2 Komplexitätstheorie

Die Komplexitätstheorie untersucht den Ressourcenbedarf (Zeitbedarf, Speicherplatz, Programmlänge,...) von Algorithmen.

Die Komplexitätstheorie ist interessiert an:

oberen Schranken:	Garantie, daß das untersuchte Problem im Rahmen der vorgegebenen Ressourcen gelöst werden kann. Ein solcher Algorithmus heißt obere Schranke.
unteren Schranken:	Aussagen über den Mindestbedarf an Ressourcen für die Lösung eines Problems.

Ein Algorithmus ist (asymptotisch) *optimal* für ein Problem, wenn er bei der Lösung des Problems mit den durch die untere Schranke beschriebenen Ressourcen auskommt.

2.1 Komplexitätsmaße und Komplexitätsklassen

Beschreibe Problem als formale Sprache "Entscheidungsproblem":

$$A \subseteq \Sigma^*$$

$x \in A \Leftrightarrow x$ "löst" Problem

Da Algorithmen untersucht werden, kann man sich auf die Klasse der entscheidbaren Sprachen beschränken.

Sei M eine Mehrband–TM, die bei allen Eingaben $x \in \Sigma^*$ hält.

$time_M : \Sigma^* \rightarrow I\!N$ "Zeitkomplexität von M".
$time_M(x)$ = Anzahl der Rechenschritte von M bei Eingabe von x.
$space_M : \Sigma^* \rightarrow I\!N$ "Speicherkomplexität von M".
$time_M(x)$ = Anzahl der bei Eingabe von x von M benutzten Felder der Bänder.

Sei $A \subseteq \Sigma^*$ entscheidbar.
$time_A(x) = min\{time_M(x) : M$ ist Mehrband–TM für $A\}$ ist Zeitkomplexität von A
$space_A(x) = min\{space_M(x) : M$ ist Mehrband–TM für $A\}$ ist Speicherkomplexität von A

Sei $f : I\!N \rightarrow I\!N$
$TIME(f(n)) = \{A \subseteq \Sigma^* : time_A(x) \leq f(|x|)\}$
$SPACE(f(n)) = \{A \subseteq \Sigma^* : space_A(x) \leq f(|x|)\}$

Satz.
Die Komplexitätsklasse $TIME(f(n))$, wobei $f(n)$ nach oben durch LOOP–Programme beschränkt werden kann, ist enthalten in der Klasse der primitiv rekursiven Sprachen (bzw. der LOOP–berechenbaren Sprachen).

Beweis.
Sei M TM mit $time_M(x) \leq f(|x|)$.
Simuliere M durch ein GOTO–Programm , wobei jeder TM–Schritt durch eine endliche Zahl von Wertzuweisungen bzw. GOTOs simuliert wird. Forme das GOTO–Programm in ein äquivalentes WHILE–Programm mit nur einer WHILE–Schleife um.
Die Anzahl der Durchläufe der WHILE–Schleife ist durch die Zahl $f(n)$ beschränkt.
Ersetze "WHILE count$\neq 0$ DO" durch "$y := f(n)$;LOOP y DO".
Das Ergebnis ist ein LOOP–Programm, das M simuliert. ■

Korollar
$TIME(n^k)$ $(k \in I\!N), TIME(2^n), TIME(2^{2^{\cdot^{\cdot^{2}}}})$ enthalten nur primitiv rekursive Mengen.

Komplexitätsmaße können auch mit Hilfe von WHILE–Programmen eingeführt werden.
Vorsicht
$x_i := x_j$ muß so groß angesetzt werden, wie die Anzahl der Bits, die bei dieser Aktion übertragen werden (also etwa $\log x_j$).
("logarithmisches Kostenmaß")
Dieses Kostenmaß ist zwar genau, aber schwierig zu analysieren.
Setzt man dagegen die Kosten für elementare Anweisungen 1, spricht man vom "uniformen Kostenmaß".

Beispiel
INPUT (n);
$x := 2$;
LOOP n DO $x := x * x$ END;
OUTPUT(x);

Der Algorithmus berechnet 2^{2^n}.
Kosten bei: "uniformem Kostenmaß": $O(n)$
"logarithmischem Kostenmaß": $O(2^n)$

Üblich
Die Komplexität eines Algorithmus wird unter dem uniformen Kostenmaß angegeben.

2.2 Die Komplexitätsklassen P und NP

Es ist sinnvoll, anstelle von Funktionen Funktionsklassen zu betrachten:

Beispiel
Problem ist mit Einband–TM M lösbar $\Leftrightarrow$
Problem ist mit Mehrband–TM N lösbar,
aber
$time_M = (time_N)^2$

Es werden deshalb Klassen betrachtet, die gegenüber solchen Modifikationen abgeschlossen sind.

Definition.
Ein *Polynom* $p : I\!N \to I\!N$ ist eine Funktion der Form
$p(n) = a_k n^k + \ldots + a_1 n + a_0 \quad a_i \in I\!N, k \in I\!N$

Komplexitätsklasse P

$$\begin{aligned} P &= \bigcup_{P\ Polynom} TIME(p(n)) \\ &= \{A : \exists \text{ TM } M \text{ mit } L(M) = A \text{ und } time_M(x) \leq p(|x|) \text{ für ein Polynom } p\} \\ \Rightarrow P &= \bigcup_{k \geq 1} TIME(O(n^k)) \end{aligned}$$

Bermerkung
Auch Algorithmen der Komplexität $n \log n$ sind polynomial, da
$n \log n = O(n^2)$.

$n^{\log n}, 2^n$ sind nicht polynomial ("exponentiell").

Allgemein akzeptiert

P = Klasse der effizienten Algorithmen(d.h. der praktisch realisierbaren Algorithmen) .

Algorithmen mit exponentieller Laufzeit sind nicht effizient.

- P könnte auch als WHILE–Programm mit logarithmischen Kostenmaß definiert werden.
- $P \subseteq TIME(2^n) \subseteq$ {Primitiv rekursive Sprache}
- P enthält alle Probleme, für die sich in polynomieller Zeit ein Beweis finden läßt.

Betrachte daneben die Klasse NP der Probleme, für die sich in polynomieller Zeit ein vorgegebener Beweis überprüfen läßt:

Definition.
Eine *nichtdeterministische TM* ist diejenige TM, bei der die Übergangsfunktion eine Übergangsrelation ist (eine Konfiguration hat i.a. mehrere mögliche Nachfolgekonfigurationen). Eine *akzeptierende Berechnung* besteht aus einer zulässigen Folge von Konfigurationen, die mit der Startkonfiguaration beginnt und in einer akzeptierenden Konfiguration endet.

Definition.
Sei M eine nichtdeterministische Mehrband–TM.

$$time_M(x) = \begin{cases} \min\{\text{Länge einer akzeptierenden Berechnung von } M \text{ auf } x\}, & x \in L(M) \\ 0, & x \notin L(M) \end{cases}$$

$f : I\!N \to I\!N$

$$NTIME(f(n)) = \{A : \exists \text{ NTM } M \text{ mit } L(M) = A \text{ und } time_M(x) \leq f(|x|) \forall x \in \Sigma^*\}$$

$$\begin{aligned} NP &:= \bigcup_{p \ Polynom} NTIME\ p(n) \\ &= \bigcup_{k \geq 1} TIME(O(n^k)) \end{aligned}$$

Offenbar gilt:

$$P \subseteq NP$$

Die Umkehrung ist unklar: "$P - NP$–Problem"

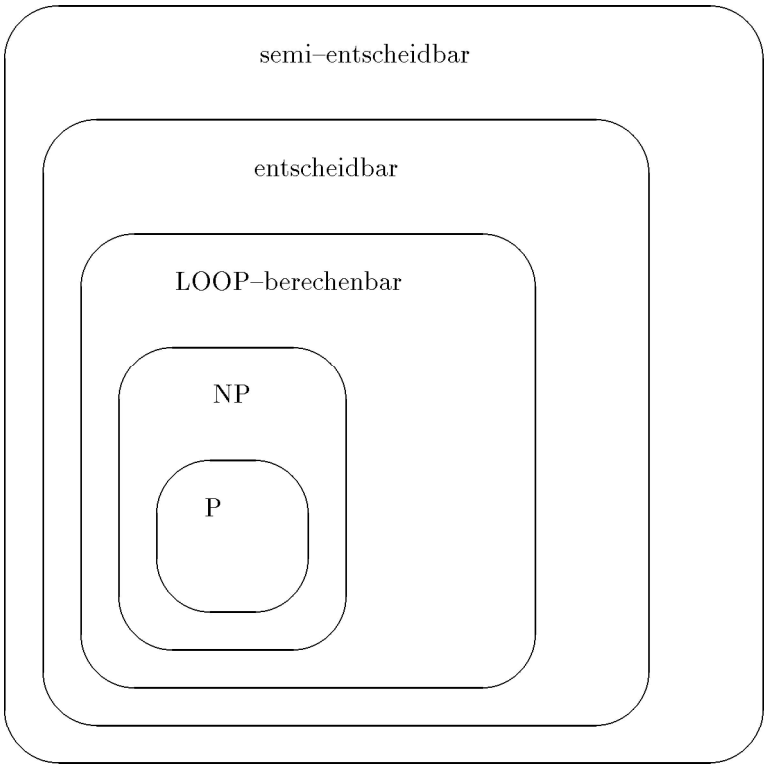
Das $P - NP$–Problem ist wegen seiner Bedeutung für die Findung effizienter Algorithmen von großer Bedeutung (von vielen als das bedeutendste Problem der Theoretischen Informatik angesehen).

Allgemeine Annahme

$$P \neq NP$$

Bei der Untersuchung des $P - NP$–Problems wurde die Theorie der NP–Vollständigkeit entwickelt. (Cook '71, Karp '72)

Es gilt:



2.3 NP–Vollständigkeit

Definition.
Seien $A, B \subseteq \Sigma^*$.
A heißt *polynomiell auf B reduzierbar* ($A \leq_p B$) , falls es eine totale und mit polynomialer Komplexität berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt, so daß für alle $x \in \Sigma^*$ gilt:

$$x \in A \Leftrightarrow f(x) \in B$$

Lemma
$\leq_p$ ist transitiv.

Beweis.
Übungsaufgabe.

Lemma
i) $A \leq_p B, B \in P \Rightarrow A \in P$
ii) $A \leq_p B, B \in NP \Rightarrow A \in NP$
Beweis.
i) Sei $A \leq_p B$ mittels f.
TM M_f berechne f in Polynomialzeit p.
Sei $B \in P$ mittels TM M in Rechenzeit q.
$\Rightarrow (M_f; M)$ berechnet A in Rechenzeit
$p(|x|) + q(|f(x)|) \leq p(|x|) + q(p(|x|))$ ist Polynom. ■

ii) analog

Definition.
Eine Sprache A heißt *NP–hart*, falls für alle Sprachen $L \in NP$ gilt: $L \leq_p A$.
Eine Sprache A heißt *NP–vollständig*, falls A NP–hart ist und $A \in NP$ gilt.

Bemerkung
NP-vollständige Sprachen sind "schwere" Sprachen in NP.

Satz.
Sei A NP–vollständig.
Dann gilt:

$$A \in P \Leftrightarrow P = NP$$

Beweis.
(—): Sei $A \in P, L \in NP$ beliebig.
da A NP–hart gilt: $L \leq_p A$
$\Rightarrow_{Lemma} L \in P$ ■
(—): $P = NP \Rightarrow A \in P$