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".

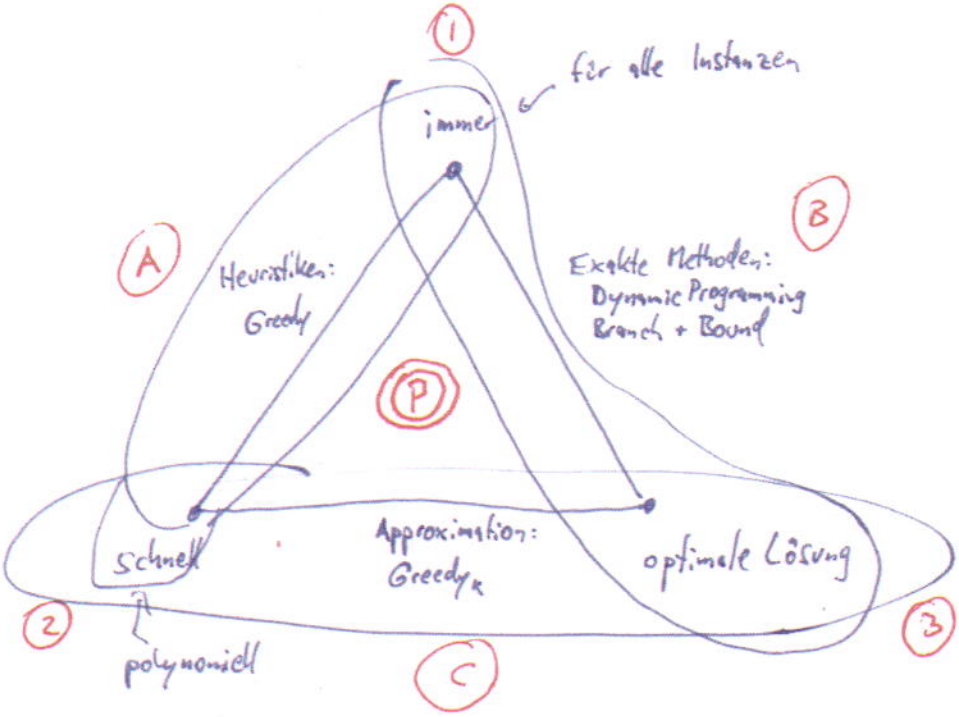
Geht das noch besser? Können wir einen „perfekten" Algorithmus finden, der

(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 ~~eine~~ 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

$P \neq NP$?

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.

$\square$

Und jetzt der Knaller:

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$_{\geq}$ ist NP-vollständig.

(3) KNAPSACK ist NP-schwer.

1.5 Approximation

Wie gut ist Greedy? (Ganzzahlige Variante von ALG 1.4, G_0)

Schon in Übung gesehen: Beliebig schlecht!

Beispiel 1.21

Betrachte: $n = 2$, $z_1 = 1$, $p_1 = 1+\varepsilon$

$z_2 = N$, $p_2 = N$

$Z = N$

Dann liefert Greedy eine Lösung von Nutzen $1+\varepsilon$ (nur Objekt 1), optimal ist aber Nutzen N (nur Objekt 2).

Für $N \gg 1+\varepsilon$ ist das beliebig weit vom Optimum!

Relevante Größe ist dabei nicht die absolute Abweichung, sondern die relative: Wieviele Prozent des Optimums können wir erreichen?

ALGORITHMUS 1.22 (Greedy_0)

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n > 0$

Ausgabe: Eine Lösung $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} z_i \leq Z$

und Lösungswert $G_0 := \sum_{i \in S} p_i$

(1) sortiere $\{1, \dots, n\}$ nach $\frac{z_i}{p_i}$ aufsteigend;

dies ergibt Permutation $\pi(1), \dots, \pi(n)$.

Setze $j := 1$.

(2) WHILE $(j \leq n)$ DO {

IF $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ {

$x_{\pi(j)} := 1$

}

$j := j + 1$

}

(3) RETURN

(Vergleiche Algorithmus 1.4 ! ⚠ $x_j \to x_{\pi(j)}$ ⚠)

ALGORITHMUS 1.23

Eingabe: $z_1, \dots, z_n, Z, p_1, \dots, p_n > 0$ (mit $z_i \leq Z$)

Ausgabe: Eine Lösung $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} z_i \leq Z$

(1) Berechne G_0 mit Algorithmus 1.22

(2) Return $\tilde{G}_0 := \max\{G_0, \max p_i\}$ und zugehöriges S

Satz 1.24

Algorithmus 1.23 berechnet eine Lösung mit Wert $\tilde{G}_0$, für den im Vergleich zum bestmöglichen Lösungswert gilt

$$\tilde{G}_0 \geq \frac{1}{2} OPT.$$

Beweis:

Nach Konstruktion ist $\tilde{G}_0 \geq G_0$

und $\tilde{G}_0 \geq \max p_i =: p^*$

Außerdem ist (vgl. Algorithmus 1.4)

$$G_0 + p^* \geq \text{Fract KP} \geq OPT$$

Also ist $2\tilde{G}_0 \geq G_0 + p^* \geq OPT$, d.h. $\tilde{G}_0 \geq \frac{1}{2} OPT$.

Das motiviert die folgende

Definition 1.25

(1) Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein c-Approximationsalgorithmus, wenn für jede Instanz I von MAX

(i) ALG in polynomieller Zeit in der Größe der Instanz eine Lösung mit Wert ALG(I) liefert.

(ii) Für den Vergleich mit dem zugehörigen Optimalwert OPT(I) gilt

$$ALG(I) \geq c \cdot OPT(I)$$

(2) Entsprechend verlangt man für ein Minimierungsproblem wiederum:

(i) polynomielle Laufzeit

(ii) $ALG(I) \leq c \cdot OPT(I)$

Merke: Maximieren $\rightarrow$ $c \leq 1$, je größer c, um so besser!

Minimieren $\rightarrow$ $c \geq 1$, je kleiner c, um so besser!

Wie gut kann man KNAPSACK approximieren?