Hintergrund und Bedeutung des $P \neq NP$ Problems

Markus Unterkircher

4. Juni 2014

Inhaltsverzeichnis

1 Einleitung

Im Folgenden wird versucht, das in der Informatik sehr bekannte $P \neq NP$ Problem zu erklären und deren Bedeutung für die *reale* Welt dem Leser näherzubringen. Zuerst wird kurz auf die Geschichte des $P \neq NP$ eingegangen, dann die wichtigsten Begriffe der Thematik erklärt und mit den Auswirkungen abgeschlossen.

Das $P \neq NP$ Problem wurde von Stephen Cook und Leonid Levin unabhängig voneinader 1971 formuliert. Cook beschreibt das $P \neq NP$ Problem als ein Entscheidungsproblem, ob eine *formale Sprache*, die von einem *nichtdeterministischen* Algorithmus mit polynomieller Laufzeit akzeptiert wird, auch von einem *deterministischen* Algorithmus mit polynomieller Laufzeit akzeptiert wird [1].

Das *Clay Mathematics Institute* hat am 24. Mai 2000 am *Collège de France* sieben *Millennium Prize Problems*[1] veröffentlicht und für die Lösung jedes dieser Probleme ein Preisgeld von $ 1 Million ausgeschrieben. Das $P \neq NP$ Problem ist eines davon.

2 Formale Sprachen und deterministischer Algorithmus

Formale Sprachen sind abstrakte Sprachen, die sich vor allem für mathematisch präzise Beschreibungen eignen, aber auch für Programmiersprachen, Eingabe für formale Maschinenmodelle usw. Eine formale Sprache L besteht aus Wörtern aus einem endlichen Eingabealphabet Σ. Ein *deterministischer Algorithmus* gibt bei gleicher Eingabe immer dieselbe Ausgabe aus.

3 Komplexitätstheorie

Die *Komplexitätstheorie* ging aus den Werken von Alan Turing, Alonzo Church, Kurt Gödel und anderen in den 1930er Jahren hervor. Das Bestreben dieser Theorie ist es, grundlegende Aussagen zu machen, mit welchem Aufwand algorithmische Probleme auf einem formalen Maschinenmodell[2] gelöst werden können. Der Aufwand wird in Hinsicht auf die Bedeutung für den Anwender meist in benötigter Rechenzeit oder Speicherplatz gemessen[3]. Die Untersuchungen in der *Komplexitätstheorie* beschränken sich fast ausschließlich auf die *worst-case-Komplexität*. Die Algorithmen werden nach ihrer Komplexität in verschiedenen Klassen eingeteilt. P und NP sind solche Klassen [4].

4 Die Klasse *P*

Die Klasse P besteht aus den in polynomialer Zeit *lösbaren* Problemen. Genauer gesagt, sind es Probleme, die in einer Laufzeit von n^k mit konstantem k gelöst werden können, wobei n die Eingabegröße angibt. Berechtigtweise kann man ein Problem mit einer Laufzeit von n^{100} als unhandlich bezeichnen. Es gibt allerdings in der Praxis äußerst wenige Probleme, deren Lösung Zeit in der Größenordnung eines Polynoms mit derart hohem Grad erfordert. Die Erfahrung hat gezeigt, wenn einmal ein Algorithmus mit polynomialer Laufzeit für ein Problem gefunden wurde, häufig effizientere Algorithmen folgen. Die Klasse P besitzt angenehme Abgeschlossenheitseigenschaften, da Polynome bezüglich der Addition, der Multiplikation und der Komposition abgeschlossen sind.

[1] http://www.claymath.org/millennium-problems, Version vom 25. Mai 2014
[2] Am Häufigsten wird die *Turingmaschine* verwendet
[3] In diesem Text wird die *Rechenzeit* oder *Laufzeit* als Aufwand verwendet.

5 Die Klasse *NP*

Die Klasse *NP* besteht aus den in polynomialer Zeit *verifizierbaren* Problemen. Damit meint man, dass zum Verifizieren der Korrektheit einer gegebenen *Testlösung* Zeit benötigt wird, die polynomial von der Eingabegröße des Problems abhängt. Die Komplexität der Suche einer geeigneten Lösung bleibt hier jedoch verborgen. Diese Definition ist äquivalent zur ursprünglichen Definition von Stephen Cook. Erfahrungsgemäß ist es viel schwieriger ein Problem zu lösen, als eine klar dargestellte Lösung zu verifizieren, besonders hinsichtlich der Rechenzeit. Aus diesen Definitionen ist sofort ersichtlich, dass jedes Problem in P auch in *NP* liegt, da ein Problem in P sogar ohne gegebene Testlösung in polynomialer Zeit lösbar ist. Noch ungeklärt ist die Frage, ob die Menge P gleich der Menge *NP* ist. Der vielleicht zwingendste Grund, weshalb theoretische Informatiker glauben, dass $P \neq NP$ gilt, ist die Existenz der Klasse der *NP-vollständigen* Probleme.

6 Die Klasse *NPC*

Die Klasse der *NP-vollständigen* Probleme wird als *NPC* (*engl.* NP-C*omplete*) bezeichnet. *NP-vollständige* Probleme sind in gewissen Sinne die *härtesten* Probleme in *NP*. Anders ausgedrückt: kein Problem in *NP* ist *härter* zu lösen, als jedes in *NPC*. Die relative *Härte* von Problemen kann mithilfe einer als *Polynomialzeitreduktion* bezeichneten Notation untereinander verglichen werden. Formal ausgedrückt kann eine Sprache L_1 auf die Sprache L_2 in *polynomialer Zeit reduziert* werden (notiert als $L_1 \leq_p L_2$), wenn eine in polynomialer Zeit berechnbare Funktion f existiert, sodass $x \in L_1 \iff f(x) \in L_2$ gilt. Eine Sprache L ist *NP-vollständig*, genau dann, wenn L in *NP* liegt und $L' \leq_p L$ für jede Sprache L' in *NP*. Dies ist auch der essentielle Punkt in der Argumentation, dass $P \neq NP$ gilt: Wenn ein *NP-vollständiges* Problem in polynomialer Zeit lösbar ist, dann besitzt jedes Problem in NP eine Lösung in polynomialer Zeit, das heißt es gilt $P = NP$. Bisher wurde kein Algorithmus mit polynomialer Laufzeit für ein NP-vollständiges Problem entdeckt [2].

7 Sudoku ist *NP-vollständig*

Als Beispiel für ein *NP-vollständiges* Problem wird hier das bekannte Logik-Puzzle *Sudoku* kurz gezeigt. Das allgemeine *Sudoku*-Problem n-ter Ordnung, n ist eine natürliche Zahl und $N = n^2$, besteht darin, auf einem $N * N$ Gitter die Ziffern $1 - N$ so zu verteilen, dass in jeder Zeile und Spalte sowie in jedem $n * n$-Block jede Ziffer genau einmal auftritt. Einige dieser Felder sind bereits mit Ziffern zwischen 1 und N gefüllt. Die Laufzeit eines Lösungsalgorithmus wächst exponentiell mit N.

Takayuki Yato und Takashiro Seta haben 2002 bewiesen, dass das verallgemeinerte *Sudoku*-Problem *NP-vollständig* ist[4]. Um sich ein Bild der Lösungsmöglichkeiten zu machen: Im Standard $9 * 9$ *Sudoku* gibt es insgesamt $\approx 6,671 * 10^{21}$ verschiedene vollständig ausgefüllte Gitter [5].

8 Auswirkungen

Die Komplexitätstheorie und das $P \neq NP$ Problem spielen in sehr vielen Bereichen eine wichtige Rolle, zum Beispiel in der Kryptographie. Die dort verwendeten Verschlüsselungen hängen von Annahmen aus der Komplexitätstheorie ab, wie etwa von der Schwierigkeit der Primfaktorzerlegung. Falls $P = NP$ gilt, sind diese Annahmen alle falsch und es gäbe effiziente Algorithmen, die Verschlüsselungen dekodieren können. Natürlich würde es dann aber auch für alle anderen *NP*-Probleme effiziente Lösungsalgorithmen geben. Michael Garey und David Johnson haben 1979 über 300 *NP-vollständige* Probleme aufgelistet [3]. Deren Lösung hätten direkte Auswirkungen auf die ganze Welt. Aber nicht nur bekannte Probleme könnten gelöst werden. Computer wären dazu in der Lage, schwierigste mathematische Beweise zu finden, physikalische Theorien zu erstellen oder sogar kreative Aufgaben wie Komponieren von Musikwerken zu übernehmen[6]. Das $P \neq NP$ Problem ist eines der wichtigsten Probleme unsere Zeit und kann vorraussichtlich auch nicht in naher Zukunft gelöst werden.

Literatur

[1] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest, Clifford Stein, and Paul Molitor. *Algorithmen - eine Einführung.* Oldenbourg, Deutschland, 2004.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[4] Karl R Reischuk. *Einführung in die Komplexitätstheorie.* B.G. Teubner, Stuttgart, 1990.

[4] http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/SIGAL87-2.pdf, Version vom 25. Mai 2014

[5] http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf, Version vom 25. Mai 2014

[6] http://www.claymath.org/sites/default/files/pvsnp.pdf, Version vom 25. Mai 2014