12 Komplexitätstheorie

In vorangegangenen Kapiteln haben wir gesehen, daß bestimmte Aufgaben durch einen Computer prinzipiell nicht gelöst werden können. Andere Problemstellungen können zwar im Prinzip durch einen Rechner behandelt werden, benötigen aber zu viel Rechenzeit oder Speicherplatz, als daß sie mit den verfügbaren Ressourcen praktisch gelöst werden können. Die Komplexitätstheorie macht quantitative Aussagen über die Rechenzeit und den Speicherplatz, der zur Lösung einer bestimmten Aufgabe und in Abhängigkeit von den Eingabedaten erforderlich ist. Da diese Abhängigkeiten nur selten genau berechnet werden können, arbeitet man mit asymptotischen Angaben. Man unterscheidet ferner

- *Worst Case Analysis*, welche den Ressourcenbedarf im ungünstigsten Fall untersucht,
- *Average Case Analysis*, die den im Mittel zu erwartenden Aufwand untersucht, wobei über alle möglichen Eingaben mit einer gewissen Wahrscheinlichkeit gemittelt wird,
- *Lower Bound Analysis*, welche die unteren Schranken für den Aufwand angibt – besser geht's nicht – und
- *Upper Bound Analysis*, welche die oberen Schranken für den Aufwand untersucht – schlimmer wird's nicht.

Wir werden in diesem Kapitel einige Techniken vorstellen, die bei der Analyse der Komplexität von Algorithmen häufig benutzt werden. Weitere Einzelheiten dieses komplexen Gebietes findet man in der Literatur.

12.1 Asymptotische Notationen

Asymptotische Notationen dienen zur Angabe des ungefähren Wachstumsverhaltens von Funktionen $g : \mathrm{N} \to \mathrm{R}$ bei großen Problemgrößen n.

467 Definition GROSS O, GROSS OMEGA UND GROSS THETA
Eine Funktion $g : \mathrm{N} \to \mathrm{R}$ liegt in der Klasse $O(f)$ einer Funktion $f : \mathrm{N} \to \mathrm{R}$, wenn

es eine positive reelle Zahl $C \in \mathrm{R}^+$ gibt, so, daß ab einer bestimmten Zahl $N_0 \in \mathrm{N}$ für alle größeren $n \in \mathrm{N}, n \geq N_0$ die Ungleichung $g(n) \leq Cf(n)$ gilt. Anschaulich heißt das: g liegt in der Klasse $O(f)$, wenn g *asymptotisch höchstens so rasch wie* f wächst.

Eine Funktion $g : \mathrm{N} \to \mathrm{R}$ liegt in der Klasse $\Omega(f)$ einer Funktion $f : \mathrm{N} \to \mathrm{R}$, wenn es eine positive reelle Zahl $C \in \mathrm{R}^+$ gibt, so, daß ab einer bestimmten Zahl $N_0 \in \mathrm{N}$ für alle größeren $n \in \mathrm{N}, n \geq N_0$ die Ungleichung $g(n) \geq Cf(n)$ gilt. Anschaulich heißt das: g liegt in der Klasse $\Omega(f)$, wenn g *asymptotisch mindestens so rasch wie* f wächst.

Eine Funktion $g : \mathrm{N} \to \mathrm{R}$ liegt in der Klasse $\Theta(f)$ einer Funktion $f : \mathrm{N} \to \mathrm{R}$, wenn es positive reelle Zahlen $C_1, C_2 \in \mathrm{R}^+$ gibt, so, daß ab einer bestimmten Zahl $N_0 \in \mathrm{N}$ für alle größeren $n \in \mathrm{N}, n \geq N_0$ die Ungleichung $C_1 f(n) \leq g(n) \leq C_2 f(n)$ gilt. Anschaulich heißt das: g liegt in der Klasse $\Theta(f)$, wenn g *asymptotisch etwa gleich rasch wie* f wächst.

Klasse	**Interpretation**
$g \in O(f)$	g wächst asymptotisch höchstens so rasch wie f
$g \in \Theta(f)$	g wächst asymptotisch genau so rasch wie f
$g \in \Omega(f)$	g wächst asymptotisch mindestens so rasch wie f

468 Bemerkung Interpretation asymptotischer Notationen
Eine Funktion g liegt in der Klasse $O(n)$, wenn es eine positive reelle Zahl C und eine natürliche Zahl N_0 gibt, daß für $n \geq N_0$ die Ungleichung $g(n) \leq C * n$ erfüllt ist. Das heißt g wächst ab einer bestimmten Anfangsphase, bei der nichts über das Verhalten der Funktion ausgesagt wird, höchstens so schnell wie eine lineare Funktion. Da wir eine beliebige Konstante C frei wählen dürfen, bezieht sich diese Aussage auch nicht auf die spezifische quantitative Steigung der linearen Funktion, sondern mehr auf das qualitative Verhalten.

Eine Funktion g liegt etwa in der Klasse $\Omega(n^2)$, wenn $\exists C, N_0 : \forall n \geq N_0 : g(n) \geq C * n^2$. Das heißt g wächst ab einer bestimmten Anfangsphase, bei der nichts über das Verhalten der Funktion ausgesagt wird, mindestens so schnell wie eine quadratische Funktion. Da wir eine beliebige Konstante C frei wählen dürfen, bezieht sich diese Aussage auch nicht auf die spezifische quantitative Steigung der quadratischen Funktion, sondern mehr auf das qualitative Verhalten.

Eine Funktion g, die in der Klasse $\Theta(f)$ liegt, kann man also anschaulich gesprochen, mit Ausnahme eines uninteressanten Anfangsverhaltens, zwischen zwei Versionen von f begrenzen. Beide Versionen von f, das sind $C_1 * f$ und $C_2 * f$, unterscheiden sich nur durch eine vorangestellte Konstante.

469 Theorem THETA THEOREM
Eine Funktion g liegt in der Klasse $\Theta(f)$ genau dann, wenn g in den Klassen $\Omega(f)$ und $O(f)$ liegt: Die Aussage *g wächst asymptotisch gleich rasch wie f* ist äquivalent zur Aussage *g wächst asymptotisch sowohl mindestens so rasch wie f, als auch höchstens so rasch wie f.*

470 Beispiel ASYMPTOTISCHES WACHSTUM
Die Funktion $n^2 + 2n + 2$ hat quadratisches Wachstum, das heißt mit $g(n) := n^2 + 2n + 2$ und $f(n) := n^2$ gilt $g \in \Theta(f)$.

Die Funktion 2^n wächst mindestens so schnell wie die Funktion n^4, das heißt mit $g(n) := 2^n$ und $f(n) := n^4$ gilt $g \in \Omega(f)$. Die Funktion 2^n wächst sogar schneller als die Funktion n^4, es gilt also weder $g \in O(f)$ noch $g \in \Theta(f)$.

BEWEIS:
Wir wählen $C_1 = 1$, $C_2 = 5$ und $N_0 = 1$. Nun kann man leicht zeigen, daß für $n \in \mathbb{N}$ mit $n \geq N_0$ die Ungleichung $C_1 f(n) \leq g(n) \leq C_2 f(n)$ gilt.

Die Ungleichung $2^n \geq Cn^4$ gilt genau dann, wenn die Ungleichung $\log(2^n) \geq \log(Cn^4)$ gilt, da der Logarithmus eine monotone Funktion ist. Umformung der Logarithmen ergibt $n \log 2 \geq \log C + 4 \log n$. Die linke Seite wächst linear: Erhöhung von n um 1 erhöht den Funktionswert immer um $\log 2$. Um das Wachstum der rechten Seite zu betrachten, bilden wir die Ableitung, sie ist $4/n$. Das bedeutet, daß bei größerem n die rechte Seite immer langsamer wächst. Wählen wir insbesondere $n \geq 8/\log 2$, so wächst ab diesem Wert die rechte Seite langsamer als die lineare Funktion $0.5 * n \log 2$. Je Erhöhung von n um 1 wächst dann die rechte Seite sicher um $0.5 * \log 2$ weniger als die linke Seite. Deshalb wird die linke Seite ab einem geeigneten N_0 die rechte aufgeholt haben. Eine analoge Argumentation können wir benutzen, um $g \notin O(f)$ und $g \notin \Theta(f)$ zu zeigen. Wir nehmen mittels indirekten Beweis das Gegenteil an. Dann müßte eine Ungleichung gelten, die aber nicht gilt. □

12.2 Wachstumsklassen

471 Definition ASYMPTOTISCHE ÄQUIVALENZ
Zwei Funktionen f und g heißen *asymptotisch äquivalent (asymptotically equivalent)*, $f \sim_a g$, wenn f in der Klasse $\Theta(g)$ liegt.

472 Definition WACHSTUMSKLASSE
Eine *(asymptotische) Wachstumsklasse (asymptotical growth class)* ist eine Äquivalenzklasse von Funktionen, die dasselbe asymptotische Wachstumsverhalten haben,

präziser ausgedrückt eine Äquivalenzklasse der oben eingeführten Relation $\sim_a$.

473 Bemerkung POLYNOMIALE WACHSTUMSKLASSE
Eine *polynomiale Wachstumsklasse* ist eine Menge von Funktionen, die als $\Theta(P(n))$ angeschrieben werden kann, wobei P ein geeignetes Polynom ist. Wir haben bereits am Beispiel $n^2+2n+2 \in \Theta(n^2)$ gesehen, daß für das asymptotische Wachstumsverhalten von Polynomen nur die höchste auftretende Potenz, also der *Grad (degree)* des Polynoms, maßgeblich ist. Wir definieren deshalb:

Die *polynomiale Wachstumsklasse vom Grad p* ist die Funktionenmenge $\Theta(n^p)$. Hat eine Funktion f polynomiales Wachstum, das heißt, liegt sie in einer Menge $\Theta(P(n))$, bei der P ein Polynom ist, dann liegt sie auch in der polynomialen Wachstumsklasse vom Grad p, wobei p der Grad von P ist.

Das Verhalten dieser Wachstumsklassen ist durch den Grad determiniert: Wir betrachten ein Problem, dessen Rechenzeit polynomial von der Eingangsgröße n abhängt. Verdoppelt man n, so verdoppelt sich der Aufwand (falls Grad $p = 1$), oder er vervierfacht sich (falls Grad $p = 2$), oder er verachtfacht sich (falls Grad $p = 3$), und so weiter.

Ist der Grad $p = 1$, so spricht man von *linearem Wachstum (linear growth)*, bei $p = 2$ von *quadratischem Wachstum (quadratic growth)* und bei $p = 3$ von *kubischem Wachstum (cubic growth)*.

474 Bemerkung EXPONENTIELLE WACHSTUMSKLASSE
Die Menge $\Theta(b^n)$ von Funktionen, $b > 1$, heißt die *exponentielle Wachstumsklasse zur Basis b*.

Charakteristisch für diese exponentiellen Klassen ist ihr extremes Wachstum. Vergrößert man das Argument um 1, so multipliziert sich ihr Wert mit der Basis. Bei Basis $b = 2$ bewirkt die Vergrößerung des Arguments um 1 bereits eine Verdopplung des Funktionswertes.

475 Bemerkung LOGARITHMISCHE WACHSTUMSKLASSE
Die Funktionsmenge $\Theta(\log(n))$ heißt *logarithmische Wachstumsklasse*. Da der Logarithmus zur Basis a und der Logarithmus zur Basis b über eine Konstante zusammenhängen, $\log_a(n) = C \log_b(n)$, ist die logarithmische Wachstumsklasse unabhängig von der Basis des benutzten Logarithmus.

Charakteristisch für die logarithmische Wachstumsklasse ist das *sublineare* Wachstum. Logarithmisch wachsende Funktionen wachsen also langsamer als lineare Funktion, also auch langsamer als ihr Argument.

476 Bemerkung WEITERE WACHSTUMSKLASSEN
Die Wachstumsklassen $\Theta(n \log n)$ und $\Theta(n^2 \log n)$ haben eine besondere Bedeutung, da sie bei vielen Algorithmen auftauchen. Weitere wichtige Klassen stellen die *gemischt logarithmisch-polynomial-exponentiellen Wachstumsklassen* der Form $\Theta(n^p q^n (\log n)^\alpha)$ und die *Wachstumsklasse der inversen Ackermannfunktion* dar.

12.3 Komplexitätsaussagen

477 Beispiel SORTIERVERFAHREN
In der Theorie der Algorithmen hat man die Rechenzeit untersucht, welche die unterschiedlichen Algorithmen zum Sortieren von n Zahlen benötigen. Für "Sortieren durch Einfügen" ergab sich:

$$A_E(n) = \frac{n(n-1)}{2} W_E(n) = \frac{n^2}{4} + \frac{3n}{4} - 1 - \sum_{i=2}^{n} \frac{1}{i}$$

$A_E(n)$ gibt im Sinne einer average case Analyse den durchschnittlichen, über alle möglichen Eingaben gemittelten Zeitbedarf an, $W_E(n)$ im Sinne einer worst case Analyse den maximalen Zeitbedarf. Unter "Zeitbedarf" ist hier weniger die genaue Rechenzeit auf einem bestimmten Computer zu verstehen, als vielmehr die Anzahl von Operationen, die der betreffende Algorithmus durchzuführen hat. Es handelt sich also um eine recht grobe Abschätzung der tatsächlichen Rechenzeit, die ja noch vom benutzten Computer, der verwendeten Programmiersprache und vielen anderen Einflüssen abhängt. Der Unterschied zwischen $A_E(n)$ und $W_E(n)$ rührt daher, daß neben der Anzahl der zu sortierenden Zahlen auch die bereits in diesen Zahlen vorhandene Ordnung eine Rolle spielt: Sind die Zahlen bereits geordnet, so muß der Algorithmus kaum Arbeit leisten.
Abschätzungen wie die oben durchgeführten liefern $A_E \in \Theta(n^2)$ und $W_E \in \Theta(n^2)$. Man sagt: "Sortieren durch Einfügen" hat im Mittel und im schlimmsten Fall quadratische Komplexität. Verdoppelt man die Anzahl zu sortierender Zahlen, so wird die Rechenzeit in etwa vervierfacht.

Für den Algorithmus "Quicksort" ergibt diese Analyse im Mittel die Zeitkomplexität $\Theta(n \log n)$ und im schlimmsten Fall die Rechenzeit $\Theta(n^2)$. Dadurch ist über die konkrete Rechenzeit des "Quicksort" im Vergleich zum anderen Algorithmus noch nicht viel gesagt: Quadratische Komplexität im schlimmsten Fall kann eine Rechenzeit von $10000 * n^2$ oder von n^2 bedeuten. Wir wissen aber, daß "Quicksort" im Mittel bei Vergrößerung der Anzahl zu sortierender Elemente besser "skaliert" als "Sortieren durch Einfügen". Wenn wir anstelle von n Zahlen $10 * n$ Zahlen sortieren, so wird "Sortieren durch Einfügen" im Mittel 100 mal so lange brauchen, da

$(10n)^2 = 100n^2$. "Quicksort" wird wegen $(10n)\log(10n) = 10n\log n + 10n\log 10 \leq 20n\log n$ höchstens 20 mal so lange brauchen, falls n mindestens 10 ist, für größere n schneidet "Quicksort" im relativen Vergleich sogar noch besser ab.

478 Bemerkung VERGLEICH DER WACHSTUMSKLASSEN
In nachfolgender Tabelle sind für vorgegebenes n die Werte typisch charakterisierender Funktionen für verschiedene Wachstumsklassen angegeben. Die Werte werden jeweils als Sekunden interpretiert:

n	$log_{10}(n)$	n	n^2	n^3	2^n	e^n	10^n
1	0	1	1	1	2	2.718	10
2	0.3	2	4	8	4	7.4	100
3	0.48	3	9	27	8	20.1	$17min$
4	0.6	4	16	64	16	54.6	$2.8h$
5	0.7	5	25	125	32	$2.5h$	$27.8h$
10	1	10	100	$16.7min$	$17min$	$6h$	$317j$
20	1.3	20	400	$2.2h$	$12d$	$15.4j$	$200w$
50	1.7	50	$41min$	$1.4d$	$35.000.000j$	$1.000w$	
100	2	100	$2.8h$	$11.6d$	$2w$		
1.000	3	1.000	$11.6d$	$31.7j$			
10.000	4	10.000	$3.2j$	$3.000j$			
100.000	5	100.000	$31.7j$	$30.000.000j$			

Ohne Angabe von Einheiten sind die Zeiten in Sekunden, ansonsten in Minuten, Stunden, Tagen, Jahren oder in Vielfachen der Lebensdauer $w = 18 * 10^9$ Jahre des Universums angegeben. Die nicht angeführten Werte lagen jenseits der numerischen Fähigkeiten des benutzten Taschenrechners.

12.4 NP–Vollständigkeit

Die Theorie der NP–vollständigen Probleme behandelt Problemklassen, die so rechenaufwendig sind, daß sie in der Praxis kaum gerechnet werden können.

479 Bemerkung FORMULIERUNG VON PROBLEMEN
Komplexitätsanalyse im Umfeld der NP–Vollständigkeit befaßt sich zunächst mit dem Problem der Zugehörigkeit eines Wortes $w \in A^*$ zu einer Sprache $\mathcal{L} \subseteq A^*$ und untersucht die Anzahl von Schritten, die eine TURING–Maschine zur Lösung dieses Problems in Abhängigkeit von der Länge des Wortes w benötigt.
Will man also wissen, ob ein Problem NP–vollständig ist, so muß man es zuerst als Spracherkennungsproblem formulieren. Da die hiermit verbundenen Codierungen von Problemen manchmal etwas aufwendig zu beschreiben sind und keine für un-

sere Zwecke wichtigen Einsichten enthalten, wollen wir die technischen Details der weiterführenden Literatur überlassen.

480 Definition KOMPLEXITÄTSKLASSEN P UND NP
Die *polynomiale Komplexitätsklasse P (complexity class P)* ist die Menge aller semientscheidbaren Sprachen $\mathcal{L}$ über einem endlichen Alphabet A, die von einer deterministischen TURING–Maschine in polynomialer Zeit erkannt werden können: Es gibt eine deterministische TURING–Maschine und ein Polynom P mit folgender Eigenschaft: Schreibt man ein Wort $w \in A^*$ auf das Eingabeband der TURING–Maschine, so führt diese genau dann irgendwann einmal die Halteoperation aus, wenn das Wort w in der Sprache $\mathcal{L}$ liegt. Ist das Wort $w \in A^*$ Element der Sprache $\mathcal{L}$, dann hält die TURING–Maschine nach höchstens $P(\mid w \mid)$ Schritten[1] an.

Die *nichtdeterministisch-polynomiale Komplexitätsklasse NP (complexity class NP)* ist die Klasse Menge aller semientscheidbaren Sprachen $\mathcal{L}$ über einem endlichen Alphabet A, die von einer nichtdeterministischen TURING–Maschine in polynomialer Zeit erkannt werden können. Es gibt also mindestens eine Folge von höchstens $P(\mid w \mid)$ Operationen, P und w wie oben, nach denen die TURING–Maschine anhält, falls $w \in \mathcal{L}$.

481 Theorem SIMULATIONSTHEOREM
Sei P ein Polynom und $\mathcal{L}$ eine Sprache, deren Wörter w von einer nichtdeterministischen TURING–Maschine in höchstens $P(\mid w \mid)$ Schritten erkannt werden. Dann gibt es eine deterministische TURING–Maschine, welche die Wörter w der Sprache in höchstens $c^{P(|w|)}$ Schritten erkennt, wobei c eine geeignete Konstante ist.

Dieses Theorem zeigt die Hauptschwierigkeit mit der Problemklasse NP auf: Hätten wir eine nichtdeterministische TURING–Maschine zur Verfügung, so könnten wir ein NP Problem in polynomialer Zeit lösen, was ja nicht so schlimm ist. Da unsere Computer aber deterministische Maschinen sind, müssen wir nichtdeterministische Prozesse auf ihnen simulieren. Wenn eine nichtdeterministische Maschine zu einem bestimmten Zeitpunkt die Auswahl zwischen zwei Möglichkeiten hat, so müssen wir auf unseren deterministischen Maschinen zuerst die eine und anschließend die andere Variante durchrechnen. Eine nichtdeterministische Maschine rechnet quasi alle Varianten gleichzeitig durch, respektive ist in der Lage, die "richtige" Variante korrekt zu erraten. Die Simulation einer nichtdeterministischen Maschine führt aber im allgemeinen auf eine exponentielle Komplexitätsklasse mit einem dementsprechend gewaltigen Wachstumsverhalten der Rechenzeit bei Vergößerung der Eingabe.

Ist $\mathcal{L}$ eine Sprache, die durch eine nichtdeterministische TURING–Maschine in li-

[1] $\mid w \mid$ bezeichnet die Länge des Wortes w.

nearer Rechenzeit $P(x) = 3x$ erkannt wird, so gibt es etwa eine deterministische TURING–Maschine, die $\mathcal{L}$ in beispielsweise der exponentiellen Rechenzeit 2^{3x} erkennen kann. Die nachfolgende Tabelle zeigt auf, was dies bereits für kleine Werte von x für die Zunahme der Rechenzeit mit wachsendem x bedeutet:

x	2^{3x}
1	8
2	64
3	512
4	4096
5	32.768
6	$2 * 10^5$
7	$2 * 10^6$
8	$2 * 10^7$
9	$1 * 10^8$
10	$1 * 10^9$
20	$1 * 10^{18}$
30	$1 * 10^{27}$

Probleme einer realistischen Größenordnung werden also sehr schnell so zeitaufwendig, daß man sie nicht lösen kann. Auch eine Beschleunigung der Rechengeschwindigkeit um einen Faktor 1000 bringt hier praktisch keine Beschleunigung. Solche Probleme sind zwar prinzipiell lösbar, praktisch aber unlösbar.

482 Definition POLYNOMIALE REDUKTION
Seien A, B endliche Alphabete und $\mathcal{L}_1 \subseteq A^*$ und $\mathcal{L}_2 \subseteq B^*$ zwei rekursiv aufzählbare Sprachen. Die Sprache $\mathcal{L}_1$ heißt *polynomial reduzierbar (polynomially reducible)* auf $\mathcal{L}_2$, $\mathcal{L}_1 \leq \mathcal{L}_2$, wenn es eine in polynomialer Zeit berechenbare Funktion $f : A \to B$ gibt, daß $x \in \mathcal{L}_1 \Leftrightarrow f(x) \in \mathcal{L}_2$ gilt.

Ist $\mathcal{L}_1 \leq \mathcal{L}_2$, so bedeutet das folgendes: Um das Spracherkennungsproblem $x \in \mathcal{L}_1$ zu lösen, genügt es, zuerst $f(x)$ zu berechnen, was in deterministisch-polynomialer Zeit geht, und dann das Problem $f(x) \in \mathcal{L}_2$ zu lösen. Bei $\mathcal{L}_1 \leq \mathcal{L}_2$ kann also das Spracherkennungsproblem für $\mathcal{L}_1$ dadurch gelöst werden, daß man jenes für $\mathcal{L}_2$ löst – samt einem kleinen zusätzlichen Transformationsaufwand.

483 Definition NP–VOLLSTÄNDIG
Eine semientscheidbare Sprache $\mathcal{L}$ heißt *NP–vollständig (NP–complete)*, wenn

(1) $\mathcal{L}$ in NP liegt und

(2) jede andere Sprache $\mathcal{L}'$ in NP auf $\mathcal{L}$ polynomial reduziert werden kann: $\forall \mathcal{L}' \in$ NP $: \mathcal{L}' \leq \mathcal{L}$.

Das heißt insbesondere, daß das Spracherkennungsproblem für eine NP–vollständige Sprache $\mathcal{L}$ so komplex ist, daß das Spracherkennungsproblem einer beliebigen NP Sprache durch das Spracherkennungsproblem für $\mathcal{L}$ gelöst werden kann – abgesehen von dem polynomialen Zeitaufwand für die Transformation f, der aber nicht ins Gewicht fällt.

484 Theorem THEOREM VON COOK
Das Problem der *Erfüllbarkeit (satisfiability)* einer aussagenlogischen Formel ist NP–vollständig.

Genauer: Sei $\mathcal{L}_A \subseteq A^*$ die bereits früher definierte Sprache aller aussagenlogischer Formeln über dem Alphabet $A = \{t, f, \wedge, \vee, \Rightarrow, \Leftrightarrow, \neg, (,), v, '\}$. Wir haben eine aussagenlogische Formel *erfüllbar (satisfiable)* genannt, wenn den atomaren Aussagen Wahrheitswerte so zugeordnet werden können, daß die Formel den Wahrheitswert W bekommt. Sei nun $\mathcal{S} \subset \mathcal{L}_A \subseteq A^*$ die Menge aller erfüllbaren aussagenlogischen Formeln. $\mathcal{S}$ kann durch eine TURING–Maschine akzeptiert werden und ist sogar eine entscheidbare Menge. Das Spracherkennungsproblem, ob ein $w \in A^*$ auch Element von $\mathcal{S}$ ist, ist NP-vollständig. Das bedeutet: Es gibt eine nichtdeterministische TURING–Maschine und ein Polynom P mit folgender Eigenschaft: Wird das Wort $w \in A^*$ auf das Eingabeband der TURING–Maschine geschrieben, so hat diese nach höchstens $P(\mid w \mid)$ Schritten festgestellt, ob $w \in \mathcal{S}$ ist – sofern $w \in \mathcal{S}$ ist. Die Zeitkomplexität dieser nichtdeterministischen TURING–Maschine ist also polynomial. Des weiteren kann das Erkennungsproblem jeder anderen Sprache, deren Erkennungsproblem in NP liegt, polynomial auf das Erkennungsproblem von $\mathcal{S}$ reduziert werden.

485 Beispiel NP–VOLLSTÄNDIGE PROBLEME
Die folgenden Probleme sind NP–vollständig:

(1) PROBLEM DES HANDLUNGSREISENDEN
Vorgegeben sind Städte und ein Straßennetz, das diese Städte verbindet. Wir kennen die Längen dieser Straßen. Ein Handlungsreisender soll nun mit dem Auto so durch alle diese Städte fahren, daß er jede Stadt genau einmal besucht und am Ende der Reise wieder am Ausgangspunkt ankommt. Unter allen möglichen Routen, die er so auswählen kann soll er jene Route suchen, bei der er am wenigsten Kilometer mit dem Firmenauto fährt.

(2) RUCKSACK PROBLEM
Vorgegeben sind n Steine mit den ganzzahligen Gewichten $a_1, \ldots, a_n$ und eine ganze Zahl $b \in \mathrm{N}$. Welche Steine muß man in den Rucksack geben, damit sein Gewicht genau b beträgt?

(3) Scheduling von Aufgaben
Es sollen n Aufgaben erledigt werden, die jeweils die Arbeitszeiten $t_1, \ldots, t_n$ benötigen. Insgesamt stehen m Maschinen zur Verfügung, von denen jede alle Aufgaben erledigen könnte. Zu ermitteln ist eine Maschinenbelegung, welche alle Aufgaben in möglichst kurzer Gesamtzeit erledigt.

(4) Chromatische Zahl
Zu einem Graph ist die chromatische Zahl zu berechnen.

(5) Registerbelegung
Einem Prozessor stehen neben dem Hauptspeicher auch eine kleine Anzahl sehr schneller Register zum Speichern von Zwischenresultaten zur Verfügung. Für einen vom Prozessor zu berechnenden Ausdruck ist jene Zuordnung von Zwischenresultaten zu Registern und zum Hauptspeicher zu ermitteln, für welche der Ausdruck möglichst schnell ausgewertet werden kann.

(6) Hamiltonsche Wege
Zu einem Graph sind alle seine Hamiltonschen Wege gesucht.

(7) Chip und Leiterplatten Entwurf
Beim Entwurf eines Chips oder einer Leiterplatte sind die elektronischen Bauteile so zu plazieren, daß die dafür benötigte Fläche minimal ist, zugleich aber gewisse durch die Elektronik bedingte Nebenbedingungen erfüllt sind, wie etwa bestimmte Mindestabstände für benachbarte Leitungen.

(8) Minimaler Verschnitt
Aus einer rechteckigen Aluminiumplatte sind unterschiedliche Formen auszustanzen. Wie sind diese Formen anzuordnen, daß möglichst wenig Verschnitt entsteht.

Bei diesen Problemen handelt es sich zunächst nicht um Spracherkennungsprobleme. Wir können die Aufgaben aber jeweils so umformulieren, daß sich ein Spracherkennungsproblem ergibt. Beim Problem des Handlungsreisenden etwa müssen wir die Angaben, also den Graphen und die Abstandsinformation in einer formalen Sprache geeignet codieren: Dies ergibt ein Wort $w \in A^*$, das die Problemstellung beschreibt. Eine mögliche Lösung dieser Aufgabe ist ein Hamiltonscher Weg durch den Graphen. Wir codieren nun alle möglichen Hamiltonschen Wege in einer weiteren Sprache. Ein Wort $w \cdot u$ aus der Konkatenation dieser beiden Sprachen stellt also durch das Wort w eine Aufgabenstellung und durch das Wort u eine mögliche Lösung dar. Nun wollen wir die Sprache aller Konkatenationen dieser Form $w \cdot u$ betrachten, bei denen u tatsächlich eine kostenminimale Rundfahrt im durch w codierten Graphen ist. Das Sprachzugehörigkeitsproblem dieser Sprache ist NP–vollständig. Die Lösung dieses Problems enthält zudem auch die Bestätigung, daß alle anderen Rundreisen im Graphen gleich hohe oder höhere Kosten verursachen, vom Aufwand

her ist sie also der Lösung des Optimierungsproblems äquivalent.

486 Bemerkung PRAKTISCHE KONSEQUENZEN
Für den Anwender ist es wichtig, typische Beispiele für NP–vollständige Probleme zu kennen und zu wissen, daß diese in der Praxis meist nicht gelöst werden können, da sie im allgemeinen zu astronomischen[2] Rechenzeiten führen. Zu diesem "im allgemeinen" gibt es einige Ausnahmen. Es kann sein, daß Sie nur *kleine Probleminstanzen* lösen müssen. So kann das Problem des Handlungsreisenden für eine kleine Anzahl von Städten noch innerhalb vernünftiger Zeit gelöst werden. Die volle Problematik dieser Aufgabe zeigt sich erst bei Städtezahlen, die ein Handlungsreisender ohnehin nicht auf einer Tour besucht. Es ist denkbar, daß ihre Aufgabenstellung gewisse *Zusatzeigenschaften* erfüllt, die eine schnellere Lösung gestatten. So haben etwa Speicherbausteine einen sehr regelmäßigen Schaltplan, der die Lösung des Plazierungsproblems erleichtert. Eine weitere Möglichkeit stellen *Näherungslösungen und Heuristiken* dar, wenn Sie nicht die optimale Lösung eines Problems benötigen, sondern bereits mit einer hinreichend günstigen zufrieden sind.

Für den Theoretiker hält das Gebiet der NP–Vollständigkeit berühmte und wichtige, noch ungelöste Fragestellungen parat. Das Simulationstheorem besagt, daß eine nichtdeterministische TURING–Maschine durch eine deterministische simuliert werden kann, falls wir bereit sind, statt polynomialer Rechenzeit exponentielle Rechenzeit in Kauf zu nehmen. Das Theorem besagt aber nicht, daß es unmöglich ist, noch raffiniertere Simulationsmethoden zu finden. Es gibt zwar sehr viele Indizien, daß wir eine nichtdeterministische TURING–Maschine nicht schneller durch eine deterministische simulieren können, als in diesem Theorem angegeben, aber man verfügt über keinen Beweis. Falls jemand eines Tages eine schnellere Simulation fände, dann könnten NP–vollständige Probleme innerhalb von polynomialer Rechenzeit gelöst werden und die Komplexitätsklassen NP und P wären identisch. Diese Frage, ob NP = P ist, gehört zu den wichtigsten offenen Fragen der Informatik überhaupt. Eine positive Antwort würde die rasche Lösung sehr vieler wichtiger Aufgaben gestatten und wäre ein phänomenaler Fortschritt für die Informatik. Man vermutet aber, daß die Antwort "nein" lautet.

[2]Das ist hier nicht im übertragenen Sinn gemeint. Manche Probleme führen auch auf den schnellsten heute verfügbaren Computern zu Rechenzeiten, die größer sind als die bisherige Lebensdauer des Universums.