

NP-vollständige Probleme. Ein Artikel für das PS Wissenschaftliches Arbeiten. WS 2001/02*

Hannes Eder[†]

18. Dezember 2001[‡]

Zusammenfassung

Dieser Artikel ist eine Arbeit im Rahmen des Proseminars „Wissenschaftliches Arbeiten, Effiziente Algorithmen“ unter der Leitung von Prof. Dr. Helmut Veith und ist eine kurze Einführung in die Theorie der NP-Vollständigkeit.

Inhaltsverzeichnis

1	Das Rucksackproblem	2
2	3 mögliche Problemstellungen	2
2.1	Entscheidungsproblem	2
2.2	Berechnungsproblem	2
2.3	Optimierungsproblem	2
3	Problem & Probleminstanz	2
4	Deterministisch vs. Nichtdeterministisch	3
5	Berechnungsmodelle	3
6	3-COL Problem	4
7	Travelling Salesperson Problem (TSP)	4
7.1	Travelling Salesperson World Tour	4
8	Was heißt überhaupt „effizient“	5
9	Problemkomplexität	5
10	NP Nichtdeterministisch Polynomiell	5
11	NP-vollständige Probleme	6

*<http://stud.hanneseder.net/ws01/wa-effalg/>

[†]email: <Hannes@HannesEder.net>

[‡]aktualisiert am 14.10.2007

12 Das 1. NP-vollständige Problem	6
13 Lösungsansätze für NP-schwere Probleme	6
14 P=NP?	7
15 Schlußworte	7

1 Das Rucksackproblem

Die allgemeine Problemformulierung beim Rucksackproblem ist wie folgt: Gegeben ist eine Liste der Form $\langle \text{Gegenstand, Gewicht, Preis} \rangle$, sowie ein maximales Gewicht G und ein gewünschter Wert W . Die Tabelle sieht z.B. so aus:

Gegenstand	Gewicht	Wert
Luster	5500	14300,-
Schatulle	3200	8000,-
Schwert	1500	8500,-
...

2 3 mögliche Problemstellungen

Beim Rucksackproblem ergeben sich die folgenden 3 Fragestellungen:

2.1 Entscheidungsproblem

Gibt es eine Menge $S \subseteq$ Gegenstände, so dass das Gesamtgewicht $\leq G$ und der Gesamtwert $\geq W$ ist?

2.2 Berechnungsproblem

Berechne eine Lösung S , so dass das Gesamtgewicht $\leq G$ und der Gesamtwert $\geq W$ ist.

2.3 Optimierungsproblem

Berechne eine Lösung S , so dass das Gesamtgewicht $\leq G$ und der Gesamtwert *maximal* ist.

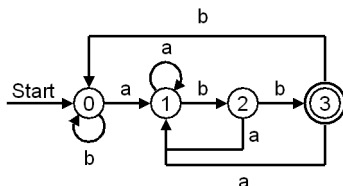
In der Komplexitätstheorie betrachtet man i.A. Entscheidungsprobleme.

3 Problem & Probleminstanz

Als *Problem* bezeichnet man die abstrakte Beschreibung der Aufgabenstellung ohne konkrete Daten, eine *Probleminstanz* oder kurz *Instanz* sind konkrete Daten zu einem Problem, i.A. existieren unendlich viele Instanzen zu einem Problem. Die Größe der Instanz wird häufig mit n bezeichnet und gibt je nach Art der Aufgabenstellung z.B. die Anzahl der Zeichen, um die Instanz zu kodieren, bei graphentheoretischen Betrachtungen die Anzahl der Knoten oder bei Sortier-Suchalgorithmen die Anzahl der Werte an.

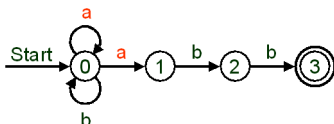
4 Deterministisch vs. Nichtdeterministisch

Bei einem deterministischen Automaten, das typischste Beispiel ist ein Computer, ist immer genau definiert, was im nächsten Schritt passieren soll, es gibt keine Mehrdeutigkeiten. Hier ein Beispiel für einen deterministischen Automaten wie man ihn beim Patternmatching finden kann. Dieser Automat erkennt die Sprache $(a|b)^*abb$.



In einem Computer sind daher keine „wirklichen Zufallszahlen“ möglich.

Im Gegensatz dazu stehen *nichtdeterministische* Automaten. Wie man für den nichtdeterministischen Automaten sieht, der auch die Sprache $(a|b)^*abb$ erkennt, gibt es im Zustand 0 *zwei* Übergänge für das Zeichen a. Der nichtdeterministische Automat hat nun die Eigenschaft *optimal zu raten*, sprich, wenn die Eingabe zum Erfolg führen kann, dann werden die passenden Übergänge richtig erraten. In der Praxis gibt es natürlich keinen derartigen Automaten, er spielt aber eine entscheidende Rolle bei der Klasse der NP Probleme. Das „N“ in NP steht für nichtdeterministisch. *Anmerkung: Beim Patternmatching haben nichtdeterministische Automaten eine große Bedeutung, sie werden mit Hilfe einer Zustandsmenge simuliert[1].*



Also z.B. um den Eingabestring **aababb**, der zur Sprache $(a|b)^*abb$ gehört zu erkennen, wird der nichtdeterministische Automat folgende Zusatzfolge annehmen: 0 0 0 0 1 2 3.

5 Berechnungsmodelle

Die Laufzeit eines Algorithmus hängt natürlich vom gewählten Berechnungsmodell ab. So kennt z.B. die Turing Maschine keine Arrays und diese müssen simuliert werden. Ein sehr häufig verwendetes Berechnungsmodell ist Pseudo-Code, der eine große Ähnlichkeit mit Pascal besitzt. Die These von Church besagt, dass jede im intuitiven Sinne berechenbare Funktion auch Turing berechenbar ist und des Weiteren, dass sich unterschiedliche Berechnungsmodelle nur um einen polynomiellen Faktor unterscheiden wie man anhand folgender Tabelle erkennen kann:

Simulierte/Simulierende Maschine	1TM	kTM	RAM
1-Band Turing Machine (1TM)	-	$O(T(n))$	$O(T(n) \log(T(n)))$
k-Band Turing Machine (kTM)	$O(T^2(n))$	-	$O(T(n) \log(T(n)))$
Random Access Machine (RAM)	$O(T^3(n))$	$O(T^2(n))$	-

man in polynomieller Zeit überprüfen, ob alle Städte besucht werden und ob die Gesamtstrecke unter der geforderten Schranke ist.

8 Was heißt überhaupt „effizient“

In dieser Tabelle ist die Laufzeit auf einem 1-MIPS Rechner von verschiedenen Zeitkomplexitätsklassen in Abhängigkeit von der Instanz Größe (n) dargestellt.

	n=10	n=20	n=30	n=40	n=50	n=60
n	10 μ s	20 μ s	30 μ s	40 μ s	50 μ s	60 μ s
n^2	100 μ s	400 μ s	900 μ s	1.6ms	2.5ms	3.6ms
n^3	1ms	8ms	27ms	64ms	125ms	216ms
n^5	0.1s	3.2s	24.3s	1.7min	5.2min	13min
2^n	1ms	1s	17.9min	12.7d	35.7a	366jhd
3^n	59ms	58min	6.5a	3855jhd	$2 \cdot 10^{10}a$	$1.3 \cdot 10^{15}a$

Wie man sieht ist für $n = 10$ und $n = 20$ ein Algorithmus mit einer Zeitkomplexität von 2^n sogar schneller als einer mit n^5 . Probleme mit einer Komplexität von 3^n sind für $n \geq 50$ praktisch nicht mehr berechenbar.

In der Komplexitätstheorie bezeichnet man alle Algorithmen, die eine polynomielle Zeitkomplexitätsfunktion haben, als *effizient*. Diese Klasse wird mit **P** bezeichnet, womit das „P“ in NP erklärt wäre. **NP heißt also „Nichtdeterministisch Polynomiell“**. In der Praxis treten jedoch keine Probleme mit einer Komplexität größer als n^8 auf.

9 Problemkomplexität

Wie man seit dem Gödel'schen Unvollständigkeitstheorem weiß, gibt es Probleme, die unentscheidbar sind. Ein Problem dieser Klasse ist z.B. das Halteproblem. Es lässt sich beweisen, dass es keinen Algorithmus gibt, der ein anderes Programm und gegebene Daten darauf hin untersucht, ob dieses Programm irgendwann einmal terminiert.

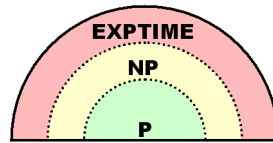
Wir wollen hier nur entscheidbare Probleme betrachten, die sich in folgende Klassen einteilen lassen:

1. Beweisbar exponentiell (Domino Problem, usw.)
2. NP (Rucksackproblem (RUCK), 3-Färbung (3-COL), Travelling Salesperson (TSP), Erfüllbarkeit (SAT), Minesweeper, usw.)
3. Beweisbar polynomiell (Sortieren, usw.)

10 NP Nichtdeterministisch Polynomiell

NP liegt also genau zwischen diesen beiden Klassen. Kein Problem in NP konnte bisher als *nicht* polynomiell bewiesen werden. Man kennt aber für viele Probleme in NP keinen polynomiellen Algorithmus.

Als Paradigma gilt: „*Guess and Check*“.



11 NP-vollständige Probleme

NP-vollständige Probleme (*englisch: NP-Complete*) bilden die „obere Schranke“ in der Klasse NP, also keine Problemstellung in NP ist schwerer als ein NP-vollständiges Problem. Alle diese NP-Vollständigen Probleme sind polynomiell ineinander überführbar, man spricht in diesem Zusammenhang auch von „Reduktion“.

Für kein NP-vollständiges Problem ist ein polynomieller Algorithmus bekannt. Würde dies aber gelingen, hätte das zur Folge, dass für alle NP-vollständigen Probleme, und somit für die ganze Klasse NP, polynomielle Algorithmen existieren. Es könnte aber auch noch für kein NPV-Problem gezeigt werden, dass kein polynomieller Algorithmus existiert.

Unter polynomieller Überführung oder Reduktion versteht man, eine Problem Instanz in eine andere in polynomieller Zeit zu transformiert und anschließend das transformierten Problems zu berechnet. Wenn ein Problem in ein anderes transformiert werden kann, sagt man, es ist „nicht schwerer als“ das andere.

12 Das 1. NP-vollständige Problem

Unabhängig voneinander veröffentlichten Stephen Cook und Leonid Levin Anfang der 70er Jahre je ein NP-vollständiges Problem. Cook veröffentlichte 1970 das Erfüllbarkeitsproblem (SAT) und zeigte, dass es NP-vollständig ist. Beim Erfüllbarkeitsproblem ist ein Boolescher Ausdruck in konjunktiver Form gegeben und die Fragestellung ist: gibt es eine Belegung der Variablen, so daß der Gesamtausdruck wahr ist. z.B. $A = (X \vee \neg Y) \wedge (\neg X \vee Y)$ ist erfüllbar ($X = 1, Y = 1$) hingegen ist $B = (X \vee Y) \wedge (X \vee \neg Y) \wedge (\neg X)$ nicht erfüllbar. Cook zeigt darüber hinaus, dass dieses Problem alle Eigenschaften eines NP-vollständigen Problems erfüllt, die Details würden aber weit über den Rahmen dieser Arbeit hinausgehen.

13 Lösungsansätze für NP-schwere Probleme

Leider ist es so, dass in der Praxis NP-vollständige Probleme häufig auftreten. Diese müssen mit geeigneten Methoden gelöst werden. Mögliche Ansätze:

1. Randomisierte lokale Suche
 - (a) Erzeuge zufälligen Lösungskandidaten
 - (b) Führe solange wie mögliche lokale Verbesserungen durch
 - (c) Wenn Lösung gefunden: Ausgeben und Programm beenden
 - (d) Wenn Zeitlimit erreicht, Abbruch: „Keine Lösung gefunden“

(e) Gehe zu Schritt a

Vorteile: Funktioniert erstaunlich gut für viele praktische Probleme und führt mit hoher Wahrscheinlichkeit zum Erfolg. Nachteile: Funktioniert schlecht, falls keine Lösung existiert; liefert keinen „Unlösbarkeitsbeweis“; funktioniert schlecht bei Instanzen mit wenigen Lösungen und funktioniert nur mit Bewertungsfunktion ist also ungeeignet zum Knacken von Codes.

2. Approximation

Ziel ist eine annähernd optimale Lösung zu finden. z.B. ist das Rucksackproblem beliebig approximierbar. Vorteile: Approximationen sind in der Praxis ausreichend und funktionieren auch für manche beweisbar exponentiellen Probleme gut; Nachteile: leider sind nur wenige praktische Probleme approximierbar; nicht auf Entscheidungsprobleme anwendbar.

3. Identifikation leicht lösbarer Subklassen

Hohe Komplexität ist oft nur „worst case“, so wird z.B. 3-COL für azyklische Graphen (Baum) trivial lösbar.

14 P=NP?

Die Frage ob die Klasse P = Klasse NP ist oder doch gilt $P \neq NP$ ist eine der wichtigsten Fragen der theoretischen Informatik, die bis dato weder bewiesen noch widerlegt werden konnte. Man nimmt heute jedoch allgemein an, dass $P \neq NP$ gilt. Diese Frage steht auf der Liste der 7 Millennium Prize Problems [5] des Clay Mathematic Institute und ist mit 1 Million US \$ dotiert.

15 Schlußworte

In Anlehnung an die letzten Zeilen in Robert Sedgewick Werk „Algorithmen“ [12], möchte ich sagen: „Die vielen effizienten Algorithmen, die wir in diesem Proseminar kennen gelernt haben, sind ein Beweis dafür, dass wir seit Euklid eine Menge über effiziente Berechnungsmethoden hinzugelernt haben, doch die Theorie der NP-Vollständigkeit zeigt, dass wir in Wirklichkeit noch sehr viel zu lernen haben“.

Dieses Manuskript wurde vollständig, im Schweiß meines Angesichts, in \LaTeX gesetzt, dabei waren das Tools TeXnicCenter [10] und die MikTeX [11] Umgebung von großem Nutzen. Eine gute und kurze Einleitung zu \LaTeX hab ich unter [2] gefunden.

Literatur

- [1] Alfred V. Aho. *Compilerbau*. Oldenbourg Wissenschaftsverlag GmbH, Wien, 1999.
- [2] Walter Schmidt et al. \LaTeX 2_ε-Kurzbeschreibung, l2kurz. <ftp://dante.ctan.org/tex-archive/info/lshort/german/>.

- [3] Dr. Georg Gottlob. Komplexität: Methoden zur Lösung schwieriger Probleme. <http://www.dbai.tuwien.ac.at/education/AIKonzepte/Folien/Komplexitaet.pdf>.
- [4] Dietmar Herrmann. *Algorithmen Arbeitsbuch*. Addison-Wesley (Deutschland) GmbH, München, 1992.
- [5] Clay Mathematics Institute. Millennium Prize Problems. <http://www.claymath.org/prizeproblems/index.htm>.
- [6] Michael R. Garey; David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. Freeman, New York, NY, 1979.
- [7] et al. Mario von Baratta. *Der Fischer Weltalmanach*. Fischer Taschenbuch Verlag, 1999. <http://www.weltalmanach.de/>.
- [8] Pablo Moscato. Tspbib homepage. http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html.
- [9] Dr. Sten Odenwald. How many atoms are there in our universe? <http://itss.raytheon.com/cafe/qadir/q1797.html>.
- [10] Paul Selormey Paul Selormey. TeXnicCenter 1 beta 5.05. <http://www.toolscenter.org>.
- [11] Christian Schenk. MikTeX 2.2. <http://www.miktex.org/>.
- [12] Robert Sedgewick. *Algorithmen*. Addison-Wesley, Bonn; München, 1991.
- [13] Steffen Reith und Dr. Heribert Vollmer. Wer wird Millionär? Komplexitätstheorie: Konzepte und Herausforderungen. *c't Magazin für Computer Technik*, 7:240–251, 2001. Eine lange Version dieses Artikels findet man unter http://user.cs.tu-berlin.de/~yxbe/p_np.pdf.