

# Einführung in die Theoretische Informatik

Johannes Köbler



Institut für Informatik  
Humboldt-Universität zu Berlin

WS 2013/14

Die Laufzeit einer NTM  $M$  bei Eingabe  $x$  ist die maximale Anzahl an Rechenschritten, die  $M(x)$  ausführt.

## Definition

- Die **Laufzeit** einer NTM  $M$  bei Eingabe  $x$  ist definiert als

$$time_M(x) = \sup\{t \geq 0 \mid \exists K : K_x \vdash^t K\},$$

wobei  $\sup \mathbb{N} = \infty$  ist.

- Sei  $t : \mathbb{N} \rightarrow \mathbb{N}$  eine monoton wachsende Funktion.
- Dann ist  $M$   **$t(n)$ -zeitbeschränkt**, falls für alle Eingaben  $x$  gilt:

$$time_M(x) \leq t(|x|).$$

Die Zeitschranke  $t(n)$  beschränkt also die Laufzeit bei allen Eingaben der Länge  $n$  (**worst-case** Komplexität).

Wir fassen alle Sprachen und Funktionen, die in einer vorgegebenen Zeitschranke  $t(n)$  entscheidbar bzw. berechenbar sind, in folgenden **Komplexitätsklassen** zusammen.

## Definition

- Die in deterministischer Zeit  $t(n)$  entscheidbaren Sprachen bilden die Sprachklasse

$$\text{DTIME}(t(n)) = \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte DTM}\}.$$

- Die in nichtdeterministischer Zeit  $t(n)$  entscheidbaren Sprachen bilden die Sprachklasse

$$\text{NTIME}(t(n)) = \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte NTM}\}.$$

- Die in deterministischer Zeit  $t(n)$  berechenbaren Funktionen bilden die Funktionenklasse

$$\text{FTIME}(t(n)) = \left\{ f \mid \begin{array}{l} \text{es gibt eine } t(n)\text{-zeitbeschränkte} \\ \text{DTM } M, \text{ die } f \text{ berechnet} \end{array} \right\}.$$

# Die wichtigsten Zeitkomplexitätsklassen

- Die wichtigsten deterministischen Zeitkomplexitätsklassen sind

$$\text{LINTIME} = \bigcup_{c \geq 1} \text{DTIME}(cn + c) \quad \text{„Linearzeit“}$$

$$\text{P} = \bigcup_{c \geq 1} \text{DTIME}(n^c + c) \quad \text{„Polynomialzeit“}$$

$$\text{E} = \bigcup_{c \geq 1} \text{DTIME}(2^{cn+c}) \quad \text{„Lineare Exponentialzeit“}$$

$$\text{EXP} = \bigcup_{c \geq 1} \text{DTIME}(2^{n^c+c}) \quad \text{„Exponentialzeit“}$$

- Die nichtdeterministischen Klassen **NLINTIME**, **NP**, **NE**, **NEXP** und die Funktionenklassen **FP**, **FE**, **FEXP** sind analog definiert.
- Für eine Klasse  $\mathcal{F}$  von Funktionen sei  $\text{DTIME}(\mathcal{F}) = \bigcup_{t \in \mathcal{F}} \text{DTIME}(t(n))$  (die Klassen **NTIME**( $\mathcal{F}$ ) und **FTIME**( $\mathcal{F}$ ) sind analog definiert).

# Asymptotische Laufzeit und Landau-Notation

## Definition

Seien  $f$  und  $g$  Funktionen von  $\mathbb{N}$  nach  $\mathbb{R}^+ \cup \{0\} = [0, \infty)$ .

- Wir schreiben  $f(n) = O(g(n))$ , falls es Zahlen  $n_0$  und  $c$  gibt mit

$$\forall n \geq n_0 : f(n) \leq c \cdot g(n).$$

**Bedeutung:** „ $f$  wächst nicht wesentlich schneller als  $g$ .“

- Formal bezeichnet der Term  $O(g(n))$  die Klasse aller Funktionen  $f$ , die obige Bedingung erfüllen, d.h.

$$O(g(n)) = \{f: \mathbb{N} \rightarrow [0, \infty) \mid \exists n_0, c \in \mathbb{N} \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}.$$

- Die Gleichung  $f(n) = O(g(n))$  drückt also in Wahrheit eine **Element-Beziehung**  $f \in O(g(n))$  aus.
- $O$ -Terme können auch auf der linken Seite vorkommen. In diesem Fall wird eine **Inklusionsbeziehung** ausgedrückt.
- So steht  $n^2 + O(n) = O(n^2)$  für die Aussage

$$\{n^2 + f \mid f \in O(n)\} \subseteq O(n^2).$$

## Beispiel

- $7 \log(n) + n^3 = O(n^3)$  ist **richtig**.
- $7 \log(n)n^3 = O(n^3)$  ist **falsch**.
- $2^{n+O(1)} = O(2^n)$  ist **richtig**.
- $2^{O(n)} = O(2^n)$  ist **falsch** (siehe Übungen).

Mit der  $O$ -Notation lassen sich die wichtigsten deterministischen Zeitkomplexitätsklassen wie folgt charakterisieren:

**L**INTIME = DTIME( $O(n)$ ) „Linearzeit“

**P** = DTIME( $n^{O(1)}$ ) „Polynomialzeit“

**E** = DTIME( $2^{O(n)}$ ) „Lineare Exponentialzeit“

**EXP** = DTIME( $2^{n^{O(1)}}$ ) „Exponentialzeit“

- Wie wir gesehen haben, sind NTMs nicht mächtiger als DTMs, d.h. jede NTM kann von einer DTM simuliert werden.
- Die Frage, wieviel Zeit eine DTM zur Simulation einer NTM benötigt, ist eines der wichtigsten offenen Probleme der Informatik.
- Wegen  $\text{NTIME}(t) \subseteq \text{DTIME}(2^{O(t)})$  erhöht sich die Laufzeit im schlimmsten Fall exponentiell.
- Insbesondere die Klasse NP enthält viele für die Praxis überaus wichtige Probleme, für die kein Polynomialzeitalgorithmus bekannt ist.
- Da jedoch nur Probleme in P als effizient lösbar angesehen werden, hat das so genannte **P-NP-Problem**, also die Frage, ob alle NP-Probleme effizient lösbar sind, eine immense praktische Bedeutung.

# Die Polynomialzeitreduktion

## Definition

- Eine Sprache  $A \subseteq \Sigma^*$  ist auf  $B \subseteq \Gamma^*$  **in Polynomialzeit reduzierbar** ( $A \leq^P B$ ), falls eine Funktion  $f : \Sigma^* \rightarrow \Gamma^*$  in FP existiert mit

$$\forall x \in \Sigma^* : x \in A \Leftrightarrow f(x) \in B.$$

- Eine Sprache  $A$  heißt  **$\leq^P$ -hart** für eine Sprachklasse  $\mathcal{C}$  (kurz:  **$\mathcal{C}$ -hart** oder  **$\mathcal{C}$ -schwer**), falls gilt:

$$\forall L \in \mathcal{C} : L \leq^P A.$$

- Eine  $\mathcal{C}$ -harte Sprache  $A$ , die zu  $\mathcal{C}$  gehört, heißt  **$\mathcal{C}$ -vollständig** (bzgl.  $\leq^P$ ).
- **NPC** bezeichnet die Klasse aller NP-vollständigen Sprachen.

## Lemma

- Aus  $A \leq^P B$  folgt  $A \leq B$ .
- Die Reduktionsrelation  $\leq^P$  ist reflexiv und transitiv (s. Übungen).



## Satz

Die Klassen P und NP sind unter  $\leq^P$  abgeschlossen.

## Beweis

- Sei  $B \in P$  und gelte  $A \leq^P B$  mittels einer Funktion  $f \in FP$ .
- Seien  $M$  und  $T$  DTMs mit  $L(M) = B$  und  $T(x) = f(x)$ .
- Weiter seien  $p$  und  $q$  polynomielle Zeitschranken für  $M$  und  $T$ .
- Betrachte die DTM  $M'$ , die bei Eingabe  $x$  zuerst  $T$  simuliert, um  $f(x)$  zu berechnen, und danach  $M$  bei Eingabe  $f(x)$  simuliert.
- Dann gilt

$$x \in A \Leftrightarrow f(x) \in B \Leftrightarrow f(x) \in L(M) \Leftrightarrow x \in L(M').$$

- Also ist  $L(M') = A$  und wegen

$$\text{time}_{M'}(x) \leq \text{time}_T(x) + \text{time}_M(f(x)) \leq q(|x|) + p(q(|x|))$$

ist  $M'$  polynomiell zeitbeschränkt und somit  $A$  in P. □

# NP-Vollständigkeit

## Satz

- 1  $A \leq^P B$  und  $A$  ist NP-hart  $\Rightarrow B$  ist NP-hart.
- 2  $A \leq^P B$ ,  $A$  ist NP-hart und  $B \in \text{NP}$   $\Rightarrow B \in \text{NPC}$ .
- 3  $\text{NPC} \cap \text{P} \neq \emptyset \Rightarrow \text{P} = \text{NP}$ .

## Beweis

- 1 Da  $A$  NP-hart ist, ist jede NP-Sprache  $L$  auf  $A$  reduzierbar. Da zudem  $A \leq^P B$  gilt und  $\leq^P$  transitiv ist, folgt  $L \leq^P B$ .
- 2 Klar, da  $B$  mit (1) NP-hart und nach Voraussetzung in NP ist.
- 3 Sei  $B$  eine NP-vollständige Sprache in P. Dann ist jede NP-Sprache  $A$  auf  $B$  reduzierbar und da P unter  $\leq^P$  abgeschlossen ist, folgt  $A \in \text{P}$ .  $\square$

# Laufzeit einer universellen NTM

## Lemma

Es gibt eine NTM  $U$  für die Sprache

$$\left\{ w\#x \mid \begin{array}{l} w \in \{0,1\}^*, x \in \{0,1,\#\}^* \text{ und} \\ \text{die NTM } M_w \text{ akzeptiert } x \end{array} \right\}$$

mit der Laufzeit

$$time_U(w\#x) = \mathcal{O}(|w| \cdot time_{M_w}(x)^2).$$

## Beweis

- Die 3-NTM  $U(w\#x)$  verschiebt zunächst den Binärstring  $w$  auf das 2. Band.
- Anschließend schreibt  $U$  die Binärkodierung des Startzustands von  $M_w$  auf das 3. Band.
- Dann ersetzt  $U$  jedes Zeichen  $a_i$  des auf dem Eingabeband verbliebenen Strings  $x$  durch  $bin(i)$  und trennt diese durch  $\#$ .
- Hierbei benutzt  $U$  Binärzahlen einer festen Länge, mit denen sich alle Arbeitszeichen von  $M_w$  darstellen lassen.

- Die 3-NTM  $U(w\#x)$  verschiebt zunächst  $w$  auf das 2. Band.
- Anschließend schreibt  $U$  den Startzustand von  $M_w$  auf das 3. Band.
- Dann kodiert  $U$  die Zeichen von  $x$  binär (getrennt durch  $\#$ ).
- Schließlich ergänzt  $U$  das 1. Band um  $k - 1$  Bandinschriften (getrennt durch  $\#\#$ ), die jeweils nur die Binärokodierung von  $\sqcup$  enthalten.
- Die aktuellen Kopfpositionen speichert  $U$  durch Markierung des vorangehenden  $\#$ -Zeichens.
- Nun simuliert  $U$  jeden einzelnen Rechenschritt von  $M_w$  in  $\mathcal{O}(|w| \cdot \text{time}_{M_w}(x))$  Schritten.



# NP-Vollständigkeit

## Satz

Folgende Sprache ist NP-vollständig:

$$L = \left\{ w\#x\#0^m \mid \begin{array}{l} w, x \in \{0, 1\}^* \text{ und } M_w \text{ ist eine NTM,} \\ \text{die } x \text{ in } \leq m \text{ Schritten akzeptiert} \end{array} \right\}$$

## Beweis

- Zunächst ist klar, dass  $L \in \text{NP}$  mittels folgender NTM  $M$  ist:  
 $M$  simuliert bei Eingabe  $w\#x\#0^m$  die NTM  $M_w$  bei Eingabe  $x$  für höchstens  $m$  Schritte. Falls  $M_w$  in dieser Zeit akzeptiert, akzeptiert  $M$  ebenfalls, andernfalls verwirft  $M$ .
- Sei nun  $A$  eine beliebige NP-Sprache. Dann ist  $A$  in Polynomialzeit auf eine Sprache  $B \subseteq \{0, 1\}^*$  in NP reduzierbar (siehe Übungen).
- Sei  $M_w$  eine durch ein Polynom  $p$  zeitbeschränkte NTM für  $B$ .
- Dann reduziert folgende FP-Funktion  $f$  die Sprache  $B$  auf  $L$ :

$$f : x \mapsto w\#x\#0^{p(|x|)}.$$

## Platzkomplexität von Turingmaschinen

- Als nächstes definieren wir den Platzverbrauch von NTMs.
- Intuitiv ist dies die Anzahl aller während einer Rechnung benutzten Bandfelder.
- Wollen wir auch sublinearen Platz sinnvoll definieren, so dürfen wir hierbei das Eingabeband offensichtlich nicht berücksichtigen.
- Um sicherzustellen, dass eine NTM  $M$  das Eingabeband nicht als Speicher benutzt, verlangen wir, dass  $M$ 
  - die Felder auf dem Eingabeband nicht verändert und
  - sich nicht mehr als ein Feld von der Eingabe entfernt.

### Definition

Eine NTM  $M$  heißt **offline-NTM** (oder NTM mit **Eingabeband**), falls für jede von  $M$  bei Eingabe  $x$  erreichbare Konfiguration

$$K = (q, u_1, a_1, v_1, \dots, u_k, a_k, v_k)$$

gilt, dass  $u_1 a_1 v_1$  ein Teilwort von  $\sqcup x \sqcup$  ist.

## Definition

- Der **Platzverbrauch** einer offline-NTM  $M$  bei Eingabe  $x$  ist definiert als

$$space_M(x) = \sup \left\{ s \geq 1 \left| \begin{array}{l} \exists K = (q, u_1, a_1, v_1, \dots, u_k, a_k, v_k) \\ \text{mit } K_x \vdash^* K \text{ und } s = \sum_{i=2}^k |u_i a_i v_i| \end{array} \right. \right\}.$$

- Sei  $s : \mathbb{N} \rightarrow \mathbb{N}$  eine monoton wachsende Funktion.
- $M$  heißt  **$s(n)$ -platzbeschränkt**, falls für alle Eingaben  $x$  gilt:

$$space_M(x) \leq s(|x|) \text{ und } time_M(x) < \infty.$$

Wir fassen alle Sprachen, die in einer vorgegebenen Platzschranke  $s(n)$  entscheidbar sind, in folgenden **Platzkomplexitätsklassen** zusammen.

### Definition

- Die auf deterministischem Platz  $s(n)$  entscheidbaren Sprachen bilden die Klasse

$$\mathbf{DSPACE}(s(n)) = \{L(M) \mid M \text{ ist eine } s(n)\text{-platzb. offline-DTM}\}.$$

- Die auf nichtdeterministischem Platz  $s(n)$  entscheidbaren Sprachen bilden die Klasse

$$\mathbf{NSPACE}(s(n)) = \{L(M) \mid M \text{ ist eine } s(n)\text{-platzb. offline-NTM}\}.$$



- Die wichtigsten deterministischen Platzkomplexitätsklassen sind

$L = \text{DSPACE}(O(\log n))$  „Logarithmischer Platz“

$LINSPACE = \text{DSPACE}(O(n))$  „Linearer Platz“

$PSPACE = \text{DSPACE}(n^{O(1)})$  „Polynomieller Platz“

- Die nichtdeterministischen Klassen  $NL$ ,  $NLINSPACE$  und  $NPSPACE$  sind analog definiert.
- Der Satz von Savitch besagt, dass

$$NPSPACE(s(n)) \subseteq \text{DSPACE}(s^2(n))$$

ist (dies wird in der VL Komplexitätstheorie gezeigt).

Daher fallen die Klassen  $NPSPACE$  und  $PSPACE$  zusammen.

## Frage

Welche elementaren Beziehungen gelten zwischen den verschiedenen Zeit- und Platzklassen?

## Satz

- Für jede Funktion  $s(n) \geq \log n$  gilt

$$\text{DSPACE}(s) \subseteq \text{NSPACE}(s) \subseteq \text{DTIME}(2^{O(s)}).$$

- Für jede Funktion  $t(n) \geq n + 2$  gilt

$$\text{DTIME}(t) \subseteq \text{NTIME}(t) \subseteq \text{DSPACE}(O(t)).$$

## Korollar

Es gilt  $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP \subseteq EXPSPACE$ .

$\text{REG} = \text{DSPACE}(O(1)) = \text{NSPACE}(O(1)) \not\subseteq L,$

$\text{DCFL} \not\subseteq \text{LINTIME},$

$\text{CFL} \not\subseteq \text{NLINTIME} \cap \text{DTIME}(O(n^3)) \not\subseteq P,$

$\text{DCSL} = \text{LINSPACE} \subseteq \text{CSL},$

$\text{CSL} = \text{NLINSPACE} \subseteq \text{PSPACE} \cap E,$

$\text{REC} = \bigcup_f \text{DSPACE}(f(n))$

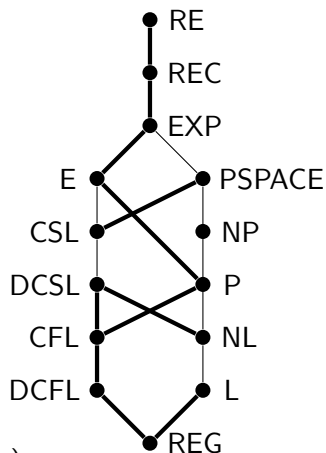
$= \bigcup_f \text{NSPACE}(f(n))$

$= \bigcup_f \text{DTIME}(f(n))$

$= \bigcup_f \text{NTIME}(f(n)),$

wobei  $f$  alle (oder äquivalent: alle berechenbaren)

Funktionen  $f : \mathbb{N} \rightarrow \mathbb{N}$  durchläuft.



# Aussagenlogische Formeln

- Die Menge der **booleschen** (oder **aussagenlogischen**) **Formeln** über den Variablen  $x_1, \dots, x_n$ ,  $n \geq 0$ , ist induktiv wie folgt definiert:
  - Die Konstanten 0 und 1 sind boolesche Formeln.
  - Jede Variable  $x_i$  ist eine boolesche Formel.
  - Mit  $G$  und  $H$  sind auch die **Negation**  $\neg G$ , die **Konjunktion**  $(G \wedge H)$  und die **Disjunktion**  $(G \vee H)$  von  $G$  und  $H$  Formeln.
- Eine **Belegung** von  $x_1, \dots, x_n$  ist ein Wort  $a = a_1 \dots a_n \in \{0, 1\}^n$ .
- Der **Wert**  $F(a)$  von  $F$  unter  $a$  ist induktiv wie folgt definiert:

$F$	0	1	$x_i$	$\neg G$	$(G \wedge H)$	$(G \vee H)$
$F(a)$	0	1	$a_i$	$1 - G(a)$	$G(a)H(a)$	$G(a) + H(a) - G(a)H(a)$

- Durch die Formel  $F$  wird also eine  **$n$ -stellige boolesche Funktion**  $F : \{0, 1\}^n \rightarrow \{0, 1\}$  definiert, die wir ebenfalls mit  $F$  bezeichnen.

## Definition

- Zwei Formeln  $F$  und  $G$  heißen **logisch äquivalent**, wenn sie dieselbe boolesche Funktion berechnen.
- Eine Formel  $F$  heißt **erfüllbar**, falls es eine Belegung  $a$  mit  $F(a) = 1$  gibt.
- Gilt dagegen für alle Belegungen  $a$ , dass  $F(a) = 1$  ist, so heißt  $F$  **Tautologie**.

# Aussagenlogische Formeln

## Notation

Wir benutzen die **Implikation**  $G \rightarrow H$  als Abkürzung für die Formel  $\neg G \vee H$ .

## Beispiel (Erfüllbarkeitstest mittels Wahrheitstabelle)

Die Formel  $((\neg x_1 \vee \neg x_2) \rightarrow (x_2 \wedge x_3))$  ist erfüllbar:

$a$	$(\neg x_1 \vee \neg x_2)$	$(x_2 \wedge x_3)$	$((\neg x_1 \vee \neg x_2) \rightarrow (x_2 \wedge x_3))$
000	1	0	0
001	1	0	0
010	1	0	0
011	1	1	1
100	1	0	0
101	1	0	0
110	0	0	1
111	0	1	1

## Präzedenzregeln zur Klammerersparnis

- Der Junktor  $\wedge$  bindet stärker als der Junktor  $\vee$  und dieser wiederum stärker als der Junktor  $\rightarrow$ .
- Formeln der Form  $(x_1 \circ (x_2 \circ (x_3 \circ \dots \circ x_n) \dots))$ ,  $\circ \in \{\wedge, \vee\}$  kürzen wir durch  $(x_1 \circ \dots \circ x_n)$  ab.

## Beispiel (Formel für die mehrstellige Entweder-Oder Funktion)

- Folgende Formel nimmt unter einer Belegung  $a = a_1 \dots a_n$  genau dann den Wert 1 an, wenn  $\sum_{i=1}^n a_i = 1$  ist:

$$G(x_1, \dots, x_n) = (x_1 \vee \dots \vee x_n) \wedge \bigwedge_{1 \leq i < j \leq n} \neg(x_i \wedge x_j)$$

- D.h. es gilt genau dann  $G(a) = 1$ , wenn genau eine Variable  $x_i$  mit dem Wert  $a_i = 1$  belegt ist.
- Diese Formel wird im Beweis des nächsten Satzes benötigt. ◀

Erfüllbarkeitsproblem für boolesche Formeln (*satisfiability*, SAT):

Gegeben: Eine boolesche Formel  $F$ .

Gefragt: Ist  $F$  erfüllbar?

Dabei kodieren wir boolesche Formeln  $F$  durch Binärstrings  $w_F$  und ordnen umgekehrt jedem Binärstring  $w$  eine Formel  $F_w$  zu.

Um die Notation zu vereinfachen, werden wir zukünftig jedoch  $F$  anstelle von  $w_F$  schreiben.

**Satz (Cook, Karp, Levin)**

SAT ist NP-vollständig.



# SAT ist NP-vollständig

## Beweis von $\text{SAT} \in \text{NP}$

Es ist leicht zu sehen, dass  $\text{SAT} \in \text{NP}$  ist:

Eine NTM kann bei Eingabe einer booleschen Formel  $F$  zunächst eine Belegung  $a$  nichtdeterministisch raten und dann in Polynomialzeit testen, ob  $F(a) = 1$  ist (*guess and verify* Strategie). □

## Beweis von SAT ist NP-hart

- Sei  $L$  eine beliebige NP-Sprache und sei  $M = (Z, \Sigma, \Gamma, \delta, q_0)$  eine durch ein Polynom  $p$  zeitbeschränkte  $k$ -NTM mit  $L(M) = L$ .
- Da sich eine  $t(n)$ -zeitbeschränkte  $k$ -NTM in Zeit  $t^2(n)$  durch eine 1-NTM simulieren lässt, können wir  $k = 1$  annehmen.
- Unsere Aufgabe besteht nun darin, in Polynomialzeit zu einer gegebenen Eingabe  $w = w_1 \dots w_n$  eine Formel  $F_w$  zu konstruieren, die genau dann erfüllbar ist, wenn  $w \in L$  ist.
- O.B.d.A. sei  $Z = \{q_0, \dots, q_m\}$ ,  $E = \{q_m\}$  und  $\Gamma = \{a_1, \dots, a_l\}$ .
- Zudem sei  $\delta(q_m, a) := \{(q_m, a, N)\}$  für alle  $a \in \Gamma$ .

## Idee:

Konstruiere  $F_w$  so, dass  $F_w$  unter einer Belegung  $a$  genau dann wahr wird, wenn  $a$  eine akzeptierende Rechnung von  $M(w)$  beschreibt.

- Wir bilden  $F_w$  über den Variablen

$$x_{t,q}, \quad \text{für } 0 \leq t \leq p(n), q \in Z,$$

$$y_{t,i}, \quad \text{für } 0 \leq t \leq p(n), -p(n) \leq i \leq p(n),$$

$$z_{t,i,a}, \quad \text{für } 0 \leq t \leq p(n), -p(n) \leq i \leq p(n), a \in \Gamma.$$

- Diese Variablen stehen für folgende Aussagen:

$x_{t,q}$ : zum Zeitpunkt  $t$  befindet sich  $M$  im Zustand  $q$ ,

$y_{t,i}$ : zur Zeit  $t$  besucht  $M$  das Feld mit der Nummer  $i$ ,

$z_{t,i,a}$ : zur Zeit  $t$  steht das Zeichen  $a$  auf dem  $i$ -ten Feld.

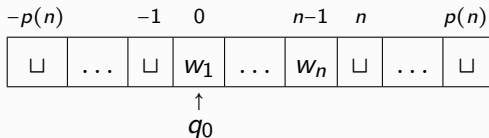
- Konkret sei  $F_w = R \wedge S_w \wedge \dot{U}_1 \wedge \dot{U}_2 \wedge E$ .

- Konkret sei  $F_w = R \wedge S_w \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E$ .
- Dabei stellt die Formel  $R = \bigwedge_{t=0}^{p(n)} R_t$  (Randbedingungen) sicher, dass wir jeder erfüllenden Belegung eindeutig eine Folge von Konfigurationen  $K_0, \dots, K_{p(n)}$  zuordnen können:

$$R_t = G(x_{t,q_0}, \dots, x_{t,q_m}) \wedge G(y_{t,-p(n)}, \dots, y_{t,p(n)}) \\ \wedge \bigwedge_{i=-p(n)}^{p(n)} G(z_{t,i,a_1}, \dots, z_{t,i,a_l}).$$

- Die Teilformel  $R_t$  sorgt also dafür, dass zum Zeitpunkt  $t$ 
  - genau ein Zustand  $q \in \{q_0, \dots, q_m\}$  eingenommen wird,
  - genau ein Bandfeld  $i \in \{-p(n), \dots, p(n)\}$  besucht wird und
  - auf jedem Feld  $i$  genau ein Zeichen  $a_j \in \Gamma$  steht.

- Die Formel  $S_w$  (wie Startbedingung) stellt sicher, dass zum Zeitpunkt 0 tatsächlich die Startkonfiguration vorliegt:



$$S_w = x_{0,q_0} \wedge y_{0,0} \wedge \bigwedge_{i=-p(n)}^{-1} z_{0,i,\sqcup} \wedge \bigwedge_{i=0}^{n-1} z_{0,i,w_{i+1}} \wedge \bigwedge_{i=n}^{p(n)} z_{0,i,\sqcup}$$

- Die Formel  $\ddot{U}_1$  sorgt dafür, dass der Inhalt von nicht besuchten Feldern beim Übergang von  $K_t$  zu  $K_{t+1}$  unverändert bleibt:

$$\ddot{U}_1 = \bigwedge_{t=0}^{p(n)-1} \bigwedge_{i=-p(n)}^{p(n)} \bigwedge_{a \in \Gamma} (\neg y_{t,i} \wedge z_{t,i,a} \rightarrow z_{t+1,i,a})$$

# Beweis von SAT ist NP-hart

- $\ddot{U}_2$  achtet darauf, dass sich bei jedem Übergang der Zustand, die Kopfposition und das gerade gelesene Zeichen gemäß einer Anweisung in  $\delta$  verändern:

$$\ddot{U}_2 = \bigwedge_{t=0}^{p(n)-1} \bigwedge_{i=-p(n)}^{p(n)} \bigwedge_{a \in \Gamma} \bigwedge_{q \in Z} (x_{t,q} \wedge y_{t,i} \wedge z_{t,i,a} \rightarrow \bigvee_{(q',b,D) \in \delta(q,a)} x_{t+1,q'} \wedge y_{t+1,i+D} \wedge z_{t+1,i,b}),$$

wobei

$$i + D = \begin{cases} i - 1, & D = L \\ i, & D = N \\ i + 1, & D = R \end{cases}$$

- Schließlich überprüft  $E$ , ob  $M$  zur Zeit  $p(n)$  den Endzustand  $q_m$  erreicht hat:

$$E = x_{p(n),q_m}$$

# Beweis von SAT ist NP-hart

- Da der Aufbau der Formel  $f(w) = F_w$  einem einfachen Bildungsgesetz folgt und ihre Länge polynomiell in  $n$  ist, folgt  $f \in \text{FP}$ .
- Es ist klar, dass  $F_w$  im Fall  $w \in L(M)$  erfüllbar ist, indem wir die Variablen von  $F_w$  gemäß einer akz. Rechnung von  $M(w)$  belegen.
- Umgekehrt führt eine Belegung  $a$  mit  $F_w(a) = 1$  wegen  $R(a) = 1$  eindeutig auf eine Konfigurationenfolge  $K_0, \dots, K_{p(n)}$ , so dass gilt:
  - $K_0$  ist Startkonfiguration von  $M(w)$  (wegen  $S_w(a) = 1$ ),
  - $K_i \vdash K_{i+1}$  für  $i = 0, \dots, p(n) - 1$  (wegen  $\ddot{U}_1(a) = \ddot{U}_2(a) = 1$ ),
  - $M$  nimmt spätestens bei Erreichen der Konfiguration  $K_{p(n)}$  den Endzustand  $q_m$  an (wegen  $E(a) = 1$ ).
- Also gilt für alle  $w \in \Sigma^*$  die Äquivalenz

$$w \in L(M) \Leftrightarrow F_w \in \text{SAT},$$

d.h. die FP-Funktion  $f : w \mapsto F_w$  reduziert  $L(M)$  auf SAT. □

# Boolesche Schaltkreise

Als nächstes betrachten wir das Erfüllbarkeitsproblem für Schaltkreise.

## Definition

- Ein **boolescher Schaltkreis** über den Variablen  $x_1, \dots, x_n$  ist eine Folge  $S = (g_1, \dots, g_m)$  von **Gattern**

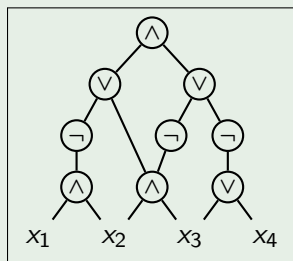
$$g_l \in \{0, 1, x_1, \dots, x_n, (\neg, j), (\wedge, j, k), (\vee, j, k)\} \text{ mit } 1 \leq j, k < l.$$

- Die am Gatter  $g_l$  berechnete  $n$ -stellige boolesche Funktion  $g_l: \{0, 1\}^n \rightarrow \{0, 1\}$  ist induktiv wie folgt definiert ( $a = a_1 \dots a_n$ ):

$g_l$	0	1	$x_i$	$(\neg, j)$	$(\wedge, j, k)$	$(\vee, j, k)$
$g_l(a)$	0	1	$a_i$	$1 - g_j(a)$	$g_j(a)g_k(a)$	$g_j(a) + g_k(a) - g_j(a)g_k(a)$

- $S$  berechnet die boolesche Funktion  $S(a) = g_m(a)$ .
- $S$  heißt **erfüllbar**, wenn eine Eingabe  $a \in \{0, 1\}^n$  mit  $S(a) = 1$  ex.

## Beispiel



Graphische Darstellung  
des Schaltkreises

$$S = (x_1, x_2, x_3, x_4, (\wedge, 1, 2),$$

$$(\wedge, 2, 3), (\vee, 3, 4), (\neg, 5),$$

$$(\neg, 6), (\neg, 7), (\vee, 6, 8),$$

$$(\vee, 9, 10), (\wedge, 11, 12)).$$

## Bemerkung

- Die Anzahl der Eingänge eines Gatters  $g$  wird als **Fanin** von  $g$  bezeichnet,
- die Anzahl der Ausgänge von  $g$  (also die Anzahl der Gatter, die  $g$  als Eingabe benutzen) als **Fanout**.
- Boolesche Formeln entsprechen also den booleschen Schaltkreisen mit (maximalem) Fanout 1 und umgekehrt.
- Eine boolesche Formel  $F$  kann somit leicht in einen äquivalenten Schaltkreis  $S$  mit  $S(a) = F(a)$  für alle Belegungen  $a$  transformiert werden.



## Erfüllbarkeitsproblem für boolesche Schaltkreise (CIRSAT):

Gegeben: Ein boolescher Schaltkreis  $S$ .

Gefragt: Ist  $S$  erfüllbar?

## Satz

CIRSAT ist NP-vollständig.

## Beweis

Klar, da  $SAT \leq^P CIRSAT$  und  $CIRSAT \in NP$  gilt.  $\square$

## Bemerkung

- Da SAT NP-vollständig ist, ist CIRSAT auf SAT reduzierbar.
- Dies bedeutet, dass sich jeder Schaltkreis  $S$  in Polynomialzeit in eine **erfüllbarkeitsäquivalente** Formel  $F_S$  überführen lässt.  
 $F_S$  und  $S$  müssen aber nicht logisch äquivalent sein.
- CIRSAT ist sogar auf eine spezielle SAT-Variante reduzierbar.

## Formeln in konjunktiver Normalform (KNF)

- Ein **Literal** ist eine Variable  $x_i$  oder eine negierte Variable  $\neg x_i$ , die wir auch kurz mit  $\bar{x}_i$  bezeichnen.
- Eine **Klausel** ist eine Disjunktion  $C = \bigvee_{j=1}^k l_j$  von Literalen  $l_1, \dots, l_k$ .
- Hierbei ist auch  $k = 0$  zulässig, d.h. die **leere Klausel** repräsentiert die Konstante 0 und wird üblicherweise mit  $\square$  bezeichnet.
- Eine boolesche Formel  $F$  ist in **konjunktiver Normalform** (kurz **KNF**), falls  $F = \bigwedge_{j=1}^m C_j$  eine Konjunktion von Klauseln  $C_1, \dots, C_m$  ist.
- Auch hier ist  $m = 0$  zulässig, wobei die **leere Konjunktion** die Konstante 1 repräsentiert.
- Enthält jede Klausel höchstens  $k$  Literale, so heißt  $F$  in  **$k$ -KNF**.
- Klauseln werden oft als Menge  $C = \{l_1, \dots, l_k\}$  ihrer Literale und KNF-Formeln als Menge  $F = \{C_1, \dots, C_m\}$  ihrer Klauseln dargestellt.
- Enthält  $F$  die leere Klausel, so ist  $F$  unerfüllbar.
- Dagegen ist die leere Formel eine Tautologie.

## Erfüllbarkeitsproblem für $k$ -KNF Formeln ( $k$ -SAT):

Gegeben: Eine boolesche Formel  $F$  in  $k$ -KNF.

Gefragt: Ist  $F$  erfüllbar?

## Not-All-Equal-SAT ( $\text{NAESAT}$ ):

Gegeben: Eine Formel  $F$  in 3-KNF.

Gefragt: Hat  $F$  eine (erfüllende) Belegung, unter der in keiner Klausel alle Literale denselben Wahrheitswert haben?

## Beispiel

- Der 3-KNF Formel  $F = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$  entspricht die Klauselmenge

$$F = \{ \{x_1, \bar{x}_2\}, \{\bar{x}_1, x_3\}, \{x_2, \bar{x}_3, x_4\} \}$$

- Offenbar ist  $F(0000) = 1$ , d.h.  $F \in 3\text{-SAT}$ . Zudem gilt  $F \in \text{NAESAT}$ .

Satz

3-SAT ist NP-vollständig.

## 3-SAT ist NP-vollständig

## Reduktion von CIRSAT auf 3-SAT

- Wir transformieren einen Schaltkreis  $S = (g_1, \dots, g_m)$  mit  $n$  Eingängen in eine 3-KNF Formel  $F_S$  über den Variablen  $x_1, \dots, x_n, y_1, \dots, y_m$ , die die Klausel  $\{y_m\}$  und für jedes Gatter  $g_i$  die Klauseln folgender Formel  $F_i$  enthält:

Gatter $g_i$	Semantik von $F_i$	Klauseln von $F_i$
0	$y_i = 0$	$\{\bar{y}_i\}$
1	$y_i = 1$	$\{y_i\}$
$x_j$	$y_i = x_j$	$\{\bar{y}_i, x_j\}, \{\bar{x}_j, y_i\}$
$(\neg, j)$	$y_i = \bar{y}_j$	$\{\bar{y}_i, \bar{y}_j\}, \{y_j, y_i\}$
$(\wedge, j, k)$	$y_i = y_j \wedge y_k$	$\{\bar{y}_i, y_j\}, \{\bar{y}_i, y_k\}, \{\bar{y}_j, \bar{y}_k, y_i\}$
$(\vee, j, k)$	$y_i = y_j \vee y_k$	$\{\bar{y}_j, y_i\}, \{\bar{y}_k, y_i\}, \{\bar{y}_i, y_j, y_k\}$

## 3-SAT ist NP-vollständig

### Reduktion von CIRSAT auf 3-SAT

- Wir zeigen, dass für alle  $a \in \{0, 1\}^n$  folgende Äquivalenz gilt:

$$S(a) = 1 \Leftrightarrow \exists b \in \{0, 1\}^m : F_S(ab) = 1.$$

- Ist nämlich  $a \in \{0, 1\}^n$  eine Eingabe mit  $S(a) = 1$ . Dann erhalten wir mit

$$b_i = g_i(a) \text{ für } i = 1, \dots, m$$

eine erfüllende Belegung  $ab_1 \dots b_m$  für  $F_S$ .

- Ist umgekehrt  $ab_1 \dots b_m$  eine erfüllende Belegung für  $F_S$ , so muss

- $b_m = 1$  sein, da  $\{y_m\}$  eine Klausel in  $F_S$  ist, und
- durch Induktion über  $i = 1, \dots, m$  folgt

$$g_i(a) = b_i,$$

d.h. insbesondere folgt  $S(a) = g_m(a) = b_m = 1$ .

## Reduktion von CIRSAT auf 3-SAT

- Wir wissen bereits, dass für alle  $a \in \{0, 1\}^n$  die Äquivalenz

$$s(a) = 1 \Leftrightarrow \exists b \in \{0, 1\}^m : F_S(ab) = 1.$$

gilt.

- Dies bedeutet, dass der Schaltkreis  $S$  und die 3-KNF-Formel  $F_S$  erfüllbarkeitsäquivalent sind, d.h.

$$S \in \text{CIRSAT} \Leftrightarrow F_S \in \text{3-SAT}.$$

- Da zudem die Reduktionsfunktion  $S \mapsto F_S$  in FP berechenbar ist, folgt  $\text{CIRSAT} \leq^P \text{3-SAT}$ . □

- Als nächstes zeigen wir, dass das Wortproblem für NFAs effizient entscheidbar ist.
- Dagegen wird sich das Äquivalenzproblem für NFAs als schwer lösbar (genauer: co-NP-hart) herausstellen.



## Satz

Das Wortproblem für NFAs,

$$WP_{\text{NFA}} = \{N\#x \mid N \text{ ist ein NFA und } x \in L(N)\},$$

ist in P entscheidbar.

P-Algorithmus für  $WP_{\text{NFA}}$ 

- 1 **Input:** Ein NFA  $N = (Z, \Sigma, \delta, Q_0, E)$  und ein Wort  $x = x_1 \dots x_n$
- 2  $Q := Q_0$
- 3 **for**  $i := 1$  **to**  $n$  **do**
- 4      $Q := \bigcup_{q \in Q} \delta(q, x_i)$
- 5 **if**  $Q \cap E \neq \emptyset$  **then accept else reject**

Es ist klar, dass dieser Algorithmus korrekt arbeitet und auch in eine polynomiell zeitbeschränkte DTM für die Sprache  $WP_{\text{NFA}}$  transformiert werden kann.

Ganz ähnlich folgt auch, dass das Leerheits- und das Schnittproblem für NFAs in Polynomialzeit lösbar sind (siehe Übungen).

## Korollar

Das Leerheitsproblem für NFAs,

$$LP_{\text{NFA}} = \{N \mid N \text{ ist ein NFA und } L(N) = \emptyset\}$$

und das Schnittproblem für NFAs,

$$SP_{\text{NFA}} = \{N_1 \# N_2 \mid N_1 \text{ und } N_2 \text{ sind NFAs mit } L(N_1) \cap L(N_2) \neq \emptyset\}$$

sind in P lösbar.

# Entscheidungsprobleme für reguläre Ausdrücke

## Korollar

Das Wort-, das Leerheits- und das Schnittproblem für reguläre Ausdrücke sind in  $P$  entscheidbar.

## Beweis

- Ein regulärer Ausdruck  $\alpha$  lässt sich in Polynomialzeit in einen äquivalenten NFA  $N_\alpha$  transformieren (siehe Übungen).
- Daher gilt  $WP_{RA} \leq^P WP_{NFA}$  mittels  $f : (\alpha \# x) \mapsto (N_\alpha \# x)$ .
- Da nach vorigem Satz  $WP_{NFA} \in P$  ist, und da  $P$  unter  $\leq^P$  abgeschlossen ist, folgt  $WP_{RA} \in P$ .
- Ähnlich zeigt man  $LP_{RA} \leq^P LP_{NFA}$  und  $SP_{RA} \leq^P SP_{NFA}$ , was die Zugehörigkeit von  $LP_{RA}$  und  $SP_{RA}$  zu  $P$  impliziert. □

Dagegen sind das Äquivalenz- und das Inklusionsproblem für reguläre Ausdrücke co-NP-hart. Dies gilt sogar für **sternfreie** reguläre Ausdrücke (kurz SFRAs), also für reguläre Ausdrücke, die keinen Stern enthalten.

Die Tabelle gibt die Komplexitäten der wichtigsten Entscheidungsprobleme für durch DFAs, NFAs oder (sternfreie) reguläre Ausdrücke gegebene reguläre Sprachen an.

	Wort- problem $x \in L?$	Leerheits- problem $L = \emptyset?$	Schnitt- problem $L_1 \cap L_2 \neq \emptyset?$	Äquivalenz- problem $L_1 = L_2?$	Inklusions- problem $L_1 \subseteq L_2$
DFA	P	P	P	P	P
SFRA	P	P	P	co-NP-vollständig	
RA	P	P	P	PSPACE-vollständig	
NFA	P	P	P	PSPACE-vollständig	

Als nächstes zeigen wir, dass die Probleme  $\text{ÄP}_{\text{SFRA}}$  und  $\text{IP}_{\text{SFRA}}$  **co-NP-vollständig** sind.

# Das Äquivalenzproblem für sternfreie reguläre Ausdrücke

## Korollar

Das Äquivalenzproblem für sternfreie reguläre Ausdrücke,

$$\text{ÄP}_{\text{SFRA}} = \{\alpha \# \beta \mid \alpha, \beta \text{ sind SFRA's mit } L(\alpha) = L(\beta)\},$$

ist co-NP-vollständig.

## Beweis

- Wir zeigen, dass das Inäquivalenzproblem für SFRA's NP-vollständig ist.
- Es ist leicht zu zeigen (durch Induktion über den Aufbau von  $\alpha$ ), dass  $L(\alpha)$  für einen SFRA  $\alpha$  der Länge  $m$  nur Wörter der Länge  $\leq m$  enthält.
- D.h. es gilt  $L(\alpha) \subseteq \{0, 1\}^{\leq m}$ , wobei  $\Sigma^{\leq m} = \bigcup_{i=0}^m \Sigma^i$  ist.
- Die Zugehörigkeit von  $\overline{\text{ÄP}_{\text{SFRA}}}$  zu NP liefert folgender Algorithmus:

```

1 Input: SFRA's  $\alpha$  und  $\beta$ 
2    $m := \max\{|\alpha|, |\beta|\}$ 
3   guess  $x \in \{0, 1\}^{\leq m}$ 
4   if  $x \in L(\alpha) \Delta L(\beta)$  then accept else reject
  
```

## Reduktion von 3-SAT auf das Inäquivalenzproblem für SFRA's

- Sei  $F = \{C_1, \dots, C_m\}$  eine 3-KNF Formel.
- Betrachte den SFRA  $\alpha_F = (\alpha_1 | \dots | \alpha_m)$  mit  $\alpha_j = \beta_{j1} \dots \beta_{jn}$  und

$$\beta_{ji} = \begin{cases} 0, & x_i \in C_j, \\ 1, & \bar{x}_i \in C_j, \\ (0|1), & \text{sonst.} \end{cases}$$

- Dann ist  $L(\alpha_j) = \{a \in \{0, 1\}^n \mid C_j(a) = 0\}$  und daher folgt

$$\begin{aligned} F \in 3\text{-SAT} &\Leftrightarrow \exists a \in \{0, 1\}^n : F(a) = 1 \\ &\Leftrightarrow \exists a \in \{0, 1\}^n \forall j = 1, \dots, m : C_j(a) = 1 \\ &\Leftrightarrow \exists a \in \{0, 1\}^n \forall j = 1, \dots, m : a \notin L(\alpha_j) \\ &\Leftrightarrow \exists a \in \{0, 1\}^n : a \notin L(\alpha_F) \\ &\Leftrightarrow L(\alpha_F) \neq \{0, 1\}^n \end{aligned}$$

- Also folgt  $3\text{-SAT} \leq^p \overline{\text{ÄP}_{\text{SFRA}}}$  mittels  $F \mapsto \alpha_F \# \underbrace{(0|1) \dots (0|1)}_{n\text{-mal}}$ . □

## Notation – ungerichtete Graphen

- Ein (**ungerichteter**) **Graph** ist ein Paar  $G = (V, E)$ , wobei
  - $V$  - eine endliche Menge von **Knoten/Ecken** und
  - $E$  - die Menge der **Kanten** ist.

Hierbei gilt

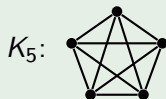
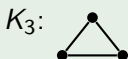
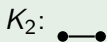
$$E \subseteq \binom{V}{2} := \{\{u, v\} \subseteq V \mid u \neq v\}.$$

- Die **Knotenzahl** von  $G$  ist  $n(G) = \|V\|$ .
- Die **Kantenzahl** von  $G$  ist  $m(G) = \|E\|$ .
- Die **Nachbarschaft** von  $v \in V$  ist  $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$  und die Nachbarschaft von  $U \subseteq V$  ist  $N_G(U) = \bigcup_{u \in U} N_G(u)$ .
- Der **Grad** von  $v \in V$  ist  $\deg_G(v) = \|N_G(v)\|$ .
- Der **Minimalgrad** von  $G$  ist  $\delta(G) := \min_{v \in V} \deg_G(v)$  und
- der **Maximalgrad** von  $G$  ist  $\Delta(G) := \max_{v \in V} \deg_G(v)$ .

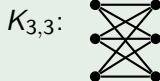
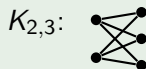
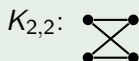
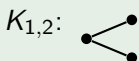
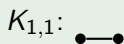
Falls  $G$  aus dem Kontext ersichtlich ist, schreiben wir auch einfach  $n$ ,  $m$ ,  $N(v)$ ,  $\deg(v)$ ,  $\delta$  usw.

## Beispiel

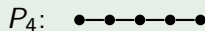
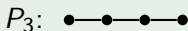
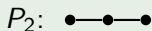
- Der **vollständige Graph**  $(V, E)$  mit  $\|V\| = n$  und  $E = \binom{V}{2}$  wird mit  $K_n$  und der **leere Graph**  $(V, \emptyset)$  wird mit  $E_n$  bezeichnet.



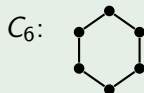
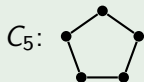
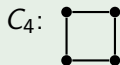
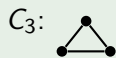
- Der **vollständige bipartite Graph**  $(A, B, E)$  auf  $a + b$  Knoten, d.h.  $A \cap B = \emptyset$ ,  $\|A\| = a$ ,  $\|B\| = b$  und  $E = \{\{u, v\} \mid u \in A, v \in B\}$  wird mit  $K_{a,b}$  bezeichnet.



- Der **Pfad der Länge  $m$**  wird mit  $P_m$  bezeichnet.



- Der **Kreis der Länge  $n$**  wird mit  $C_n$  bezeichnet.





- Ein Graph  $H = (V', E')$  heißt **Sub-/Teil-/Untergraph** von  $G = (V, E)$ , falls  $V' \subseteq V$  und  $E' \subseteq E$  ist.
- Ein **Weg** ist eine Folge von Knoten  $v_0, \dots, v_j$  mit  $\{v_i, v_{i+1}\} \in E$  für  $i = 0, \dots, j - 1$ .  
Ein Weg heißt **einfach** oder **Pfad**, falls alle durchlaufenen Knoten verschieden sind.  
Die **Länge** des Weges ist die Anzahl der Kanten, also  $j$ .  
Ein Weg  $v_0, \dots, v_j$  heißt auch  **$v_0$ - $v_j$ -Weg**.
- Ein **Zyklus** ist ein  $u$ - $v$ -Weg der Länge  $j \geq 2$  mit  $u = v$ .
- Ein **Kreis** ist ein Zyklus  $v_0, v_1, \dots, v_{j-1}, v_0$  der Länge  $j \geq 3$ , für den  $v_0, v_1, \dots, v_{j-1}$  paarweise verschieden sind.

## Cliquen, Stabilität und Matchings

- Eine Knotenmenge  $U \subseteq V$  heißt **Clique**, wenn jede Kante mit beiden Endpunkten in  $U$  in  $E$  ist, d.h. es gilt  $\binom{U}{2} \subseteq E$ .

Die **Cliquenzahl** ist

$$\omega(G) := \max\{\|U\| \mid U \text{ ist Clique in } G\}.$$

- Eine Knotenmenge  $U \subseteq V$  heißt **stabil** oder **unabhängig**, wenn keine Kante in  $G$  beide Endpunkte in  $U$  hat, d.h. es gilt  $E \cap \binom{U}{2} = \emptyset$ .

Die **Stabilitätszahl** ist

$$\alpha(G) := \max\{\|U\| \mid U \text{ ist stabile Menge in } G\}.$$

- Zwei Kanten  $e, e' \in E$  heißen **unabhängig**, falls  $e \cap e' = \emptyset$  ist.

Eine Kantenmenge  $M \subseteq E$  heißt **Matching** in  $G$ , falls alle Kanten in  $M$  paarweise unabhängig sind.

Die **Matchingzahl** von  $G$  ist

$$\mu(G) = \max\{\|M\| \mid M \text{ ist ein Matching in } G\}.$$

# Kantenüberdeckungen und Färbungen

- Eine Knotenmenge  $U \subseteq V$  heißt **Kantenüberdeckung** (engl. *vertex cover*), wenn jede Kante  $e \in E$  mindestens einen Endpunkt in  $U$  hat, d.h. es gilt  $e \cap U \neq \emptyset$  für alle Kanten  $e \in E$ .

Die **Überdeckungszahl** ist

$$\beta(G) = \min\{\|U\| \mid U \text{ ist eine Kantenüberdeckung in } G\}.$$

- Eine Abbildung  $f: V \rightarrow \mathbb{N}$  heißt **Färbung** von  $G$ , wenn  $f(u) \neq f(v)$  für alle  $\{u, v\} \in E$  gilt.

$G$  heißt  **$k$ -färbbar**, falls eine Färbung  $f: V \rightarrow \{1, \dots, k\}$  existiert.

Die **chromatische Zahl** ist

$$\chi(G) = \min\{k \in \mathbb{N} \mid G \text{ ist } k\text{-färbbar}\}.$$

## Algorithmische Graphprobleme

Seien ein Graph  $G$  und eine Zahl  $k \geq 1$  gegeben.

### CLIQUE:

Gefragt: Hat  $G$  eine Clique der Größe  $k$ ?

### MATCHING:

Gefragt: Hat  $G$  ein Matching der Größe  $k$ ?

### INDEPENDENT SET (IS):

Gefragt: Hat  $G$  eine stabile Menge der Größe  $k$ ?

### VERTEX COVER (VC):

Gefragt: Hat  $G$  eine Kantenüberdeckung der Größe  $k$ ?

### FÄRBBARKEIT (COLORING):

Gefragt: Ist  $G$   $k$ -färbbar?

## FÄRBBARKEIT (COLORING):

Gefragt: Ist  $G$   $k$ -färbbar?

## $k$ -FÄRBBARKEIT ( $k$ -COLORING):

Bei diesem Problem ist  $k$  fest und nur ein Graph  $G$  gegeben.

## Satz

- CLIQUE, IS, VC, COLORING und 3-COLORING sind NP-vollständig.
- 2-COLORING und MATCHING sind in P entscheidbar.

## IS ist NP-vollständig

## Reduktion von 3-SAT auf IS

- Sei  $F = \{C_1, \dots, C_m\}$  mit  $C_j = \{l_{j,1}, \dots, l_{j,k_j}\}$  für  $j = 1, \dots, m$  eine 3-KNF-Formel über den Variablen  $x_1, \dots, x_n$ .

- Betrachte den Graphen  $G = (V, E)$  mit

$$V = \{v_{ji} \mid 1 \leq j \leq m, 1 \leq i \leq k_j\} \text{ und}$$

$$E = \{\{v_{ji}, v_{j'i'}\} \in \binom{V}{2} \mid j = j' \text{ oder } l_{ji} = \bar{l}_{j'i'} \text{ oder } l_{j'i'} = \bar{l}_{ji}\}.$$

- Nun gilt

$F \in 3\text{-SAT} \Leftrightarrow$  es gibt eine Belegung, die in jeder Klausel  $C_j$  (mindestens) ein Literal  $l_{j,i_j}$  wahr macht

$\Leftrightarrow$  es gibt  $m$  Literale  $l_{1,i_1}, \dots, l_{m,i_m}$ , die paarweise nicht komplementär sind (d.h.  $l_{j,i_j} \neq \bar{l}_{j',i_{j'}}$  für  $j \neq j'$ )

$\Leftrightarrow$  es gibt  $m$  Knoten  $v_{1,i_1}, \dots, v_{m,i_m}$ , die nicht durch Kanten verbunden sind

$\Leftrightarrow G$  besitzt eine stabile Menge von  $m$  Knoten. □

## Korollar

CLIQUE ist NP-vollständig.

## Beweis

- Es ist leicht zu sehen, dass jede Clique in einem Graphen  $G = (V, E)$  eine stabile Menge in dem zu  $G$  komplementären Graphen  $\bar{G} = (V, \bar{E})$  mit  $\bar{E} = \binom{V}{2} \setminus E$  ist und umgekehrt.
- Daher lässt sich IS mittels

$$f : (G, k) \mapsto (\bar{G}, k)$$

auf CLIQUE reduzieren. □

## Korollar

VC ist NP-vollständig.

## Beweis

- Offensichtlich ist eine Menge  $I$  genau dann stabil, wenn ihr Komplement  $V \setminus I$  eine Kantenüberdeckung ist.
- Daher lässt sich IS mittels

$$f : (G, k) \mapsto (G, n(G) - k)$$

auf VC reduzieren.





## Not-All-Equal-SAT ( $\text{NAESAT}$ ):

Gegeben: Eine Formel  $F$  in 3-KNF.

Gefragt: Hat  $F$  eine (erfüllende) Belegung, unter der in keiner Klausel alle Literale denselben Wahrheitswert haben?

## Satz

$\text{NAESAT}$  ist NP-vollständig.

## Beweis

- $\text{NAESAT} \in \text{NP}$  ist klar.
- Wir reduzieren  $\text{CIRSAT}$  auf  $\text{NAESAT}$ , indem wir die Reduktion von  $\text{CIRSAT}$  auf 3-SAT geeignet anpassen.

# NAESAT ist NP-vollständig

## Reduktion von CIRSAT auf NAESAT

- Wir reduzieren CIRSAT auf NAESAT, indem wir die Reduktion von CIRSAT auf 3-SAT geeignet anpassen:

Gatter $g_i$	Klauseln von $F'_i$
0	$\{\bar{y}_i, z\}$
1	$\{y_i, z\}$
$x_j$	$\{\bar{y}_i, x_j, z\}, \{\bar{x}_j, y_i, z\}$
$(\neg, j)$	$\{\bar{y}_i, \bar{y}_j, z\}, \{y_j, y_i, z\}$
$(\wedge, j, k)$	$\{\bar{y}_i, y_j, z\}, \{\bar{y}_i, y_k, z\}, \{\bar{y}_j, \bar{y}_k, y_i\}$
$(\vee, j, k)$	$\{\bar{y}_j, y_i, z\}, \{\bar{y}_k, y_i, z\}, \{\bar{y}_i, y_j, y_k\}$

- Es ist leicht zu sehen, dass alle **Dreierklauseln** unter jeder erfüllenden Belegung von  $F_S$  bereits beide Wahrheitswerte annehmen.
- Zu den übrigen Klauseln können wir eine neue Variable  $z$  hinzufügen.

## Reduktion von CIRSAT auf NAESAT

- Sei also  $F'_S(x_1, \dots, x_n, y_1, \dots, y_m, z)$  die Formel, die die Klausel  $\{y_m, z\}$  und für jedes Gatter  $g_i$  die Klauseln folgender Formel  $F'_i$  enthält:

Gatter $g_i$	Klauseln von $F'_i$
0	$\{\bar{y}_i, z\}$
1	$\{y_i, z\}$
$x_j$	$\{\bar{y}_i, x_j, z\}, \{\bar{x}_j, y_i, z\}$
$(\neg, j)$	$\{\bar{y}_i, \bar{y}_j, z\}, \{y_j, y_i, z\}$
$(\wedge, j, k)$	$\{\bar{y}_i, y_j, z\}, \{\bar{y}_i, y_k, z\}, \{\bar{y}_j, \bar{y}_k, y_i\}$
$(\vee, j, k)$	$\{\bar{y}_j, y_i, z\}, \{\bar{y}_k, y_i, z\}, \{\bar{y}_i, y_j, y_k\}$

- $F'_S$  entsteht also aus  $F_S$ , indem wir zu jeder Klausel mit  $\leq 2$  Literalen die neue Variable  $z$  hinzufügen.

# NAESAT ist NP-vollständig

## Reduktion von CIRSAT auf NAESAT

- Da die Funktion  $f : S \mapsto F'_S$  offenbar in FP berechenbar ist, bleibt nur noch die Korrektheit von  $f$  zu zeigen:

$$S \in \text{CIRSAT} \Leftrightarrow F'_S \in \text{NAESAT}.$$

- Ist  $S \in \text{CIRSAT}$ , so existiert eine Belegung  $ab$  mit  $F_S(ab) = 1$ .
- Folglich enthalten unter dieser Belegung alle Klauseln von  $F_S$  (und damit auch alle Klauseln von  $F'_S$ ) ein wahres Literal.
- Tatsächlich wird unter  $ab$  in jeder Dreierklausel von  $F_S$  (und damit in jeder Klausel von  $F'_S$ , die  $z$  nicht enthält) auch ein Literal falsch, da  $ab$  neben jeder Dreierklausel der Form
  - $\{\bar{y}_i, y_j, y_k\}$  die Klauseln  $\{\bar{y}_j, y_i\}$  und  $\{\bar{y}_k, y_i\}$  und neben
  - $\{y_i, \bar{y}_j, \bar{y}_k\}$  die Klauseln  $\{y_j, \bar{y}_i\}$  und  $\{y_k, \bar{y}_j\}$  erfüllt.
- Setzen wir also  $z = 0$ , so wird in jeder Klausel von  $F'_S$  mindestens ein Literal wahr und mindestens ein Literal falsch.

## Reduktion von CIRSAT auf NAESAT

- Für die umgekehrte Implikation sei nun  $F'_S \in \text{NAESAT}$  angenommen.
- Dann existiert eine Belegung  $abc \in \{0, 1\}^{n+m+1}$  für

$$F'_S(x_1, \dots, x_n, y_1, \dots, y_m, z),$$

unter der in jeder Klausel ein wahres und ein falsches Literal vorkommen.

- Da dies auch unter der komplementären Belegung  $\overline{abc}$  der Fall ist, können wir  $c = 0$  annehmen.
- Dann erfüllt aber die Belegung  $ab$  die Formel  $F_S$ .
- Also ist  $S(a) = 1$  und damit  $s \in \text{CIRSAT}$ . □

## 3-COLORING ist NP-vollständig

### Reduktion von NAESAT auf 3-COLORING

- Sei eine 3-KNF-Formel  $F = \{C_1, \dots, C_m\}$  über den Variablen  $x_1, \dots, x_n$  mit Klauseln

$$C_j = \{l_{j,1}, \dots, l_{j,k_j}\}, \quad k_j \leq 3$$

gegeben.

- Wir können annehmen, dass  $F$  keine Einerklauseln enthält.
- Wir konstruieren einen Graphen  $G_F = (V, E)$ , der genau dann 3-färbbar ist, wenn  $F \in \text{NAESAT}$  ist.
- Wir setzen

$$V = \{s, x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\} \cup \{v_{jk} \mid 1 \leq j \leq m, 1 \leq k \leq k_j\}$$

und

$$E = \left\{ \{s, x_i\}, \{s, \bar{x}_i\}, \{x_i, \bar{x}_i\} \mid 1 \leq i \leq n \right\} \cup \left\{ \{s, v_{jk}\} \mid k_j = 2 \right\} \cup \\ \left\{ \{v_{jk}, v_{jl}\} \mid k \neq l \right\} \cup \left\{ \{v_{jk}, x_i\} \mid l_{jk} = \bar{x}_i \right\} \cup \left\{ \{v_{jk}, \bar{x}_i\} \mid l_{jk} = x_i \right\}.$$

## 3-COLORING ist NP-vollständig

## Reduktion von NAESAT auf 3-COLORING

- Wir setzen

$$V = \{s\} \cup \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\} \cup \{v_{jk} \mid 1 \leq j \leq m, 1 \leq k \leq k_j\},$$

und

$$E = \left\{ \{s, x_i\}, \{s, \bar{x}_i\}, \{x_i, \bar{x}_i\} \mid 1 \leq i \leq n \right\} \cup \left\{ \{s, v_{jk}\} \mid k_j = 2 \right\} \cup \\ \left\{ \{v_{jk}, v_{jl}\} \mid k \neq l \right\} \cup \left\{ \{v_{jk}, x_i\} \mid l_{jk} = \bar{x}_i \right\} \cup \left\{ \{v_{jk}, \bar{x}_i\} \mid l_{jk} = x_i \right\}.$$

- Sei  $a = a_1 \dots a_n$  eine Belegung für  $F$ , unter der in jeder Klausel  $C_j = \{l_{j1}, \dots, l_{jk_j}\}$  ein Literal wahr und eines falsch wird.
- Wir können annehmen, dass  $l_{j1}(a) = 0$  und  $l_{j2}(a) = 1$  ist.
- Dann lässt sich  $G_F$  wie folgt mit den 3 Farben 0, 1, 2 färben:

Knoten $v$	$s$	$x_i$	$\bar{x}_i$	$v_{j1}$	$v_{j2}$	$v_{j3}$ (falls $k_j = 3$ )
Farbe $c(v)$	2	$a_i$	$\bar{a}_i$	0	1	2

## 3-COLORING ist NP-vollständig

### Reduktion von NAESAT auf 3-COLORING

- Sei nun umgekehrt  $c : V \rightarrow \{0, 1, 2\}$  eine 3-Färbung von  $G_F$ .
- Dann können wir annehmen, dass  $c(v) = 2$  ist.
- Also gilt für  $i = 1, \dots, n$ , dass  $\{c(x_i), c(\bar{x}_i)\} = \{0, 1\}$  ist.
- Zudem müssen die Knoten  $v_{j1}, \dots, v_{jk_j}$  im Fall  $k_j = 2$  mit 0 und 1 und im Fall  $k_j = 3$  mit allen drei Farben 0, 1 und 2 gefärbt sein.
- Wir können annehmen, dass  $c(v_{j1}) = 0$  und  $c(v_{j2}) = 1$  ist.
- Wegen  $\{v_{jk}, \bar{l}_{jk}\} \in E$  muss  $c(v_{jk}) \neq c(\bar{l}_{jk})$  für  $k = 1, \dots, k_j$  und daher  $c(v_{jk}) = c(l_{jk})$  für  $k = 1, 2$  gelten.
- Also macht die Belegung  $a = c(x_1) \dots c(x_n)$  die Literale  $l_{j1}$  falsch und  $l_{j2}$  wahr.
- Insgesamt gilt also

$$F \in \text{NAESAT} \Leftrightarrow G_F \in \text{3-COLORING.}$$



# Notation – gerichtete Graphen

## Definition

- Ein **gerichteter Graph** oder **Digraph** ist ein Paar  $G = (V, E)$ , wobei
  - $V$  - eine endliche Menge von **Knoten/Ecken** und
  - $E$  - die Menge der **Kanten** ist.

Hierbei gilt

$$E \subseteq V \times V = \{(u, v) \mid u, v \in V\},$$

wobei  $E$  auch Schlingen  $(u, u)$  enthalten kann.

- Die **Nachfolgermenge** von  $v$  ist  $N^+(v) = \{u \in V \mid (v, u) \in E\}$ .
- Die **Vorgängermenge** von  $v$  ist  $N^-(v) = \{u \in V \mid (u, v) \in E\}$ .
- Die **Nachbarmenge** von  $v$  ist  $N(v) = N^+(v) \cup N^-(v)$ .
- Der **Ausgangsgrad** von  $v$  ist  $\deg^+(v) = \|N^+(v)\|$ .
- Der **Eingangsgrad** von  $v$  ist  $\deg^-(v) = \|N^-(v)\|$ .
- Der **Grad** von  $v$  ist  $\deg(v) = \deg^+(v) + \deg^-(v)$ .

# Gerichtete Zyklen und Kreise

## Definition

Sei  $G = (V, E)$  ein Digraph.

- Ein (**gerichteter**)  $v_0$ - $v_j$ -Weg in  $G$  ist eine Folge von Knoten  $(v_0, \dots, v_j)$  mit  $(v_i, v_{i+1}) \in E$  für  $i = 0, \dots, j-1$ .
- Ein (**gerichteter**) Zyklus in  $G$  ist ein gerichteter  $u$ - $v$ -Weg der Länge  $j \geq 1$  mit  $u = v$ .
- Ein gerichteter Weg heißt **einfach** oder (**gerichteter**) Pfad, falls alle durchlaufenen Knoten paarweise verschieden sind.
- Ein (**gerichteter**) Kreis in  $G$  ist ein gerichteter Zyklus  $(v_0, \dots, v_{j-1}, v_0)$  der Länge  $j \geq 1$ , für den  $v_0, \dots, v_{j-1}$  paarweise verschieden sind.
- $G$  heißt **stark zusammenhängend**, wenn es von jedem Knoten  $u$  in  $G$  zu jedem Knoten  $v$  in  $G$  einen Weg gibt.

## Definition

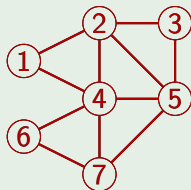
Sei  $G = (V, E)$  ein Graph (oder Digraph) und sei  $s = (v_0, v_1, \dots, v_l)$  ein (gerichteter) Weg in  $G$ , d.h.  $\{v_i, v_{i+1}\}$  bzw.  $(v_i, v_{i+1}) \in E$  für  $i = 0, \dots, l-1$ .

- $s = (v_0, v_1, \dots, v_l)$  heißt **Eulerlinie** (auch **Eulerzug** oder **Eulerweg**) in  $G$ , falls  $s$  jede Kante in  $E$  genau einmal durchläuft, d.h. es gilt  $l = \|E\|$  und

$$\left\{ \{v_i, v_{i+1}\} \text{ bzw. } (v_i, v_{i+1}) \mid i = 0, \dots, l-1 \right\} = E.$$

## Beispiel (Eulerlinie in einem ungerichteten Graphen)

$$s = (4, 1, 2, 3, 5, 7, 6, 4, 5, 2, 4, 7)$$



## Definition

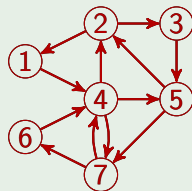
- $s = (v_0, v_1, \dots, v_l)$  in  $G$  heißt **Eulerlinie** (auch **Eulerzug** oder **Eulerweg**), falls  $s$  jede Kante in  $E$  genau einmal durchläuft, d.h. es gilt  $l = \|E\|$  und

$$\left\{ \{v_i, v_{i+1}\} \text{ bzw. } (v_i, v_{i+1}) \mid i = 0, \dots, l-1 \right\} = E.$$

- Ist  $s$  zudem ein Zyklus, d.h. es gilt  $v_l = v_0$ , so heißt  $s$  **Eulerkreis** (auch **Eulerzyklus** oder **Eulertour**).

## Beispiel (Eulerkreis in einem Digraphen)

$$s = (1, 4, 5, 2, 3, 5, 7, 4, 7, 6, 4, 2, 1)$$



# Hamiltonpfade

## Definition

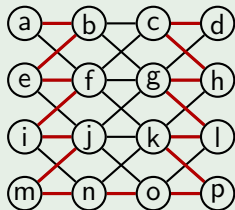
Sei  $G = (V, E)$  ein Graph (oder Digraph) und sei  $s = (v_0, v_1, \dots, v_l)$  ein (gerichteter) Pfad in  $G$ , d.h.  $\{v_i, v_{i+1}\}$  bzw.  $(v_i, v_{i+1}) \in E$  für  $i = 0, \dots, l-1$  und die Knoten  $v_0, \dots, v_l$  sind paarweise verschieden.

- $s$  heißt **Hamiltonpfad** in  $G$ , falls  $s$  jeden Knoten in  $V$  genau einmal durchläuft, d.h. es gilt

$$V = \{v_0, \dots, v_l\} \text{ und } l = \|V\| - 1.$$

## Beispiel (Hamiltonpfad in einem ungerichteten Graphen)

$$s = (a, b, e, f, i, j, m, n, o, p, k, l, g, h, c, d)$$



# Hamiltonpfade

## Definition

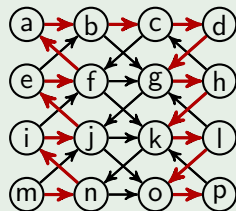
Sei  $G = (V, E)$  ein Graph (oder **Digraph**) und sei  $s = (v_0, v_1, \dots, v_l)$  ein (**gerichteter**) Pfad in  $G$ , d.h.  $\{v_i, v_{i+1}\}$  bzw.  $(v_i, v_{i+1}) \in E$  für  $i = 0, \dots, l-1$  und die Knoten  $v_0, \dots, v_l$  sind paarweise verschieden.

- $s$  heißt **Hamiltonpfad** in  $G$ , falls  $s$  jeden Knoten in  $V$  genau einmal durchläuft, d.h. es gilt

$$V = \{v_0, \dots, v_l\} \text{ und } l = \|V\| - 1.$$

## Beispiel (Hamiltonpfad in einem Digraphen)

$$s = (m, n, i, j, e, f, a, b, c, d, g, h, k, l, o, p)$$



## Definition

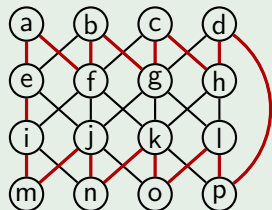
- $s = (v_0, v_1, \dots, v_l)$  in  $G$  heißt **Hamiltonpfad**, falls  $s$  jeden Knoten in  $V$  genau einmal durchläuft, d.h. es gilt

$$V = \{v_0, \dots, v_l\} \text{ und } l = \|V\| - 1.$$

- Ist zudem  $\{v_0, v_l\}$  bzw.  $(v_l, v_0) \in E$ , d.h.  $s' = (v_0, v_1, \dots, v_l, v_0)$  ist ein (**gerichteter**) Kreis, so heißt  $s'$  **Hamiltonkreis**.

## Beispiel (Hamiltonkreis in einem ungerichteten Graphen)

$$s = (a, f, b, g, c, h, d, p, l, o, k, n, j, m, i, e, a)$$



# Euler- und Hamiltonkreise

## Definition

Wir betrachten für einen gegebenen Graphen (bzw. Digraphen)  $G$  und zwei Knoten  $s$  und  $t$  folgende Entscheidungsprobleme:

- **Das Eulerlinienproblem (EULERPATH bzw. DIEULERPATH):**  
Hat  $G$  eine Eulerlinie von  $s$  nach  $t$ ?
- **Das Hamiltonpfadproblem (HAMPATH bzw. DIHAMPATH):**  
Hat  $G$  einen Hamiltonpfad von  $s$  nach  $t$ ?

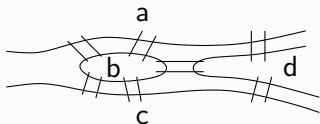
Zudem betrachten wir für einen gegebenen Graphen (bzw. Digraphen)  $G$  folgende Probleme:

- **Das Eulerkreisproblem (EULERCYCLE bzw. DIEULERCYCLE):**  
Hat  $G$  einen Eulerkreis?
- **Das Hamiltonkreisproblem (HAMCYCLE bzw. DIHAMCYCLE):**  
Hat  $G$  einen Hamiltonkreis?



# Das Königsberger Brückenproblem

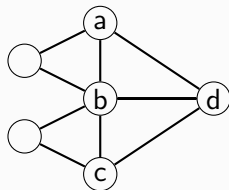
## Die 7 Königsberger Brücken



## Frage

Gibt es einen Spaziergang über alle 7 Brücken, bei dem keine Brücke mehrmals überquert wird und der zum Ausgangspunkt zurückführt?

Gelöst von Euler (1707 – 1783) durch Betrachtung des folgenden Graphen, der offenbar genau dann einen Eulerkreis hat, wenn die Antwort auf obige Frage „ja“ ist.



## Satz (Euler, 1736)

Ein zusammenhängender Graph  $G = (V, E)$  besitzt genau dann einen Eulerkreis, wenn all seine Knoten geraden Grad haben.

## Satz (Euler, 1736)

Ein zusammenhängender Graph  $G = (V, E)$  besitzt genau dann einen Eulerkreis, wenn all seine Knoten geraden Grad haben.

## Beweis

- Falls  $G$  einen Eulerkreis  $s$  besitzt, existiert zu jeder Kante, auf der  $s$  zu einem Knoten gelangt, eine weitere Kante, auf der  $s$  den Knoten wieder verlässt.
- Daher muss jeder Knoten geraden Grad haben.
- Ist umgekehrt  $G$  zusammenhängend und hat jeder Knoten geraden Grad, so können wir wie folgt einen Eulerkreis  $s$  konstruieren.

## Eulerkreise und -linien

- Ist umgekehrt  $G$  zusammenhängend und hat jeder Knoten geraden Grad, so können wir wie folgt einen Eulerkreis  $s$  konstruieren.

### Algorithmus zur Berechnung eines Eulerkreises in $G = (V, E)$

- 1 Wähle  $u \in V$  beliebig und initialisiere  $s$  zu  $s = (u)$
- 2 Wähle einen beliebigen Knoten  $u$  auf dem Weg  $s$ , der mit einer unmarkierten Kante verbunden ist.
- 3 Folge ausgehend von  $u$  den unmarkierten Kanten auf einem beliebigen Weg  $z$  solange wie möglich und markiere dabei jede durchlaufene Kante. (Da von jedem erreichten Knoten  $v \neq u$  ungerade viele markierte Kanten ausgehen, muss der Weg  $z$  zum Ausgangspunkt  $u$  zurückführen.)
- 4 Füge den Zyklus  $z$  an der Stelle  $u$  in  $s$  ein.
- 5 Wenn noch nicht alle Kanten markiert sind, gehe zu 2.
- 6 **Output:**  $s$



## Eulerkreise und -linien

### Satz (Euler, 1736)

- i) Ein zusammenhängender Graph  $G = (V, E)$  besitzt im Fall  $s \neq t$  genau dann eine Eulerlinie von  $s$  nach  $t$ , wenn  $s$  und  $t$  ungeraden und alle übrigen Knoten geraden Grad haben.
- ii) Ein stark zusammenhängender Digraph besitzt genau dann einen Eulerkreis, wenn für jeden Knoten  $u$  der Eingangs- und der Ausgangsgrad übereinstimmen.

### Beweis

- i) Da  $G$  im Fall  $s \neq t$  genau dann eine Eulerlinie von  $s$  nach  $t$  hat, wenn der Graph  $G' = (V \cup \{u_{neu}\}, E \cup \{\{t, u_{neu}\}, \{u_{neu}, s\}\})$  einen Eulerkreis hat, folgt dies aus dem vorigen Satz.
- ii) Dies folgt vollkommen analog zum ungerichteten Fall.

### Korollar

EULERCYCLE, EULERPATH, DIEULERCYCLE, DIEULERPATH  $\in$  P.

## Satz

HAMPATH, HAMCYCLE, DIHAMPATH und DIHAMCYCLE sind NP-vollständig.

## Bemerkung

Bevor wir den Satz beweisen, betrachten wir ein weiteres Problem, das mit dem Hamiltonkreisproblem eng verwandt ist und große praktische Bedeutung hat.

- Gegeben sind die Entfernungen  $d_{ij}$  zwischen  $n$  Städten  $i, j \in \{1, \dots, n\}$ .
- Gesucht ist eine Rundreise  $(i_1, \dots, i_n)$  mit minimaler Länge  $d_{i_1, i_2} + \dots + d_{i_{n-1}, i_n} + d_{i_n, i_1}$ , die jede Stadt genau einmal besucht.
- Die Entscheidungsvariante dieses Optimierungsproblems ist wie folgt definiert.

## Problem des Handlungsreisenden (TSP; *traveling salesman problem*)

**Gegeben:** Eine  $n \times n$  Matrix  $D = (d_{i,j}) \in \mathbb{N}^{n \times n}$  und eine Zahl  $k$ .

**Gefragt:** Existiert eine Permutation  $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ , so dass die Rundreise  $(\pi(1), \dots, \pi(n))$  die Länge  $\leq k$  hat?

## Satz

$$3\text{-SAT} \leq^P \text{DIHAMPATH} \leq^P \frac{\text{HAMPATH}}{\text{DIHAMCYCLE}} \leq^P \text{HAMCYCLE} \leq^P \text{TSP}.$$

## Reduktion von HAMCYCLE auf TSP

- Sei  $G = (V, E)$  ein Graph, wobei wir  $V = \{1, \dots, n\}$  annehmen.
- Dann lässt sich  $G$  in Polynomialzeit auf die TSP Instanz  $(D, n)$  mit  $D = (d_{i,j})$  und

$$d_{i,j} = \begin{cases} 1, & \text{falls } \{i, j\} \in E, \\ 2, & \text{sonst,} \end{cases}$$

transformieren.

- Diese Reduktion ist korrekt, da  $G$  genau dann einen Hamiltonkreis hat, wenn es in dem Distanzgraphen  $D$  eine Rundreise  $(\pi(1), \dots, \pi(n))$  der Länge  $L(\pi) \leq n$  gibt. □

## Hamiltonkreise, Hamiltonpfade und TSP

## Satz

$$3\text{-SAT} \leq^p \text{DIHAMPATH} \leq^p \frac{\text{HAMPATH}}{\text{DIHAMCYCLE}} \leq^p \text{HAMCYCLE} \leq^p \text{TSP}.$$

## Reduktion von DIHAMPATH auf DIHAMCYCLE

- Seien ein Digraph  $G = (V, E)$  und zwei Knoten  $s, t \in V$  gegeben.
- Wir transformieren  $G$  in den Digraphen  $G' = (V', E')$  mit

$$V' = V \cup \{u_{\text{neu}}\} \text{ und}$$

$$E' = E \cup \{(t, u_{\text{neu}}), (u_{\text{neu}}, s)\}.$$

- Offenbar ist  $G'$  in Polynomialzeit aus  $G$  berechenbar und besitzt genau dann einen Hamiltonkreis, wenn  $G$  einen  $s$ - $t$ -Hamiltonpfad besitzt.  $\square$

## Reduktion von HAMPATH auf HAMCYCLE

Analog.



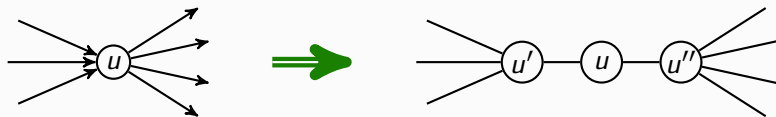
# Hamiltonkreise, Hamiltonpfade und TSP

## Satz

$$3\text{-SAT} \leq^p \text{DIHAMPATH} \leq^p \begin{matrix} \text{HAMPATH} \\ \text{DIHAMCYCLE} \end{matrix} \leq^p \text{HAMCYCLE} \leq^p \text{TSP}.$$

## Reduktion von DIHAMCYCLE auf HAMCYCLE

- Wir reduzieren einen Digraphen  $G$  auf einen Graphen  $G'$ , indem wir lokal für jeden Knoten  $u \in V$  die folgende Ersetzung durchführen:



- Dann ist klar, dass die Funktion  $G \mapsto G'$  in FP berechenbar ist, und  $G$  genau dann einen Hamiltonkreis enthält, wenn dies auf  $G'$  zutrifft.  $\square$

## Reduktion von DIHAMPATH auf HAMPATH

Analog.

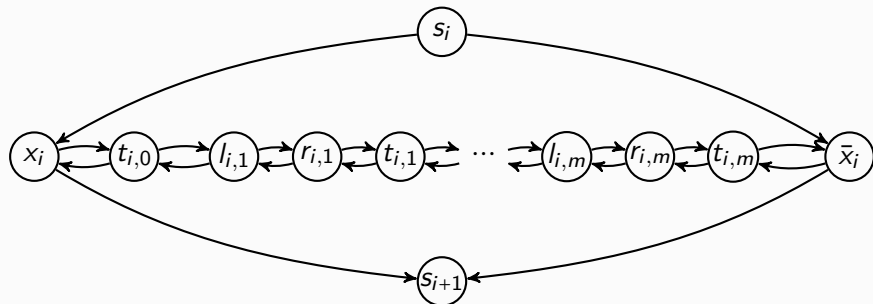
## Satz

$$3\text{-SAT} \leq^P \text{DIHAMPATH} \leq^P \begin{matrix} \text{HAMPATH,} \\ \text{DIHAMCYCLE} \end{matrix} \leq^P \text{HAMCYCLE} \leq^P \text{TSP}.$$

## Reduktion von 3-SAT auf DIHAMPATH

- Sei  $F = \{C_1, \dots, C_m\}$  mit  $C_j = \{l_{j1}, \dots, l_{jk_j}\}$  für  $j = 1, \dots, m$  eine 3-KNF-Formel über den Variablen  $x_1, \dots, x_n$ .
- Wir transformieren  $F$  in Polynomialzeit in einen gerichteten Graphen  $G_F = (V, E)$  mit zwei ausgezeichneten Knoten  $s$  und  $t$ , der genau dann einen hamiltonschen  $s$ - $t$ -Pfad besitzt, wenn  $F$  erfüllbar ist.
- Jede Klausel  $C_j$  repräsentieren wir durch einen Knoten  $c_j$  und jede Variable  $x_i$  repräsentieren wir durch folgenden Graphen  $X_i$ .

- Jede Klausel  $C_j$  repräsentieren wir durch einen Knoten  $c_j$  und jede Variable  $x_i$  repräsentieren wir durch folgenden Graphen  $X_i$ :



- Ein Pfad von  $s_1$  nach  $s_{n+1}$  kann ausgehend von  $s_i$  ( $i = 1, \dots, n$ ) entweder zuerst den Knoten  $x_i$  oder zuerst den Knoten  $\bar{x}_i$  besuchen.
- Daher können wir jedem  $s_1$ - $s_{n+1}$ -Pfad  $P$  eine Belegung  $b_P = b_1 \cdots b_n$  zuordnen mit  $b_i = 1$  gdw.  $P$  den Knoten  $x_i$  vor dem Knoten  $\bar{x}_i$  besucht.

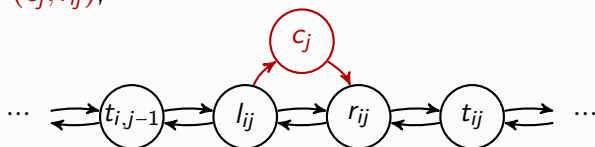
## Reduktion von 3-SAT auf DIHAMPATH

- Die Knotenmenge  $V$  von  $G_F$  ist also

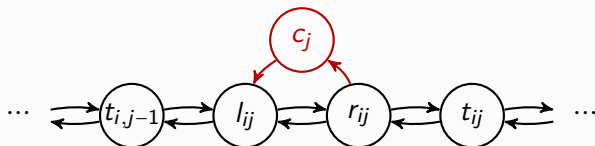
$$V = \{c_1, \dots, c_m\} \cup \{s_1, \dots, s_{n+1}\} \cup \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\} \cup \bigcup_{i=1}^n \{t_{i,0}, l_{i,1}, r_{i,1}, t_{i,1}, \dots, l_{i,m}, r_{i,m}, t_{i,m}\}.$$

- Dabei haben die Graphen  $X_{i-1}$  und  $X_i$  den Knoten  $s_i$  gemeinsam.
- Als Startknoten wählen wir  $s = s_1$  und als Zielknoten  $t = s_{n+1}$ .
- Jetzt fehlen nur noch die Verbindungskanten zwischen den Teilgraphen  $X_i$  und den Klauselknoten  $c_j$ .
- Diese Kanten sollen einem  $s$ - $t$ -Pfad  $P$  genau dann einen Abstecher nach  $c_j$  ermöglichen, wenn die Belegung  $b_P$  die Klausel  $C_j$  erfüllt.

- Für jedes Literal  $l \in C_j$  fügen wir zu  $E$  im Fall  $l = x_i$  die beiden Kanten  $(l_{ij}, c_j)$  und  $(c_j, r_{ij})$ ,



und im Fall  $l = \bar{x}_i$  die Kanten  $(r_{ij}, c_j)$  und  $(c_j, l_{ij})$  hinzu:



- Man beachte, dass ein  $s$ - $t$ -Pfad  $P$  über diese Kanten genau dann einen Abstecher zu  $c_j$  machen kann, wenn die Belegung  $b_P$  das Literal  $l$  wahr macht.

## Reduktion von 3-SAT auf DIHAMPATH

- Zunächst ist klar, dass die Reduktionsfunktion  $F \mapsto (G_F, s, t)$  in Polynomialzeit berechenbar ist.
- Es bleibt also zu zeigen, dass  $F$  genau dann erfüllbar ist, wenn in  $G_F$  ein Hamiltonpfad von  $s = s_1$  nach  $t = s_{n+1}$  existiert.
- Falls  $F(b) = 1$  ist, so erhalten wir einen Hamiltonpfad, indem wir in jeder Klausel  $C_j$  ein wahres Literal  $l = x_j$  bzw.  $l = \bar{x}_j$  auswählen und in den zu  $b$  gehörigen  $s$ - $t$ -Pfad  $P$  einen „Abstecher“ vom Knotenpaar  $l_{ij}, r_{ij}$  zum Klauselknoten  $c_j$  einbauen.
- Ist umgekehrt  $P$  ein  $s$ - $t$ -Hamiltonpfad in  $G_F$ , so müssen der Vorgänger- und Nachfolgerknoten jedes Klauselknotens  $c_j$  ein Paar  $l_{ij}, r_{ij}$  bilden, da  $P$  andernfalls nicht beide Pufferknoten  $t_{i,j-1}$  und  $t_{i,j}$  besuchen kann.
- Da aber  $P$  alle Klauselknoten besucht und ausgehend von dem Paar  $l_{ij}, r_{ij}$  nur dann ein Abstecher zu  $c_j$  möglich ist, wenn die Belegung  $b_P$  die Klausel  $C_j$  erfüllt, folgt  $F(b_P) = 1$ . □

## Beispiel

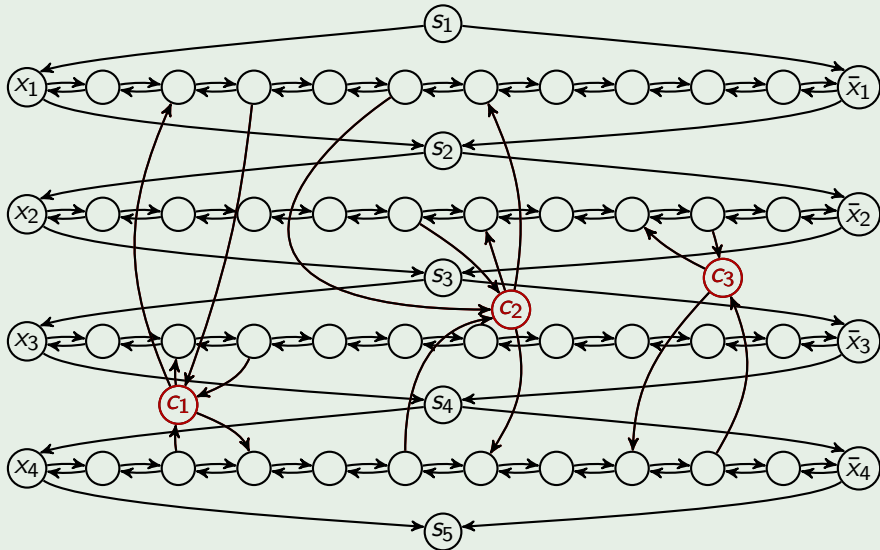
Die 3-SAT-Instanz

$$F = (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4).$$

lässt sich auf folgenden Digraphen  $G$  mit Startknoten  $s = s_1$  und Zielknoten  $t = s_5$  reduzieren:

## Reduktion von 3-SAT auf DIHAMPATH

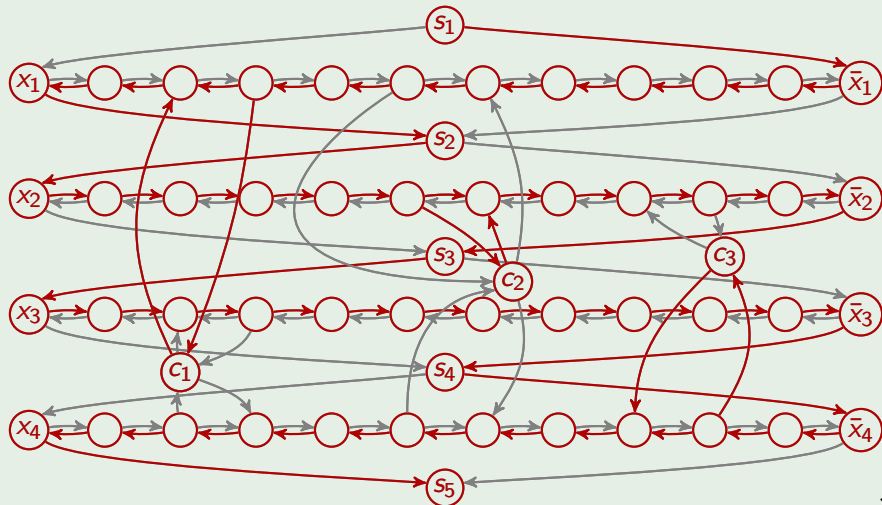
$$F = (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4).$$





# Reduktion von 3-SAT auf DIHAMPATH

Der Belegung  $b = 0110$  von  $F = (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4)$  entspricht beispielsweise folgender Hamiltonpfad:



## Das Rucksack-Problem

- Wie schwierig ist es, einen Rucksack der Größe  $w$  mit einer Auswahl aus  $k$  Gegenständen der Größe  $u_1, \dots, u_k$  möglichst voll zu packen?
- Dieses Optimierungsproblem lässt sich leicht auf folgendes Entscheidungsproblem reduzieren.

### RUCKSACK:

**Gegeben:** Eine Folge  $(u_1, \dots, u_k, v, w)$  von natürlichen Zahlen.

**Gefragt:** Ex. eine Auswahl  $S \subseteq \{1, \dots, k\}$  mit  $v \leq \sum_{i \in S} u_i \leq w$ ?

Beim SubsetSum-Problem möchte man dagegen nur wissen, ob der Rucksack randvoll gepackt werden kann.

### SUBSETSUM:

**Gegeben:** Eine Folge  $(u_1, \dots, u_k, w)$  von natürlichen Zahlen.

**Gefragt:** Ex. eine Auswahl  $S \subseteq \{1, \dots, k\}$  mit  $\sum_{i \in S} u_i = w$ ?

# Rucksack und SubsetSum sind NP-vollständig

## RUCKSACK:

**Gegeben:** Eine Folge  $(u_1, \dots, u_k, v, w)$  von natürlichen Zahlen.

**Gefragt:** Ex. eine Auswahl  $S \subseteq \{1, \dots, k\}$  mit  $v \leq \sum_{i \in S} u_i \leq w$ ?

## SUBSETSUM:

**Gegeben:** Eine Folge  $(u_1, \dots, u_k, w)$  von natürlichen Zahlen.

**Gefragt:** Ex. eine Auswahl  $S \subseteq \{1, \dots, k\}$  mit  $\sum_{i \in S} u_i = w$ ?

## Satz

RUCKSACK und SUBSETSUM sind NP-vollständig.

## Beweis

- Es ist leicht zu sehen, dass beide Probleme in NP enthalten sind.

# Rucksack und SubsetSum sind NP-vollständig

## Satz

RUCKSACK und SUBSETSUM sind NP-vollständig.

## Beweis

- Es ist leicht zu sehen, dass die beiden Probleme in NP enthalten sind.
- Zum Nachweis der NP-Härte zeigen wir die folgenden Reduktionen:

$$3\text{-SAT} \leq^P \text{SUBSETSUM} \leq^P \text{RUCKSACK}.$$

## Reduktion von SUBSETSUM auf RUCKSACK

Da SUBSETSUM einen Spezialfall des RUCKSACK-Problems darstellt, lässt es sich leicht darauf reduzieren:

$$(u_1, \dots, u_k, w) \mapsto (u_1, \dots, u_k, w, w).$$

□

# SubsetSum ist NP-vollständig

## Reduktion von 3-SAT auf SUBSETSUM

- Sei  $F = \{C_1, \dots, C_m\}$  mit  $C_j = \{l_{j1}, \dots, l_{jk_j}\}$  für  $j = 1, \dots, m$  eine 3-KNF-Formel über den Variablen  $x_1, \dots, x_n$ .
- Betrachte die Reduktionsfunktion

$$f : F \mapsto (u_1, \dots, u_n, u'_1, \dots, u'_n, v_1, \dots, v_m, v'_1, \dots, v'_m, w),$$

wobei  $u_i$  und  $u'_i$  die Dezimalzahlen

$$u_i = b_{i1} \dots b_{im} 0^{i-1} 10^{n-i-1} \quad \text{und} \quad u'_i = b'_{i1} \dots b'_{im} 0^{i-1} 10^{n-i-1}$$

mit

$$b_{ij} = \begin{cases} 1, & x_i \in C_j, \\ 0, & \text{sonst,} \end{cases} \quad \text{und} \quad b'_{ij} = \begin{cases} 1, & \bar{x}_i \in C_j, \\ 0, & \text{sonst,} \end{cases}$$

und  $v_j = v'_j = 0^{j-1} 10^{m-j-1} 0^n$  sind.

- $w$  setzen wir auf  $w = \underbrace{3 \dots 3}_{m\text{-mal}} \underbrace{1 \dots 1}_{n\text{-mal}}$ .

# SubsetSum ist NP-vollständig

## Beispiel

- Betrachte die 3-KNF Formel

$$F = (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4).$$

- Die zu  $F$  gehörige SUBSETSUM-Instanz  $f(F)$  ist

$$(u_1, u_2, u_3, u_4, u'_1, u'_2, u'_3, u'_4, v_1, v_2, v_3, v'_1, v'_2, v'_3, w)$$

mit

$$\begin{array}{llll} u_1 = 0101000, & u_2 = 0100100, & u_3 = 0000010, & u_4 = 1100001, \\ u'_1 = 1001000, & u'_2 = 0010100, & u'_3 = 1000010, & u'_4 = 0010001, \\ v_1 = 1000000, & v_2 = 0100000, & v_3 = 0010000, & \\ v'_1 = 1000000, & v'_2 = 0100000, & v'_3 = 0010000, & \end{array}$$

sowie  $w = 3331111$ .

- Der erfüllenden Belegung  $a = 0100$  entspricht dann die Auswahl  $(u'_1, u_2, u'_3, u'_4, v_1, v_2, v'_2, v_3, v'_3)$ .



# SubsetSum ist NP-vollständig

Beweis von  $F \in 3\text{-SAT} \Rightarrow f(F) \in \text{SUBSETSUM}$

- Sei  $a = a_1 \cdots a_n$  eine erfüllende Belegung für  $F$ .
- Da  $a$  in jeder Klausel mindestens ein und höchstens drei Literale wahr macht, hat die Zahl

$$\sum_{a_j=1} u_j + \sum_{a_j=0} u'_j$$

eine Dezimaldarstellung der Form  $b_1 \cdots b_m 1 \cdots 1$  mit  $1 \leq b_j \leq 3$  für  $j = 1, \dots, m$ .

- Durch Addition von

$$\sum_{b_j \leq 2} v_j + \sum_{b_j = 1} v'_j$$

erhalten wir  $w$ .

# SubsetSum ist NP-vollständig

Beweis von  $f(F) \in \text{SUBSETSUM} \Rightarrow F \in \text{3-SAT}$

- Sei  $S = P \cup N \cup I \cup J$  eine Auswahlmenge für  $f(F)$  mit

$$\sum_{i \in P} u_i + \sum_{i \in N} u'_i + \sum_{j \in I} v_j + \sum_{j \in J} v'_j = \underbrace{3 \cdots 3}_{m\text{-mal}} \underbrace{1 \cdots 1}_{n\text{-mal}}.$$

- Da die Teilsumme  $\sum_{j \in I} v_j + \sum_{j \in J} v'_j$  die Form  $c_1 \cdots c_m 0 \cdots 0$  mit  $c_j \leq 2$  hat, muss die Teilsumme  $\sum_{i \in P} u_i + \sum_{i \in N} u'_i$  die Form  $b_1 \cdots b_m 1 \cdots 1$  mit  $b_j \geq 1$  haben.
- Da keine Überträge auftreten, muss also  $P = \{1, \dots, n\} - N$  sein, und jede Klausel  $C_j$  muss mindestens ein Literal aus der Menge  $\{x_i \mid i \in P\} \cup \{\bar{x}_i \mid i \in N\}$  enthalten.
- Folglich erfüllt folgende Belegung  $a_1 \cdots a_n$  die Formel  $F$ :

$$a_i = \begin{cases} 1, & i \in P, \\ 0, & i \in N. \end{cases}$$

- Damit haben wir die Korrektheit von  $f$  gezeigt. □



In vielen Anwendungen tritt das Problem auf, eine ganzzahlige Lösung für ein System linearer Ungleichungen zu finden.

## Ganzzahlige Programmierung (IP; *integer programming*)

Gegeben: Eine ganzzahlige  $m \times n$  Matrix  $A = (a_{ij}) \in \mathbb{Z}^{m \times n}$  und ein ganzzahliger Vektor  $\mathbf{b} \in \mathbb{Z}^m$ .

Gefragt: Existiert ein ganzzahliger Vektor  $\mathbf{x} \in \mathbb{Z}^n$  mit

$$A\mathbf{x} \geq \mathbf{b},$$

wobei  $\geq$  komponentenweise zu verstehen ist.

## Satz

IP ist NP-hart.

## Reduktion von 3-SAT auf IP

- Sei  $F = \{C_1, \dots, C_m\}$  mit  $C_j = \{l_{j1}, \dots, l_{jk_j}\}$  für  $j = 1, \dots, m$  eine 3-KNF-Formel über den Variablen  $x_1, \dots, x_n$ .
- Wir transformieren  $F$  in ein Ungleichungssystem  $A\mathbf{x} \geq \mathbf{b}$  für den Lösungsvektor  $\mathbf{x} = (x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n)$ , das

- für  $i = 1, \dots, n$  die vier Ungleichungen

$$x_i + \bar{x}_i \geq 1, \quad -x_i - \bar{x}_i \geq -1, \quad x_i \geq 0, \quad \bar{x}_i \geq 0 \quad (*)$$

- und für jede Klausel  $C_j = \{l_{j1}, \dots, l_{jk_j}\}$  folgende Ungleichung enthält:

$$l_{j1} + \dots + l_{jk_j} \geq 1. \quad (**)$$

- Die Ungleichungen (\*) sind für ganzzahlige  $x_i, \bar{x}_i$  genau dann erfüllt, wenn  $x_i$  den Wert 0 und  $\bar{x}_i$  den Wert 1 hat oder umgekehrt.
- Die Klauselungleichungen (\*\*) stellen sicher, dass mindestens ein Literal in jeder Klausel  $C_j$  wahr wird.
- Nun ist leicht zu sehen, dass jede Lösung  $\mathbf{x}$  von  $A\mathbf{x} \geq \mathbf{b}$  einer erfüllenden Belegung von  $F$  entspricht und umgekehrt. □

# (Ganzzahlige) lineare Programmierung

## Bemerkungen zu IP

- Es ist nicht leicht zu sehen, dass IP in NP entscheidbar ist.
- Ein nichtdeterministischer Algorithmus kann zwar eine Lösung raten, aber a priori ist nicht klar, ob eine Lösung  $\mathbf{x}$  ex., deren Binärkodierung polynomiell in der Länge der Eingabe  $(A, \mathbf{b})$  ist.
- Mit Methoden der linearen Algebra lässt sich jedoch zeigen, dass jede lösbare IP-Instanz  $(A, \mathbf{b})$  auch eine Lösung  $\mathbf{x}$  hat, deren Kodierung polynomiell in der Länge von  $(A, \mathbf{b})$  ist.
- Wenn wir nicht verlangen, dass die Lösung  $\mathbf{x}$  der IP-Instanz ganzzahlig ist, dann spricht man von einem **linearen Programm**.
- Für **LP** (Lineare Programmierung) gibt es Polynomialzeitalgorithmen (von Khachiyan 1979 und von Karmarkar 1984).

# SAT als Optimierungsproblem

- In manchen Anwendungen ist es wichtig zu wissen, wieviele Klauseln einer KNF-Formel  $F$  maximal erfüllbar sind.
- Die Schwierigkeit dieses Optimierungsproblems lässt sich durch folgendes Entscheidungsproblem charakterisieren.

## MAX- $k$ -SAT:

**Gegeben:** Eine Formel  $F$  in  $k$ -KNF und eine Zahl  $l$ .

**Gefragt:** Existiert eine Belegung  $a$ , die mindestens  $l$  Klauseln in  $F$  erfüllt?

Hierbei können Klauseln in  $F$  auch mehrfach vorkommen.

## Satz

- 1 MAX-1-SAT  $\in$  P.
- 2 MAX-2-SAT  $\in$  NPC.

# SAT als Optimierungsproblem

## Reduktion von 3-SAT auf MAX-2-SAT

- Für eine Dreierklausel  $C = \{l_1, l_2, l_3\}$ , in der die Variable  $v$  nicht vorkommt, sei  $G(l_1, l_2, l_3, v)$  die 2-KNF Formel, die aus folgenden 10 Klauseln besteht:

$$\{l_1\}, \{l_2\}, \{l_3\}, \{v\}, \{\bar{l}_1, \bar{l}_2\}, \{\bar{l}_2, \bar{l}_3\}, \{\bar{l}_1, \bar{l}_3\}, \{l_1, \bar{v}\}, \{l_2, \bar{v}\}, \{l_3, \bar{v}\}$$

- Die folgenden 3 Eigenschaften von  $G$  sind leicht zu verifizieren:
  - Keine Belegung von  $G$  erfüllt mehr als 7 Klauseln von  $G$ .
  - Jede Belegung  $a$  von  $C$  mit  $C(a) = 1$  ist zu einer Belegung  $a'$  von  $G$  erweiterbar, die 7 Klauseln von  $G$  erfüllt.
  - Keine Belegung  $a$  von  $C$  mit  $C(a) = 0$  ist zu einer Belegung  $a'$  von  $G$  erweiterbar, die 7 Klauseln von  $G$  erfüllt.
- Sei  $F$  eine 3-KNF-Formel über  $x_1, \dots, x_n$  mit  $m$  Klauseln.
- Wir nehmen an, dass  $C_j = \{l_{j1}, l_{j2}, l_{j3}\}$ ,  $j = 1, \dots, k$ , die Dreier- und  $C_{k+1}, \dots, C_m$  die Einer- und Zweierklauseln von  $F$  sind.

## Reduktion von 3-SAT auf MAX-2-SAT

- Wir nehmen an, dass  $C_j = \{l_{j1}, l_{j2}, l_{j3}\}$ ,  $j = 1, \dots, k$ , die Dreier- und  $C_{k+1}, \dots, C_m$  die Einer- und Zweierklauseln von  $F$  sind.
- Betrachte die 2-KNF Formel  $F'$  mit  $10k + (m - k)$  Klauseln, die wie folgt aus  $F$  entsteht:
  - Ersetze jede Dreierklausel  $C_j = \{l_{j1}, l_{j2}, l_{j3}\}$  in  $F$  durch die 10 Klauseln der 2-KNF-Formel  $G_j = G(l_{j1}, l_{j2}, l_{j3}, v_j)$ .
- Dann gilt
$$F \in 3\text{-SAT} \Leftrightarrow (F', m + 6k) \in \text{MAX-2-SAT}.$$
- Die Vorwärtsrichtung ergibt sich unmittelbar aus der Tatsache, dass jede erfüllende Belegung  $a$  für  $F$  zu einer Belegung  $a'$  für  $F'$  erweiterbar ist, die  $7k + m - k = m + 6k$  Klauseln von  $F'$  erfüllt.

## Reduktion von 3-SAT auf MAX-2-SAT

- Dann gilt

$$F \in 3\text{-SAT} \Leftrightarrow (F', m + 6k) \in \text{MAX-2-SAT}.$$

- Die Vorwärtsrichtung ergibt sich unmittelbar aus der Tatsache, dass jede erfüllende Belegung  $a$  für  $F$  zu einer Belegung  $a'$  für  $F'$  erweiterbar ist, die  $7k + m - k = m + 6k$  Klauseln von  $F'$  erfüllt.
- Für die Rückwärtsrichtung sei  $a$  eine Belegung, die mindestens  $m + 6k$  Klauseln von  $F'$  erfüllt.
- Da in jeder 10er-Gruppe  $G_j$ ,  $j = 1, \dots, k$ , maximal 7 Klauseln erfüllbar sind, muss  $a$  in jeder 10er-Gruppe genau 7 Klauseln und zudem alle Klauseln  $C_j$  für  $j = k + 1, \dots, m$  erfüllen.
- Dies ist aber nur möglich, wenn  $a$  alle Klauseln  $C_j$  von  $F$  erfüllt.
- Zudem ist leicht zu sehen, dass  $g : F \mapsto (F', m + 6k)$  in FP berechenbar ist, woraus  $3\text{-SAT} \leq^P \text{MAX-2-SAT}$  folgt. □

### Das Heiratsproblem

- Sei  $V$  eine Gruppe von heiratswilligen Personen.
- Wir verbinden  $u, w \in V$  durch eine Kante, falls aus Sicht von  $u$  und  $w$  die Möglichkeit einer Heirat zwischen  $u$  und  $w$  besteht.
- Sind Viehlen ausgeschlossen, so lässt sich jedes mögliche Heiratsarrangement durch ein Matching beschreiben.



# Matchings und der Heiratssatz

## Definition

Sei  $G = (V, E)$  ein Graph (d.h.  $E \subseteq \binom{V}{2}$ ) oder allgemeiner ein  $d$ -uniformer Hypergraph (d.h.  $E \subseteq \binom{V}{d}$ ,  $d \geq 3$ ).

- Zwei Kanten  $e, e' \in E$  heißen **unabhängig**, falls  $e \cap e' = \emptyset$  ist.
- Eine Kantenmenge  $M \subseteq E$  heißt **Matching** in  $G$ , falls die Kanten in  $M$  paarweise unabhängig sind.
- Die **Matchingzahl** von  $G$  ist

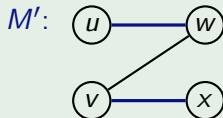
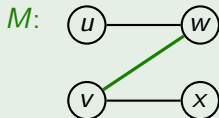
$$\mu(G) = \max\{\|M\| \mid M \text{ ist ein Matching in } G\}.$$

- Ein Matching  $M$  der Größe  $\|M\| = \mu(G)$  heißt **optimal**.
- $M$  heißt **gesättigt**, falls  $M \cup \{e\}$  für keine Kante  $e \in E \setminus M$  ein Matching ist.
- $M$  heißt **perfekt**, falls  $\underbrace{\|\{v \in V \mid \exists e \in M : v \in e\}\|}_{=: V(M)} > n - d$  ist.

# Matchings und der Heiratssatz

## Beispiel

- Jedes perfekte Matching ist optimal, aber ein optimales Matching muss nicht perfekt sein.
- Jedes optimale Matching ist gesättigt, aber ein gesättigtes Matching muss nicht optimal sein:



- Das Matching  $M = \{\{v, w\}\}$  ist gesättigt, da es sich nicht zu einem größeren Matching erweitern lässt.
- $M$  ist jedoch nicht optimal, da  $M' = \{\{u, w\}, \{v, x\}\}$  größer ist.
- Die Greedy-Methode, ausgehend von  $M = \emptyset$  solange Kanten zu  $M$  hinzuzufügen, bis sich  $M$  nicht mehr zu einem größeren Matching erweitern lässt, funktioniert also i.a. nicht.

## Matchings und der Heiratssatz

Falls beim Heiratsproblem Homoeen ausgeschlossen sind, ist der resultierende Graph  $G = (V, E)$  bipartit.

### Definition

- Sei  $G = (V, E)$  ein Graph und seien  $U, W \subseteq V$ . Dann bezeichne

$$E(U, W) = \left\{ \{u, w\} \in E \mid u \in U, w \in W \right\}$$

die Menge aller Kanten zwischen  $U$  und  $W$  in  $G$ .

- $G$  heißt **bipartit**, falls sich  $V$  in zwei Mengen  $U$  und  $W$  mit  $E(U, W) = E$  zerlegen lässt.
- In diesem Fall notieren wir  $G$  auch in der Form  $G = (U, W, E)$ .

# Matchingprobleme

## Bipartites Matching (BIMATCHING)

Gegeben: Ein bipartiter Graph  $G$  und eine Zahl  $k \geq 1$ .

Gefragt: Hat  $G$  ein Matching  $M$  der Größe  $\|M\| \geq k$ ?

## Matching (MATCHING)

Gegeben: Ein Graph  $G$  und eine Zahl  $k \geq 1$ .

Gefragt: Hat  $G$  ein Matching  $M$  der Größe  $\|M\| \geq k$ ?

## Perfektes Matching (PERFECTMATCHING)

Gegeben: Ein Graph  $G$ .

Gefragt: Hat  $G$  ein perfektes Matching?

# Matchingprobleme

## $d$ -dimensionales Matching ( $d$ -MATCHING)

Gegeben: Ein  $d$ -uniformer Hypergraph  $G$  und eine Zahl  $k \geq 1$ .

Gefragt: Hat  $G$  ein Matching  $M$  der Größe  $\|M\| \geq k$ ?

## $d$ -dimensionales perfektes Matching ( $d$ -PERFECTMATCHING)

Gegeben: Ein  $d$ -uniformer Hypergraph  $G$ .

Gefragt: Hat  $G$  ein perfektes Matching?

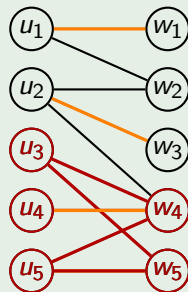
## Satz

- BIMATCHING, MATCHING und PERFECTMATCHING sind in P entscheidbar.
- $d$ -MATCHING und  $d$ -PERFECTMATCHING sind für  $d \geq 3$  NP-vollständig.

# Bipartites Matching und der Heiratssatz

## Beispiel

- Der bipartite Graph  $G$  enthält ein Matching  $M$  der Größe 4.
- Die Nachbarschaft von  $A = \{u_3, u_4, u_5\}$  ist  $N(A) = \{w_4, w_5\}$ .
- Daher kann jedes Matching höchstens 2 der 3 Knoten in  $A$  und somit höchstens 4 der 5 Knoten in  $U$  überdecken.
- Dies zeigt, dass  $M$  optimal ist.



$$G = (U, W, E)$$



- Sei  $G = (U, W, E)$  ein bipartiter Graph und sei  $A \subseteq U$ .
- Da ein Matching  $M$  höchstens  $\|N(A)\|$  Knoten in  $A$  überdecken kann, gilt

$$\|U\| - \|M\| \geq \|A\| - \|N(A)\|.$$

- Folglich ist  $\|U\| - \mu(G) \geq \max_{A \subseteq U} (\|A\| - \|N(A)\|)$ .

**Frage:** Ist diese Schranke in jedem bipartiten Graphen scharf?

Anders ausgedrückt: Lässt sich die Optimalität von  $M$  immer durch Angabe einer Menge  $A \subseteq U$  mit  $\|U\| - \|M\| = \|A\| - \|N(A)\|$  beweisen?

**Satz (Heiratssatz von Hall)**

Sei  $G = (U, W, E)$  ein bipartiter Graph. Dann gibt es in  $G$  genau dann ein Matching  $M$  der Größe  $\|M\| = \|U\|$ , wenn für alle Teilmengen  $A \subseteq U$  gilt:

$$\|N(A)\| \geq \|A\|.$$

Allgemeiner gilt

$$\mu(G) = \|U\| - \max_{A \subseteq U} (\|A\| - \|N(A)\|).$$



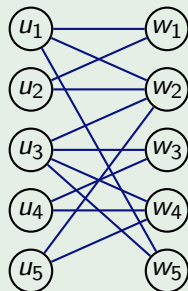
## Beweis des Heiratssatzes

- Sei  $D = \max_{A \subseteq U} (\|A\| - \|N(A)\|)$ .
- Wir konstruieren zu jedem Matching  $M$  der Größe  $\|M\| < \|U\| - D$  ein Matching  $M'$  mit  $\|M'\| > \|M\|$ .
- Hierzu betrachten wir den Digraphen  $G_M$ , der aus  $G$  dadurch entsteht, dass wir alle Kanten in  $M$  von  $W$  nach  $U$  und alle Kanten in  $E - M$  von  $U$  nach  $W$  orientieren.
- Für eine Menge  $A \subseteq V$  sei  $A^+ = \{u \in A \mid \exists e \in M : u \in e\}$  und  $A^- = A - A^+$ .
- Angenommen, es ex. in  $G_M$  ein Pfad von  $U^-$  nach  $W^-$ :  

$$P : u_0 \rightarrow w_0 \rightarrow u_1 \rightarrow w_1 \rightarrow \dots \rightarrow u_{k-1} \rightarrow w_{k-1} \rightarrow u_k \rightarrow w_k$$
mit  $u_0 \in U^-$  und  $w_k \in W^-$ .
- Dann hat  $P$  ungerade Länge  $2k + 1$ ,  $k \geq 0$ , und wir können  $M$  zu  $M'$  vergrößern, indem wir die  $k$  Kanten  $\{w_i, u_{i+1}\} \in M$ ,  $i = 0, \dots, k - 1$ , durch die  $k + 1$  Kanten  $\{u_i, w_i\} \in E - M$ ,  $i = 0, \dots, k$ , ersetzen:

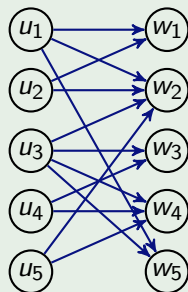
$$M' = \left( M \setminus \left\{ \{w_i, u_{i+1}\} \mid 0 \leq i < k \right\} \right) \cup \left\{ \{u_i, w_i\} \mid 0 \leq i \leq k \right\}.$$

## Beispiel



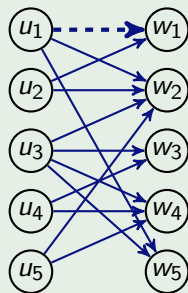
$$G = (U, W, E)$$

## Beispiel



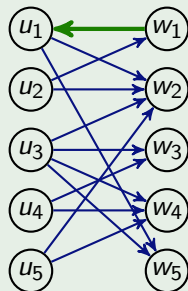
$G_M$  für  $M = \emptyset$

## Beispiel



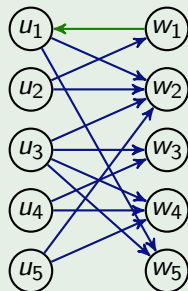
$G_M$  für  $M = \emptyset$

## Beispiel



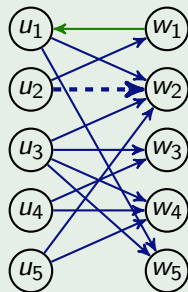
$G_M$  für  $M = \{\{u_1, w_1\}\}$

## Beispiel



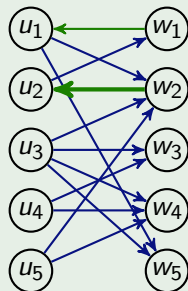
$G_M$  für  $M = \{\{u_1, w_1\}\}$

## Beispiel



$G_M$  für  $M = \{\{u_1, w_1\}\}$

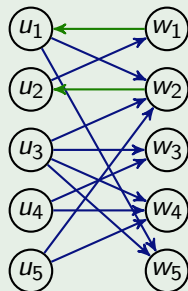
## Beispiel



$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}\}$

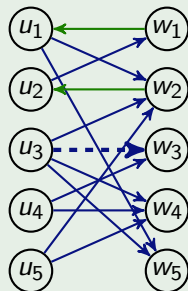


## Beispiel



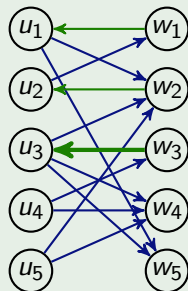
$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}\}$

## Beispiel



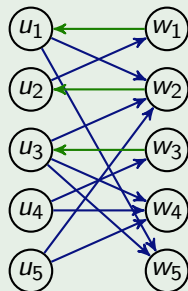
$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}\}$

## Beispiel



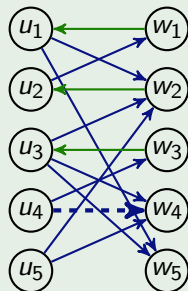
$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_3\}\}$

## Beispiel



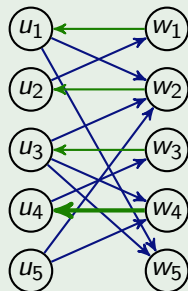
$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_3\}\}$

## Beispiel



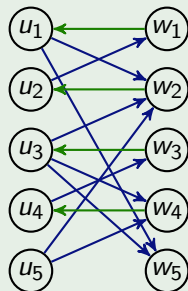
$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_3\}\}$

## Beispiel



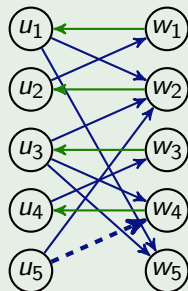
$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_3\}, \{u_4, w_4\}\}$

## Beispiel



$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_3\}, \{u_4, w_4\}\}$

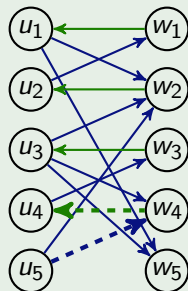
## Beispiel



$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_3\}, \{u_4, w_4\}\}$

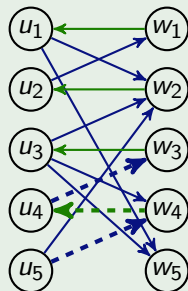


## Beispiel



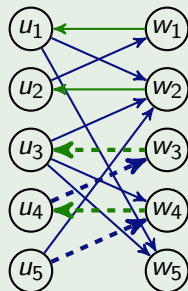
$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_3\}, \{u_4, w_4\}\}$

## Beispiel



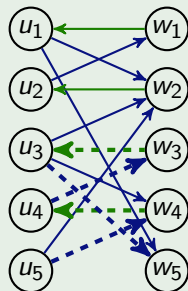
$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_3\}, \{u_4, w_4\}\}$

## Beispiel



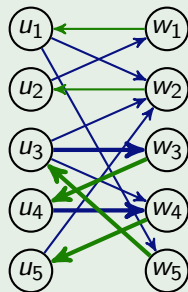
$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_3\}, \{u_4, w_4\}\}$

## Beispiel



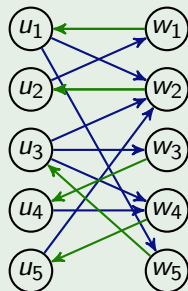
$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_3\}, \{u_4, w_4\}\}$

## Beispiel



$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_5\}, \{u_4, w_3\}, \{u_5, w_4\}\}$

## Beispiel



$G_M$  für  $M = \{\{u_1, w_1\}, \{u_2, w_2\}, \{u_3, w_5\}, \{u_4, w_3\}, \{u_5, w_4\}\}$

## Beweis des Heiratssatzes

- Noch zu zeigen:  $\|U^-\| \leq D$ , falls  $G_M$  keinen Pfad von  $U^-$  nach  $W^-$  hat.
- Sei  $X$  die Menge aller in  $G_M$  von  $U^-$  aus erreichbaren Knoten.
- Da  $U^-$  in der Menge  $A = X \cap U$  enthalten ist, ist  $A^- = U^-$ .
- Wegen

$$\|U^-\| = \|A^-\| = \|A\| - \|A^+\| \quad \text{und} \quad \|A\| - \|N(A)\| \leq D$$

reicht es zu zeigen, dass  $\|A^+\| \geq \|N(A)\|$  ist.

- Hierzu zeigen wir, dass die Abbildung  $f : A^+ \rightarrow N(A)$ , die jedem Knoten  $u \in A^+$  seinen Matchingpartner zuordnet, eine Surjektion ist.
- Sei also  $w \in N(A)$  und sei  $u' \in A$  mit  $\{u', w\} \in E$ .
- Falls  $\{u', w\} \in M$  ist, folgt sofort  $u' \in A^+$  und  $w = f(u')$ .
- Andernfalls ist mit  $u'$  auch  $w$  in  $X$  und wegen  $X \cap W^- = \emptyset$  in  $W^+$ .
- Folglich existiert ein  $u \in U^+$  mit  $\{u, w\} \in M$ .
- $u$  ist sogar in  $A^+$ , da  $u$  in  $G_M$  von  $w$  aus erreichbar und  $w \in X$  ist.  $\square$

## Korollar (zum Beweis des Heiratssatzes)

In bipartiten Graphen ist ein optimales Matching in Polynomialzeit berechenbar, d.h. `BIMATCHING`  $\in$  P.

## Algorithmus zur Bestimmung eines optimalen Matchings

- 1 **Input:** Ein bipartiter Graph  $G = (U, W, E)$
- 2  $M := \emptyset$
- 3 **while**  $\exists$  Pfad  $P$  von  $U^-$  nach  $W^-$  in  $G_M$  **do**
- 4     sei  $P : u_0 \rightarrow w_0 \rightarrow u_1 \rightarrow \dots \rightarrow w_{k-1} \rightarrow u_k \rightarrow w_k$
- 5      $M := \left( M \setminus \left\{ \{w_i, u_{i+1}\} \mid 0 \leq i < k \right\} \right) \cup \left\{ \{u_i, w_i\} \mid 0 \leq i \leq k \right\}$
- 6 **Output:**  $M$

**Frage:** Wie lässt sich  $P$  effizient berechnen?



- Wir haben eine Reihe von Optimierungsproblemen kennengelernt, für die sich in Polynomialzeit keine optimalen Lösungen berechnen lassen (außer wenn  $NP = P$  ist):
  - MAX- $k$ -SAT,  $k \geq 2$ ,
  - MAXCLIQUE,
  - MAXIMUM INDEPENDENT SET (MAXIS),
  - MINIMUM VERTEX COVER (MINVC),
  - MINCOLORING,
  - MINIMUM TRAVELING SALESMAN PROBLEM (MINTSP),
  - MAXRUCKSACK.
- Da diese Probleme in der Praxis dennoch gelöst werden müssen, stellt sich die Frage, wie nahe man der optimalen Lösung in Polynomialzeit kommen kann.
- In vielen Anwendungen ist eine Lösung, die maximal 1% vom Optimum abweicht, durchaus zufriedenstellend.

# NP-Optimierungsprobleme

Bei allen hier betrachteten Optimierungsproblemen handelt es sich um so genannte NP-Optimierungsprobleme. Diese sind wie folgt definiert.

## Definition

- Ein **NP-Optimierungsproblem**  $\pi$  besteht aus
  - einer Funktion  $F$ , die jeder Instanz  $x$  eine nichtleere Menge  $F(x)$  von **zulässigen Lösungen**  $y$  zuordnet, und
  - einer **Kostenfunktion**  $c$ , die jeder Lösung  $y \in F(x)$  einen **Wert**  $c(y) \in \mathbb{N}^+$  zuweist und in FP berechenbar ist.
- Dabei müssen die Lösungen  $y \in F(x)$  eine polynomiell beschränkte Länge  $|y| = |x|^{O(1)}$  haben und die Sprache  $\{x\#y \mid y \in F(x)\}$  muss in P entscheidbar sein.
- Eine Lösung  $y_{opt} \in F(x)$  heißt **optimal**, falls

$$c(y_{opt}) = \text{opt}\{c(y) \mid y \in F(x)\}$$

ist, wobei **opt** bei einem **Minimierungsproblem** der min-Operator und bei einem **Maximierungsproblem** der max-Operator ist.

- Beim Optimierungsproblem **MINIMUM VERTEX COVER** wird für einen gegebenen Graphen  $G = (V, E)$  eine möglichst kleine Menge  $U$  von Knoten gesucht, die alle Kanten in  $G$  überdecken.

## MINIMUM VERTEX COVER (MINVC):

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

**Lösungen:** alle Kantenüberdeckungen  $U$  in  $G$ ,

**Kosten:**  $\|U\|$ .

- Formal lässt sich das Optimierungsproblem **MINVC** also durch die Funktion

$$F(G) = \{U \subseteq V \mid \forall e \in E : e \cap U \neq \emptyset\}$$

und die Kostenfunktion  $c(U) = \|U\|$  beschreiben.

# Minimum Vertex Cover

Betrachte folgenden Algorithmus:

## Algorithmus `naiv-Finde-VC(G)`

```

1   Input: Graph  $G = (V, E)$ 
2    $U := \emptyset$ 
3   while  $E \neq \emptyset$  do
4       wähle einen Knoten  $u$  mit maximalem Grad in  $G$ 
5        $U := U \cup \{u\}$ 
6        $G := G - \{u\}$  (d.h.  $V := V \setminus \{u\}$  und  $E := \{e \in E \mid u \notin e\}$ )
7   Output:  $U$ 
  
```

- Es ist klar, dass der Algorithmus eine Kantenüberdeckung berechnet.
- Bezeichne  $OPT(G)$  die minimale Größe und  $naiv-Finde-VC(G)$  die Größe der durch den Algorithmus berechneten Kantenüberdeckung.

## Frage

Wie stark kann  $naiv-Finde-VC(G)$  von  $OPT(G)$  abweichen?

# Ein Maß für die Approximationsgüte

## Definition

- $A$  heißt **Approximationsalgorithmus** für ein NP-Optimierungsproblem, falls  $A$  für jede Instanz  $x$  in Polynomialzeit eine zulässige Lösung  $y \in F(x)$  ausgibt.
- Der **relative Gütequotient** (kurz **Güte**) von  $A$  bei Instanz  $x$  ist

$$Q_A(x) = \frac{|OPT(x) - A(x)|}{\max\{OPT(x), A(x)\}}.$$

- Die (**worst-case**) **Güte** von  $A$  ist

$$Q_A = \sup\{Q_A(x) \mid x \text{ ist eine Problem Instanz}\}.$$

## Bemerkung

- $Q_A(x)$  nimmt nur Werte im halboffenen Intervall  $[0, 1)$  an.
- Dabei bedeutet  $Q_A(x) = 0$ , dass  $A(x) = OPT(x)$  ist.
- Im Gegensatz zu  $Q_A(x)$  kann  $Q_A$  auch den Wert 1 annehmen.

## Minimum Vertex Cover

**Algorithmus** `naiv-Finde-VC(G)`

```

1  Input: Graph  $G = (V, E)$ 
2   $U := \emptyset$ 
3  while  $E \neq \emptyset$  do
4  wahle einen Knoten  $u$  mit maximalem Grad in  $G$ 
5   $U := U \cup \{u\}$ 
6   $G := G - \{u\}$  (d.h.  $V := V \setminus \{u\}$  und  $E := \{e \in E \mid u \notin e\}$ )
7  Output:  $U$ 

```

- Es gibt eine Folge von Graphen  $G_n$ ,  $n \geq 1$ , mit  $OPT(G_n) = n$  und  $\text{naiv-Finde-VC}(G_n) \geq n \lfloor \log_2 n \rfloor / 2$ .
- Folglich ist

$$Q_{\text{naiv-Finde-VC}}(G_n) = 1 - \frac{OPT(G_n)}{\text{naiv-Finde-VC}(G_n)} \geq 1 - 2 / \lfloor \log_2 n \rfloor$$

und somit  $Q_{\text{naiv-Finde-VC}} = 1$ .

# Minimum Vertex Cover

- Betrachte beispielsweise die bipartiten Graphen  $G_n = (U, W, E)$ ,  $n \geq 1$ , mit

$$U = \{u_0, \dots, u_{n-1}\},$$

$$W = \{v_j^i \mid i = 1, \dots, n, j = 0, \dots, \lfloor n/i \rfloor - 1\} \text{ und}$$

$$E = \left\{ \{u_k, v_j^i\} \mid k \text{ div } i = j \right\}.$$

- $G_n$  hat eine optimale Kantenüberdeckung  $U$  der Größe  $n$ .
- Es ist aber möglich, dass naive-Finde-VC die Menge  $W$  ausgibt.
- Diese hat für  $k = \lfloor \log_2 n \rfloor$  die Größe

$$\begin{aligned} \|W\| &= n + \lfloor n/2 \rfloor + \lfloor n/3 \rfloor + \dots + \lfloor n/n \rfloor \\ &\geq n/2 + n/3 + \dots + n/n \\ &\geq n/2 + \underbrace{n/3 + n/4 + n/5 + \dots + n/8}_{\geq n/2} + \dots + \underbrace{n/(2^{k-1} + 1) + \dots + n/2^k}_{\geq n/2} \\ &\geq kn/2. \end{aligned}$$