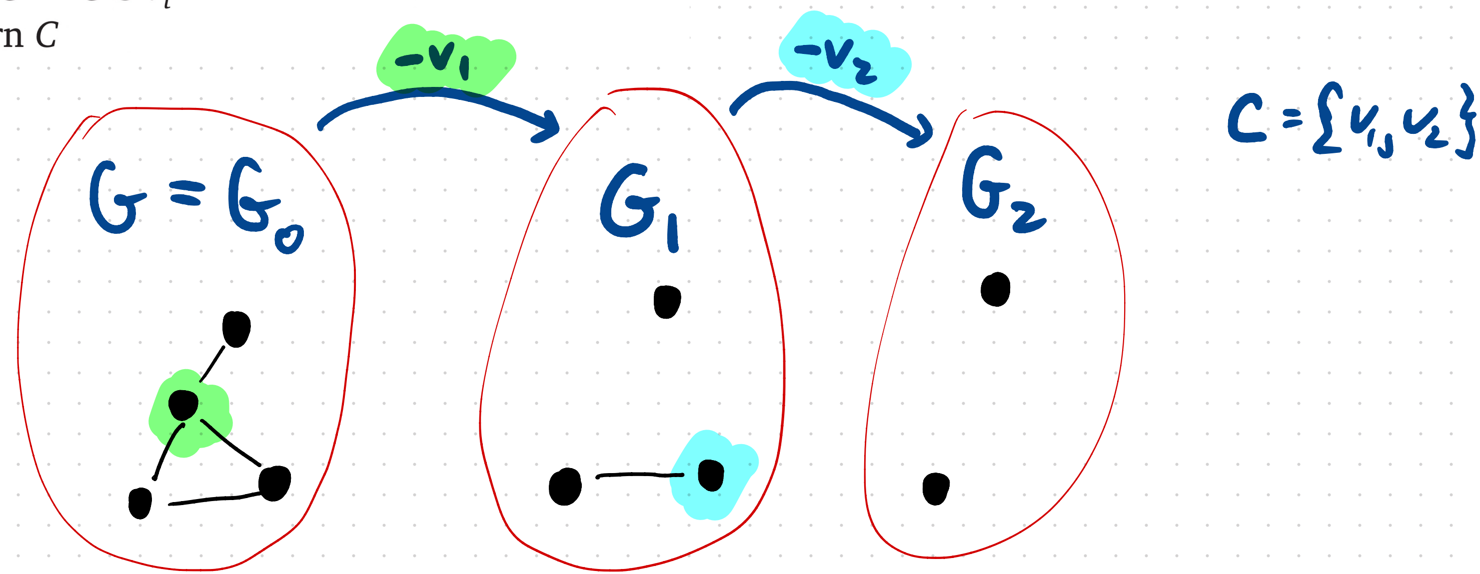
  $G \leftarrow G \setminus v$

  $C \leftarrow C \cup v$

return C

Frage: Wie gut ist die Approximation?

```
GreedyVertexCover(G):
    C ← ∅
    G_0 ← G
    i ← 0
    while G_i has at least one edge
        i ← i + 1
        v_i ← vertex in G_{i−1} with maximum degree
        d_i ← deg_{G_{i−1}}(v_i)
        G_i ← G_{i−1} \ v_i
        C ← C ∪ v_i
    return C
```



Satz: Greedy VC ist ein $\mathcal{O}(\log n)$-Approx-Alg

Sei C^* ein optimales VC für G. $\rightsquigarrow OPT = |C^*|$

$\Rightarrow C^*$ ist ein VC für alle G_{i-1}

$$\Rightarrow \sum_{v \in C^*} \deg_{G_{i-1}}(v) \geq |G_{i-1}| := \text{\# Kanten in } G_{i-1}$$

$$\Rightarrow d_i \geq \frac{|G_{i-1}|}{|C^*|} = \frac{|G_{i-1}|}{OPT}$$

$$\Rightarrow \sum_{i=1}^{OPT} d_i \geq \sum_{i=1}^{OPT} \frac{|G_{i-1}|}{OPT} \geq \sum_{i=1}^{OPT} \frac{|G_{OPT}|}{OPT} = |G_{OPT}| = |G| - \sum_{i=1}^{OPT} d_i$$

$$\Rightarrow \underbrace{\sum_{i=1}^{OPT} d_i}_{\text{\# durch } v_1, \dots, v_{OPT} \text{ abgedeckte Kanten}} \geq \frac{1}{2} |G|$$

$\Rightarrow$ Alle OPT Runden wird die Hälfte der Kanten abgedeckt

$$\Rightarrow \quad OPT \leq \widehat{OPT} \leq O(\log n) \cdot OPT$$

<u>DUMBVERTEXCOVER(G):</u>

$C \leftarrow \varnothing$
while G has at least one edge
  $uv \leftarrow$ any edge in G
  $G \leftarrow G \setminus \{u, v\}$
  $C \leftarrow C \cup \{u, v\}$
return C

optimale Lösung C^* muss u oder v enthalten!

Hier werden also höchstens doppelt so viele Knoten genommen wie in C^*

$$\Rightarrow \quad OPT \le \overset{|C| \,=}{\widetilde{OPT}} \le 2 \cdot OPT$$

Übersicht Approximationsalgorithmen für VC

		Approx.-faktor
LP	Runden	2
LP	Randomisiert Runden	$O(\log n)$
	Gieriges VC	$O(\log n)$
	Dummes VC	2

E:V.1 Zwei Approximationsalgorithmen für Load Balancing (Scheduling)

Load Balancing

Gegeben: jobs $1, \dots, n$ machines $1, \dots, m$

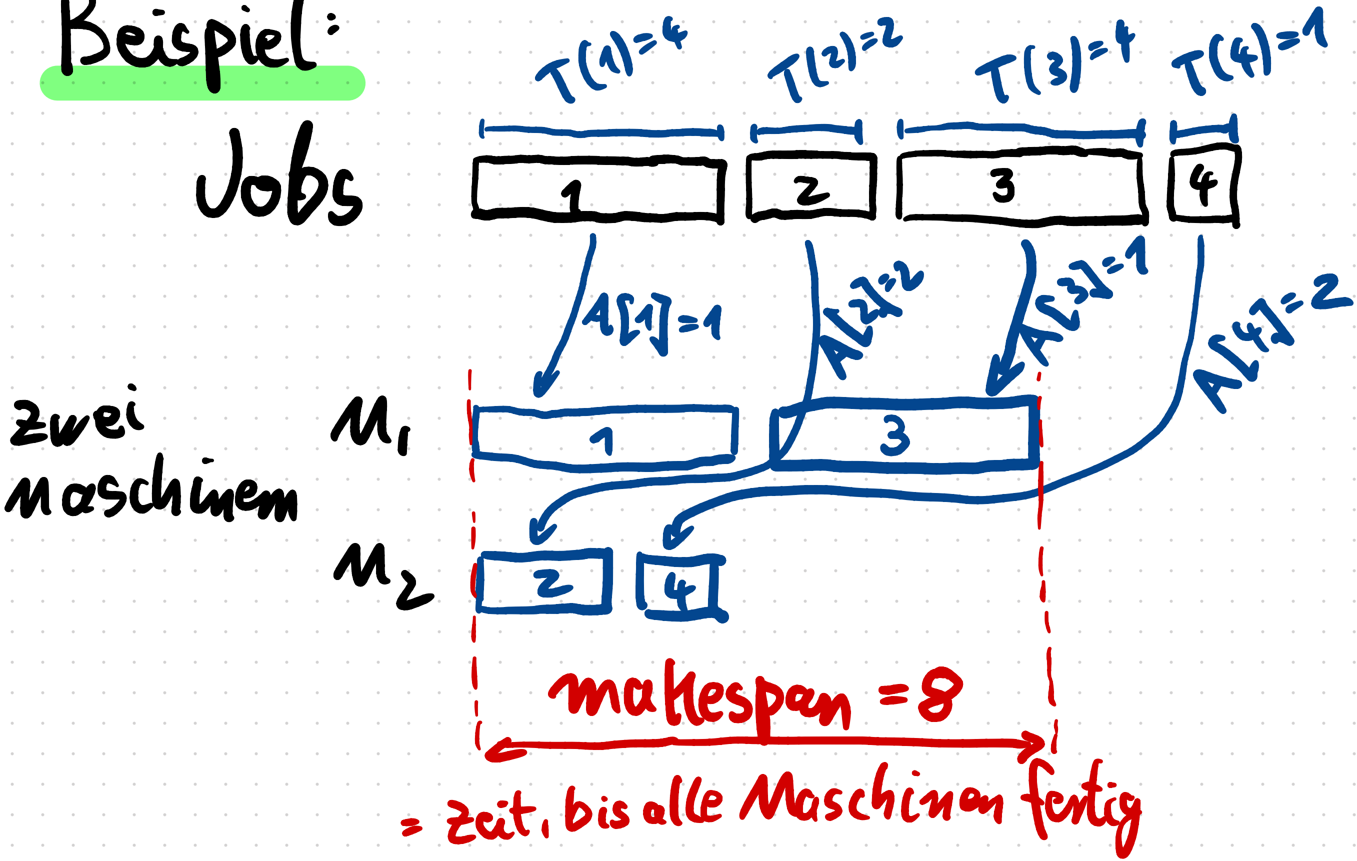
job j braucht Zeit $T[j]$

Ziel: Berechne ein Assignment $A[1..n]$,

($A[j] = i$ heißt, wir führen job j auf Maschine i aus)

sodass der makespan minimiert wird.

Beispiel:



$$makespan(A) = \max_i \sum_{A[j]=i} T[j]$$

Gierige Heuristik:

Gib den nächsten Job immer der Maschine, die derzeit am frühsten fertig ist!

$\underline{\textsc{GreedyLoadBalance}(T[1..n], m):}$

for $i \leftarrow 1$ to m

  $Total[i] \leftarrow 0$

for $j \leftarrow 1$ to n

  $mini \leftarrow \arg\min_i Total[i]$

  $A[j] \leftarrow mini$

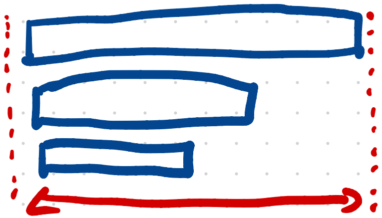
  $Total[mini] \leftarrow Total[mini] + T[j]$

return $A[1..m]$

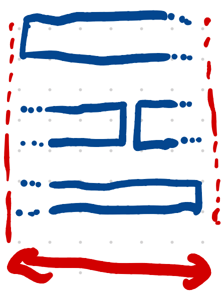
Satz J.1:

$$\widetilde{OPT} = \text{makespan}\left(\text{von Greedy Load Balance berechnetes Assignment } A\right) \le 2 \cdot \underbrace{\text{makespan}\left(\text{optimales Assignment } A^*\right)}_{OPT}$$

Zwei Beobachtungen:



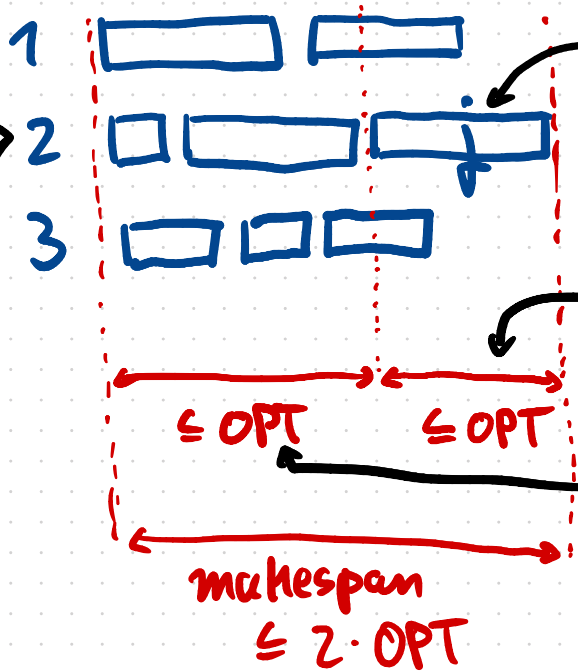
$$\mathrm{OPT} \geq \max_j T[j]$$



$$\mathrm{OPT} \geq \frac{1}{m} \sum_{j=1}^{n} T[j]$$

Beweis des Satzes

Sei i die Maschine mit maximalem $Total[i]$



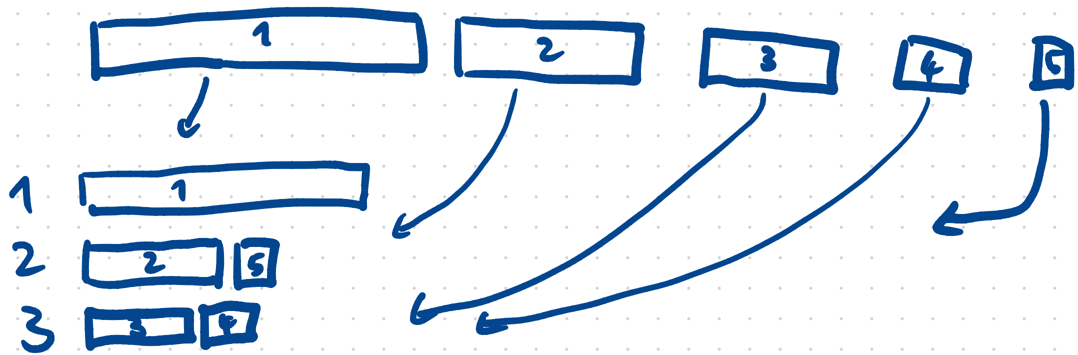
Sei j der als letztes auf i eingefügte Job

$T[j] \leq OPT$

$$Total[i] - T[j] \leq \frac{1}{m} \sum_i Total[i]$$
$$\leq OPT$$

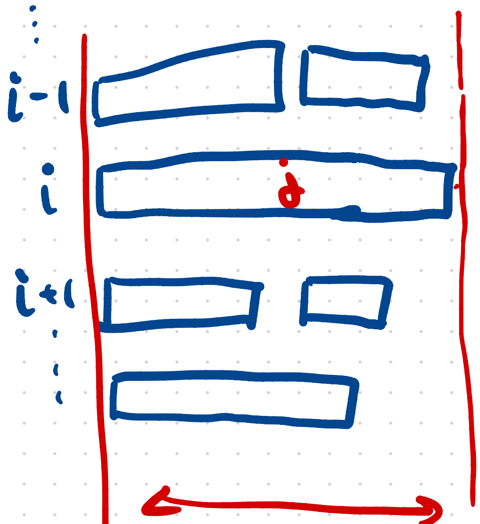
SORTEDGREEDYLOADBALANCE($T[1..n], m$):
    sort T in decreasing order
    return GREEDYLOADBALANCE(T, m)

Satz J.2. Algorithmus berechnet $\frac{3}{2}$-Approximation.



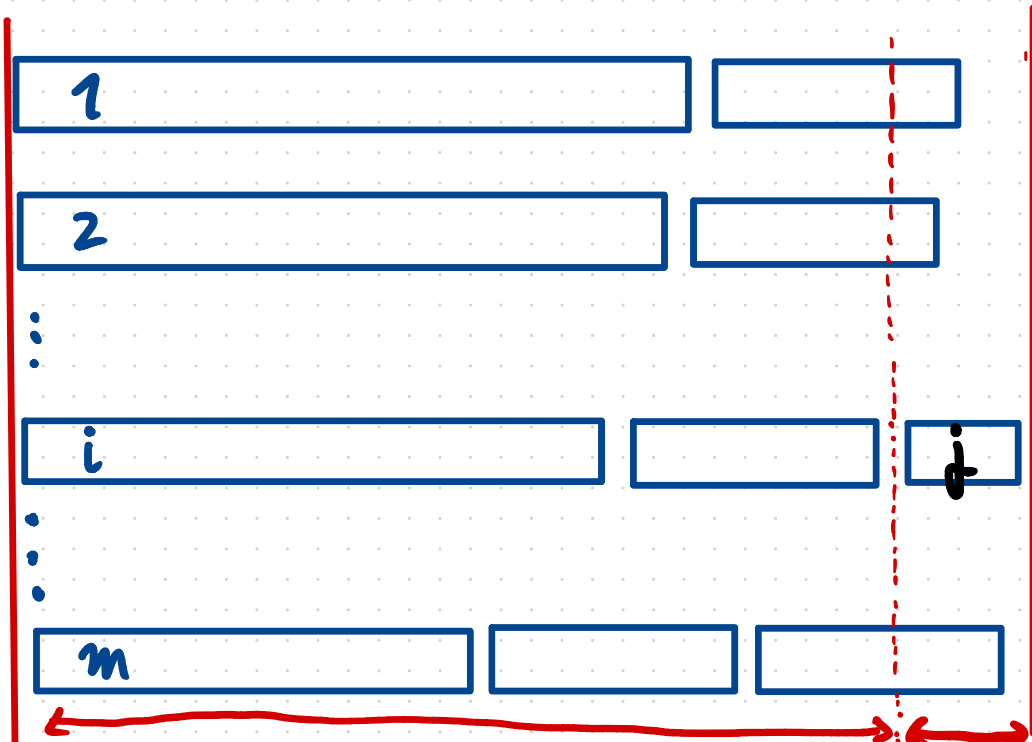
Sei i Maschine mit maximalem Total[i]

Fall: i enthält nur <u>einen</u> Job



$\Rightarrow$ makespan $= T[j] =$ OPT

Fall: Maschine i enthält mehrere Jobs



$j \geq m+1$

$\text{Total}[i] - T[j] \leq OPT$ (wie vorher)

$T[j] \leq \frac{OPT}{2}$ (neu)

$\Big\}$ makespan $\leq \frac{3}{2} OPT$

Neue Beobachtung

Unter den ersten $m+1$ Jobs müssen ≥ 2 auf dieselbe Maschine!

$\Rightarrow \quad OPT \geq T[k] + T[\ell]$

$j \geq m+1$ $\qquad$ $m+1 \geq k, \ell$

$$T[j] \leq T[m+1] \leq T[\max\{k,\ell\}] = \min\{T[k], T[\ell]\} \leq OPT/2$$

Bounded Search Tree Algorithmus für Vertex-Cover

KT 10.1

Vertex-Cover als Entscheidungsproblem

Eingabe: Graph G, Zahl $k \in \mathbb{N}$

Frage: Hat G ein Vertex-Cover $S \subseteq V(G)$ mit $|S| \leq k$?

Neues Ziel: Exakter Algorithmus

Exakte Algorithmen für Vertex-Cover

Erschöpfende Suche:

Für alle Teilmengen S mit $|S|=k$: $\binom{n}{k}$ Mengen

Prüfe, ob S ein VC ist. Zeit $\mathcal{O}(k\cdot n)$

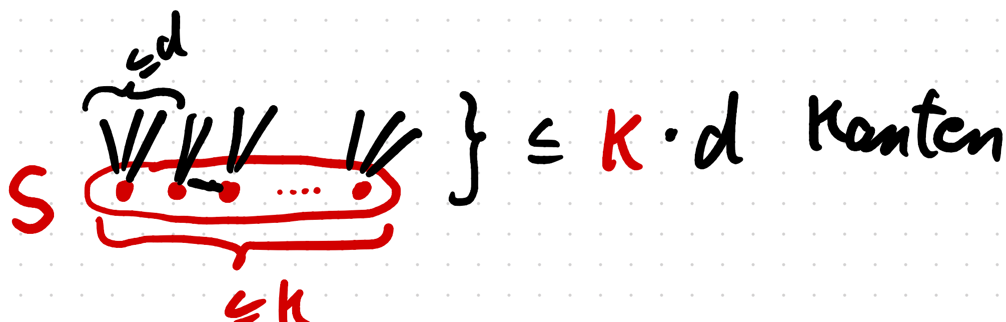
$\rightsquigarrow$ Zeit $\mathcal{O}\left(k\cdot n\cdot\binom{n}{k}\right) = \mathcal{O}(k\,n^{k+1})$

Ziel jetzt: Zeit $\mathcal{O}(2^k\cdot k\,n)$

(10.1) Wenn G:

- $\to$ n Knoten hat,
- $\to$ jeder Knoten Grad $\leq d$ hat, und
- $\to$ ein VC S mit $|S| \leq k$ hat,

dann hat G höchstens kd Kanten



Rekursive Suche

G mit Kante $\{u, v\}$

$$\begin{tikzcd}
 & G \arrow[dl, "u \in S"'] \arrow[dr, "v \in S"] & \\
G-u & & G-v
\end{tikzcd}$$

Beobachtung

G hat VC $\leq k$

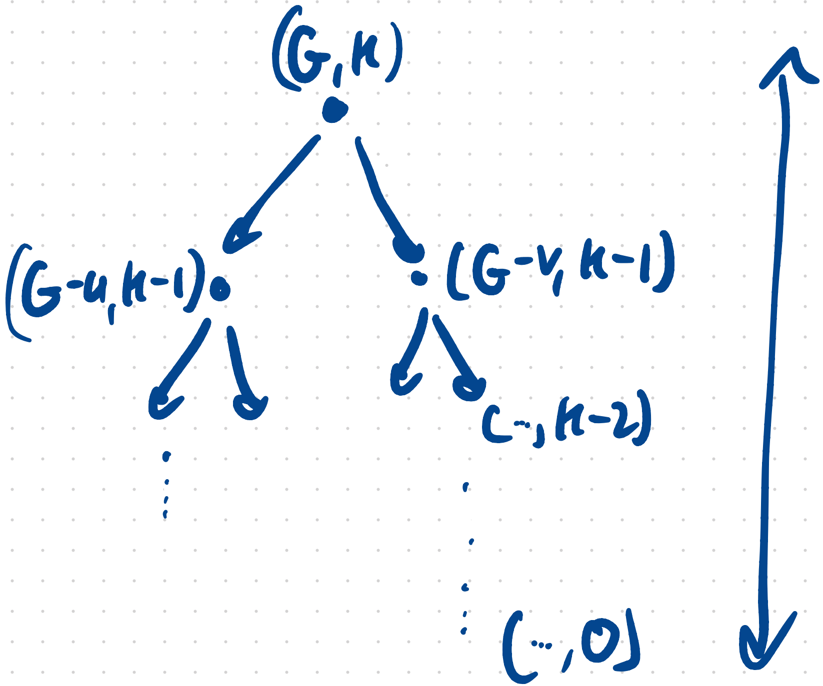
$\iff$ $G-u$ oder $G-v$ hat VC $\leq k-1$

Bounded Search Tree Algorithmus für VC

Algorithmus $A(G, k)$

1. If $|E| = 0$, return true
2. If $|E| > k \cdot |V|$, return false
3. Let $e = \{u, v\}$ be any edge of G.
4. Return $A(G-u, k-1)$ or $A(G-v, k-1)$

Analyse 1: Rekursionsbaum von A



Tiefe $\leq k$

$\Rightarrow \leq 2^{k+1}$ Knoten

Zeit $O(kn)$ je Knoten

$\rightsquigarrow$ Zeit $O(2^k \cdot kn)$

Analyse 2: Rekursionsgleichung

$T(n,k)$ = Laufzeit von A

Es gilt:

$T(n,1) \leq c \cdot n$ für eine Konstante c

$T(n,k) \leq 2 \cdot T(n,k-1) + c \cdot kn$

Durch vollständige Induktion über $k \geq 1$ folgt $T(n,k) \leq c \cdot 2^k kn$ (siehe Buch)

Übersicht

Vertex-Cover kann in Zeit $\mathcal{O}(2^k kn)$ gelöst werden.