1.6 KOMPLEXITÄT

1.6.1 Einstieg

Wir haben bislang 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
- Branch-and-Bound)

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

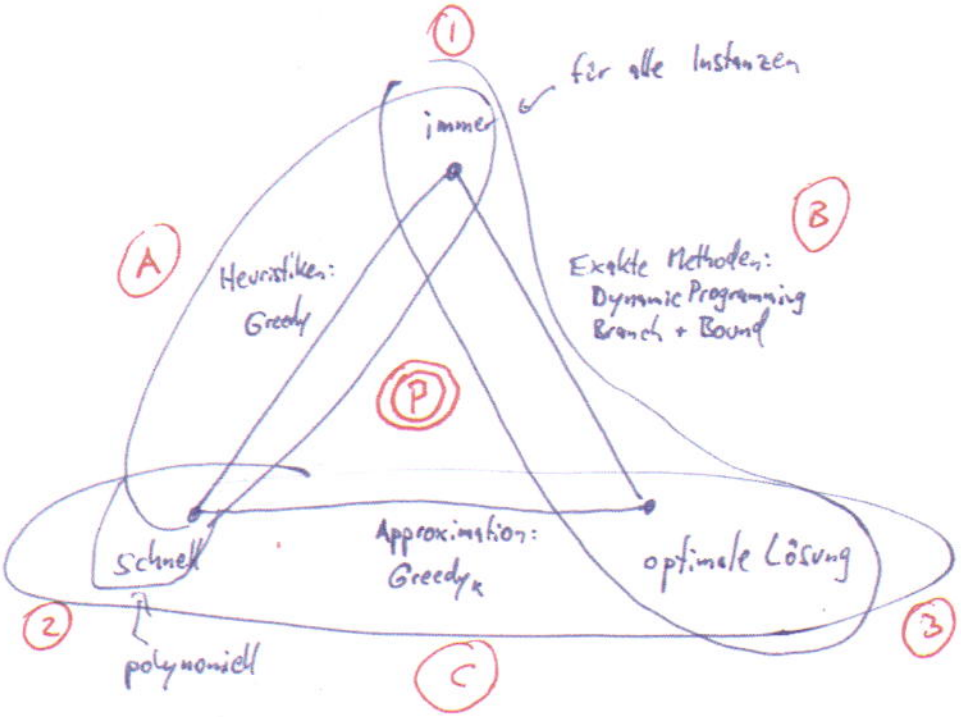
Geht das noch besser? Können wir einen „perfekten" Algorithmus finden, der

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

} Klasse P

berechnet?

Bild:



Geht 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 „perfekter" Algorithmus existiert, der

(1) für jede Instanz
(2) in polynomieller Zeit
(3) eine optimale (oder korrekte) Lösung findet.

Schwächer; d.h. potentiell größere Klasse:

Definition 1.31 (Klasse NP)

Ein algorithmisches (oder logisches) Problem gehört zur Klasse NP, wenn sich die Existenz einer Lösung in polynomieller Zeit Nachprüfen lässt.

Beobachtung 1.32

Klar: SUBSET SUM $\in$ NP
Unklar: SUBSET SUM $\in$ P

Lösungswert mindestens so groß wie eine Schranke!

Klar: KNAPSACK$_{\geq}$ $\in$ NP
Unklar: KNAPSACK$_{\geq}$ $\in$ 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 $z_i = p_i$ sowie $Z = 111444$:

$z_1 = p_1 =$	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

Besondere Struktur!

Gibt es $S \subseteq \{1, \dots, 12\}$ mit $\sum_{i \in S} p_i = 111444$? SUBSET SUM

Man sieht:

(1) Man muss p_1 ODER p_2 auswählen, aber nicht beide ← 1. Ziffer

(2) Man muss p_3 ODER p_4 auswählen, aber nicht beide ← 2. Ziffer

(3) Man muss p_5 ODER p_6 auswählen, aber nicht beide ← 3. Ziffer

(4) Man muss p_1 oder p_3 oder p_6 auswählen ← 4. Ziffer
(Dann kann man mit p_7 oder p_8 eine 4 erreichen!)

(5) Man muss p_1 oder p_4 oder p_6 auswählen ← 5. Ziffer
(mit p_7 oder p_{10} erreicht man eine 4!)

(6) Man muss p_2 oder p_3 oder p_5 auswählen ← 6. Ziffer
(4 mit p_{11} oder p_{12}.)

Setzen wir einfach einmal die Boolschen Variablen:

$$x_1 := \begin{cases} 1 & p_1 \text{ wird gewählt} \\ 0 & p_1 \text{ wird nicht gewählt} \end{cases}$$

$$x_2 := \begin{cases} 1 & p_3 \text{ wird gewählt} \\ 0 & p_3 \text{ wird nicht gewählt} \end{cases}$$

$$x_3 := \begin{cases} 1 & p_5 \text{ wird gewählt} \\ 0 & p_5 \text{ wird nicht gewählt} \end{cases}$$

Dann sehen wir: (SUBSET SUM)

Es gibt ein $S \in \{1, \dots, 12\}$ mit $\sum_{i \in S} p_i = Z$

$\Leftrightarrow$ Wir können die logischen Variablen x_1, x_2, x_3 so wählen, dass die Folgende Formel erfüllt ist (3SAT)

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

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)SFIABILITY)

Gegeben: Eine Boolsche Formel I, bestehend aus:

- m Klauseln $c_1, \dots, c_m$, jeweils der Form $c_j: (\ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3})$, wobei jedes „Literal" $\ell_{j,k}$ eine negierte oder unnegierte Variable ist:
$$\ell_{j,k} \in \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\}$$
- n Boolsche Variable, $x_1, \dots, x_n$

Gesucht: Eine Wahrheitsbelegung für die n Variablen, so dass für jede Klausel mindestens ein Literal WAHR ist

BEOBACHTUNG 1.36

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

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

PROBLEM 1.37 ($3SAT \in P$?)

Gibt es für 3SAT einen Algorithmus, der für jede Instanz I in polynomieller Zeit (in m und n von I) entscheidet, ob I erfüllbar ist?

SATZ 1.38 (KNAPSACK $\in P \Rightarrow$ 3SAT $\in P$)

Wenn KNAPSACK polynomiell lösbar ist, dann ist auch 3SAT polynomiell 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 polynomiell beschränkt durch die Codierungsgröße von I_{3SAT}.

(B) Es gibt für I_{KN} eine Teilmenge mit Lösungswert Z $\Leftrightarrow$ I_{3SAT} ist erfüllbar.

Wenn wir einen polynomiellen Algorithmus für KNAPSACK haben, dann können wir damit in polynomieller 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 Kracher:

KOROLLAR 1.39

Wenn KNAPSACK polynomiell lösbar ist, dann gilt $P = NP$.

Denn:

SATZ 1.36 (Satz von Cook, 1971)
Wenn 3SAT polynomiell 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 Π in NP heißt NP-vollständig, wenn $\Pi \in P \Rightarrow P = NP$ gilt.

(2) Ein Problem Π heißt NP-schwer, wenn $\Pi \in P \Rightarrow P = NP$ gilt.

Also:

KOROLLAR 1.38
(1) 3SAT ist NP-vollständig.
(2) KNAPSACK$_{\neq}$ ist NP-vollständig.
(3) KNAPSACK ist NP-schwer.

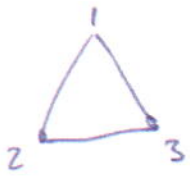
Idealer Finanzberater:

(1) ehrlicher
(2) intelligenter
(3) Investmentbanker

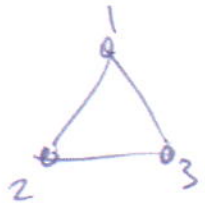
Perfekter Algorithmus:

(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 — BWL
(B) Hart arbeiten — Inf
(C) Erwartungen herabschrauben — WiInf
(D) Mit dem Schicksal hadern + diskutieren — Jura

Hier:

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