NP-Vollständigkeit Spezialfälle

Algorithmen und Datenstrukturen
VU 186.866, 5.5h, 8 ECTS, 2023S
Letzte Änderung: 15. Mai 2023

Vorlesungsfolien

ALGORITHMS AND COMPLEXITY GROUP
TU WIEN Informatics

NP-Vollständigkeit meistern (Wiederholung)

Frage: Angenommen, wir müssen ein NP-vollständiges Problem lösen. Wie sollen wir vorgehen?
Antwort: Die Theorie besagt, dass es unwahrscheinlich ist, einen polynomiellen Algorithmus zu finden.

Man muss eine der gewünschten Eigenschaften opfern:

- Löse Problem optimal
  $\rightarrow$ Approximationsalgorithmen, Heuristische Algorithmen
- Löse Problem in Polynomialzeit
  $\rightarrow$ Algorithmen mit exponentieller Laufzeit
- Löse beliebige Instanzen des Problems
  $\rightarrow$ Identifiziere effizient lösbare Spezialfälle

Überblick

Finden von kleinen Vertex Covers

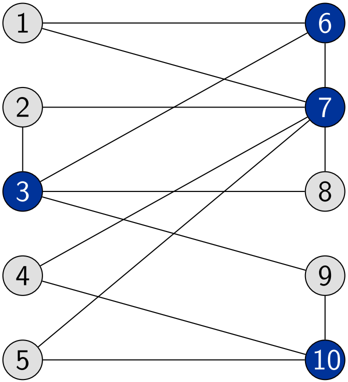
NP-vollständige Probleme auf Bäumen

Knotenfärben in Intervallgraphen

Finden von kleinen Vertex Covers

Vertex Cover (Knotenüberdeckung)

$\textsc{Vertex Cover}$: Gegeben sei ein Graph $G = (V, E)$ mit $|V| = n$ Knoten und eine ganze Zahl k. Existiert eine Teilmenge von Knoten $S \subseteq V$, sodass $|S| \leq k$, und für jede Kante $(u, v) \in E$ gilt, dass $u \in S$ oder $v \in S$ (oder beides)?



$k = 4$

$S = \{3, 6, 7, 10\}$

Finden kleiner Vertex Covers

Frage: Was ist möglich, wenn k klein ist?

Brute-Force. $O\left(kn^{k+1}\right)$.

- Probiere alle $\binom{n}{k}=O\left(n^k\right)$ Teilmengen der Größe k aus.
- Benötigt $O(kn)$ Zeit um zu überprüfen, ob eine Teilmenge ein Vertex Cover ist.

Ziel: Beschränke die exponentielle Abhängigkeit von k.

Beispiel: $n = 1000$, $k = 10$.
Brute-Force: $kn^{k+1} = 10^{34} \Rightarrow$ undurchführbar.
Besser: $2^k kn \approx 10^7 \Rightarrow$ durchführbar.

Anmerkung: Wenn k eine Konstante ist, dann ist der Algorithmus polynomiell. Wenn k eine kleine Konstante ist, dann ist er auch praktikabel.

Finden kleiner Vertex Covers

Behauptung: Sei (u, v) eine Kante von G. G hat ein Vertex Cover der Größe $\leq k$ genau dann, wenn zumindest einer der Graphen $G - \{u\}$ und $G - \{v\}$ ein Vertex Cover der Größe $\leq k - 1$ hat.

Lösche u und alle inzidenten Kanten.

Beweis: $\Rightarrow$

- Angenommen G hat ein Vertex Cover S der Größe $\leq k$.
- S enthält entweder u oder v (oder beide). Angenommen, S enthält u.
- Dann ist $S - \{u\}$ ein Vertex Cover von $G - \{u\}$.

Beweis: $\Leftarrow$

- Angenommen S ist ein Vertex Cover von $G - \{u\}$ der Größe $\leq k - 1$.
- Dann ist $S \cup \{u\}$ ein Vertex Cover von G. $\square$

Finden kleiner Vertex Covers

Behauptung: Wenn G ein Vertex Cover der Größe k hat, dann hat $G \leq k(n-1)$ Kanten.

Beweis: Jeder Knoten überdeckt höchstens $n-1$ Kanten. $\square$

Finden kleiner Vertex Covers: Algorithmus

Behauptung: Der folgende Algorithmus bestimmt mit einer Laufzeit in $O\left(2^k kn\right)$, ob G ein Vertex Cover der Größe $\leq k$ hat.

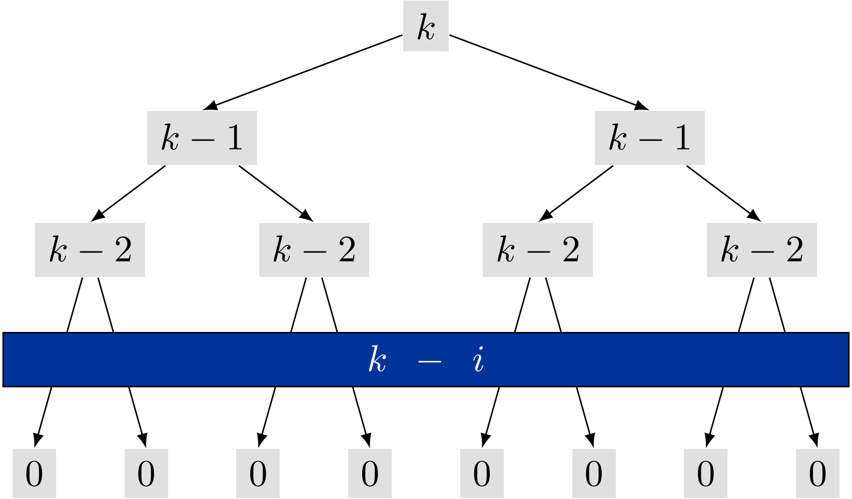
```
Vertex-Cover(G,k):
if G enthält keine Kanten
    return true
if G enthält > k(n − 1) Kanten
    return false
Sei (u, v) eine beliebige Kante von G
a ← Vertex-Cover(G − {u}, k − 1)
b ← Vertex-Cover(G − {v}, k − 1)
return a OR b
```

Beweis:

- Korrektheit folgt aus den zwei vorherigen Behauptungen.
- Es existieren $\leq 2^{k+1}$ Knoten im Rekursionsbaum. Jeder Aufruf benötigt $O(kn)$ Zeit. □

Finden kleiner Vertex Covers: Rekursionsbaum

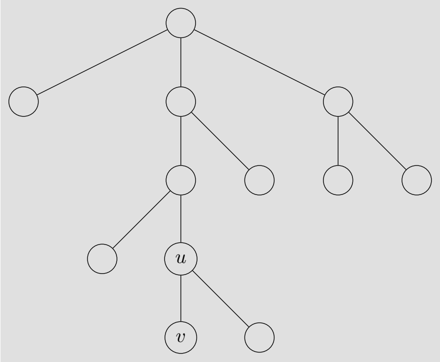
$$T(n,k) \leq \begin{cases} c & \text{falls } k = 0 \\ cn & \text{falls } k = 1 \\ 2T(n-1,k-1) + ckn & \text{falls } k > 1 \end{cases} \quad \Rightarrow \quad T(n,k) \leq 2^k ckn$$



Lösen NP-vollständiger Probleme auf Bäumen

Independent Set auf Bäumen

Independent Set auf Bäumen: Gegeben sei ein Baum. Finde die größte Teilmenge von Knoten, sodass keine zwei Knoten durch eine Kante verbunden werden.



Tatsache: Ein Baum mit zumindest zwei Knoten hat zumindest zwei Blätter.

- *Grad = 1*

Beobachtung: Wenn v ein Blatt ist, dann existiert ein maximales Independent Set, das v enthält.

Beweis: (Austauschargument)

- Wir gehen von einem maximalen Independent Set S aus.
- Wenn $v \in S$, dann ist man fertig.
- Wenn $v \notin S$ und $u \notin S$, dann ist $S \cup \{v\}$ unabhängig $\Rightarrow$ S ist nicht maximal.
- Wenn $v \notin S$ und $u \in S$, dann ist $S \cup \{v\} - \{u\}$ unabhängig.

Independent Set auf Bäumen: Greedy-Algorithmus

Theorem: Der nachfolgende Greedy-Algorithmus findet ein maximales Independent Set in einem Wald $T = (V, E)$
(jede Zusammenhangskomponente des Graphen ist ein Baum).

```
Independent-Set-In-A-Forest(T):
S ← ∅
while T hat zumindest eine Kante
    Sei e = (u, v) eine Kante, sodass v ein Blatt ist
    Füge v zu S hinzu
    Lösche aus V die Knoten u und v (und alle
    zu diesen Knoten inzidenten Kanten)
S ← S ∪ V
return S
```

Beweis: Korrektheit folgt aus der vorherigen Beobachtung. □

Anmerkung: Kann man in $O(n)$ Zeit implementieren, indem man die Knoten in Postorder durchmustert.

Gewichtetes Independent Set auf Bäumen

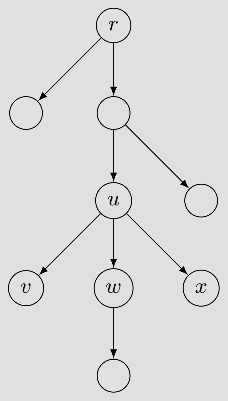
Gewichtetes Independent Set auf Bäumen: Gegeben sei ein Baum und Knotengewichte $w_v > 0$. Finde ein Independent Set S, das $\sum_{v \in S} w_v$ maximiert.

Beobachtung: Wenn (u, v) eine Kante ist, sodass v ein Blatt ist, dann beinhaltet die Lösung entweder u oder sie enthält alle Blattknoten inzident zu u.

Gewichtetes Independent Set auf Bäumen

Lösung mit dynamischer Programmierung: Wähle einen Knoten, z.B. r, als Wurzel aus.

- $OPT_{\text{in}}(u)$ = maximales Gewicht eines Independent Sets des Unterbaums mit Wurzel u, das u enthält.
- $OPT_{\text{out}}(u)$ = maximales Gewicht eines Independent Sets des Unterbaums mit Wurzel u, das u nicht enthält.



Nachfolger$(u) = \{v, w, x\}$

Formulierung:

$$
\begin{aligned}
OPT_{\text{in}}(u) &= w_u + \textstyle\sum_{v \in \text{Nachfolger}(u)} OPT_{\text{out}}(v) \\
OPT_{\text{out}}(u) &= \textstyle\sum_{v \in \text{Nachfolger}(u)} \max\{OPT_{\text{in}}(v), OPT_{\text{out}}(v)\}
\end{aligned}
$$

Gewichtetes Independent Set auf Bäumen

Formulierung:

$$\begin{aligned} OPT_{\mathsf{in}}(u) &= w_u + \sum_{v \in \mathsf{Nachfolger}(u)} OPT_{\mathsf{out}}(v) \\ OPT_{\mathsf{out}}(u) &= \sum_{v \in \mathsf{Nachfolger}(u)} \max\{OPT_{\mathsf{in}}(v), OPT_{\mathsf{out}}(v)\} \end{aligned}$$

Erklärung:

- $OPT_{\mathsf{in}}(u)$ addiert das Gewicht von u (u ist enthalten) und die maximalen Gewichte aller Unterbäume, bei denen die Wurzeln (Nachfolger von u) nicht enthalten sind. Wäre eine Wurzel enthalten, dann wäre es kein Independent Set mehr, da dann diese Wurzel mit u eine Kante gemeinsam hätte.
- $OPT_{\mathsf{out}}(u)$ addiert die maximalen Gewichte aller Unterbäume. Bei einem Unterbaum kann die Wurzel v enthalten sein oder nicht. Zwei Wurzeln zweier Unterbäume von u können niemals miteinander verbunden sein.

Gewichtetes Independent Set auf Bäumen: Implementierung

```
Weighted-Independent-Set-In-A-Tree(T):
Wähle eine Wurzel r aus
foreach Knoten u von T in Postorder
    if u ist ein Blatt
        M_in[u] ← w_u
        M_out[u] ← 0
    else
        M_in[u] ← w_u + Σ_{v∈Nachfolger(u)} M_out[v]
        M_out[u] ← Σ_{v∈Nachfolger(u)} max{M_in[v], M_out[v]}
return max{M_in[r], M_out[r]}
```

- *Stellt sicher, dass ein Knoten nach seinen Nachfolgern besucht wird.*

Gewichtetes Independent Set auf Bäumen: Dynamische Programmierung

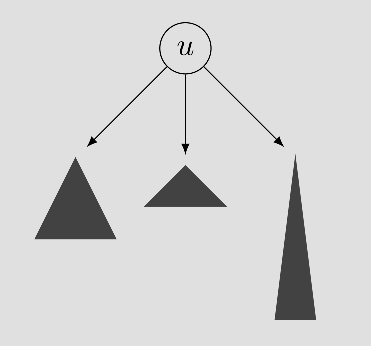
Theorem: Der Algorithmus auf der vorherigen Folie findet ein maximal gewichtetes Independent Set in einem Baum mit einer Laufzeit in $O(n)$.

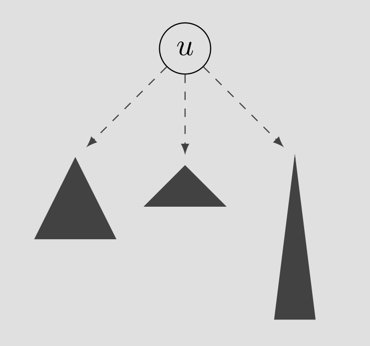
Kann auch das Independent Set (und nicht nur den Wert) finden.

Beweis: Erfordert $O(n)$ Zeit, da wir Knoten in Postorder durchmustern und jede Kante genau einmal überprüfen. $\square$

Kontext

Independent Set auf Bäumen: Dieser strukturierte Spezialfall ist handhabbar, weil wir einen Knoten finden können, der die Verbindung zwischen den Subproblemen in verschiedenen Subbäumen unterbrechen kann.





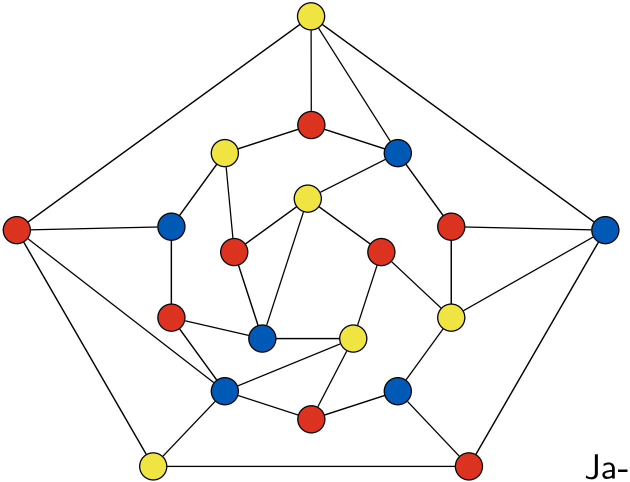
Graphen mit beschränkter Baumweite (elegante Generalisierung von Bäumen):

- Erfassen eine reichhaltige Klasse von Graphen, die in der Praxis auftritt.
- Erlauben die Aufteilung in unabhängige Teile.

Knotenfärben in Intervallgraphen

Knotenfärbeproblem

Erinnerung 3-COLOR: Gegeben sei ein ungerichteter Graph G. Kann man die Knoten des Graphen mit den Farben Rot, Gelb und Blau so einfärben, dass benachbarte Knoten nicht die gleiche Farbe besitzen?



Ja-Instanz

Knotenfärbeproblem

Erinnerung 3-$\textsc{Color}$: Gegeben sei ein ungerichteter Graph G. Kann man die Knoten des Graphen mit den Farben Rot, Gelb und Blau so einfärben, dass benachbarte Knoten nicht die gleiche Farbe besitzen?

k-$\textsc{Color}$: Gegeben sei ein ungerichteter Graph G. Kann man die Knoten des Graphen mit $k \geq 3$ Farben so einfärben, dass benachbarte Knoten nicht die gleiche Farbe besitzen?

> k-$\textsc{Color}$ *ist NP-vollständig für* $k \geq 3$

Als Optimierungsproblem:

$\textsc{Opt-Color}$: Gegeben sei ein ungerichteter Graph G. Färbe die Knoten des Graphen mit einer minimalen Anzahl Farben so, dass benachbarte Knoten nicht die gleiche Farbe besitzen.

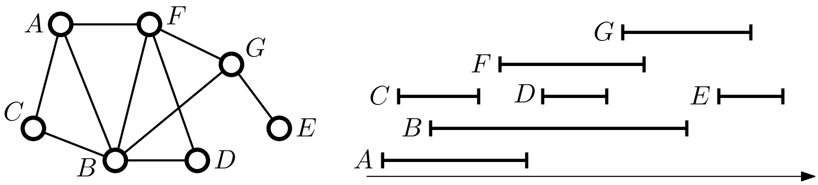
Ziel: Betrachte effizient lösbaren Spezialfall Intervallgraphen

Intervallgraphen

Definition: Intervallgraphen sind Graphen, die sich wie folgt als Schnittgraph von Intervallen in $\mathbb{R}$ repräsentieren lassen:

- ungerichteter Graph $G = (V, E)$
- Intervallmenge $\mathcal{I} = \{I_v \subset \mathbb{R} \mid v \in V\}$
- Kante $(u, v) \in E \Leftrightarrow I_u \cap I_v \neq \emptyset$

Beispiel:

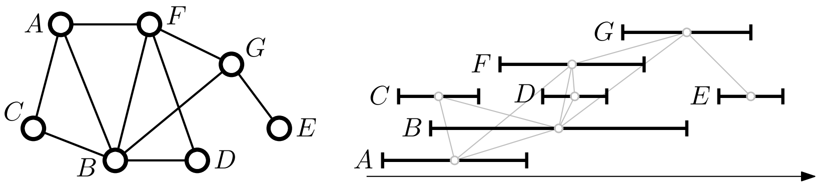


Intervallgraphen

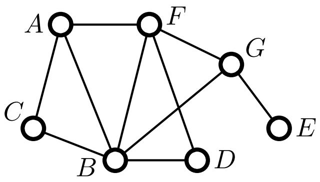
Definition: Intervallgraphen sind Graphen, die sich wie folgt als Schnittgraph von Intervallen in $\mathbb{R}$ repräsentieren lassen:

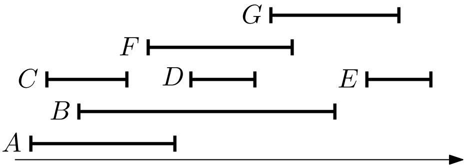
- ungerichteter Graph $G = (V, E)$
- Intervallmenge $\mathcal{I} = \{I_v \subset \mathbb{R} \mid v \in V\}$
- Kante $(u, v) \in E \Leftrightarrow I_u \cap I_v \neq \emptyset$

Beispiel:



Färben von Intervallgraphen

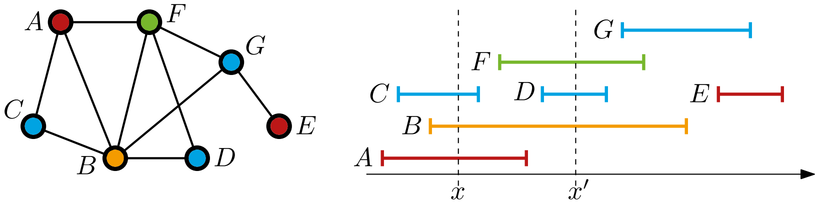




Färben mit einer Farbe:

- äquivalent zu Maximum Independent Set
- äquivalent zu Interval Scheduling (Kap. Greedy-Algorithmen)
- Greedy-Algorithmus EDF (earliest deadline first)?

Färben von Intervallgraphen: Untere Schranke



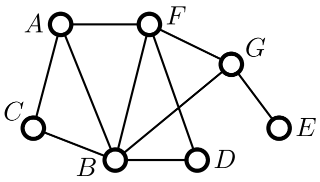
Beobachtungen:

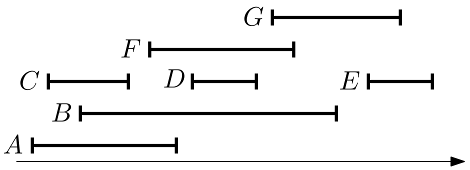
- Liegt ein Punkt $x \in \mathbb{R}$ in k Intervallen, braucht man mindestens k Farben
- Für Tiefe $d := \max_{x \in \mathbb{R}} \{|\{I \in \mathcal{I} \mid x \in I\}|\}$ gilt Färbung von G benötigt $\geq d$ Farben

Frage: Geht es auch immer mit d Farben?

Färben von Intervallgraphen: Algorithmus

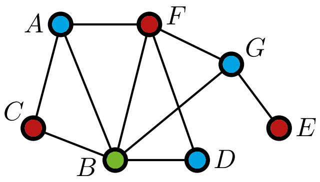
```
Interval-Coloring(𝓘):
Sortiere Intervallgrenzen aufsteigend in Liste 𝓛
col_max ← 0
Initialisiere leere Queue Q
foreach Punkt a von 𝓛
    if a ist ein Startpunkt
        if Q ist leer
            färbe I = [a, b] mit Farbe col_max
            col_max ← col_max + 1
        else
            entferne c = Q.head und färbe I = [a, b] mit Farbe c
    else //a ist ein Endpunkt
        füge Farbe von I = [b, a] in Q ein
return Färbung von 𝓘 und Anzahl col_max
```

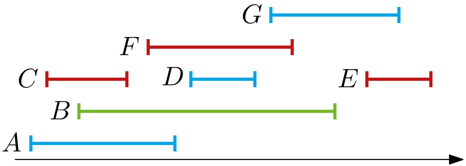




Färben von Intervallgraphen: Algorithmus

```
Interval-Coloring(𝓘):
Sortiere Intervallgrenzen aufsteigend in Liste 𝓛
col_max ← 0
Initialisiere leere Queue Q
foreach Punkt a von 𝓛
    if a ist ein Startpunkt
        if Q ist leer
            färbe I = [a, b] mit Farbe col_max
            col_max ← col_max + 1
        else
            entferne c = Q.head und färbe I = [a, b] mit Farbe c
    else //a ist ein Endpunkt
        füge Farbe von I = [b, a] in Q ein
return Färbung von 𝓘 und Anzahl col_max
```





Färben von Intervallgraphen: Analyse

Theorem: Der Algorithmus `Interval-Coloring` berechnet eine minimale Färbung einer Intervallmenge $\mathcal{I}$ und des zugehörigen Intervallgraphen G mit n Knoten in $O(n \log n)$ Zeit.

Beweis:

- Q enthält zu jedem Zeitpunkt nur "freie" Farben
- Wenn keine Farbe frei ist, wird eine neue Farbe gewählt
- Am Ende jedes Intervalls wird dessen Farbe wieder freigegeben
- Färbung gültig, da überlappende Intervalle verschiedene Farben erhalten, also mindestens d
- Es werden genau d Farben $0, \ldots, d-1$ verwendet:
  - sonst wären zum Zeitpunkt der ersten Wahl der Farbe d die Queue Q leer und damit die Tiefe $d+1 \rightarrow$ Widerspruch
- Färbung selbst erfolgt in Zeit $O(n)$, wird allerdings dominiert durch Sortierung der Intervallgrenzen in $O(n \log n)$ Zeit

Ergänzende Literatur

J. Kleinberg and E. Tardos. *Algorithm Design*. Pearson, 2005. Kapitel 10.