

Berechenbarkeit und Komplexitätstheorie

Mitschrift von Thomas Battermann

3. Semester

Inhaltsverzeichnis

1	Turingmaschinen	1
1.1	Erweiterungen	2
1.1.1	Mehrband-Turingmaschine	2
1.1.2	Nichtdeterministische Turingmaschine (NTM)	3
1.1.3	Nichtdeterministische Mehrband Turingmaschine	3
1.2	Kodierung von Turingmaschinen	5
1.3	Das Halteproblem	5
1.4	Die Universalsprache	5
1.5	Diagonalsprache	7
1.6	Das spezielle Halteproblem	8
1.6.1	Satz von Rice	9
1.6.2	Post'sches Korrespondenz Problem (PCP)	9
2	Eulerkreise	10
3	Hamilton Kreise	10
4	Erfüllbarkeit logischer Formeln, SAT	11
5	Einteilung in Klassen	12
6	Traveling Salesman (TSP)	14
7	k-Färbbarkeit	18
8	Unabhängige Knotenmenge (Independent Set (IS))	20
9	Clique	20
10	Knotenüberdeckung (Vortex Cover (VC))	21
11	Dominating Set (DS)	21
12	Subset Sum	22
13	Knapsack (Rucksack)	23
14	Partition	23
14.1	Bin Packing	24

15 Approximationsalgorithmen	25
15.1 Bin Packing	25
15.2 Multi Processor Scheduling (MPS)	26
15.3 Vortex Cover	27
15.4 Traveling Salesman Problem	30
15.4.1 Nearest Neighbor (NN)	30
15.4.2 Nearest Insertion	31
15.4.3 Christofides	32
15.5 Subset Sum	34
16 PSPACE	34
16.1 True Quantified Boolean Formular (TQBF)	35

1 Turingmaschinen

Eingabe und Arbeitsband

x_1	x_2	x_3	$\dots$	x_n
-------	-------	-------	---------	-------

 Lese- Schreibkopf

Abhängig vom Zustand und vom gelesenen Zeichen auf dem Band kann die Maschine:

- den Zustand wechseln
- das gelesene Zeichen überschreiben
- den Lese-Schreib-Kopf um ≤ 1 Feld auf dem Band nach links oder rechts bewegen.

Formal:

Ein Turingmaschine (TM) M hat 8 Komponenten,

$$M = (Z, \Sigma, \Gamma, \delta, Z_0, Z_a, Z_V, \square)$$

wobei:

- Z – Zustände
- Σ – Eingabealphabet
- Γ – Arbeitalphabet
- Z_0 – Anfangszustand
- Z_a – akzeptierender Zustand
- Z_V – verwerfender Zustand
- $\square$ – Blank $\in \Gamma - \Sigma$

Die Übergangsfunktion S ,

$$\delta : Z \times \Gamma \rightarrow Z \times \Gamma \times \{L, N, R\}$$

D.h. typische Anweisung ist

$$\delta(z, a) = (Z', b, x)$$

Wenn M im Zustand z ist und das Zeichen a liest, dann wechselt M in den Zustand Z' , überschreibt a mit b und bewegt den Lese-Schreibkopf um ein Feld nach links, falls $X = L$

- um ein Feld nach rechts, falls $X = R$
- bleibt stehen, falls $X = N$

Rechnung von M auf Eingabe $x = x_1 \dots x_n \in \Sigma^x$:

Initial steht x auf dem Eingabeband und der Lese-Schreibkopf auf dem 1. Zeichen x_1 .

Links und rechts der Eingabe steht $\square$ auf allen Feldern (Beidseitig unbeschränktes Band).

Anfangszustand ist Z_0 . Dann fährt die Maschine Schritte gemäß δ . Die Rechnung stoppt, wenn Z_a oder Z_V erreicht wird. x wird akzeptiert, falls Z_a erreicht wird, sonst wird x verworfen.

Die von M akzeptierte Sprache ist

$$L(M) = \{x \in \Sigma^* \mid M \text{ akzeptiert } x\} \text{ Bsp. Sei } \Sigma = \{0, 1\}$$

Berechne $f(x) = x + 1$

$$\begin{array}{|c|c|c|c|c|} \hline \square & 1 & 0 & 1 & \square \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|c|} \hline \square & 1 & 1 & 0 & \square \\ \hline \end{array}$$

Der berechnete Funktionswert ist definiert als das Wort $\in \Sigma^*$ von der Position des Lese-Schreibkopfs nach rechts, bis zu 1. Blank.

In Z_a .

$$\sigma(Z_0, 0) = (Z_0, 0, R)$$

$$\begin{aligned}
\sigma(Z_0, 1) &= (Z_0, 1, R) \\
\sigma(Z_0, \square) &= (Z_1, \square, L) \\
\sigma(Z_1, 0) &= (Z_2, 1, L) \\
\sigma(Z_1, 1) &= (Z_1, 0, L) \\
\sigma(Z_2, 0) &= (Z_2, 0, L) \\
\sigma(Z_2, 1) &= (Z_2, 1, L) \\
\sigma(Z_2, \square) &= (Z_a, \square, R) \\
\sigma(Z_1, \square) &= (Z_a, 1, N)
\end{aligned}$$

Als Diagramm

$$L = \{0^{2^n} \mid n \geq 0\}$$

0, 00, 0⁴, 0⁸, ...

0	0	0	0	
---	---	---	---	--

1. Vorschlag: Zähle

	0	0	...	0		0
--	---	---	-----	---	--	---

2. Vorschlag:

a) streiche jede 2. Null, teste dabei, ob Anzahl gerade

b) iteriere Schritt 1

bis nur noch eine Null bleibt.

0000 $\rightarrow$ 0x0x $\rightarrow$ 0xxx

1.1 Erweiterungen

1.1.1 Mehrband-Turingmaschine

Turingmaschine mit fester Anzahl k von Bändern. Davon eins ausgezeichnet für die Eingabe.

$$\delta : Z \times \Gamma^k \rightarrow Z \times \Gamma^k \times \{L, N, R\}^k$$

Bsp.:

$$\delta(Z, a, b) = \delta(Z', c, d, L, R)$$

$$\text{Bsp.: } L = \{w\#w \mid w \in \{0, 1\}^k\}$$

$$100\#100 \rightarrow x00\#x00 \dots$$

2-Band-Turingmaschine

kopiert w auf 2. Band

$$\delta(Z_0, a, \square) = (Z_0, a, a, R, R) \text{ für } a \in \{1, 0\}$$

$$\delta(Z_0, \#, \square) = (Z_1, \#, \square, N, L)$$

$$\delta(Z_1, \#, \square) = (Z_1, \#, a, N, L)$$

$$\delta(Z_1, \#, \square) = (Z_2, \#, \square, R, R)$$

$$\delta(Z_2, a, a) = (Z_2, a, a, R, R)$$

$$\delta(Z_2, \square, \square) = (Z_a, \square, \square, N, N)$$

Alles was fehlt geht nach Z_V

1.1.2 Nichtdeterministische Turingmaschine (NTM)

Definiert wie Turingmaschine, mit:

$$\delta : Z \times \Gamma \rightarrow \mathcal{P}(Z \times \Gamma \times \{L, N, R\})$$

$$\text{z. B. } \delta(Z, a) = \{(Z_1, b, L), (Z_2, c, N)\}$$

M akzeptiert x, falls es eine akzeptierende Rechnung gibt.

1.1.3 Nichtdeterministische Mehrband Turingmaschine

Bsp. $L = \{x0y \mid |x| = |y|\}$

über $\Sigma = \{0, 1\}$

$$\delta(Z_0, 1, \square) = \{(Z_0, 1, 1, R, R)\}$$

$$\delta(Z_0, 0, \square) = \{(Z_0, 0, 0, R, R), (Z_1, 0, \square, R, L)\}$$

$$\delta(Z_1, a, b) = \{(Z_1, a, b, R, L)\} \quad a, b \rightarrow \{0, 1\}$$

$$\delta(Z_1, \square, \square) = \{(Z_a, \square, \square, N, N)\}$$

$$\delta(Z_0, \square, \square) = \{(Z_v, \square, \square, N, N)\}$$

(fehlende Anweisungen verwerfen.)

Deterministisch: $\delta : Z \times \Gamma \rightarrow Z \times \Gamma \times \{L, N, R\}$

Nicht Deterministisch: $\delta : Z \times \Gamma \rightarrow \mathcal{P}(Z \times \Gamma \times \{L, N, R\})$

Satz: Zu jeder k-Band Turingmaschine gibt es eine äquivalente (1-Band) Turingmaschine

Bew.: (Idee)

k-Band Turingmaschine M

	x			
		y		
			z	

TM M'

	#	x			#		y		#			z	#	
--	---	---	--	--	---	--	---	--	---	--	--	---	---	--

M:

□	a	b	a	c	□
□	c	c	□		
□	b	a	a	a	□

M':

#	□	a	b	a	c	□	#	□	c	a	□	#	□	b	a	a	a	□	#
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$\dot{b} \rightarrow \Gamma' - \Gamma$

M' läuft 2x mal über das gesamte Band (und zurück)

1. M' merkt sich alle Zeichen, die die Lese-Schreibköpfe aktuell lesen
2. M' führt die Änderungen gemäß δ von M auf allen Bändern aus.

Merke Zeichen im Zustand:

$$Z_{Z,b,c,\square}$$

d. h. $|Z| \cdot |\Gamma|^3$ viele Zustände, also endlich.

Initial:

#	x	#	□	#	□	#
---	---	---	---	---	---	---

Zeit: M macht t Schritte auf Eingabe X .

Alle Bänder von M' sind also mit Wörtern beschriftet mit Länge $\leq t$

→ Beschrifteter Teil des Bands von M hat Länge $\leq k \cdot t$

Pro Schritt von M macht M'

- $\leq 2 \cdot 2 \cdot t$ Schritte für die beiden Durchläufe.
- verschieben zum Platz schaffen $\leq k \cdot 2 \cdot t$
- Zusammen $\leq (k + 2) \cdot 2 \cdot t = \mathcal{O}(t)$.
 M macht t Schritte $\Rightarrow M'$ macht $\mathcal{O}(t^2)$ Schritte

Satz: Zu jeder NTM gibt es eine äquivalente TM.

Bew. (Idee):

Sei M NTM

Strategie: durchsuche Baum "Breite zuerst". Dazu zunächst Mehrband-TM (det.).

Pro Adresse:

- Kopiere x auf das (leere) Simulationsband,
- führe Rechnung von M auf x aus, wobei jeweils die Anweisung gemäß Adresse benutzt wird.
- falls M akzeptiert wird, dann akzeptierte.

Bann lösche Simulationsband, erhöhe Adresse um 1 (zur Basis d) und wiederhole.

Zeit: Habe jede Rechnung von M die Länge $\leq t$

$\Rightarrow$ Anzahl der Knoten im Baum ist $\leq d^{t+1} - 1$

$\Rightarrow$ Rechenzeit von M' : $\mathcal{O}(t + n) \cdot d^t$

$\Rightarrow$ Rechenzeit der det. 1-Band TM M'' ist dann $\mathcal{O}\left((t + n) \cdot d^t\right)^2$

$$\begin{aligned}
 & \mathcal{O}(t^2 2^{2t \cdot \log d}) \\
 &= \mathcal{O}(2^{2 \log t} \cdot 2^{2 \cdot \log d}) \\
 &= \mathcal{O}(2^{2t \log d + 2 \log t}) \\
 &= \mathcal{O}(2^{2t(\log d + 1)}) \\
 &= 2^{\mathcal{O}(t)}
 \end{aligned}$$

Weitere Rechenmodelle haben sich alle als äquivalent zur TM erwiesen. Insbesondere lässt sich jedes C-Programm in eine äquivalente TM umschreiben.

These von Turing/Church

Algorithmus = TM intuitiver Begriff = formal

1.2 Kodierung von Turingmaschinen

Sei M TM mit $Z = \{Z_1, Z_2, \dots, Z_k\}$ und $\Sigma \leq \Gamma = \{a_1, a_2, \dots, a_m\}$

Sei $\delta(Z_i, a_j) = (Z_{i'}, a_{j'}, d)$

$d = \{L, M, R\}$

Kodiere mit:

$$x_{i,j} = 01^i 01^j 01^{i'} 01^{j'} 01^{d'} 0$$

$$\text{mit } d = \begin{cases} 1 & , \text{falls } d = L \\ 2 & , \text{falls } d = N \\ 3 & , \text{falls } d = R \end{cases}$$

Kodiere M durch

$$x = x_{1,1} x_{1,2} \dots x_{1,m} x_{2,1} \dots x_{k,m}$$

Dabei lege fest: Z_1 ist Startzustand; Z_{k-1} ist Z_V ; Z_k ist Z_a

und $a_1 = \square$

Damit besitzt jede Turingmaschine M eine Kodierung $x \in \{0,1\}^*$. x heißt auch Index oder Gödelnummer von M

Nicht jedes Wort aus $\{0,1\}^*$ taucht als Kodierung gemäß obigem Schema auf.

Deswegen: Für $x \in \{0,1\}^*$

$$x \rightarrow \begin{cases} M_x & , \text{falls Kodierung von } M_x \text{ ergibt} \\ M_0 & , \text{sonst} \end{cases}$$

M_0 fest, mit $L(M_0) = \emptyset$

Mit dieser Festlegung kodiert jedes Wort $x \in \{0,1\}^*$ eine TM M_x .

D. h. $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$

dies ist eine Aufzählung aller 0-1-Wörter und damit auch aller Turingmaschinen.

1.3 Das Halteproblem

$$H = \{(x, y) \mid M_y(x) \text{ hält}\}$$

dabei sei $(x, y) = x_1 x_1 x_2 x_2 \dots x_n x_n 01 y$

$$(010, 10) = \begin{cases} 000101010 \\ 00110010 \underbrace{10}_y \end{cases}$$

1.4 Die Universalsprache

$$U = \{(x, y) \mid M_y \text{ akzeptiert } x\}$$

Eine Sprache L heißt (Turing-) akzeptierbar, falls es eine Turingmaschine M gibt, mit $L(M) = L$.

L heißt (Turing-) entscheidbar, falls M zusätzlich auf alle Eingaben hält.

U ist akzeptierbar:

Beschreibe zunächst eine 3-Band-Turingmaschine

M für U

M :

	(x, y)	
--	----------	--

$$\begin{array}{c} \rightarrow \\ Z_1 \text{ Startzustand} \end{array}$$

	x_1	x_2	$\dots$	x_n	
Z_1					
y					

dann simuliere M_y auf dem 1. Band

d. h. Suche in y die Stelle wo $\delta(Z_1, x_1)$ definiert ist und führe dann Änderungen auf dem 1. Band aus. * Dann gehe auf Band 2 und 3 wieder nach links und wiederhole. Abbruch wenn $M_y Z_a$ oder Z_v erreicht.

* Auf Band 2 ändere den Zustand.

M akzeptiert, falls $M_y Z_a$ erreicht und verwirft bei Z_v .

Dann gilt:

- $(x, y) \in U \Rightarrow M_y(x)$ akzeptiert.
 $\Rightarrow M_y(x)$ kommt nach endlich vielen Schritten nach Z_a
 $\Rightarrow M(x, y)$ akzeptiert
- $(x, y) \notin U \Rightarrow M_y(x)$ verwirft.
 - a) $] M_y(x)$ erreicht Z_v
 $\Rightarrow M(x, y)$ verwirft.
 - b) $M_y(x)$ hält nicht
 $\Rightarrow M(x, y)$ hält nicht.
 $\Rightarrow M(x, y)$ verwirft.

Es gibt also $L(M) = U$.

Sei M_U die zu M äquivalente (1-Band) Turingmaschine.

M_U heißt universelle Turingmaschine.

Analog ist H akzeptierbar.

(nehme M von oben und akzeptiere auch, wenn Z_v erreicht wird.)

Bezeichnungen:

- akzeptierbar
 - rekursiv aufzählbar
 - semi-entscheidbar
 - partiell rekursiv
- entscheidbar
 - rekursiv

Wir Konstruieren eine Sprache D , die nicht nicht akzeptierbar ist.

D. h. $D \neq L(M)$ für alle Turingmaschinen M.

TM \ Σ^*	x_1	x_2	x_3	x_4	$\dots$
x_1	0	1	0	0	$\dots$
x_2	1	0	1	1	
x_3	0	0	1	1	
x_4	1	0	1	0	$\dots$
$\vdots$				$\vdots$	

$M_{x_1}(x_3)$ verwirft

$M_{x_1}(x_4)$ akzeptiert

Jede Zeile in der Tabelle beschreibt eine Sprache:

Zeile i : die von M_{x_i} akzeptierte Sprache.

Wir konstruieren D so, dass D mit keiner Zeile der Tabelle übereinstimmt.

$$D = \{x \mid x \notin L(M_x)\}$$

Dann ist D nicht akzeptierbar:

Angenommen $D = L(M) = L(M_x)$

für ein $x \in \Sigma^*$

Es gilt aber $x \in D \leftrightarrow x \notin L(M_x)$

Widerspruch!

1.5 Diagonalsprache

$D = \{x \mid x \notin L(M_x)\}$ ist nicht akzeptierbar.

$$U = \{(x, y) \mid \underbrace{i M_y \text{ akzeptiert } x}_{x \in L(M_y)}\}$$

Satz: H und U sind nicht entscheidbar.

Beweis zu U : Angenommen U wäre entscheidbar. D.h. es gibt eine Turingmaschine M , die U akzeptiert die auf alle Eingaben hält.

Dann entscheidet folgende Turingmaschine M' die Sprache D :

$$D = \{x \mid (x, x) \notin U\}$$

$$H = \{(x, y) \mid M_y \text{ auf } x \text{ hält}\}$$

M' :

Eingabe x .

Starte M auf Eingabe (x, x)

Falls M akzeptiert, dann verwirft, sonst akzeptiere.

Da M immer hält, hält auch M' auf alle Eingaben und $L(M') = D$.

Widerspruch, da D nicht entscheidbar. $\square$

Zu H : Angenommen H ist entscheidbar, $H = L(M)$.

Konstruiere M' für D :

M' :

Eingabe x

Starte M auf Eingabe (x, x) .

Falls M akzeptiert

dann starte M_x auf x
 falls M_x akzeptiert, dann verwirfe
 sonst akzeptiere

sonst akzeptiere

1. $(x, x) \in H \rightarrow M_x(x)$ hält
 und $x \in D \leftrightarrow M_x$ verwirft
2. $(x, x) \notin H \Rightarrow M_x(x)$ hält nicht
 $\Rightarrow M_x$ verwirft x
 $\Rightarrow x \in D$
 Es gilt.: M' hält auf alle Eingaben und $L(M') = D$. Widerspruch $\square$

1.6 Das spezielle Halteproblem

$$H_0 = \{x \mid M_x(x) \text{ hält} \}$$

H_0 ist akzeptierbar und entscheidbar.

Zu H_0 : Angenommen H_0 ist entscheidbar, $H_0 = L(M)$.

$$T = \{x \mid M_x \text{ hält auf alle Eingabe} \}$$

Turingmaschine M heißt total, falls M auf alle Eingaben hält.

$$L_E = \{x \mid M_x \text{ hat Eigenschaft } E\}$$

Satz: T ist nicht entscheidbar.

Beweis: Sonst wäre H entscheidbar.

Sei M_T Turingmaschine für T (die immer hält)

Turingmaschine M für H : Eingabe (x, y) .

Betrachte folgende Turingmaschine M_0 :

M_0 auf Eingabe $w \in \Sigma^*$:
 Simuliere $M_y(x)$
 Falls $M_y(x)$ hält, dann akzeptiere

1. Fall: $(x, y) \in H \Rightarrow M_y(x)$ hält.
 $\Rightarrow M_0$ akzeptiert
 $L(M_0) = \Sigma^*$
2. Fall: $(x, y) \notin H \Rightarrow M_0$ hält nicht
 $\Rightarrow M_0$ verwirft
 $\Rightarrow L(M_0) = \emptyset$

Sei y_0 Kodierung von M_0 .

Dann gilt:

$$(x, y) \in H \Leftrightarrow y_0 \in T$$

M für H :

Eingabe (x, y)
 Konstruiere Turingmaschine M_0 (aus x wird y)
 Sei y_0 Kodierung von M_0
 Falls $y_0 \in T$, dann akzeptiere
 Sonst verwerfe

1. Fall $(x, y) \in H \Rightarrow M_x(x)$ hält
 $\Rightarrow L(M_0) = \Sigma^*$
 $\Rightarrow$ insbesondere hält M_0 auf alle Eingaben
 $\Rightarrow y_0 \in T$
 $\Rightarrow M$ akzeptiert (x, y)
2. Fall $(x, y) \notin H \Rightarrow L(M_0) = \emptyset$ und M_0 hält auf keine Eingabe
 d. h. M entscheidet H Widerspruch $\square$

$L_E = \{x \mid \text{Turingmaschine } M_x \text{ hat } \underbrace{\text{Eigenschaft}}_{\text{Spracheigenschaft}} E\}$

ist unentscheidbar für jede Eigenschaft E .

1.6.1 Satz von Rice

Bsp.: $F = \{x \mid L(M_x) \text{ ist endlich}\}$

finite: $\bar{F} = \{x \mid L(M_x) \text{ ist unendlich}\}$

empty: $E = \{x \mid L(M_x) = \emptyset\}$

$S = \{x \mid L(M_x) = \Sigma^*\}$

Äquivalenz = $\{(x, y) \mid L(M_x) = L(M_y)\}$

Äquivalenz für alle Sprachen
 $= \{(x, y) \mid x, y \text{ sind PDAs und } L(M_x) = L(M_y)\}$
 ist unentscheidbar.

Gödel: mehrere Arithmetische Formeln:

$\forall n \exists m : (n = 2m \text{ oder } n = 2m - 1) \text{ wahr}$

$\forall n \exists m : n = 2m \text{ falsch}$

1.6.2 Post'sches Korrespondenz Problem (PCP)

gegeben sind Dominos der Form $\begin{bmatrix} x \\ y \end{bmatrix}$

wobei $x, y \in \Sigma^*$, endlich viele Typen, von jedem Type beliebig viele.

Bsp.: $D = \left\{ \begin{bmatrix} b \\ ca \end{bmatrix}, \begin{bmatrix} a \\ ab \end{bmatrix}, \begin{bmatrix} ca \\ a \end{bmatrix}, \begin{bmatrix} abc \\ c \end{bmatrix} \right\}$

Ziel: lege Dominos so, dass das Wort, das oben steht gleich dem ist, das unten steht.

$\begin{bmatrix} a \\ ab \end{bmatrix} \begin{bmatrix} b \\ ca \end{bmatrix} \begin{bmatrix} ca \\ a \end{bmatrix} \begin{bmatrix} abc \\ c \end{bmatrix}$ Widerspruch!
 $\begin{bmatrix} a \\ ab \end{bmatrix} \begin{bmatrix} b \\ ca \end{bmatrix} \begin{bmatrix} ca \\ a \end{bmatrix} \begin{bmatrix} a \\ ab \end{bmatrix} \begin{bmatrix} abc \\ c \end{bmatrix}$

D hat Lösung, also $D \in PCP$

$$D_1 = \left\{ \begin{bmatrix} ab \\ abc \end{bmatrix} \begin{bmatrix} c \\ ab \end{bmatrix} \begin{bmatrix} ac \\ cab \end{bmatrix} \begin{bmatrix} ba \\ acb \end{bmatrix} \right.$$

$$\left. \begin{bmatrix} ab \\ abc \end{bmatrix} \begin{bmatrix} c \\ ab \end{bmatrix} \right\}$$

hat keine Lösung, da Wörter unten immer Länger als Wörter oben sind. $D_1 \notin PCP$

Bsp.: $\left\{ \begin{bmatrix} 0 \\ 011 \end{bmatrix} \begin{bmatrix} 001 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 00 \end{bmatrix} \begin{bmatrix} 11 \\ 110 \end{bmatrix} \right\}$

Kürzeste Lösung hat 515 Dominos.

PCP ist unentscheidbar

2 Eulerkreise

Rundweg in G , der über jede Kante genau einmal führt.

Kreis = (1, 5, 2, 3, 1, 2, 4, 6, 3, 4)

Ein Eulerscher Kreis besucht jeden Knoten gerade oft, d.h. jeder Knoten muss geraden Grad haben.

Satz: (Euler) G ungerichtet, zusammenhängend
 G hat Eulerkreis $\Leftrightarrow$ jeder Knoten hat geraden Grad.

Zu " $\Leftarrow$ ": Konstruiere Eulerkreis

Starte an beliebigem Knoten, der noch unbesuchte Knoten hat. Laufe in beliebige Richtung, solange bis es nicht mehr weiter geht.

Falls es noch unbesuchte Kanten gibt, wiederhole.

Füge (rekursiv) die Kreise zu einem Kreis zusammen.

Der Algorithmus funktioniert, weil in jeder Iteration in dem Restgraph mit dem noch unbesuchten Kanten jeder Knoten geraden Grad hat. (Invariante)

3 Hamilton Kreise

= Kreise, in denen jeder Knoten genau einmal besucht wird.

Brute force: teste alle Möglichkeiten.

n Knoten. Liste alle Permutationen der n Knoten auf.

1, 2, 3, ..., n

2, 1, 3, ..., n

...

für jede Permutation teste, ob sie ein Hamilton Kreis ist d. h. ob alle Kreiskanten in G existieren.

Laufzeit $\geq n!$ (im schlechtesten Fall)

$n! \approx 2^{n \log n} \geq 2^n$

also exponentielle Laufzeit.

(Sehr schneller) Rechner mit $10^9 \frac{\text{instructions}}{s}$

$n = 100$

$t(n)$	n	n^2	n^3	2^n
Zeit	$\frac{1}{10^7}$	$\frac{1}{10^4}$	$\frac{1}{10^2}$	$\frac{2^{100}}{10^9} = \frac{(2^{10})^{10}}{10^9} \approx \frac{(10^3)^{10}}{10^9} = 10^{21} s = 3 * 10^{13} \text{ Jahre}$

Schnellere Hardware:

Sei n_0 = Eingabegröße, die in 1h gelöst wird.

Neue Rechner, 1000-mal schneller als bisher.

Berechne n_1 = Eingabegröße, die in einer Stunde mit den neuen Rechnern gelöst wird.

1. $t(n) = n$
 1h. $t(n_1) = 1000t(n_0)$
 $n_1 = 1000 * n_0$
2. $t(n) = n^2 : n_1^2 = 1000 * n_0^2$
 $n_1 = \underbrace{\sqrt{1000} * n_0}_{\approx 33}$
3. $t(n) = n^3$
 $n_1^3 = 1000 * n_0^3$
 $n_1 = 10 * n_0$
4. $t(n) = 2^n$
 $2^{n_1} = 1000 * 2^{n_0} = 2^{n_0 + \log 1000}$
 $\Rightarrow n_1 = n_0 + \underbrace{\log 1000}_{\leq 10}$

Nichtdeterministischer Algorithmus
für hamilton Kreis:

Eingabe G mit n Knoten

- Permutation ρ
- Prüfe, ob ρ einen hamilton Kreis beschreibt
falls ja, akzeptiere, sonst verwirfe

HAM nichtdeterministisch effizient lösbar.

4 Erfüllbarkeit logischer Formeln, SAT

$$F(x_1, \dots, x_n) = (\overline{x_1} \wedge \overline{x_2} \vee x_3) \rightarrow (x_4 \wedge \overline{x_5})$$

Frage: gibt es eine Belegung $a \in \{0, 1\}^n$ so dass $F(a) = 1$.

Deterministischer Algorithmus: durchlaufe alle $a \in \{0, 1\}^n$ und teste jeweils, $F(a) = 1$.
 Laufzeit $\geq 2^n$, exponentiell.

Nichtdeterministisch: Rate nichtdeterministisch ein $a \in \{0, 1\}^n$ und teste ob $F(a) = 1$.

$\overline{\text{F}}$			
	x	x	...
	x		x

markiere n Bandfelder.

fülle diese nichtdeterministisch mit 0 oder 1 aus

$$\delta(Z_1, x) = \{(Z_1, 0, N, R), (Z_1, 1, N, R)\}$$

KNF-SAT = Erfüllbarkeit für Formeln F in KNF

$$\text{Bsp. } F = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_5 \vee \overline{x_6} \vee x_9 \vee \overline{x_1})$$

DNF-SAT = analog

$$F = (x_1 \wedge \overline{x_2} \wedge x_3) \vee (x_2 \wedge \overline{x_5} \wedge x_4) \vee (x_5 \wedge \overline{x_6} \wedge x_9 \wedge \overline{x_1})$$

Es genügt, eine Klausel zu erfüllen $\Rightarrow$ DNF-SAT

$\Rightarrow$ DNF-SAT ist effizient lösbar.

5 Einteilung in Klassen

Sei $t : \mathbb{N} \rightarrow \mathbb{N}$

$DTIME(t) = \{L \leq \Sigma^* \mid \text{es gibt eine Turingmaschine } M \text{ mit } L(M) = L \text{ und } M \text{ macht } O(t(m))$

Schnitte auf Eingaben der Länge $n\}$

Analog: $NTIME(t)$ für NTM.

Analog für Speicherplatz:

Hier mit 2-Band Turingmaschine. Dabei darf das Eingabeband nicht verändert werden. Speicherbedarf wird nur in Bezug auf das 2. Band gemessen.

Für $S : \mathbb{N} \rightarrow \mathbb{N}$

$DSPACE(s) = \{L \mid \text{es gibt } M \text{ mit } L(M) = L \text{ und } M \text{ braucht } O(s(m)) \text{ Platz auf Eingaben der Länge } n\}$

Bsp.: Reguläre Sprachen $\leq DTIME(n)$.

Kontextfreie Sprachen in CNF mit CYK-Algorithmus $\leq DTIME(n^3)$ (evtl. n^4)

Polynomische Rechenzeit:

$$P = \bigcup_{k \geq 1} DTIME(n^k)$$

$$P = DTIME(n) \cup DTIME(n^2) \cup \dots$$

Nichtdeterministische polynomische Rechenzeit

$$NP = \bigcup_{k \geq 1} NTIME(n^k)$$

Bsp.: $HAM \in NP$, $SAT, CNF - SAT \in NP$

$DNF - SAT \in P$

Offenes Problem: $HAM, SAT \in P$?

In P sind auch Probleme mit Laufzeit n^{1000} . Aber auch mit dieser Laufzeit ist kein Algorithmus für SAT bekannt!

$\left. \begin{array}{l} \text{Registernmaschinen} \\ \text{RAM} \end{array} \right\}$ moderne Hardware statt Turingmaschine.

Gegenseitige Simulation möglich. Typischer Zeitzuwachs ist dabei ein Faktor von n^3 oder n^4 .

D.h. P bleibt gleich bei anderer Hardware, genauso NP.

Exponentielle Zeit

$$EXP = \bigcup_{k \geq 0} DTIME(2^{n^k})$$

$NEXP$ = analog mit $NTIME$

Logarithmischer Platz

$$L = DSPACE(\log n)$$

$$NL = NSPACE(\log n)$$

Polynomischer Platz

$$PSPACE = \bigcup_{k \geq 0} DSPACE(n^k)$$

$$NSPACE = \bigcup_{l \geq 0} NSPACE(n^k)$$

Es gilt:

1. $DTIME(t) \leq DSPACE(t)$
 pro Schritt kann höchstens ein weiteres Bandfeld beschrieben werden.
 $NTIME(t) \leq NSPACE(t)$
 Folgerungen:
 $P \leq PSPACE$
 $NP \leq NPSPACE$
2. $DTIME(t) \leq NTIME(t)$
 $DSPACE(s) \leq NSPACE(s)$
 Folgerungen:
 $P \leq NP$
 $EXP \leq NEXP$
 $L \leq NL$
 $PSPACE \leq NPSPACE$
3. $NTIME(t) \leq DTIME(2^{O(t)})$
 Folgerung:
 $NP \leq EXP$
4. $DSPACE(t) \leq DTIME(2^{O(t)})$ (sogar für $NSPACE(t)$)
 ohne Beweis
 Folgerungen:
 $PSPACE \leq EXP$
 $NL \leq P$
5. Satz von Savitch
 Sei $S(m) \geq \log n$
 $NSPACE(s) \leq DSPACE(s^2)$
 Folgerungen:
 $NPSPACE \leq PSPACE$
 d. h. $PSPACE = NPSPACE$

Zusammenfassung:

$$L \leq NL \leq P \leq NP \leq PSPACE \leq EXP \leq NEXP$$

Alle Inklusionen sind offen (ob sie echt sind).

$$\text{z. B. } P \stackrel{?}{=} NP \leftarrow \text{Preis}$$

$$L \stackrel{?}{=} NL$$

$$NL \stackrel{?}{=} P$$

$$MP \stackrel{?}{=} PSPACE$$

Bekannt ist:

$$NL \neq PSPACE$$

aber $NL \stackrel{?}{=} NP$ ist offen.

$$P \neq EXP$$

aber $P \stackrel{?}{=} PSPACE$ ist offen

$$NP \neq NEXP$$

aber $NP \stackrel{?}{=} EXP$ ist offen.

offen ist $L \stackrel{?}{=} NP$

Reduktionen:

Seien $A, B \subseteq \Sigma^*$ Sprachen.

A ist auf B reduzierbar, schreibe $A \leq^P B$, falls es eine in polynomieller Zeit berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ gibt, so dass gilt:

$$\forall x \in \Sigma^* \quad x \in A \leftrightarrow f(x) \in B.$$

$$\text{d. h. } x \in A \Rightarrow f(x) \in B$$

$$\text{und } x \notin A \Rightarrow f(x) \notin B$$

6 Traveling Salesman (TSP)

Bsp.: Traveling Salesman (TSP)

Finde kürzeste Rundreise die jede Stadt genau einmal besucht.

TSP

gegeben: $n, L, d : \{0, \dots, n\}^2 \rightarrow \mathbb{N}$

gefragt: gibt es eine Permutationen (=Rundreise)

der Länge $\leq L$

dabei ist $d(i, j) = d(j, i) = \text{Abstand von Stadt zu Stadt.}$

Satz: $HAM \leq^P TSP$

Beweis: gegeben sei $G = (V, E)$ mit $|V| = n$.

Definiere:

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

1. Ein hamilton Kreis ind G liefert eine TSP-Tour der Länge n.
2. Hat G keinen hamilton Kreis, dann benützt jede TSP-Tour mindestens eine Kante der Länge 2
 $\Rightarrow \text{Gesamtlänge} \geq n + 1$
 Dann gilt also
 $G \in HAM \Leftrightarrow (n, L, d) \in TSP.$
 d. h. $f(G) = (n, L, d)$

Satz: Sei $A \leq^P B$.

1. $B \in P \Rightarrow A \in P$
2. $B \in NP \Rightarrow A \in NP$

Beweis: zu 1.

Sei M_B eine Turingmaschine für B die in polynomieller Zeit arbeitet.

Sei M_f eine Turingmaschine, die die Reduktionsfunktion in polynomieller Zeit berechnet.

Sei $c \in \Sigma^*$ Eingabe für A.

$$x \rightarrow M_f \xrightarrow{f(x)} M_B \rightarrow \{0, 1\}$$

Sei p ein Polynom, das die Rechenzeit von M_f beschränkt, z. B. $p(n) = n^k$.

Ebenso q Polynom für M_B , z. B. $q(n) = n^e$

Rechenzeit von M_A auf x, $|x| = n$:

$$p(n) + q(\underbrace{p(n) + n}_{\text{max. Länge von } f(x)})$$

Dies ist ein Polynom, alsp $A \in P$.

Bsp.: $n^k + (n^k + n)^l = O(n^{k+l})$

Beweis: zu 2.

M_B nichtdeterministisch $\Rightarrow M_A$ nichtdeterministisch. Zeit bleibt unverändert.

Def.: A, B heißen (polynomiell) äquivalent, schreibe $A \equiv^P B$, falls $A \leq^P B$ und $B \leq^P A$.

Lemma: $\equiv^P$ ist Äquivalenzrelation.

Beweis: (R) $A \equiv^P A$, da $A \leq^P A$ mittels $f(x) = x$.

(S) $A \equiv^P V \Rightarrow B \equiv^P A \square$

(T) $A \leq^P B$ und $B \leq^P C \Rightarrow A \leq^P C$

Sei $A \leq^P B$ mittels f und $B \leq^P C$ mittels g

Definiere $h(x) = g(f(x))$.

Rechenzeit für h ist Polynom $\square$

Def.: Sei $A \subseteq \Sigma^*$.

A ist NP-hart, falls $\forall L \in NP \quad L \leq^P A$.

A heißt NP-Vollständig, falls A NP-Hart ist und $A \in NP$

Satz: Sei A NP-Vollständig.

$A \in P \Rightarrow P = NP$.

Beweis: Sei $L \in NP$. Zeige: $L \in P$

Da A NP-vollständig ist, gilt: $L \leq^P A$.

Nach obigem Satz: $A \in P \Rightarrow L \in P$

Satz: Sind A, B NP-vollständig, dann ist $A \equiv^P B$.

Satz von Cook (1971)

SAT ist NP-Vollständig.

Beweis: $SAT \in NP$

Sei $M = (Z, \Sigma, \Gamma, \delta, Z_a, Z_v, \square)$ eine NTM mit Rechenzeit $p(n)$ mit $L = L(M)$.

Sei $Z = \{Z_0, Z_1, \dots, Z_k\}, \Gamma = \{a_1, \dots, a_m\}$
 $\subseteq \Sigma$

Sei $x \in \Sigma^*$ Eingabe für M . Gebe Reduktionsfunktion f an, so dass $f(x) = F_x$ eine boolesche Formel ist, so dass gilt:

$$x \in L(M) \Leftrightarrow F_x \in SAT$$

$$|x| = n$$

Variablen von F_x :

$zust_{t,z}, t = 0, 1, \dots, P(n), z \in Z$

(Intuitiv: = 1, falls M zum Zeitpunkt t im Zustand z , sonst 0)

$pos_{t,i}, t = 0, 1, \dots, p(n), i$

(Intuitiv: = 1, falls der Lese/Schreibkopf von M zum Zeitpunkt t auf Feld i steht)

$band_{t,i,q}, t = 0, 1, \dots, P(n), i = -p(n), \dots, p(n), a \in \Gamma$ (Intuitiv: = 1, falls auf dem Band von M zum Zeitpunkt t auf Feld i das Zeichen a steht, 0 sonst)

F_x setzt sich zusammen aus

$$F_x = R \wedge A \wedge U_1 \wedge U_2 \wedge E \quad R \text{ ist die Randbedingung.}$$

Zu jedem Zeitpunkt t ist genau eine Variable von $zust_{t,z_0}, \dots, zust_{t,z_k}$ wahr. Ebenso von $pos_{t,-p(m)}, \dots, pos_{t,p(m)}$

Variable $y_1, \dots, y_n$

$$G(y_1, \dots, y_n) = \begin{cases} 1, & \text{falls genau eine der Variable } y_i = 1 \text{ ist} \\ 0, & \text{sonst} \end{cases}$$

$$G = y_1, \dots, y_n = \left(\bigvee_{i=1}^n y_i \right) \wedge \left(\bigwedge_{1 \leq i \leq j \leq n} \overline{y_i \wedge y_j} \right)$$

1. Teil heißt: ≥ 1 Variablen ist 1

2. Teil heißt: keine 2 Variablen sind $1 \equiv \leq 1$ Variable ist 1

Damit

$$R = \bigwedge_{t=0}^{p(n)} \left(G(zust_{t,t_0}, \dots, zust_{t,t_k}) \wedge G(pos_{t,-p(n)}, \dots, pos_{t,p(n)}) \wedge \bigwedge_{i=-p(n)}^{p(n)} G(band_{t,i,a}, \dots, band_{t,i,a_m}) \right)$$

$$|R| = O(p(n) * (1 + (2 * p(n))^2 + 2 * p(n) * 1))$$

$$= O(p^3(n))$$

A, die Anfangsbedingung beschreibt die Variablen zum Zeitpunkt $t = 0$.

$$A = zust_{0,Z_0} \wedge pos_{0,1} \wedge \left(\bigwedge_{i=-p(n)}^0 band_{0,1,\square} \wedge \bigwedge_{i=1}^n band_{0,i,x_i} \wedge \bigwedge_{i=n+1}^{p(n)} band_{0,i,\square} \right)$$

$$|A| = O(p(n))$$

$\ddot{U}_1$ = Beschreibt den Übergang von t nach $t+1$ an der Stelle, an der der Lese/Schreib-Kopf steht.

$$= \bigwedge_{t=0}^{p(n)-1} \bigwedge_{z \in Z} \bigwedge_{i=-p(n)}^{p(n)} \bigwedge_{a \in \Gamma} \left((zust_{t,z} \wedge pos_{t,i} \wedge band_{t,i,a}) \rightarrow \bigvee_{Z',a',d} (zust_{t+1,Z'} \wedge pos_{t+1,i+d} \wedge band_{t+1,i,a'}) \right)$$

$$(Z', a', d) \in \delta(Z, a) \text{ mit } d = \begin{cases} 1 & \text{für R} \\ 0 & \text{für N} \\ -1 & \text{für L} \end{cases}$$

$$|\ddot{U}_1| = O(p(n) * 1 * (2 * p(n)) * 1 * 1) = O(p^2(n))$$

$\ddot{U}_2$ beschreibt die restlichen Übergänge.

$$\ddot{U}_2 = \bigwedge_{t=0}^{p(n)} \bigwedge_{i=-p(n)}^{p(n)} \bigwedge_{a \in \Gamma} (\neg pos_{t,i} \wedge band_{t,i,a} \rightarrow band_{t+1,i,a}) \quad |\ddot{U}_2| = O(p^2(n))$$

E beschreibt das Ende, dass M akzeptiert

$$E = \bigvee_{t=1}^{p(n)} zust_{t,Z_a}$$

$$|E| = O(p(n))$$

$$F_x = R \wedge A \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E$$

$$|F_x| = O(p(n)^3)$$

Dann gilt:

zu jeder akzeptierenden Rechnung von $M(x)$ gibt es eine zugehörige Belegung der Variablen von F_x , die F_x erfüllt, und umgekehrt.

Es gilt also: $M(x)$ hat gleich viele akzeptierende Rechnungen wie F_x erfüllende Belegungen.

Insbesondere gilt also:

$$x \in L(M) \Leftrightarrow F_x \in SAT \quad \square$$

Um von einem weiteren Problem A zu zeigen, dass A NP-vollständig ist, genügt es jetzt zu zeigen

$$SAT \leq^P A \text{ (und } A \in NP)$$

da dann für $L \in NP$ gilt:

$$L \leq^P SAT \text{ und } SAT \leq^P A \Rightarrow L \leq^P A.$$

Cook SAT ist NP-Vollständig

Satz: Sei A NP-Vollständig und $B \in NP$.

Dann gilt: $B = \text{NP-Vollständig} \Leftrightarrow A \leq^P B$

$$KNF - SAT = \{F \in SAT \mid F \text{ ist in KNF}\}$$

$$k - SAT = \{F \in KNF - SAT \mid \text{jede Klausel von } F \text{ hat } \leq 3 \text{ Literale}\}$$

Satz: 3-SAT ist NP-Vollständig.

Beweis: Zeige $SAT \leq^P 3 - SAT$

Bsp.: Sei F boolsche Formel.

$$F = ((x_1 \vee \overline{x_3}) \wedge x_4) \vee (\overline{x_2} \wedge x_3)$$

1. Schritt: Bringe alle Negationszeichen nach innen $\neg(x_1 \vee \overline{x_2}) \equiv (\overline{x_1} \wedge x_2)$

2. Schritt: nehme neue Variablen $y_1, y_2, \dots$ und ordne jedem Gatter eine Variable zu

3. Schritt: Definiere G

$$G = (y_1 \leftrightarrow (x_1 \vee \overline{x_3})) \wedge (y_2 \leftrightarrow (y_1 \wedge x_4)) \wedge (y_3 \leftrightarrow (\overline{x_2} \wedge x_3)) \wedge (y_4 \leftrightarrow (y_2 \vee y_3)) \wedge y_4$$

Dann gilt: $F \in SAT \Leftrightarrow G \in SAT$

$$x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1$$

$$y_1 = 1, y_2 = 1, y_3 = 1, y_4 = 1$$

4. Schritt: forme einzelne Klammern in KNF um:

$$a \leftrightarrow (b \vee c) \equiv (a \vee \overline{b}) \wedge (\overline{a} \vee b \vee c) \wedge (a \vee \overline{c})$$

$$a \leftrightarrow (b \wedge c) \equiv (\overline{a} \vee b) \wedge (a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee c)$$

Dann ist G in 3-KNF. ■

Es gilt: 2-SAT $\in P$

Not-all-equal-SAT, kurz: NAE-SAT

gegeben: F in 3-KNF

gefragt: gibt es eine erfüllende Belegung für F , so dass in keiner Klausel alle Literale den Wert 1 haben.

$$F = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$$

Satz: $NAE - SAT$ ist NP-Vollständig

Beweis: $NAE - SAT \in NP$:

wähle nicht-deterministisch Belegung a und teste dass jede Klausel erfüllt ist und nicht alle 3 Literale Wert 1 haben.

Zeige $SAT \leq^P NAE - SAT$

Sei F beliebige Formel und G die oben konstruierte in 3-KNF.

Es gilt: eine erfüllbare Belegung von G erfüllt nicht alle Literale in Klauseln mit 3 Literalen.

- $\overline{a} \vee b \vee c$: haben b oder c den Wert 1, dann muss $a = 1$ sein, da $a \leftrightarrow a \vee b$

- ist $b = c = 0$, dann ist $a = 0$, also $\overline{a} = 1$

$a \vee \overline{b} \vee \overline{c}$: hat $\overline{b}$ oder $\overline{c}$ Wert 1, dann ist $a = 0$, da $a \leftrightarrow (a \vee c)$.

- ist $\overline{b} = \overline{c} = 0$, dann ist $a = a$

Für die Klauseln mit 2 Literalen nehme neue Variable Z hinzu und bringe Z in die Klauseln ein:

$$a \vee \bar{b} \rightarrow a \vee \bar{b} \vee z$$

$$a \vee \bar{c} \rightarrow a \vee \bar{c} \vee z$$

Sei H die so entstehende Formel.

Dann gilt: $F \in SAT \leftrightarrow G \in 3 - SAT$

$F \in SAT \leftrightarrow G \in NAE - SAT$.

" $\Rightarrow$ ": Sei $a = (a_1, \dots, a_n)$ eine erfüllende Belegung von G .

Dann wähle $z = 0$.

Dann ist $(a_1, \dots, a_n, 0)$ erfüllende Belegung von H .

" $\Leftarrow$ ": Sei $(a_1, a_2, \dots, a_n, a_{n+1})$ erfüllende Belegung von H im NAE-Sinn.

1. Fall: $a_{n+1} = 0$ dann ist, $a = (a_1, \dots, a_n)$ erfüllende Belegung von G

2. Fall: $a_{n+1} = 1$ Es ist auch $(\bar{a}_1, \bar{a}_2, \dots, \bar{a}_n, \bar{a}_{n+1})$ eine erfüllende Belegung von H im NAE-Sinn.

$$(0, 0, 1) \rightarrow (1, 1, 0)$$

Jetzt ist $t = 0$. Dann nach Fall 1.

NP-Vollständig sind: $SAT, 3 - SAT, NAE - SAT$

7 k-Färbbarkeit

gegeben: $G = (V, E)$ ungerichtet

gefragt: $\exists f : V \rightarrow \{1, \dots, k\}$ mit $f(u) \neq f(v) \quad \forall (u, v) \in E$

Satz: 3-Färbbarkeit ist NP-Vollständig

Beweis in NP: gebe nichtdeterministisch jedem Knoten eine der 3 Farben. Dann prüfe, ob es eine 3-Färbung ist.

Beachte: Suchraum aller Zuordnungen von Farben zu V hat Größe 3^n

Es gibt k^n Funktionen $f : V \rightarrow \{1, \dots, k\}$

Zeige: $NAE - SAT \leq^P$ 3-Färbbarkeit.

gegeben: F in 3-KNF

Konstruiere einen Graph G , so dass $F \in NAE - SAT \Leftrightarrow G$ 3-Färbbar

Bsp.: $F = (x_1 \vee \bar{a}_x \vee x_3) \wedge \dots = c_1 \wedge c_2 \wedge \dots \wedge c_m$

1) Sei $F \in NAE - SAT$ und $a = (a_1, a_2, \dots, a_n)$ eine erfüllende Belegung für F . Dann ist G 3-Färbbar.

x_i Farbe a_i und $\bar{x}_i$ Farbe $\bar{a}_i$.

Bei den Klauseln: pro $\triangle$:

- wähle ein Literal mit Wert 1 und gebe Knoten die Farbe 1
- wähle ein Literal mit Wert 0 und gebe Knoten Farbe 0
- Knoten vom dritten Literal bekommt Farbe 2.

2) Sei G 3-Färbbar:

Der Wurzelknoten habe Farbe 2 (sonst permutiere Farben in G).

Sei a_i die Farbe von x_i im oberen Teil, d.h. $a_i \in \{0, 1\}$. Belege Variable x_i in F mit a_i .

Dann ist $a = (a_1, \dots, a_n)$ eine erfüllende Belegung von F .

In Klausel- $\triangle$ gibt es alle 3 Farben. Literal mit Farbe 0 hat Wert 0 in der Klausel, Literal mit Farbe 1 hat Wert 1 in der Klausel.

$\Rightarrow F$ ist erfüllt.

G bipartit.

$\Leftrightarrow G$ 2-färbbar

$\Leftrightarrow$ alle Kreise in G haben gerade Länge.

Allgemein für 2-färbbar:

1) starte mit beliebigem Knoten v und gebe v Farbe 1

2) Wiederhole:

gebe Nachbarn von v andere Farbe, wähle nächstes v .

Trifft man dabei auf einen bereits gefärbten Knoten mit anderer Farbe als er jetzt bekommen sollte, dann ist G nicht 2-färbbar. $\Rightarrow$ 2-Färbbarkeit $\in P$

Satz: HAM ist NP-Vollständig

Zeige: $3-SAT \leq^P HAM$

Gegeben: $F(x_1, \dots, x_n) = c_1 \wedge c_2 \wedge \dots \wedge c_n$

Konstruiere einen Graph G , so dass $F \in 3-SAT \Leftrightarrow G \in HAM$

G : Für jede Variable x_i konstruiere folgenden Teilgraph.

Soweit hat der Graph 2^n ham. Kreise und diese lassen sich je einer Belegung der Variablen von F zuordnen.

Bsp.: $F = (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3})$

1) Sei $F \in 3-SAT$ und $a = (a_1, \dots, a_n)$ eine erfüllende Belegung für F .

Dann hat G einen ham. Kreis:

gehe bei Teilgraph für x_i nach links, falls $a_i = 0$ und nach rechts, falls $a_i = 1$ ist.

- Kommt x_i in C_j vor und ist $a_i = 1$ und C_j noch nicht besucht worden, dann verzweige zu Klauselknoten C_j .
- kommt $\overline{x_i}$ in C_j vor und ist $a_i = 0$ und C_j noch nicht besucht, dann verzweige zu Knoten für C_j .

Da a F erfüllt, wird so jeder Klauselknoten erreicht.

2) Sei $G \in HAM$.

Es gibt nur ham. Kreise wie unter 1) beschrieben und diese entsprechen dann erfüllender Belegung von F .

$HAM_{ger} \leq^P HAM_{unger}$

Gegeben: G gerichtet.

Konstruiere G' ungerichtet, so dass:

$G \in HAM_{ger} \Leftrightarrow G' \in HAM_{unger}$

1) Ein ham. Kreis in G liefert direkt einen ham. Kreis in G' .

2) Ham. Kreise in G' müssen die Richtung von G einhalten (oder komplett in Gegenrichtung). Andernfalls bleibt man stecken, z. B. bei v^1 .

8 Unabhängige Knotenmenge (Independent Set (IS))

gegeben: $G = (V, E), k$

gefragt: $\exists I \subseteq V, |I| \geq k$

innerhalb von I gibt es keine Kanten.:w

$IS \in NP$: rate $I \subseteq V, |I| = k$

prüfe, ob I unabhängig

Deterministisch: durchlaufe alle $I \subseteq V$ mit $|I| = k$

Es gibt $\binom{n}{k}$ solche Teilmengen.

$$n^{\frac{n}{2}} \leq \binom{n}{k} \leq n^k$$

$$3 - SAT \leq^P IS$$

Sei $F(x_1, \dots, x_n) = c_1 \wedge c_2 \wedge \dots \wedge c_n$ in $3 - KNF$

Konstruiere Graph G und k , so dass gilt $F \in 3 - SAT \Leftrightarrow (G, k) \in IS$

Bsp.: $F = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$

Verbinde zwischen allen C_i und C_j $i \neq j$ komplementäre Literale

Beobachtung: eine unabhängige Menge I in G kann aus jedem Klauseldreieck höchstens 1 Knoten haben $\Rightarrow |I| \leq m$

Beh.: $F \in 3 - SAT \Leftrightarrow (G, m) \in IS$.

' $\Rightarrow$ ' Sei $a = (a_1, \dots, a_n)$ erfüllende Belegung von F .

Definiere I : wähle aus jedem Klausel- Δ einen Knoten, dessen Literal Wert 1 hat.

Im **Bsp.:** $a = (1, 0, 0)$

$\rightarrow I = \{x_1 \text{ aus } C_1, \overline{x_2} \text{ aus } C_2, \overline{x_3} \text{ aus } C_3\}$

Dann ist I unabhängig, da nie x_i und $\overline{x_i}$ ausgewählt wurden (sonst wäre a nicht erfüllbar) und nur diese durch Kanten verbunden sind.

' $\Leftarrow$ ' Sei I unabhängig, $|I| = m$.

Dann muss I nach obiger Beobachtung aus jedem Klausel- Δ genau einen Knoten haben.

Konstruiere Belegung a für F :

Wähle aus Klausel- Δ für C_i das Literal das in I ist und gebe Wert 1.

Noch nicht belegte Variablen können jetzt beliebig belegt werden. Diese Belegung erfüllt F , da keine komplementären Literale auftauchen.

9 Clique

gegeben: $G = (V, E), k$

gefragt: $\exists C \subseteq V, |C| \geq k$

so dass je 2 Knoten in C durch eine Kante verbunden sind.

Clique $\in NP$: rate $C \subseteq V, |C| = k$ prüfe, ob C Clique ist.

$IS =^P$ Clique

Sei G, k Eingabe für IS .

Konstruiere G', k' , so dass $(G, k) \in IS \Leftrightarrow (G', k') \in \text{Clique}$.

Der Komplementgraph von G ist $\overline{G} = (V, \overline{E})$

D. h. $\overline{E} = \{(u, v) \mid (u, v) \notin E\}$

Dann gilt: $(G, k) \in IS \Leftrightarrow (\overline{G}, k) \in \text{Clique}$

10 Knotenüberdeckung (Vortex Cover (VC))

gegeben: $G = (V, E), k$

gefragt: $\exists C \subseteq V, |C| \leq k$, so dass jede Kante in G mindestens einen Endpunkt in C hat.

$VC \in NP$: rate $C \subseteq V, |C| = k$ prüfe, ob C cover ist.

$IS \leq^P VC$

Sei G, k Eingabe für IS .

Ziel: konstruiere G', k' , so dass gilt: $(G, k) \in IS \Leftrightarrow (G', k') \in VC$

D. h. $G' = G$ und $C = V - I = \bar{I}$

Dann gilt: $(G, k) \in IS \Leftrightarrow (G, n - k) \in VC$

11 Dominating Set (DS)

Gegeben: $G = (V, E), k$

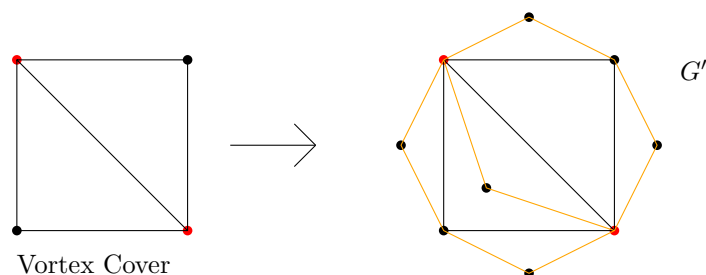
Gefragt: $\exists D \subseteq V, |D| \leq k$

so dass für jeden Knoten $v \in V$ gilt: v oder ein Nachbar von v ist in D .

$VC \leq^P DS$:

Sei G, k eine Eingabe für VC .

Ziel: konstruiere G' , so dass $(G, K) \in VC \Leftrightarrow (G', k) \in DS$



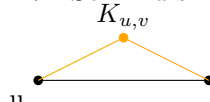
Für jede Kante (u, v) in G führe neuen Knoten $K_{(u,v)}$ ein und Kanten $(u, K_{(u,v)})$ und $(v, K_{(u,v)})$ (ungerichtet).

' $\Rightarrow$ ' Sei C VC in G , d. h. für jede Kante $(u, v) \in E$ ist $u \in C$ oder $v \in C$.

$\Rightarrow$ der neue Knoten $K_{u,v}$ hat einen Nachbarn in C

$\Rightarrow C$ ist d.s. in G' .

' $\Leftarrow$ ' Sei D d.s. in G'



u v Enthält D neue Knoten $K_{u,v}$ dann ersetze $K_{u,v}$ durch u in D . Danach entsteht D' . Dann ist D' (immer noch d.s.).

Dann gilt $|D'| \leq k$ und D' ist V.C. in G :

wäre für Kante (u, v) weder $u \in D'$ noch $v \in D'$, dann hätte $K_{u,v}$ keinen Nachbarn in D' .

Dann wäre D' kein D.S. $\nexists$

12 Subset Sum

Gegeben: $a_1, a_2, \dots, a_n, b$.

Gefragt: $\exists I \subseteq \{1, 2, \dots, n\} \quad \sum_{i \in I} a_i = b$

Bsp.: $1, 2, 5, 7, 9, \underbrace{14}_{=b} \in \text{Subset Sum}$

$4 \notin \text{Subset Sum}$

$\in NP$: rate I , teste Summe.

Subset Sum ist NP -Vollständig.

$3 - SAT \leq^P \text{Subset Sum}$:

Sei $F(x_1, \dots, x_n) = C_1 \wedge \dots \wedge C_m$.

$b = \underbrace{33\dots3}_m \mid \underbrace{11\dots1}_n$ dezimal.

Bsp.: $F = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge (\overline{x_2} \vee \overline{x_4} \vee x_5)$

Zahlen $y_1, \dots, y_n$ für $x_1, \dots, x_n$

$y_1 = \underbrace{100}_{m=3} \mid \underbrace{1000}_{n=5}$

x_1 kommt in C_1 vor

X_1 kommt nicht in C_2, C_3 vor.

$y_2 = 010 \mid 01000$

$y_3 = 100 \mid 00100$

$y_4 = 010 \mid 00010$

$y_5 = 001 \mid 00001$

Analog $Z_1, \dots, Z_n$ für $\overline{x_1}, \dots, \overline{x_n}$

$Z_1 = 010 \mid 10000$

$Z_2 = 101 \mid 01000$

$Z_3 = 000 \mid 00100$

$Z_4 = 001 \mid 00010$

$Z_5 = 000 \mid 00001$

Füllzahlen $f_1, \dots, f_m, g_1, \dots, g_m$

$f_1 = g_1 = 100 \mid 00000$

$f_2 = g_2 = 010 \mid 00000$

$f_3 = g_3 = 001 \mid 00000$

Es gibt nie Überträge, da im vorderen Teil kann man höchstens auf 3 Einsen kommen pro Klauselspalte mit y - und z -Zahlen + max. 2 Einsen mit f - und g -Zahlen.

$\Rightarrow$ Summe pro Klauselspalte ≤ 5

Im hinteren Teil höchstens 2 Einsen pro Spalte.

' $\Rightarrow$ ' Sei $a = (a_1, \dots, a_n)$ erfüllende Belegung von F .

Wähle y_i , falls $a_i = 1$,

wähle z_i , falls $a_i = 0$

Summe ergibt $s_1 s_2 \dots s_m 11 \dots 1$, wobei $s_i \in \{1, 2, 3\}$, da in jeder Klausel ≥ 1 Literal den Wert 1 hat.

Dann wähle passende Füllzahlen um auf b zu kommen.

D.h. f_i und g_i , falls $s_i = 1$ und f_i , falls $s_i = 2$.

' $\Leftarrow$ ' Habe Subset Sum Problem eine Lösung.

Wegen 1-er Block in b muss für jedes i entweder y_i oder z_i in der Lösung sein.

Def.: $a_i = 1$, falls y_i in Lösung,
 $a_i = 0$, falls z_i in Lösung.

Dann ist $a = (a_1, \dots, a_n)$ erfüllende Belegung von F:

Da im vorderen Teil mit den Füll-Zahlen maximal 2 erreicht werden kann, muss die Summe der ausgewählten y - und z -Zahlen ≥ 1 je Klauselspalte sein.

$\Rightarrow$ nach Konstruktion enthält also jede Klausel ≥ 1 erfülltes Problem.

13 Knapsack (Rucksack)

Gegeben: $s_1, \dots, s_n, w_1, \dots, w_n, s, w$

Gefragt: $\exists I \subseteq \{1, \dots, n\}$

$$\sum_{i \in I} s_i \leq S \text{ und } \sum_{i \in I} w_i \geq W$$

$\in NP$

Knapsack ist NP -Vollständig. Subset Sum $\leq^P$ Knapsack.

Sei $a_1, \dots, a_n, b$ Eingabe für Subset Sum. Konstruiere Eingabe für Knapsack:

$$S_i = a_i, S = b$$

$$w_i = a_i, W = b$$

$$\text{Dann ist } \sum_{i \in I} a_i = b \Leftrightarrow \sum_{i \in I} a_i \leq b \text{ und } \sum_{i \in I} a_i \geq b$$

14 Partition

Gegeben: $a_1, \dots, a_n$

Gefragt: $\exists I \subseteq \{1, \dots, n\}$

$$\sum_{i \in I} a_i = \sum_{i \notin I} a_i$$

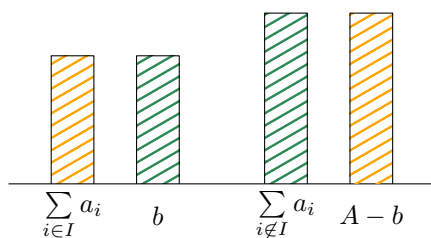
$\in NP$ -Vollständig

Subset Sum $\leq^P$ Partition.

Sei $a_1, a_2, \dots, a_n, b$ Eingabe für Subset Sum.

$$\text{Sei } A = \sum_{i=1}^n a_i$$

$$\text{Sei } I \subseteq \{1, \dots, n\} \text{ mit } \sum_{i \in I} a_i = b$$



$$\text{Dann ist } \sum_{i \in I} a_i + A - b = \sum_{i \notin I} a_i + b$$

D. h. Eingabe $a_1, \dots, a_n, b, A - b$ für Partition hat Lösung.

Aber: die hat immer die Lösung: $\underbrace{\sum_{i=1}^n a_i}_{=A} = \underbrace{b + (A - b)}_{=A}$

Stattdessen: $a_i, \dots, a_n, b+1, A-b+1$

Dann ist $\sum_{i \in I} a_i + (A-b+1) = \sum_{i \notin I} a_i + (b+1)$

Aber $\sum_{i=1}^n a_i = A \neq (b+1) + (A-b+1) = A+2$

' $\Leftarrow$ ' Sei I eine Lösung für das Partitionsproblem $a_1, \dots, a_n, \underbrace{b+1}_{=a_{n+1}}, \underbrace{A-b+1}_{=a_{n+2}}$

Da $(b+1) + (A-b+1) = A+2 > A$ kann nur eine von beiden Zahlen zu I gehören. Sei o. B. d. A.¹ sei $n+2 \in I$ (sonst betrachte $\bar{I}$).

Es gilt aber:

$$\sum_{i \in I} a_i + A - b + 1 = \sum_{i \notin I} a_i + b + 1$$

$$\Rightarrow \sum_{i \in I} a_i + A - \underbrace{\sum_{i \notin I} a_i}_{=\sum_{i \in I} a_i} = 2b$$

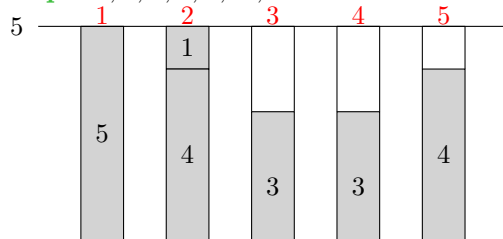
$$\Rightarrow \sum_{i \in I} a_i = b$$

14.1 Bin Packing

Gegeben: $a_1, \dots, a_n, B, k$

Gefragt: Kann man $a_1, \dots, a_n$ auf k Bins der Größe B verteilen?

Bsp.: 1, 3, 3, 7, 5, 4, $B = 5$



$\in NP$

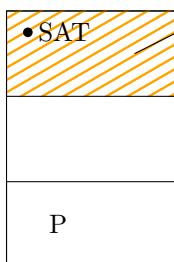
Partition $\leq^P$ Bin Packing.

Sei $a_1, \dots, a_n$ Eingabe für Partition.

Konstruiere Eingabe für Bin Packing:

$a_1, \dots, a_n, B = \frac{A}{2}, k = 2$ mit $A = \sum_{i=1}^n a_i$

$NP \subseteq EXP$



NP-vollständig

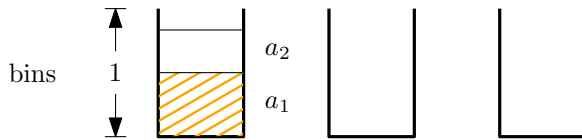
$SAT \equiv^P HAM \equiv^P VertexCover \equiv^P \dots$

¹o. B. d. A.: ohne Beschränkung der Allgemeinheit

15 Approximationsalgorithmen

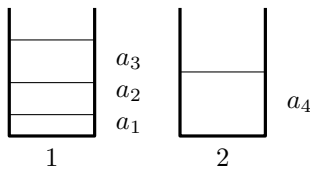
15.1 Bin Packing

geg. $a_1, a_2, \dots, a_n$



1. Strategie

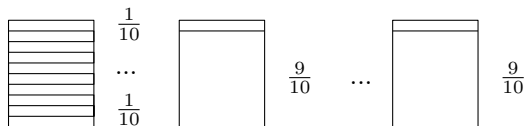
- sortiere Eingabe aufsteigend zu $a_1 \leq a_2 \leq \dots \leq a_n$
- fülle bin i so weit wie möglich, für $i = 1, 2, \dots$ mit Gegenständen in sortierter Reihenfolge



2. Strategie

- sortiere Eingabe absteigend zu $a_1 \geq a_2 \geq \dots \geq a_n$
- setze a_i in den ersten bin, in den es noch hinein passt, für $i = 1, \dots, n$.

1. Strategie



11 bins

2. Strategie

10 bins = optimal,
da alle bins voll sind.

Strategie 3: First Fit (FF)

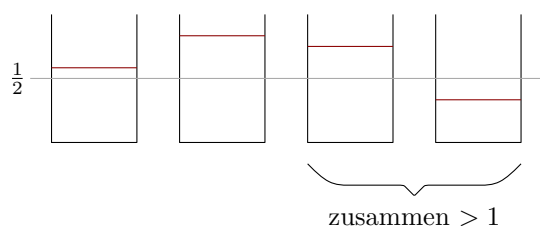
setze a_i in den ersten bin, in den es noch hinein passt, für $i = 1, 2, \dots, n$

Betrachtung zu FF:

Sei k^* die Anzahl der bins bei einer optimalen Lösung und k die Anzahl der bins, die FF auf $a_1, \dots, a_n$ benützt.

Es gilt: $\sum_{i=1}^n a_i \leq k^*$

Für FF gilt: alle bins, bis auf evtl. den letzten, haben Füllhöhe $> \frac{1}{2}$



$$\Rightarrow 2 \sum_{i=1}^n a_i > k$$

$$= \# \text{ bins, wenn alle genau } \frac{1}{2} \text{ voll sind.}$$

Folglich gilt:

$$k < 2 \sum_{i=1}^n a_i \leq 2k^*$$

$$\Rightarrow k < 2k^*$$

Man kann sogar zeigen: $l \leq \frac{17}{10}k^*$

Für FFD gilt sogar: $k \leq \frac{11}{9}k^*$

15.2 Multi Processor Scheduling (MPS)

geg. m Prozessoren, n Jobs

mit Laufzeiten $t_1, t_2, \dots, t_n$ und deadline D .

gefr. kann man die Jobs so auf die Prozessoren verteilen, dass alle Jobs in Zeit D fertig sind.

MPS ist NP-vollständig:

BinPacking $\leq^P$ MPS

$a_1, \dots, a_n, k, B \rightarrow t_i = a_i, i = 1, \dots, n, m = k, D = B$

Greedy-Strategie:

- Ordne Job i dem Prozessor zu, der aktuell die kleinste Ladung hat.

Sei T die maximale Laufzeit bei der Greedy-Strategie und sei T^* die Laufzeit bei der optimalen Lösung.

Für T gilt:

$$1) T^* \geq \frac{1}{m} \sum_{i=1}^n t_i$$

$$2) T^* \geq \max_i t_i$$

Für T gilt:

Nach Ausführung von Greedy habe Prozess i Ladung T_i .

Sei i_0 ein Prozessor mit $T_{i_0} = T$.

Sei Job j der letzte Job, der Prozess i_0 zugewiesen wurde.

D.h. alle anderen Prozessoren haben zu diesem Zeitpunkt bereits eine größere Ladung. Es gilt also $T_i \geq T_{i_0} - t_j \forall i$

$$\Rightarrow \underbrace{\sum_{i=1}^n T_i}_{=\sum_{i=1}^n t_i} \geq m \underbrace{(T_{i_0} - t_j)}_{=T}$$

$$\Rightarrow \sum_{i=1}^m T_i \geq m(T - t_j)$$

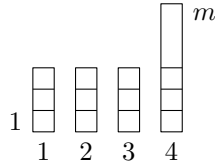
$$\Rightarrow \frac{1}{m} \sum_{i=1}^m t_i \geq T - t_j$$

$$\Rightarrow T \leq \underbrace{\frac{1}{m} \sum_{i=1}^m t_i}_{\leq T^*} + \underbrace{t_j}_{\leq T^*}$$

$$\Rightarrow \boxed{T \leq 2T^*}$$

Bsp.: $t_i = 1, i = 1, \dots, n-1$

$$t_n = m$$



wähle $n = m(m-1) + 1$

$$T = 2m - 1$$

$$T^* = m$$

Greedy-Sortiert:

- sortiere Jobs, so dass
 $t_1 \geq t_2 \geq \dots \geq t_n$
- dann wie Greedy

Lemma: $2t_{m+1} \leq T^*$

Beweis: Betrachte Jobs $1, \dots, m+1$

Nach Schubfachschluss hat ein Prozessor ≥ 2 Jobs (bei optimaler Lösung), d.h. Laufzeit $\geq$

$$\underbrace{t_i}_{\geq t_{m+1}} + \underbrace{t_j}_{\geq t_{m+1}}$$

Gleiche Überlegung wie oben:

Prozessor i_0 mit maximaler Ladung T .

Prozessor i_0 habe ≥ 2 Jobs.

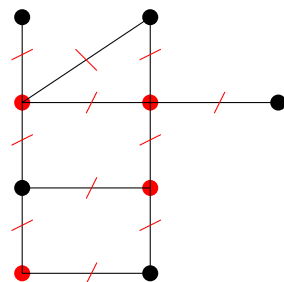
(Hätte Prozessor i_0 nur einen Job, dann ist $T = t_j = T^*$)

Es ist $j \geq m+1$, also $t_j \leq t_{m+1}$, wobei Job j wieder der letzte Job ist, der Prozessor i_0 zugewiesen wurde.

Abschätzung von vorher:

$$T \leq \underbrace{\frac{1}{m} \sum_{i=1}^n t_i}_{\leq T^*} + \underbrace{t_j}_{\leq t_{m+1} \leq \frac{T^*}{2}}$$

15.3 Vortex Cover

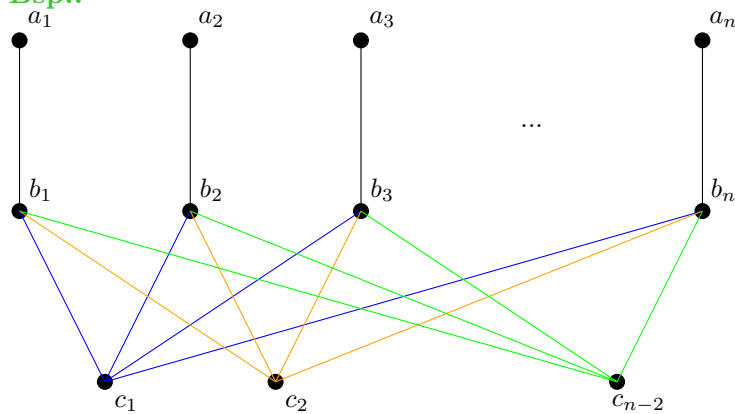


Greedy:

- wähle Knoten v mit maximalem Grad
- entferne v (mit seinen Kanten)

- wiederhole

Bsp.:



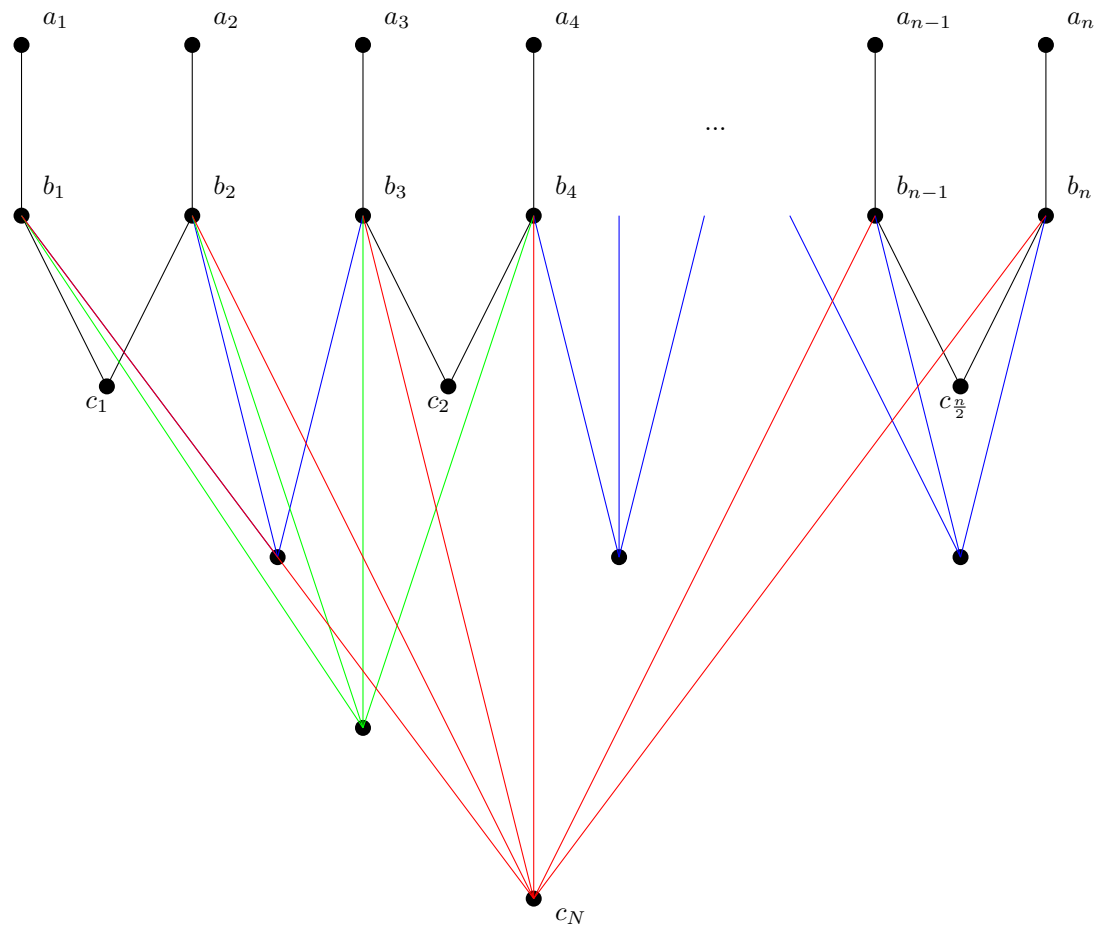
$$\deg(a_i) = 1$$

$$\deg(b_i) = (n - 2) + 1 = n - 1$$

$$\deg(c_i) = n$$

Greedy wählt $c_1, c_2, \dots, c_{n-2}, a_1, a_2, \dots, a_n$, insgesamt als $2n - 2$ Knoten.
Optimal wäre $b_1, \dots, b_n$ zu nehmen, also n Knoten.

Bsp.:



$$\begin{aligned}
 N &= \sum_{i=2}^n \left\lfloor \frac{n}{i} \right\rfloor \\
 &\geq \sum_{i=2}^n \left(\frac{n}{i} - 1 \right) \\
 &= \sum_{i=2}^n \frac{n}{i} - (n-1) \\
 &= n * \underbrace{\sum_{i=2}^n \frac{1}{i}}_{\geq \ln - 1} - (n-1)
 \end{aligned}$$

[Harmonische Reihe $\ln n \leq \sum_{i=1}^n \frac{1}{i} \leq \ln n + 1$]

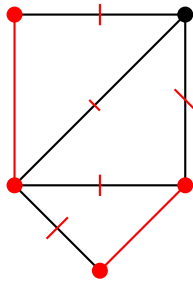
$$\Rightarrow N \geq n(\ln n - 1) - n + 1$$

$$= n \ln n - 2n + 1$$

greedy nimmt $c_N, c_{N-1}, \dots, 1, a_1, \dots, a_n$, also $N + n$ Knoten.

$$\text{Verhältnis: } \frac{N+n}{2} = \frac{n \ln n - n + 1}{n} = \ln n - 1 + \frac{1}{n}$$

D.h. die Strategie kann beliebig schlecht werden.



- 1) $C \leftarrow \emptyset$
- 2) wähle beliebige Kante (u, v)
- 3) $C \leftarrow C \cup \{u, v\}$
- 4) $G \leftarrow G - u, v$
- 5) wiederhole bis G leer ist (2-4).

Da eine vertex cover jede Kante abdecken muss, ist auch in einer optimalen Lösung mindestens einer der beiden Endpunkte u oder v enthalten.

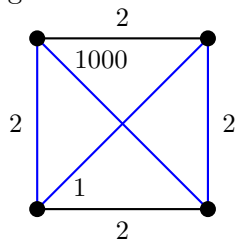
$$\underbrace{|C|}_{\text{vc. vom Approx Alg.}} \leq 2 \underbrace{|C^*|}_{\text{optimale Lösung}}$$

Besser: mit Approx Faktor $2 - \frac{2}{\sqrt{u}}$ für eine Konstante c .

15.4 Traveling Salesman Problem

15.4.1 Nearest Neighbor (NN)

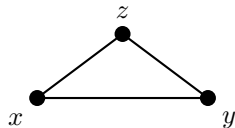
gehe immer zum nächstliegenden noch nicht besuchten Punkt.



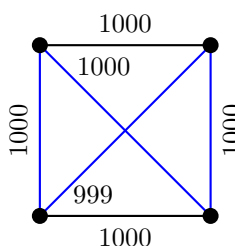
NN: 1005

Optimal: 8

Metrik: d erfüllt die Dreiecksungleichung



$$d(x, y) \leq d(x, z) + d(z, y)$$



NN: 4999

Optimal: 4000

Allgemein:

T = NN-Lösung

T^* = optimale Lösung

$$|T| \leq \log_n |T|^*$$

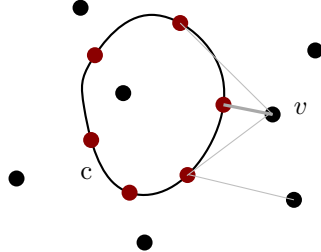
(TSP mit $\triangle$ -Ungleichung)

15.4.2 Nearest Insertion

bilde Kreis und füge Knoten ein

- Start mit 3 (beliebigen) Knoten und verbinde zu Dreieck.
- wähle Knoten v der noch nicht besucht ist und füge v in den Kreis ein.

Auswahl von v :



wähle v mit minimalem Abstand zum Kreis:

$$d(c, v) = \min_{u \in V} d(u, v) \quad u \in V$$

füge v so ein, dass sich der Kreis möglichst wenig verlängert. D. h. wähle Nachbar w von v so, dass

$$Cost(v) = d(u, v) + d(v, w) - d(u, w)$$

minimal ist.

Noch $\triangle$ -Ungleichung ist

$$\begin{aligned} d(v, w) &\leq d(u, v) + d(u, w) \\ \Rightarrow d(v, w) - d(u, w) &\leq d(u, v) \\ \Rightarrow cost(v) &\leq 2 * d(u, v) \end{aligned}$$

- Wiederhole Insertion-Schritt bis alle Knoten im Kreis sind.

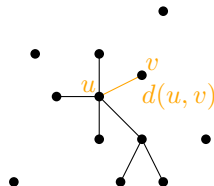
Beh.: Sei C der Kreis der durch NI berechnet wird und C^* eine optimale TSP-Tour.

Dann ist $d(C) \leq 2 * d(C^*)$

$$d(C) = \sum_{(u,v) \in C} d(u, v)$$

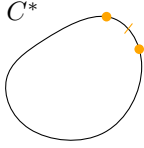
Beweis: Vergleich mit der Berechnung aufspannender Bäume, wähle jeweils Knoten v der am nächsten zum aktuellen Baum liegt.

NI wählt ebenfalls diesen Knoten v aus.



Aufspannender Baum T vergrößert sich um $d(u, v)$, Kreis vergrößert sich um $cost(v) \leq d(u, v)$.
 $\Rightarrow$ am ende gilt: $d(c) \leq 2 * d(T)$

Sei C^* Kreis minimaler Länge (= opt. TSP-Tour).



Lasse irgendeine Kante weg. Dann entsteht ein Baum T' , ein aufspannender Baum. Folglich ist $d(T) \leq d(T')$, da T minimal ist.

Außerdem ist $d(T') < d(C^*)$

$$\Rightarrow d(T) \leq d(C^*)$$

$$\Rightarrow d(C) \leq 2 * d(T) \leq 2 * d(C^*)$$

$$\Rightarrow \boxed{d(C) \leq 2 * d(C^*)} \quad \square$$

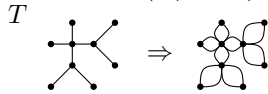
Varianten:

- Farthest Insertion (FI)
- Random Insertion (RI)

15.4.3 Christofides

1) Konstruiere minimal aufspannenden Baum T .

Dann ist $d(T) \leq d(C^*)$

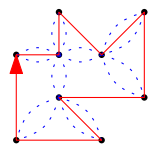


2) verdopple alle Kanten

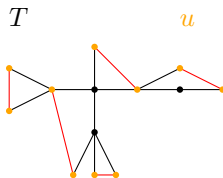
3) Konstruiere Euler-Tour E (mittels DFS)

$$D(E) = 2 * d(T)$$

4) Konstruiere daraus TSP-Tour durch Abkürzungen



Verbesserung



Erweitere T um Kanten, so dass jeder Knoten geraden Grad hat.

Betrachte $U =$ Knoten mit ungeradem Grad in T

Lemma: $|u|$ ist gerade

Beweis: $\sum_{v \in V} \text{grad}(v) = 2m$ ($m =$ Anzahl Kanten), da in der Summe jede Kante zweimal gezählt wird.

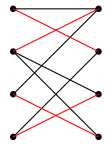
$$\Rightarrow \underbrace{2m}_{\text{gerade}} = \sum_{u \in U} \text{grad}(u) + \underbrace{\sum_{v \in V-U} \text{grad}(v)}_{\text{gerade}}$$

$$\Rightarrow \sum_{u \in U} \text{grad}(u) \text{ ist gerade}$$

$$\Rightarrow |n| \text{ gerade}$$

Verbinde Knoten in U paarweise, so dass die Summe der hinzugefügten Kanten möglichst klein ist.

Matching auf U .

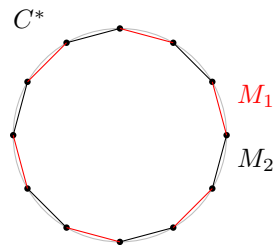


Perfect Matching M ist eine Menge von Kanten, so dass jeder Knoten zu genau einer Kante aus M gehört.

Es gibt effiziente Verfahren zu Berechnung von minimalem perfect Matching.

Dann weiter wie vorher: konstruiere Eulertour E und daraus TSP-Tour C .

Es gilt: $d(E) = d(T) + d(M) \geq d(C)$



Nehme jede zweite Kante

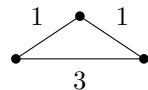
$\Rightarrow$ perfect Matching M_1

Rest ist ebenfalls perfect Matching M_2

$$\begin{aligned} \Rightarrow d(C^*) &= d(M_1) + d(M_2) \\ &\geq d(M) + d(M) \\ &= 2 * d(M) \\ \Rightarrow d(M) &= \frac{1}{2} * d(C^*) \\ \Rightarrow d(C) &\leq d(T) + d(M) \\ &\leq d(C^*) + \frac{1}{2} d(C^*) \\ &= \frac{3}{2} d(C^*) \end{aligned}$$

Δ TSP

TSP ohne Δ -Ungleichung:



Satz: Sei $c > 0$ beliebig.

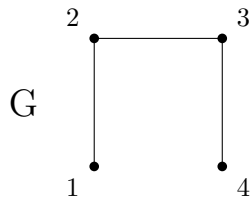
$P \neq NP \Rightarrow$ es gibt keine c -Approximation für TSP.

Beweis: Betrachte folgende Reduktion von $HAM \leq^P TSP$.

Sei $G = (V, E)$ ein Graph mit $|V| = n$.

Konstruiere Eingabe für TSP mit n Städte und Abständen d :

$$d(i, j) = \begin{cases} 1, & \text{falls } (i, j) \in E \\ c * n, & \text{sonst} \end{cases}$$



Dann gilt:

1. $G \in HAM \Rightarrow L^* = \text{optimale Länge einer Rundreise} = n$
2. $G \notin HAM \Rightarrow L^* \geq n - 1 + c * n > c * n$

Zeige: Wenn es einen Algorithmus A gibt, eine c -Approximation an TSP, dann ist $HAM \in P$ und folglich ist dann $P = NP$.

A ist c -Approximation, d.h. sei L die Länge der TSP-Tour, die $A(n, d)$ berechnet, dann gilt:
 $L \leq c * L^*$

Sei $G = (V, E)$ Eingabe für HAM . Reduziere G wie oben zu (n, d) . Sei $L = \text{Länge von } A(n, d)$.

Dann gilt:

1. $G \in HAM \Rightarrow L^* = n \Rightarrow L \leq c * n$
2. $G \notin HAM \Rightarrow L^* > c * n \Rightarrow L > c * n$

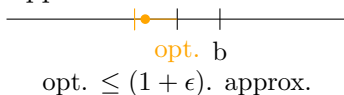
D.h. $G \in HAM \Leftrightarrow L \leq c * n$
 $\Rightarrow HAM \in P$

Bsp.: Vertex Cover

$P \neq NP \Rightarrow$ es gibt keine c -Approximation für $c < \frac{16}{15}$

15.5 Subset Sum

Für Subset Sum gibt es eine Approximation, die auf Eingabe $(a_1, \dots, a_n, b, \epsilon), \epsilon > 0$ eine $(1 + \epsilon)$ -Approximation liefert.



Optimierungsversion: $I \subseteq \{1, \dots, n\}$ sodass $\sum_{i \in I} a_i$ maximal und $\leq b$

16 PSPACE

$$P \subseteq NP \subseteq \underline{\underline{PSPACE}} \leq EXP$$

16.1 True Quantified Boolean Formular (TQBF)

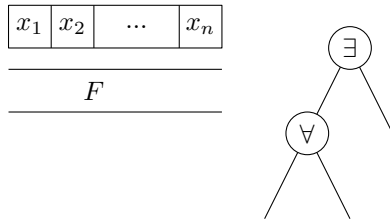
$$\forall x \exists y (x \wedge \exists z (y \wedge z))$$

Voll quantifizierte Formeln sind wahr oder falsch.

$$F(x_1, \dots, x_n) \in SAT$$

$$\Leftrightarrow \exists x_1 \exists x_2 \dots \exists x_n F(x_1, \dots, x_n) \in TQBF$$

$$TQBF \in PSPACE$$



Stelle Formel als Schaltkreis dar und werte mittels Tiefensuche aus. Es gilt: $TQBF$ ist $PSPACE$ -vollständig.

Jede QBF -Formel lässt sich äquivalent umschreiben in $Q_1 x_1 Q_2 x_2 \dots Q_n x_n F(x_1, \dots, x_n)$ mit $Q_1 = \exists, Q_2 = \forall, \dots$ alternierend und F in 3-KNF ist. (quantorenfrei)

Geographie

Gegeben: Graph G , gerichtet.

Knoten S (Startknoten).

Zwei Spieler ziehen Abwechselnd.

Spieler I started in S und zieht zu einem Nachbar von S . Dann zieht Spieler II, ...

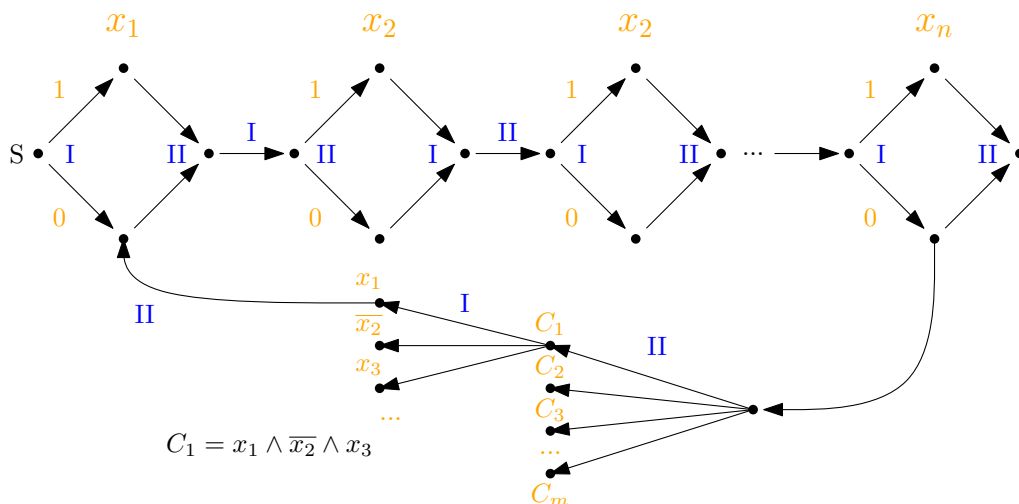
Dann immer abwechselnd.

Kein Knoten darf mehrfach besucht werden. Es verliert der Spieler, der nicht mehr Ziehen kann.

$GEO = \{(G, s) \mid \text{es gibt eine Gewinnstrategie für Spieler I}\}$

$$TQBF \leq^P GEO : F = \exists x_1 \forall x_2 \dots Q_n x_n (C_1 \wedge C_2 \wedge \dots \wedge C_m)$$

Konstruiere G, s :



GEO ist $PSPACE$ -Vollständig

Ebenso:

- *NFA*-Äquivalenz
- Reguläre Ausdrücke-Äquivalenz

$P \subseteq NP \subseteq PSPACE \subseteq EXP$

Es ist $P \neq EXP$