VL-16: NP-vollständige Zahlprobleme

(Berechenbarkeit und Komplexität, WS 2018)

Gerhard Woeginger

WS 2018, RWTH

Organisatorisches

- Nächste Vorlesung:
  Donnerstag, Januar 17, 12:30–14:00 Uhr, Aula

- Webseite:
  http://algo.rwth-aachen.de/Lehre/WS1819/BuK.php
  ($\longrightarrow$ Arbeitsheft zur Berechenbarkeit)
  ($\longrightarrow$ Arbeitsheft zur NP-Vollständigkeit)

Wdh.: NP-schwer & NP-Vollständig

Definition

- Ein Problem L heisst NP-schwer, falls $\forall L' \in NP : \; L' \leq_p L$
- Ein Problem L heisst NP-vollständig, falls $L \in$ NP und L NP-schwer.

Satz

Wenn L NP-vollständig ist, dann gilt: $\quad L \in P \;\Rightarrow\; P = NP$

Unter der Annahme $P \neq NP$ (Standardannahme) besitzt also kein NP-vollständiges Problem einen polynomiellen Algorithmus.

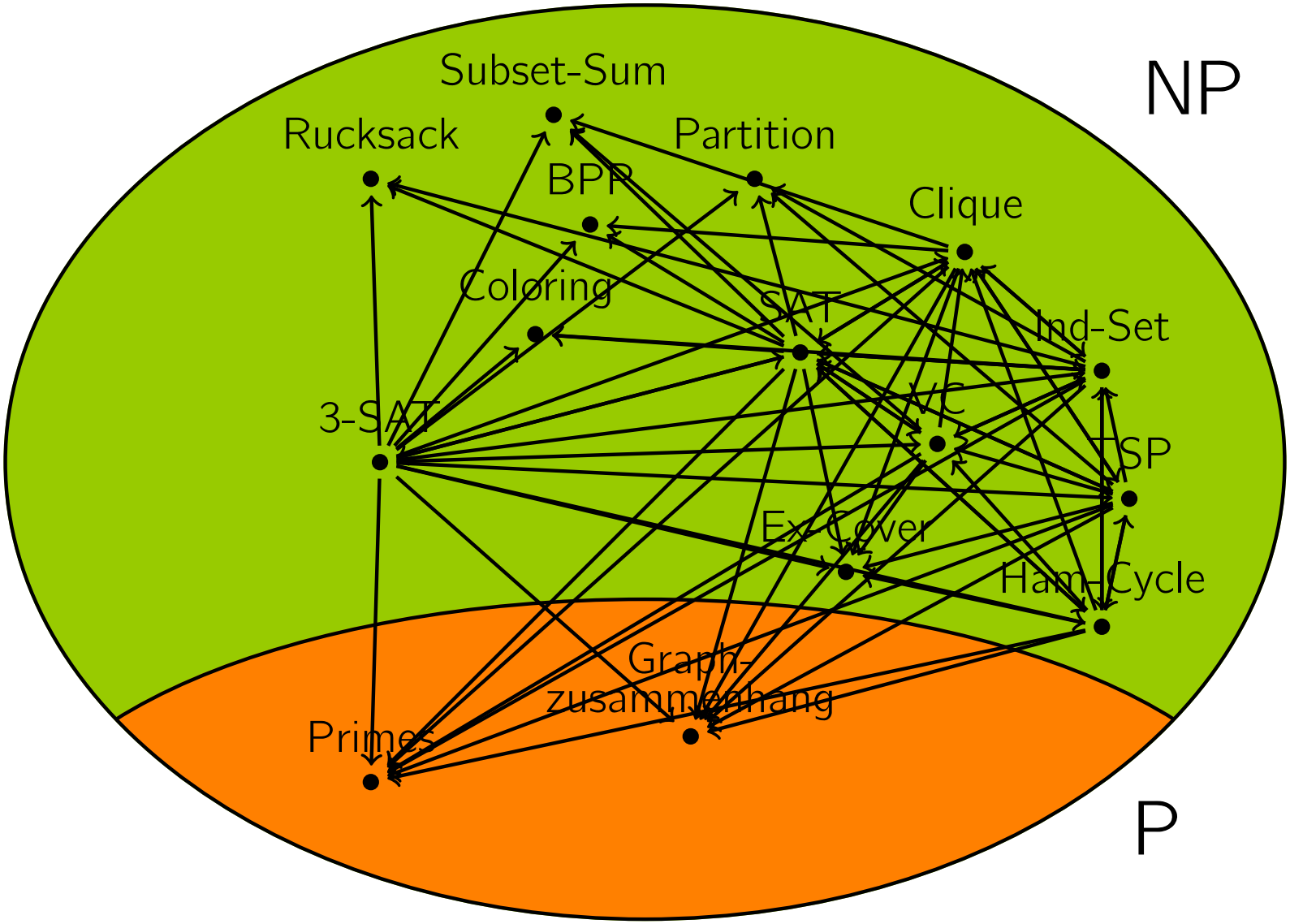
Wdh.: Kochrezept für NP-Vollständigkeitsbeweise

Kochrezept:

1. Man zeige $L \in NP$.

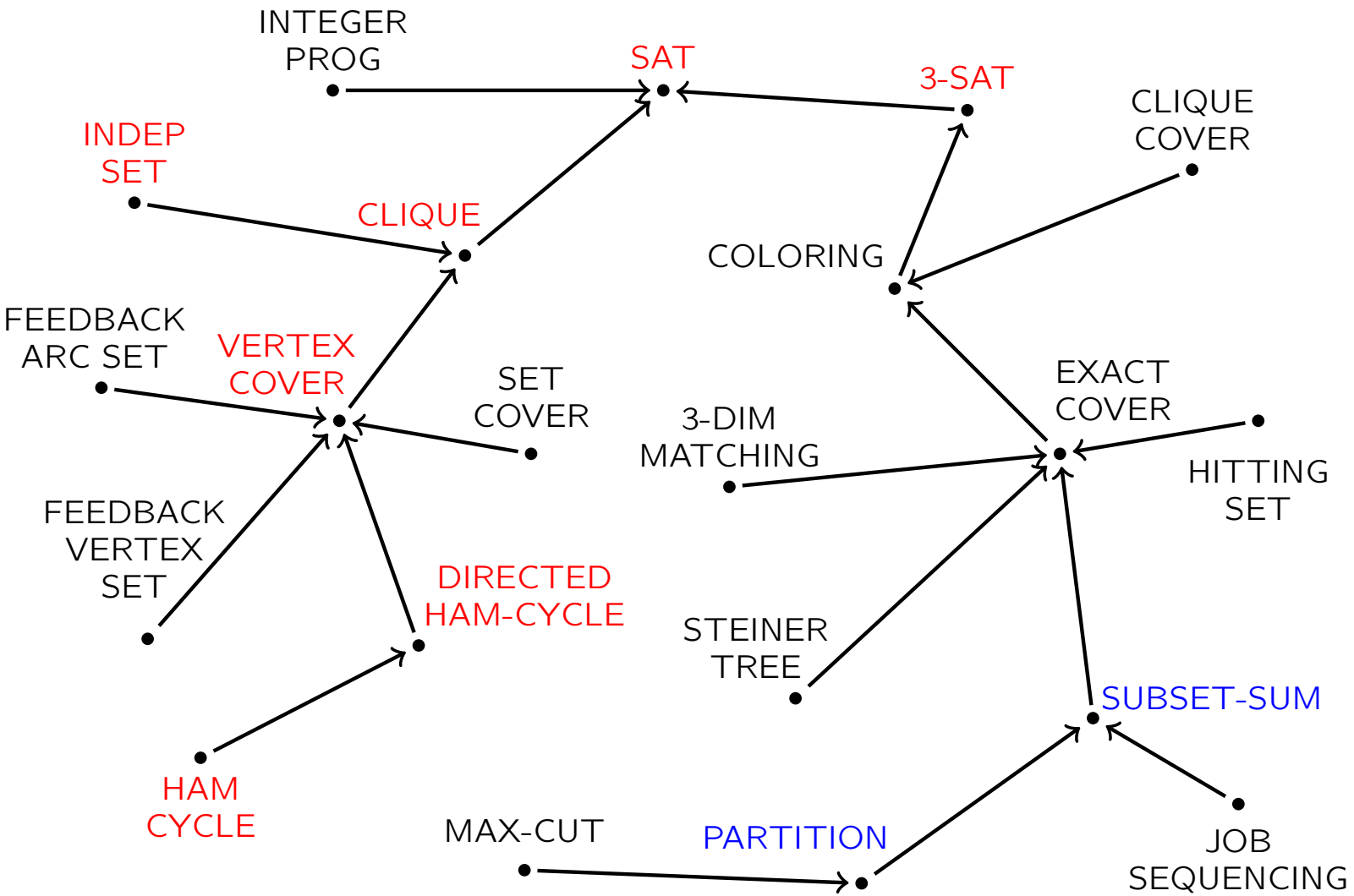
2. Man wähle eine NP-vollständige Sprache L^*.
3. **(Reduktionsabbildung):** Man konstruiere eine Funktion f, die Instanzen von L^* auf Instanzen von L abbildet.
4. **(Polynomielle Zeit):** Man zeige, dass f in polynomieller Zeit berechnet werden kann.
5. **(Korrektheit):** Man beweise, dass f tatsächlich eine Reduktion ist. Für $x \in \{0,1\}^*$ gilt $x \in L^*$ genau dann, wenn $f(x) \in L$.

Wdh.: Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $P \neq NP$ zu Grunde.

Wdh.: Landkarte mit Karp's 20 Reduktionen



Vorlesung VL-16
Einige NP-vollständige Zahlprobleme

- NP-Vollständigkeit von SUBSET-SUM
- NP-Vollständigkeit von PARTITION
- NP-Vollständigkeit von Bin Packing und Rucksack

- Pseudo-polynomielle Zeit
- Starke NP-Schwere
- Das THREE-PARTITION Problem

NP-Vollständigkeit von SUBSET-SUM

SUBSET-SUM: Definition

Problem: SUBSET-SUM

Eingabe: Positive ganze Zahlen $a_1, \ldots, a_n$; eine ganze Zahl b

Frage: Existiert eine Indexmenge $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = b$?

Beispiel: Eingabe für SUBSET-SUM

Zahlen $1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024$ und $b = 999$

Satz

SUBSET-SUM ist NP-vollständig.

SUBSET-SUM: Reduktion

Satz

SUBSET-SUM ist NP-vollständig.

Beweis:

- SUBSET-SUM liegt in NP
- Wir zeigen 3-SAT $\leq_p$ SUBSET-SUM
- Die Boole'sche Formel φ in 3-CNF sei eine Instanz von 3-SAT
- Die Formel hat Klauseln $c_1, \ldots, c_m$ mit den Variablen $x_1, \ldots, x_n$

In der Reduktion arbeiten wir mit Dezimalzahlen mit jeweils $n + m$ Ziffern. Die k-te Ziffer einer Zahl z bezeichnen wir dabei mit $z(k)$.

Reduktion (1a): Var-Zahlen / Definition

Wir definieren:

$$S^+(i) = \{j \in \{1,\ldots,m\} \mid \text{Klausel } c_j \text{ enthält Literal } x_i\}$$
$$S^-(i) = \{j \in \{1,\ldots,m\} \mid \text{Klausel } c_j \text{ enthält Literal } \bar{x}_i\}$$

Für jede Boolesche Variable x_i mit $1 \leq i \leq n$ erzeugen wir zwei entsprechende Var-Zahlen a_i^+ und a_i^- mit den folgenden Ziffern:

$$a_i^+(i) = 1 \quad \text{und für alle } j \in S^+(i): \; a_i^+(n+j) = 1$$
$$a_i^-(i) = 1 \quad \text{und für alle } j \in S^-(i): \; a_i^-(n+j) = 1$$

Alle anderen Ziffern in diesen Dezimaldarstellungen sind 0.

Reduktion (1b): Var-Zahlen / Beispiel

Als Beispiel betrachten wir die Formel

$$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4})$$

Die folgenden Var-Zahlen werden erzeugt:

$$\begin{aligned}
a_1^+ &= 100010 \\
a_1^- &= 100000 \\
a_2^+ &= 010011 \\
a_2^- &= 010000 \\
a_3^+ &= 001010 \\
a_3^- &= 001001 \\
a_4^+ &= 000100 \\
a_4^- &= 000101
\end{aligned}$$

Reduktion (2): Dummy-Zahlen

- Wir definieren für jede Klausel c_j zwei entsprechende Dummy-Zahlen d_j und d'_j.
- Dummy-Zahlen haben nur an der Ziffernposition $n + j$ eine Ziffer 1; alle anderen Ziffern sind 0.

Fortsetzung des Beispiels

Wir betrachten wieder die Formel

$$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4})$$

Die Dummy-Zahlen für die beiden Klauseln lauten dann:

$$\begin{aligned} d_1 &= 000010 \\ d'_1 &= 000010 \\ d_2 &= 000001 \\ d'_2 &= 000001 \end{aligned}$$

Reduktion (3): Zielsumme

Die Zielsumme b definieren wir folgendermassen:

- $b(k) = 1$ für $1 \leq k \leq n$,
- $b(k) = 3$ für $n + 1 \leq k \leq n + m$.

Abschluss des Beispiels

Wir betrachten wieder die Formel

$$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4})$$

Die Zielsumme lautet dann:

$$b \;=\; 111133$$

Reduktion (4): Illustration

Mögliche Zahlwerte für eine Formel mit n Variablen und m Klauseln:

	1	2	3	$\cdots$	n	$n+1$	$n+2$	$\cdots$	$n+m$
a_1^+	1	0	0	$\cdots$	0	1	0	$\cdots$	$\cdots$
a_1^-	1	0	0	$\cdots$	0	0	0	$\cdots$	$\cdots$
a_2^+	0	1	0	$\cdots$	0	0	1	$\cdots$	$\cdots$
a_2^-	0	1	0	$\cdots$	0	1	0	$\cdots$	$\cdots$
a_3^+	0	0	1	$\cdots$	0	1	1	$\cdots$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_n^+	0	0	0	$\cdots$	1	0	0	$\cdots$	$\cdots$
a_n^-	0	0	0	$\cdots$	1	0	1	$\cdots$	$\cdots$
d_1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
d_1'	0	0	0	$\cdots$	0	1	0	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
d_m	0	0	0	$\cdots$	0	0	0	$\cdots$	1
d_m'	0	0	0	$\cdots$	0	0	0	$\cdots$	1
b	1	1	1	$\cdots$	1	3	3	$\cdots$	3

Beweis (1): Keine Carry-Overs

- Für jede Dezimalstelle $i \in \{1, \ldots, n\}$ gilt: Nur zwei der Var-Zahlen und Dummy-Zahlen haben an dieser Stelle die Ziffer 1; alle anderen Zahlen haben an dieser Stelle die Ziffer 0.
- Für jede Dezimalstelle $i \in \{n+1, \ldots, n+m\}$ gilt: Nur fünf der Var-Zahlen und Dummy-Zahlen haben an dieser Stelle die Ziffer 1; alle anderen Zahlen haben an dieser Stelle die Ziffer 0.

Beobachtung: Keine Carry-Overs

Wird eine beliebige Menge von Var-Zahlen und Dummy-Zahlen addiert, so tritt von keiner Dezimalstelle zur nächsten ein Additionsübertrag auf.

Beweis (2): Laufzeit der Reduktion

- Die SAT Instanz φ besteht aus n Variablen und m Klauseln.
  Die Eingabelänge ist $\geq m + n$.
- Die konstruierte SUBSET-SUM Instanz besteht aus $2n + 2m + 1$
  Dezimalzahlen mit je $m + n$ Dezimalstellen.
- Die Reduktion wird in polynomieller Zeit $O\left((m + n)^2\right)$ durchgeführt.

Beweis (3a): Korrektheit

Lemma A: Formel φ erfüllbar $\Rightarrow$ SUBSET-SUM Instanz ist lösbar

Es gibt eine erfüllende Belegung x^* für die Formel φ.

- Falls $x_i^* = 1$, so wählen wir a_i^+ aus; andernfalls wählen wir a_i^-
- Die Summe der ausgewählten Var-Zahlen bezeichnen wir mit A
- Da für jedes $i \in \{1, \ldots, n\}$ entweder a_i^+ oder a_i^- ausgewählt wurde, gilt $A(i) = 1$
- Ausserdem gilt $A(n+j) \in \{1, 2, 3\}$ für $1 \leq j \leq m$, weil in jeder Klausel ein oder zwei oder drei Literale erfüllt sind.
- Falls $A(n+j) \in \{1, 2\}$, so wählen wir zusätzlich d_j und/oder d'_j aus, um die Ziffer 3 an Ziffernposition $n+j$ der Summe zu erhalten.

Also gibt es eine Teilmenge mit der gewünschten Zielsumme b.

Beweis (3b): Korrektheit

Lemma B: SUBSET-SUM Instanz ist lösbar $\Rightarrow$ Formel φ erfüllbar

Es gibt eine Teilmenge K_A der Var-Zahlen (mit Summe A) und eine Teilmenge K_D der Dummy-Zahlen (mit Summe H), die sich zur Zielsumme b aufaddieren; also: $A + H = b$.

- Die Menge K_A enthält für jedes $i \in \{1, \ldots, n\}$ genau eine der beiden Var-Zahlen a_i^+ und a_i^-; andernfalls wäre $A(i) \neq 1$.
- Wir setzen $x_i = 1$ falls $a_i^+ \in K_A$, und andernfalls $x_i = 0$.
- Es gilt $A(n+j) \geq 1$ für $1 \leq j \leq m$.
  Ansonsten wäre $A(n+j) + H(n+j) \leq A(n+j) + 2 < 3$.
- Dadurch ist sichergestellt, dass in jeder Klausel mindestens eines der Literale den Wert 1 hat.

Die Formel φ ist also erfüllbar.

NP-Vollständigkeit von PARTITION

PARTITION: Definition

Problem: PARTITION

Eingabe: Positive ganze Zahlen $a'_1, \ldots, a'_n$; mit $\sum_{i=1}^{n} a'_i = 2A'$

Frage: Existiert eine Indexmenge $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a'_i = A'$?

PARTITION ist der Spezialfall von SUBSET-SUM mit $b := (\sum a_i)/2$

Satz

PARTITION ist NP-vollständig.

Beweis:

- PARTITION liegt in NP
- Wir zeigen SUBSET-SUM $\leq_p$ PARTITION

PARTITION: Reduktion

- Es sei $a_1, \ldots, a_n \in \mathbb{N}$ und $b \in \mathbb{N}$ eine beliebige Instanz von SUBSET-SUM
- Es sei $S := \sum_{i=1}^{n} a_i$, und o.B.d.A. gilt $S \geq b$

Wir bilden diese SUBSET-SUM Instanz auf eine PARTITION Instanz ab, die aus den folgenden $n+2$ Zahlen $a'_1, \ldots, a'_{n+2}$ besteht:

- $a'_i = a_i$ für $1 \leq i \leq n$
- $a'_{n+1} = 2S - b$ und $a'_{n+2} = S + b$

Die Summe dieser $n+2$ Zahlen beträgt $\sum_{i=1}^{n+2} a'_i = 4S$.
Daher gilt $A' := 2S$ für die PARTITION Instanz.

Die Reduktion wird in polynomieller Zeit durchgeführt.

Beweis (1): Korrektheit

Lemma A: SUBSET-SUM Instanz lösbar $\Rightarrow$ PARTITION Instanz lösbar

- Wenn es in der SUBSET-SUM Instanz eine Teilmenge der Zahlen $a_1, \ldots, a_n$ mit der Summe b gibt, so haben die enstprechenden Zahlen $a'_1, \ldots, a'_n$ in der PARTITION Instanz ebenfalls die Summe b.
- Wir fügen die Zahl $a'_{n+1} = 2S - b$ zu dieser Teilmenge dazu und erhalten eine Teilmenge mit der gewünschten Zielsumme $A' = 2S$.

Beweis (2): Korrektheit

Lemma B: PARTITION Instanz lösbar $\Rightarrow$ SUBSET-SUM Instanz lösbar

- In der Lösung der PARTITION Instanz sind die beiden Zahlen $a'_{n+1} = 2S - b$ und $a'_{n+2} = S + b$ nicht in derselben Teilmenge, da $a'_{n+1} + a'_{n+2} = 3S > 2S = A'$ gilt.
- Eine der Teilmengen besteht aus $a'_{n+1} = 2S - b$ und einer Teilmenge der Zahlen $a'_1, \ldots, a'_n$ mit Gesamtsumme $A' = 2S$.
- Die entsprechenden Zahlen in der SUBSET-SUM Instanz haben dann die Summe b.

NP-Vollständigkeit von Bin Packing und Rucksack

Bin Packing

Beim Bin Packing sollen n Objekte mit Gewichten $w_1, \ldots, w_n$ auf eine möglichst kleine Anzahl von Kisten mit Gewichtslimit B verteilt werden.

Problem: Bin Packing (BPP)

Eingabe: Zahlen B und $w_1, \ldots, w_n \in \{1, \ldots, B\}$; eine Schranke γ

Frage: Können Objekte mit den gegebenen Grössen $w_1, \ldots, w_n$ in γ Kisten der Grösse B gepackt werden?

Satz

Bin Packing ist NP-vollständig.

Beweis:

- Wir zeigen PARTITION $\leq_p$ Bin Packing
- Wir setzen $\gamma = 2$, und $w_i = a'_i$ für $1 \leq i \leq n$, und $B = A'$ $\square$

Rucksack

Beim Rucksack Problem sollen Objekte ausgewählt werden, die in einen Rucksack mit Gewichtsschranke B passen und den Gesamtprofit maximieren.

Problem: Rucksack / Knapsack (KP)

Eingabe: Natürliche Zahlen $w_1, \ldots, w_n$, $p_1, \ldots, p_n$, B, γ

Frage: Existiert eine Teilmenge der Objekte mit Gesamtgewicht höchstens B und Gesamtprofit mindestens γ?

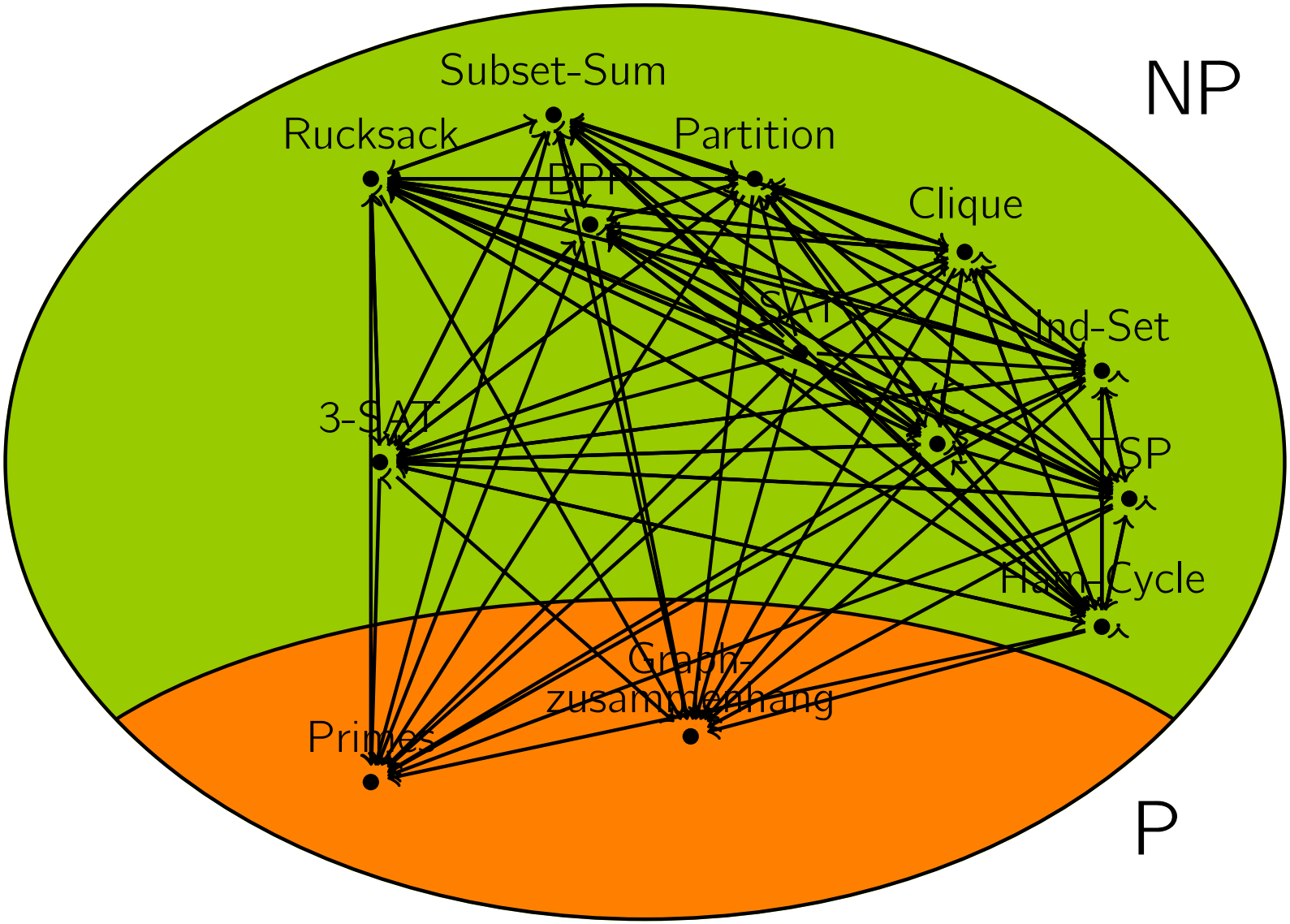
Satz

Rucksack ist NP-vollständig.

Beweis:

- Wir zeigen SUBSET-SUM $\leq_p$ Rucksack
- Wir setzen $w_i = a_i$ und $p_i = a_i$ für $1 \leq i \leq n$, und $B = \gamma = b$ $\square$

Die Komplexitätslandschaft



Warnung: Dieser Abbildung liegt die Annahme $P \neq NP$ zu Grunde.

Pseudo-polynomielle Zeit und Starke NP-Schwere

Kodierungslänge (1)

- Es sei X ein algorithmisches Problem
- Die Laufzeit eines Algorithmus A für Problem X messen wir in der Kodierungslänge der Instanzen I von X
- Die Kodierungslänge $|I|$ ist die Anzahl der Symbole in einer "vernünftigen" Beschreibung der Instanz I
- Kleine (polynomiell grosse) Änderungen in derartigen Beschreibungen sind für unsere Definitionen / Sätze / Beweise / Resultate irrelevant

Kodierungslänge (2)

Beispiel: Ungerichtete Graphen

Vernünftige Beschreibungen von ungerichteten Graphen $G = (V, E)$ sind

- Adjazenzlisten mit Länge $\ell_1(G) = O(|E| \log |V|)$
- Adjazenzmatrizen mit Länge $\ell_2(G) = O(|V|^2)$

Es gilt:

- $\ell_1(G)$ ist polynomiell beschränkt in $\ell_2(G)$
- $\ell_2(G)$ ist polynomiell beschränkt in $\ell_1(G)$

Kodierungslänge (3)

Beispiel: Natürliche Zahlen

Vernünftige Beschreibungen von natürlichen Zahlen n sind

- Dezimaldarstellung mit Länge $\approx \log_{10} n$
- Binärdarstellung mit Länge $\approx \log_2 n$
- Oktaldarstellung mit Länge $\approx \log_8 n$
- Hexadezimaldarstellung mit Länge $\approx \log_{16} n$

Für alle reellen Zahlen $a, b > 1$ gilt: $\log_a n = \log_a b \cdot \log_b n$
Die verschiedenen Kodierungslängen unterscheiden sich daher nur um einen konstanten Faktor.

Anmerkung

Die Zahl n stellt den Wert n mit Kodierungslänge $O(\log n)$ dar.
Der Wert hängt also exponentiell von der Kodierungslänge ab.

Zahlenwert versus Kodierungslänge

Definition: Number

Für eine Instanz I eines Entscheidungsproblems bezeichnen wir mit $Number(I)$ den Wert der grössten in I vorkommenden Zahl.

Beispiel

- Für eine TSP Instanz I ist $Number(I)$ der Wert der grössten Städtedistanz $max_{i,j} d(i,j)$ oder der Wert γ.
- Für eine SUBSET-SUM Instanz I ist $Number(I)$ das Maximum der Zahlen $a_1, \ldots, a_n$ und b.
- Für eine SAT Instanz I ist $Number(I)$ das Maximum der Zahlen n und m. (Ergo: $Number(I) \leq |I|$.)

Der Parameter $Number(I)$ ist nur für Probleme relevant, in denen Distanzen, Kosten, Gewichte, Längen, Profite, Zeitintervalle, Abstände, etc eine Rolle spielen.

Pseudo-polynomielle Zeit (1): Definition

Definition: Pseudo-polynomielle Zeit

Ein Algorithmus A löst ein Problem X in pseudo-polynomieller Zeit, falls die Laufzeit von A auf Instanzen I von X polynomiell in $|I|$ und $Number(I)$ beschränkt ist.

Satz

Die Probleme SUBSET-SUM, PARTITION und Rucksack sind pseudo-polynomiell lösbar.

Pseudo-polynomielle Zeit (2): Dynamisches Programm

Problem: SUBSET-SUM

Eingabe: Positive ganze Zahlen $a_1, \ldots, a_n$; eine ganze Zahl b

Frage: Existiert eine Indexmenge $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = b$?

Satz

SUBSET-SUM ist in pseudo-polynomieller Zeit $O(n \cdot b)$ lösbar.

Beweis:

- Dynamische Programmierung: Für $k = 0, \ldots, n$ und $c = 0, \ldots, b$ setzen wir $F[k, c] = \mathit{TRUE}$ genau dann, wenn es eine Indexmenge $I \subseteq \{1, \ldots, k\}$ mit $\sum_{i \in I} a_i = c$ gibt.
- $F[0, c] = (c == 0)$ für $c = 0, \ldots, b$
  $F[k, c] = F[k-1, c - a_k] \vee F[k-1, c]$
- Schlussendlich findet man die Antwort in $F[n, b]$

Starke NP-Schwere (1): Definition

Definition: Stark NP-schwer (engl.: NP-hard in the strong sense)

Ein Entscheidungsproblem X ist stark NP-schwer,
wenn es ein Polynom $q : \mathbb{N} \to \mathbb{N}$ gibt, sodass die Restriktion X_q von X auf Instanzen I mit $Number(I) \leq q(|I|)$ NP-schwer ist.

Also: Das Problem X ist sogar dann NP-schwer, wenn alle Zahlenwerte in der Instanz I nur polynomiell gross (gemessen in $|I|$) sind.

Übung (unter der Annahme P≠NP)

Welche der folgenden Probleme sind stark NP-schwer?

- SAT und 3-SAT
- CLIQUE, INDEP-SET, Vertex Cover
- Ham-Cycle und TSP
- SUBSET-SUM und PARTITION
- Bin Packing

Starke NP-Schwere (2)

Satz

Es sei X ein stark NP-schweres Entscheidungsproblem.
Falls X pseudo-polynomiell lösbar ist, so gilt P=NP.

Also: Pseudo-polynomiell und stark NP-schwer schliessen einander aus (unter unserer Standardannahme P≠NP)

Beweis:

- X ist stark NP-schwer
- Ergo gibt es ein Polynom $q : \mathbb{N} \to \mathbb{N}$, für das die Restriktion X_q von X auf Instanzen I mit $Number(I) \leq q(|I|)$ NP-schwer ist.
- Ein pseudo-polynomieller Algorithmus A für X hat Laufzeit polynomiell beschränkt in $|I|$ und $Number(I)$
- Wendet man Algorithmus A auf X_q an, so ist die Laufzeit polynomiell beschränkt in $|I|$ und $q(|I|)$, und daher polynomiell beschränkt in $|I|$
- (X_q NP-schwer) und (X_q polynomiell lösbar) $\Rightarrow$ P=NP

Starke NP-Schwere (3)

Problem: THREE-PARTITION

Eingabe: Positive ganze Zahlen $a_1, \ldots, a_n$, $b_1, \ldots, b_n$, und $c_1, \ldots, c_n$ mit $\sum_{i=1}^{n}(a_i + b_i + c_i) = nS$

Frage: Gibt es zwei Permutationen α, β von $1, \ldots, n$, sodass $a_{\alpha(i)} + b_{\beta(i)} + c_i = S$ für $1 \leq i \leq n$ gilt?

Satz (ohne Beweis)

THREE-PARTITION ist stark NP-schwer.

Übung

Zeigen Sie: Bin Packing ist stark NP-schwer.