1.6 Komplexität

1.6.1 Einstieg

Wir haben bisher eine ganze Reihe von Algorithmen kennengelernt, die Knapsack unterschiedlich angehen:

(A) Heuristisch: Einfache "Probiermethoden", die oft ganz ordentliche Lösungen liefern - manchmal sogar optimale! (Beispiel: Greedy)

(B) Exakt: Algorithmen, die immer optimale Lösungen liefern, aber manchmal sehr lange dafür brauchen. (Beispiele: Dynamic Programming, Bruch- und Band)

(C) Approximiert: Algorithmen, die in polynomieller Zeit Lösungen liefern, die nicht unbedingt optimal sind, aber zumindest mit einer "Gütegarantie".

Gehört das noch besser? Können wir einen Algorithmus finden, der

1. immer
2. in polynomieller Zeit
3. eine optimale Lösung berechnet?

Klasse P
Gelöst das noch besser?

Gibt es einen Algorithmus, der

(1) immer für alle Instanzen
(2) in polynomieller Zeit
(3) eine optimale Lösung

berechnet?
**Definition 1.30**  (Klasse P)
Ein algorithmisches (oder logisches) Problem gehört zur Klasse P, wenn dafür ein "perfecter" Algorithmus existiert, der
(1) für jede Instanz
(2) in polynomieller Zeit
(3) eine optimale (oder korrekte) Lösung findet.

Schwieriger; d.h. potenziell größere Klasse:

**Definition 1.31**  (Klasse NP)
Ein algorithmisches (oder logisches) Problem gehört zur Klasse NP, wenn sich diese die Existenz einer Lösung in polynomieller Zeit nachprüfen lässt.

**Beobachtung 1.32**
Klar: SUBSET SUM \in NP
Unklar: SUBSET SUM \in P

Klar: KNAPSACK \in NP
Unklar: KNAPSACK \in P

**Problem 1.33**
P \neq NP ?
1.6.2 Ein Beispiel mit Logik

Beispiel 1.34

Wir betrachten die folgende Instanz mit $n=12$ und $z = p_i$

sowie $z = 111444$:

\[
\begin{array}{c}
z_1 = p_1 = 100110 \\
z_2 = p_2 = 100001 \\
z_3 = p_3 = 10101 \\
z_4 = p_4 = 10010 \\
z_5 = p_5 = 1001 \\
z_6 = p_6 = 1110 \\
z_7 = p_7 = 200 \\
z_8 = p_8 = 100 \\
z_9 = p_9 = 20 \\
z_{10} = p_{10} = 10 \\
z_{11} = p_{11} = 2 \\
z_{12} = p_{12} = 1 \\
\end{array}
\]

Gibt es $S = \{1, \ldots, 12\}$ mit $\sum_{i \in S} p_i = 111444$?

Man sieht:

1. Man muss $p_1$ oder $p_2$ auswählen, aber nicht beide in 1. Stelle
2. Man muss $p_3$ oder $p_4$ auswählen, aber nicht beide in 2. Stelle
3. Man muss $p_5$ oder $p_6$ auswählen, aber nicht beide in 3. Stelle
4. Man muss $p_1$ oder $p_3$ oder $p_6$ auswählen
   (dann kann man mit $p_3$ oder $p_6$ eine 4 erreichen!)
5. Man muss $p_1$ oder $p_4$ oder $p_6$ auswählen
   (mit $p_1$ oder $p_4$ erreicht man eine 4!)
6. Man muss $p_2$ oder $p_3$ oder $p_5$ auswählen
   (4 mit $p_2$ oder $p_5$)
Setzen wir einfach einmal die
Booleschen Variablen:

\[ X_1 := \begin{cases} 
1 & \text{P}_1 \text{ wird gewählt} \\
0 & \text{P}_1 \text{ wird nicht gewählt}
\end{cases} \]

\[ X_2 := \begin{cases} 
1 & \text{P}_2 \text{ wird gewählt} \\
0 & \text{P}_2 \text{ wird nicht gewählt}
\end{cases} \]

\[ X_3 := \begin{cases} 
1 & \text{P}_3 \text{ wird gewählt} \\
0 & \text{P}_3 \text{ wird nicht gewählt}
\end{cases} \]

Damit sehen wir:

Es gibt ein \( S \subseteq \{1, \ldots, 12\} \) mit \( \sum_{i \in S} P_i = 2 \)

\[ \iff \] Wir können die logischen Variablen \( x_1, x_2, x_3 \) so wählen, dass die folgende Formel erfüllt ist:

\[ (x_1 \lor x_2 \lor \overline{x_3}) \land (x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (\overline{x_1} \lor \overline{x_2} \lor x_3) \]

Wenn wir also ein \( S \) berechnen können,
können wir eine Lösung für die logische Formel bestimmen (und umgekehrt)!

Das geht für jede logische Formel vom Typ 3SAT!
**Definition 1.35** (3SAT) 

Gegeben: Eine logische Formel $\varphi$, bestehend aus:
- $m$ Klauseln $c_1, ..., c_m$, jeweils in der Form $c_i : (l_{i,1} \lor l_{i,2} \lor l_{i,3})$
- wobei jedes "Literal" $l_{i,k}$ eine negierte oder unnegierte Variable ist: $l_{i,k} \in \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}\}$
- $n$ logische Variable, $x_1, ..., x_n$

Gesucht: Eine Wahrheitsbelegung für die $n$ Variablen, so dass für jede Klaussel mindestens ein Literal wahr ist.

**Bemerkung 1.36**

Wenn eine 3SAT-Instanz eine erfüllende Wahrheitsbelegung hat, lässt sich diese solche leicht verifizieren.

Wenn eine 3SAT-Instanz keine erfüllende Wahrheitsbelegung hat, ist das nicht so ohne Weiteres schnell nachzuweisen.

**Problem 1.37** (3SAT ∈ P?)

Gibt es für 3SAT einen Algorithmus, der für jede Instanz $\varphi$ in polynomialer Zeit (in $m$ und $n$ von I) entscheidet, ob $\varphi$ erfüllbar ist?
Satz 1.38 \[ (\text{Knapsack} \in \mathbb{P} \Rightarrow \text{3SAT} \in \mathbb{P}) \]

Wenn Knapsack polynomial lösbar ist, dann ist auch 3SAT polynomial entscheidbar.

Beweis:

Analog zu Beispiel 1.34: Baue aus einer 3SAT-Instanz $I_{3sat}$ eine Instanz $I_{kn}$ von Knapsack mit folgenden Eigenschaften:

(A) Die Codierungsgröße von $I_{kn}$ ist polynomial beschränkt durch die Codierungsgröße von $I_{3sat}$.

(B) Es gibt für $I_{kn}$ eine Teilmenge mit Lösungswert 2

$\Rightarrow I_{3sat}$ ist erfüllbar.

Wenn wir einen polynomialen Algorithmus für Knapsack haben, dann können wir damit in polynomialer Zeit jede Instanz $I_{3sat}$ von 3SAT entscheiden:

(1) Baue zu $I_{3sat}$ eine Instanz $I_{kn}$ von Knapsack.

(2) Löse $I_{kn}$.

(3) Betrachte die Lösung und verwende Eigenschaft (B), um die Lösbarkeit von $I_{3sat}$ zu entscheiden.

Und jetzt der Knaller:

Korollar 1.39

Wenn Knapsack polynomial lösbar ist, dann gilt $P = \mathbb{NP}$. 
Denn:

**Satz 1.36 (Satz von Cook 1971)**

Wenn 3SAT polynomial lösbar ist, dann gilt P = NP.

**Beweisidee:**
Man kann zeigen, dass sich jedes Problem in NP als äquivalentes 3SAT-Problem codieren lässt.

**Definition 1.37**

1. Ein Problem \( \Pi \) in NP heißt NP-vollständig, wenn \( \Pi \in P \implies P = NP \) gilt.
2. Ein Problem \( \Pi \) heißt NP-schwer, wenn \( \Pi \in P \implies P = NP \) gilt.

**Also:**

**Korollar 1.38**

1. 3SAT ist NP-vollständig.
2. Knapsack ist NP-vollständig.
Idealre Finanzberater:
1. ehrlicher
2. intelligenter
3. Investmentbanker

Perfekter Algorithmus:
1. immer
2. schnell
3. optimale Lösung

In Zeiten der Finanzkrise:
Schnitt ist leer!

Bei NP-Vollständigkeit:
Schnitt ist leer!

Was tun in schwierigen Situationen?!

(A) Auf Glück vertrauen
(B) Hart arbeiten
(C) Erwartungen herabschreiben
(D) Mit dem Schicksal hedonistisch diskutieren

BWC
Inf
WInf
Jung

Hier:
(A) Nicht "immer" : Heuristiken
(B) Nicht "schnell" : Exakte Algorithmen
(C) Nicht "optimal" sondern "gut" : Approximationsalgorithmen
(D) Nicht NP-vollständig : Komplexitätsanalyse