

# Zusammenfassung - Kryptographie

Marc Meier

8. November 2015

Korrektheit und Vollständigkeit der Informationen sind nicht gewährleistet. Macht euch eigene Notizen oder ergänzt/korrigiert meine Ausführungen!

## Inhaltsverzeichnis

<b>1 Grundlagen und Wiederholungen</b>	<b>1</b>
<b>2 Approximationsalgorithmen</b>	<b>10</b>

Planung der Veranstaltung

1. Wiederholung (ca 3-4 Termine)

- Probleme, Kodierung von Problemen, Laufzeit von Algorithmen
- P, PN, NPC...
- Grundlegende Probleme & Algorithmen

2. Algorithmen für schwierige Probleme (ca. 15 Termine)

- parallele / randomisierte Algorithmen
- Approximationsalgorithmen
- parametrisierte Algorithmen

3. Kryptographie (ca 10 Termine)

- Public-Key-Kryptographie

## 1 Grundlagen und Wiederholungen

### 1.1 Probleme, deren Kodierung und Laufzeit von Algorithmen

Anhand einiger Beispiele soll die Wichtigkeit der Kodierung der Probleme bzw. der Eingabe verdeutlicht werden.

#### 1.1.1 Sortieren

**Eingabemenge:** Natürliche Zahlen  $a_1, a_2, \dots, a_n$

**Ausgabe:** Eingabe aufsteigend sortiert

Beim Sortieren handelt es sich um ein *Suchproblem*. Beispiele für bekannte Sortieralgorithmen und deren Laufzeitkomplexität sind Bubblesort  $\mathcal{O}(n^2)$  und Mergesort  $\mathcal{O}(n \log n)$ <sup>1</sup>.

---

<sup>1</sup>Sofern nicht anders angegeben entspricht  $\log n$  immer dem dualen Logarithmus von  $n$  (Basis 2)

### 1.1.2 Eingabelänge N

Die Länge der Eingabe entspricht *nicht* der Anzahl der zu sortierenden Elemente  $n$ . Dies ist aufgrund der Kodierung in eine maschinenlesbare Form der Fall. Für gewöhnlich verwenden heutige Computer eine Darstellung im Binärsystem. Daher gilt:

$$N := \sum_{i=1}^n \log a_i \quad (1)$$

Im Folgenden soll die Eingabe [11, 13, 113] kodiert werden. Für eine Turingmaschine mit dem Eingabealphabet  $\Sigma := \{0, 1, \$\}$  würde diese folgendermaßen aussehen: 1011\$1101\$1110001. Um eine Lesbarkeit für Binärrechner zu ermöglichen, werden weiterhin folgende Ersetzungen durchgeführt:  $1 \rightarrow 11$   $0 \rightarrow 00$   $\$ \rightarrow 01$ . Die resultierende Kodierung wäre nun: 1100111101111100110111111100000011. Sei nun  $L$  die maximale Wortlänge, d.h. die größte Binärdarstellung der Eingabebezahlen  $a_i$ .

$$L := \max_{i=1}^n \log a_i \quad (2)$$

Daraus folgt für Bubblesort:

$$\begin{aligned} \mathcal{O}(n^2) &= \mathcal{O}(L \cdot n^2) \quad | \text{ Algo. } n\text{-mal durchlaufen mit max. Eingabelänge } L \text{ als obere Abschätzung} \\ &= \mathcal{O}(N \cdot N^2) \quad | L \leq N (\text{offensichtlich wegen } N := \sum_{i=1}^n \log a_i) \\ &= \mathcal{O}(N^3) \quad | n \leq N^2 \end{aligned}$$

### 1.1.3 Primzahl

**Eingabemenge:** Eine natürliche Zahl  $n$

**Ausgabe:** Ist  $n$  eine Primzahl?

Es handelt sich um ein *Entscheidungsproblem*. Die Eingabelänge ist  $N := \log n$ . Ein naiver Algorithmus wird im Folgenden beschrieben.

```
1: if  $n = 2$  then
2:   return Ja
3: else
4:   for  $d := 2$  to  $n - 1$  do
5:     if  $n \bmod d = 0$  then
6:       return Nein
7:     else
8:       end if
9:   end for
10:  return Ja
11: end if
```

Die Komplexität lässt sich wie folgt herleiten: Für die Verzweigungen (Zeilen 1 und 5), sowie für die return-Statements in den Zeilen 2 und 6 ist eine beschränkte Komplexität  $\mathcal{O}(1)$  anzunehmen. Alle Statements in der for-Schleife werden  $n - 2$  mal wiederholt, die Komplexität ist  $\mathcal{O}(n)$ .

Die daraus resultierende Komplexität  $\mathcal{O}(n)$  ist äquivalent zu  $\mathcal{O}(2^{\log n})$ . Weil  $\log n$  der Eingabelänge  $N$  entspricht, ist die Komplexität entsprechend  $\mathcal{O}(2^N)$ , also exponentiell. Demnach ist der Algorithmus nicht effizient.

### 1.1.4 Cliquensuche im Graphen

**CLIQUE** (gehört zu den Entscheidungsproblemen)

**Eingabemenge:** Graph  $G = (V, E)$  und  $k \in \mathbb{N}$

**Ausgabe:** Hat  $G$  mindestens paarweise verbundene Knoten (eine Clique  $C$  mit  $\geq k$  Knoten)?

**CLIQUE (Suchproblem)** (gehört zu den Optimierungsproblemen)

**Eingabemenge:** Graph  $G = (V, E)$

**Ausgabe:** Bestimme eine Clique  $C$  in  $G$  mit maximaler Knotenzahl.

**Behauptung:** Wenn CLIQUE (Entscheidungsproblem) in polynomieller Zeit lösbar ist, so ist auch die Suchversion von CLIQUE in polynomieller Zeit lösbar.<sup>3</sup>

**Beweis**

$\Leftarrow$  Zur Eingabe  $(G, k)$  von CLIQUE bestimmen wir zunächst mit dem Algorithmus für CLIQUE-Suchversion eine größte Clique  $C$  von  $G$ . Es wird verglichen, ob  $|C| \geq k$  gilt. Ist dies der Fall, gibt der Algorithmus *Ja* zurück, andernfalls *Nein*. Die Laufzeit entspricht offensichtlich der Clique-Suchversion.

$\Rightarrow$  Betrachte folgende Zwischenversion für CLIQUE: Betrachte folgende Zwischenversion für CLIQUE<sup>4</sup>:

**CLIQUE-Zwischenversion:**

**Eingabe:** Graph  $G=(V,E)$

**Aufgabe:** Bestimme die Maximalzahl  $\omega(G)$  von paarweise verbundenen Knoten in  $G$

**Behauptung:** CLIQUE in polynomieller Zeit lösbar  $\Rightarrow^{(1)}$  CLIQUE-Zwischenversion in polynomieller Zeit  $\Rightarrow^{(2)}$  CLIQUE-Suchversion in polynomieller Zeit lösbar.

**zu (1):** Sei  $A$  ein Algorithmus für CLIQUE in polynomieller Zeit. Zur Eingabe  $G$  von CLIQUE-Zwischenversion wende  $A$  auf Eingaben  $(G, n), (G, n-1), \dots, (G, 1)$  an! Ausgabe ist  $\omega(G) =$  erstes  $k$  mit  $A(G, k) = Ja$

```
1: for  $k := n$  to 1 do
2:   if  $A(G, k) = 'ja'$  then  $\text{return } \omega(G) = k$ 
3:   end if
4: end for
```

Laufzeit:  $n \cdot \mathcal{O}(n^c) = \mathcal{O}(n^{c+1})$

**zu (2):** Sei  $A$  ein polyzeit Algorithmus für Clique-Zwischenversion. d.h.  $A(G) = \omega(G)$  in  $\mathcal{O}(n^d)$  für eine Konstante  $d$ .

Zur Eingabe  $G$  von Clique-Suchversion wende folgendes an:

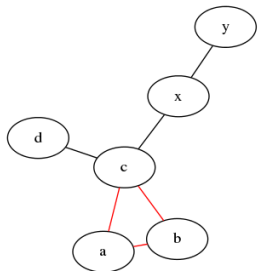
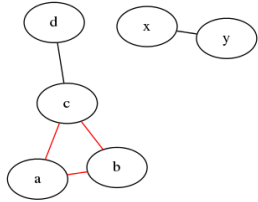
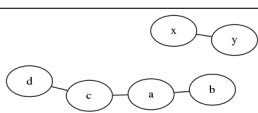
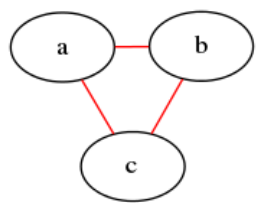
```
1: for jede Kante  $e \in E$  do
2:   if  $\omega(G - e) = \omega(G)$  then  $G := G - e$ 
3:   end if
4: end for
5: return  $C := V(G)$  ohne isolierte Knoten
```

**Hinweis:** Ich bin mir nicht sicher, ob es sich um das korrekte Beispiel aus der Vorlesung handelt.

**Laufzeit:**  $m \cdot \mathcal{O}(n^d) = \mathcal{O}(n^{d+2})$

<sup>3</sup>In polynomieller Zeit, abhängig von der Knotenzahl  $|V|$  des Graphen

<sup>4</sup> $\omega(G) = \max\{|C| : C \text{ ist eine Clique in } G\}$  ist die Cliquenzahl

$G$	$\omega(G) = 3$		Vollständiger Graph
$G_2 := G - cx$	$\omega(G) = 3$		Nach Entfernen der Kante $cx$ hat sich die Cliquenzahl nicht geändert. Die Kante $cx$ ist also nicht für die Clique notwendig.
$G_3 := G_2 - bc$	$\omega(G) = 2$		Nach Entfernen der Kante $bc$ hat sich die Cliquenzahl verringert. Offensichtlich wird $bc$ also für die Clique benötigt.
$C := \{a, b, c\}$	$\omega(G) = 3$		Im weiteren Verlauf wird (offensichtlich) noch die Kante $cd$ entfernt. Anschließend werden die isolierten Knoten d, x und y entfernt und die Lösung $C := \{a, b, c\}$ zurückgegeben.

### 1.1.5 Gegenüberstellung „einfache Formulierung“ vs „formale Formulierung“

Die genaue Formulierung des Problems ist wichtig:

„Ist  $n$  prim?“

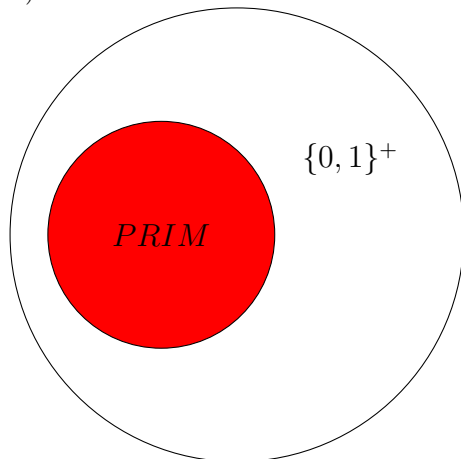
genauer:

**Eingabemenge:** Eine natürliche Zahl  $n$

**Ausgabe:** Ist  $n$  eine Primzahl?

**Formal:**  $PRIM^5 = \{bin(n) | n \in \mathbb{N} \text{ ist prim}\} \subset \{0, 1\}^+$

Venn-Diagramm  $PRIM \subset \{0, 1\}^+$  TODO (Und offensichtlich falsch, da ich noch keinen Plan von TIKZ habe)



## 1.2 $\mathbb{P}$ , NP, NP-vollständig

$\mathbb{P}$  = Menge aller *Entscheidungsprobleme*, für die ein (deterministischer) polynomialzeit-Algorithmus existiert (formal via Deterministischer Turing Maschine, „effizient lösbare Probleme“). Ein *Optimierungsproblem* ist lösbar, wenn die entsprechende Entscheidungsversion in  $\mathbb{P}$  ist.

### 1.2.1 Grundlegende Probleme in $\mathbb{P}$

- **Sortieren**
- **PRIM** - Allerdings nicht der vorgestellte Algorithmus. Effizienter Algorithmus wurde 2002 vorgestellt. Komplexität  $\mathcal{O}(\log n)^{12}$ , zur Zeit sogar  $\mathcal{O}(\log n)^6$  <https://de.wikipedia.org/wiki/AKS-Primzahltest>
- **2-SAT** (Modellierung mit Graphenproblem für Beweis)

#### Übrige Notizen

In der Praxis ermittelt man die Komplexität der Einfachheit halber mit uniformen Kostenmaß. Sortierung: Eingabe:  $a_1, a_2, \dots, a_n$  beziehungsweise  $bin(a_1) \# bin(a_2) \# \dots \# bin(a_n)$  mit Ersetzung Aufgabe: Sortiere Zahlen aufsteigend Mergesort:  $\mathcal{O}(n \cdot \log n)$  Problem -<sub>i</sub> Eingabelänge (siehe oben, hatten wir schon)

## 1.3 NP-Vollständigkeit Reduktionsbeispiele

### 3-SAT $\leq_p$ INDSET

„ $\Rightarrow$ “:

Eingabe: Sei  $F = K_1 \wedge K_2 \wedge \dots \wedge K_m$  und  $K_j = (l_{j,1} \vee l_{j,2} \vee l_{j,3})$ , wobei  $l_{j,1}, l_{j,2}, l_{j,3} \in \{x_1, \dots, x_n\} \cup \{\overline{x_1}, \dots, \overline{x_n}\}$ .

Erklärung der Notation am Beispiel:

$$F = \underbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}_{K_1} \wedge \underbrace{(\overline{x_1} \vee x_2 \vee x_3)}_{K_2} \wedge \underbrace{(x_1 \vee x_2 \vee x_4)}_{K_3}$$

$K_1 : l_{1,1} = x_1; l_{1,2} = \overline{x_2}, \text{ usw.}$

Der Beweis erfolgt in zwei Schritten:

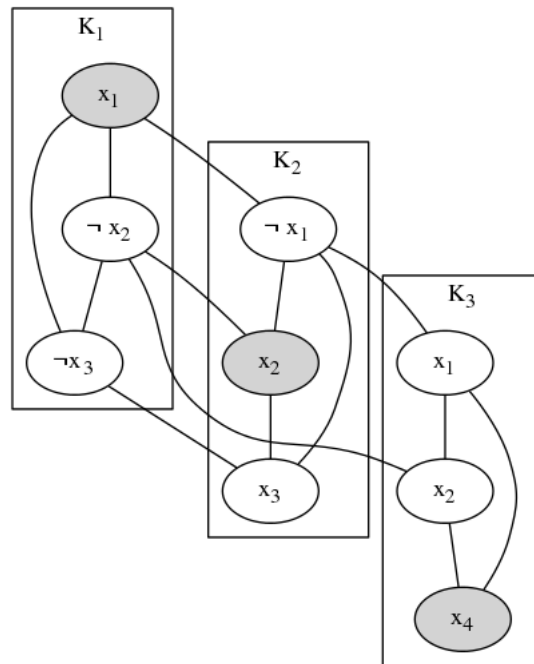
---

<sup>5</sup>bin(n) - Binärdarstellung von n

1. Konstruiere einen Graphen  $G$  und eine natürliche Zahl  $k$  aus der Formel  $F$ .
2. Zeige dass der Graph eine unabhängige Knotenmenge  $U$  mit  $|U| = k$  hat.

Konstruiere den Graphen  $G$  wie folgt:

- Literale entsprechen Knoten
- Verbinde die drei Literale innerhalb einer Klausel  $K_j$  miteinander
- Verbinde die Knoten unterschiedlicher Klauseln, wenn die entsprechenden Literale zueinander komplementär sind (Bsp:  $x_1$  mit  $\bar{x}_1$ )



Beispiel Formel  $F$ : Verbinden Knoten der Knoten

- Setze  $k$  wie folgt:  $k = m$  (= die Anzahl der Klauseln in  $F$ )

Annahme: Sei  $b$  eine erfüllende Belegung für  $F$ , d.h. in jeder Klausel  $K_j$  existiert ein Literal  $l_{j,k}$  mit  $b(l_{j,k}) = 1$ . Wähle in Graph  $G$  aus jeder Klauselmengeng genau einen Knoten mit entsprechendem wahren Literal. Aus der Konstruktion von  $G$  folgt, dass diese  $k$  Knoten unverbunden sind und eine unabhängige Menge bilden. Am Beispiel:  $b(x_1) = b(x_2) = 1; b(x_3) = b(x_4) = 0$ . Daraus folgen die ausgewählten Knoten  $x_1, x_2$  und  $\bar{x}_3$ . Diese ergeben eine unabhängige Menge.

" $\Leftarrow$ ":

Annahme: Sei  $U$  eine unabhängige Menge von  $G$  mit  $|U| \geq k = m$  Knoten. Dann gilt  $|U| = k$ , weil jede der  $m$  Klauselmengen ein Dreieck bildet, d.h.  $|UK_j| = 1$ , für jede Klauselmengeng  $K_j$ .  $U$  definiert nun eine Belegung für  $F$  wie folgt:

$$b(l_{j,k}) = 1 \quad , \text{falls } l_{j,k} \in U \\ = 0 \quad , \text{sonst}$$

$$b(l_{j,k}) = 1 \Rightarrow b(\bar{l}_{j,k}) \notin U \Rightarrow b(\bar{l}_{j,k}) = 0 \\ \Rightarrow F \text{ ist erfüllbar unter } b!$$

Beispiel Belegung:  $b(x_1) = b(x_2) = b(x_3) = 0; b(x_4) = 1$

### 3-SAT $\leq_p$ 3-Färbung

Eingabe: Sei  $F = K_1 \wedge K_2 \wedge \dots \wedge K_m$  und  $K_j = (l_{j,1} \vee l_{j,2} \vee l_{j,3})$ , wobei  $l_{j,1}, l_{j,2}, l_{j,3} \in \{x_1, \dots, x_n\} \cup \{\bar{x}_1, \dots, \bar{x}_3\}$ .

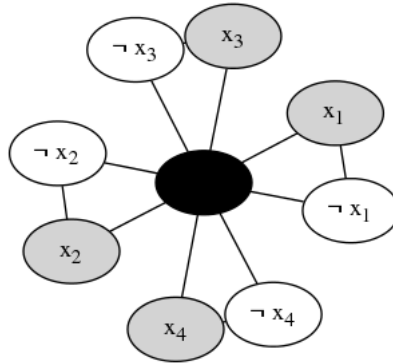
Beispiel:

$$F = \underbrace{(x_1 \vee \bar{x}_2 \vee \bar{x}_3)}_{K_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee x_3)}_{K_2} \wedge \underbrace{(x_1 \vee x_2 \vee x_4)}_{K_3}$$

" $\Rightarrow$ ": Graph  $G$  ist 3-färbbar  $\Rightarrow F$  ist erfüllbar.

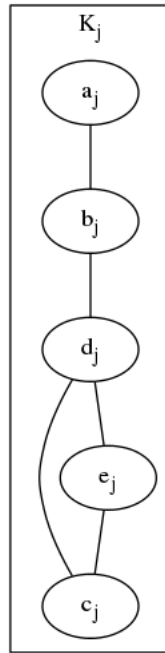
Konstruiere den Graphen  $G$  wie folgt:

- Literale entsprechen Knoten und verbinde komplementäre Literale miteinander. Idee: Füge einen Hilfsknoten  $R$  ein (im Bild schwarz) und färbe diesen, sodass alle Literale  $x_i$  und  $\bar{x}_i$  mit genau zwei anderen Farben gefärbt werden müssen.



Einfügen des Hilfsknoten  $R$

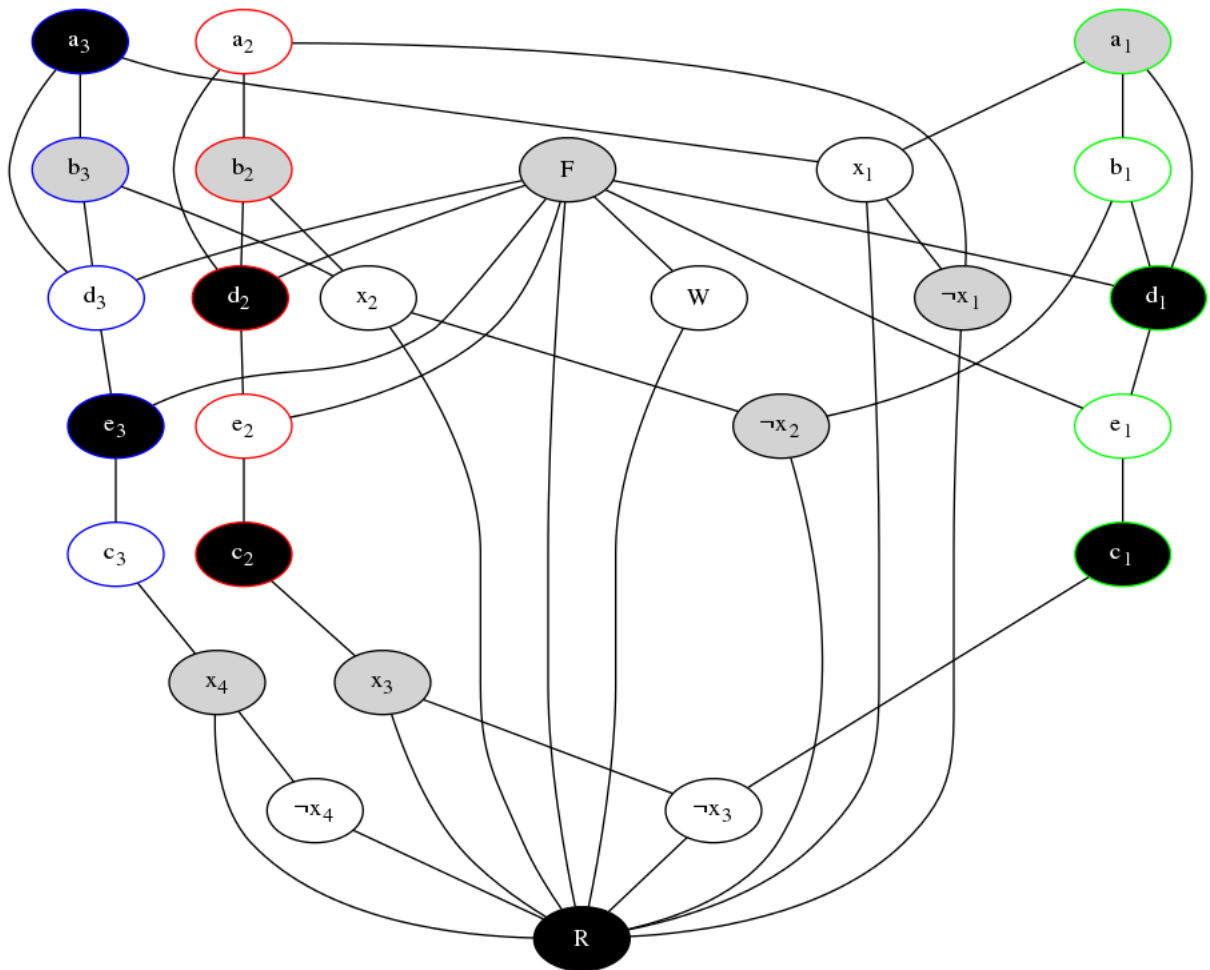
- Für jede Klausel  $K_j$  konstruiere den folgenden Teilgraphen:



Teilgraph für Klausel  $K_j$

- Für  $K_j = \{l_{j,1}, l_{j,2}, l_{j,3}\}$  füge die folgenden Kanten ein:  $al_{j,1}, bl_{j,2}, cl_{j,3}$ .
- Füge nun zwei zusätzliche Knoten  $W$  und  $F$  ein, sowie die Kanten  $RW, RF, FW$ . Zusätzlich die Kanten  $Fd_j, Fe_j$  für alle  $j$ .





Beispielgraph, konstruiert aus Formel  $F$

" $\Leftarrow$ ": Sei  $F$  eine erfüllbare Formel (unter 3-SAT)  $\Rightarrow$  Graph  $G$  ist 3-färbbar.

Sei  $b$  eine erfüllende Belegung für  $F$ . Färbe nun alle Knoten  $x_i$  mit  $b(x_i)$  und alle Knoten  $\bar{x}_i$  mit  $b(\bar{x}_i)$ . Nun erweitern wir den Graphen auf 3 Farben indem wir zunächst die Knoten aller Klauseln  $K_j$  färben. Abschließend werden  $F, W$  und  $R$  gefärbt. Dies ergibt eine 3-Färbung des Graphen.

## 2 Approximationsalgorithmen

**VertexCover (Entscheidungsversion):**

**Eingabe:** Graph  $G = (V, E), k \in \mathbb{N}$

**Frage:** Hat  $G$  ein VC  $C$  mit  $|C| \leq k$ ?

Behauptung: VC ist NP-vollständig.

Beweis über poly. Reduktion: 3-SAT  $\leq_p$  INDSET  $\leq_p$  VC:

Die Eingabe für INDSET sei  $(G, k)$  und für VC  $(G', k')$ , wobei  $G' = G$  und  $k' = n - k$ ; ( $n = |V(G)|$ )

- $G$  hat eine unabhängige Menge  $U \subseteq V(G)$  mit  $|U| \geq k$
- $G'$  hat ein VC  $C$  mit  $|C| \leq k'$

Füge Skizze ein

" $\Rightarrow$ ": geg. sei ein  $U$ , zeige das Graph ein VC  $C$  hat.

$C := V(G) \setminus U$  ist ein VC mit  $|C| = n - k = k'$

" $\Leftarrow$ ": geg. sei ein VC  $C$ , zeige das der Graph eine unabhängige Menge  $U$  hat.

$U := V(G') \setminus C$  ist eine unabhängige Menge in  $G$  mit  $|U| = n - k' = k$

### Approximationsalgorithmus für MinVC

Gegeben sei der folgende Beispielgraph:

Beispielgraph Approximationsalgorithmen

Beispielhaftes Durchlaufen des Algorithmus:

1. Wähle Kante  $ab$  aus, sodass  $E' = \{cv, uv, uw\}$
2. Wähle Kante  $uv$  aus, sodass  $E' = \emptyset$

Daraus folgt das Ergebnis  $C = \{a, b, u, w\}$

### MaxCUT

Gegeben sei der folgende Beispielgraph:

Beispielgraph Approximationsalgorithmen

wobei  $E(S, \bar{S}) = \{ac, bu, bc, vc, vu, wu\}$ .

Durchlaufen des Algorithmus liefert uns beispielsweise:

1. Schritt:  $S = \{w\}, \bar{S} = \emptyset$
2. Schritt:  $S = \{w\}, \bar{S} = \{u\}$
3. Schritt:  $S = \{w, v\}, \bar{S} = \{u\}$
4. Schritt:  $S = \{w, v\}, \bar{S} = \{c, u\}$
5. Schritt:  $S = \{w, v, b\}, \bar{S} = \{c, u\}$

6. Schritt:  $S = \{w, v, b, a\}$ ,  $\bar{S} = \{c, u\}$

## Vorlesung 3.11.15

### Job-Scheduling (Entscheidungsversion):

**Eingabe:**  $m$  Maschinen,  $n$  Aufgaben mit einer Bearbeitungszeit  $d_1, \dots, d_n$  und  $T$

**Frage:** Existiert eine Verteilung der  $n$  Jobs auf die  $m$  identischen Maschinen, sodass die Gesamtausführungszeit  $\sum_{i=1}^n d_i \leq T$ ?

### PARTITION (Entscheidungsversion):

**Eingabe:**  $n$  Elemente mit den Werten  $a_1, \dots, a_n$

**Frage:** Existiert eine Aufteilung der  $n$  Elemente in zwei disjunkte Mengen, sodass die Summe der Werte identisch ist?

$$\exists A \dot{\cup} B = \{1, \dots, n\} \text{ mit } \sum_{i \in A} a_i = \sum_{i \in B} a_i$$

Beweis das Job-Scheduling  $\in \text{NPC}$  (bereits für  $m = 2$ ):  $\text{PARTITION} \leq_p \text{Job-Scheduling}$   
"⇒":

- Sei  $A \dot{\cup} B = \{1, \dots, n\}$  mit  $\sum_{i \in A} a_i = \sum_{i \in B} a_i \Rightarrow \frac{1}{2} \sum_{i=1}^n a_i = T$
- $M_1 : J_k, k \in A$
- $M_2 : J_k, k \in B$

Daraus folgt die Ausführungsdauer  $= \max\{\sum_{k \in A} d_k, \sum_{k \in B} d_k\} = \max\{\sum_{k \in A} a_k, \sum_{k \in B} a_k\} = T$   
✓

"⇐":

- Sei  $M_1 : J_k, k \in X$  und  $M_2 : J_k, k \in Y$ ,  
wobei  $X \dot{\cup} Y = \{1, \dots, n\}$  mit  $\max\{\sum_{k \in X} d_k, \sum_{k \in Y} d_k\} \leq T = \frac{1}{2} \sum_{i=1}^n a_i$

Definiere:  $A := X$  und  $B := Y$ . Dann gilt

$$\begin{aligned} \sum_{i \in A} a_i &= \sum_{i \in X} d_i \leq \sum_{i=1}^n a_i \\ \sum_{i \in B} a_i &= \sum_{i \in Y} d_i \leq \sum_{i=1}^n a_i \end{aligned}$$

Addition der zwei Ungleichungen ergibt:

$$\sum_{i=1}^n a_i = \sum_{i \in A} a_i + \sum_{i \in B} a_i \leq 2 \sum_{i=1}^n a_i$$

Überall muss Gleichheit gelten, sodass

$$\sum_{i \in A} a_i = \frac{1}{2} \sum_{i=1}^n a_i = \sum_{i \in B} a_i$$

✓

Eingabelänge für Job-Scheduling:

$$\log m + \log n + \underbrace{\sum_{k=1}^n \log d_k}_{\geq n} \Rightarrow n \cdot \mathcal{O}(m) = \mathcal{O}(n \cdot m)$$

## Vorlesung 5.11.15

Beispiel FPAS Knapsack DynKS Algorithmus:

$i$								
4	2	3	3	5	7	9	12	14
3	2	3	3	5	$\infty$	$\infty$	$\infty$	$\infty$
2	3	3	5	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
1	2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$n$	1	2	3	4	5	6	7	8