Überlegungen zum P-NP-Problem

In meinem Informatikstudium hat mich das *P-NP*-Problem ungemein fasziniert, weil es sich augenscheinlich um ein sehr schwieriges Problem handelt, an dem sich schon viele kluge Köpfe versucht haben, ohne eine Lösung gefunden zu haben. Man weiß nicht, ob *P* gleich oder ungleich *NP*. Da aber eine Gleichheit leicht zu beweisen wäre, nimmt man an, dass *P* ungleich *NP*. Nur: Wir sind hier nicht in den Naturwissenschaften, wo es zulässig ist, mit unbewiesenen Annahmen zu arbeiten, sondern in der Mathematik bzw. Theoretischen Informatik, wo eine Aussage nur dann als wissenschaftlich gesichert gilt, wenn man sie schlüssig beweisen kann. Gerade der Fall, dass *P* ungleich *NP* sein sollte, ist aber, sollte er richtig sein, sehr schwer zu beweisen.

Ich habe in den Jahren 2013 und 2014 immer wieder über das *P-NP*-Problem nachgedacht und meine Gedanken niedergeschrieben. Hier eine Zusammenfassung meiner wichtigsten Überlegungen aus jener Zeit.

Claus D. Volko, cdvolko@gmail.com

Einführung

Das *P*-*NP*-Problem ist eines der schwierigsten noch ungelösten Probleme der Informatik. Es zählt zu den Millennium-Prize-Problemen, und die erste Person, die eine Lösung publiziert, wird vom Clay Mathematics Institute 1 Million Dollar bekommen.

P und *NP* sind Mengen von Entscheidungsproblemen, die eine bestimmte Komplexität haben. Umgangssprachlich nennt man *P* und *NP*, wenn auch nicht ganz korrekt, deswegen auch Komplexitätsklassen. Dies bedeutet, dass es einen Algorithmus gibt, der das jeweilige Entscheidungsproblem löst und dessen Laufzeit selbst im schlechtesten Fall ein gegebenes Limit nicht überschreiten wird. Egal mit welchen Eingabedaten man den Algorithmus ausführt, benötigt der Algorithmus, wenn das zu ihm gehörende Entscheidungsproblem in die Klasse *P* fällt, nur eine polynomielle Anzahl von Schritten in Bezug auf die Größe der Eingabedaten (also eine Konstante multipliziert mit der Größe der Eingabedaten hoch eine andere Konstante), um die Ausgabedaten zu berechnen. Im Gegensatz dazu bedeutet *NP*, dass es möglich ist, eine Kandidatenlösung in einer polynomiellen Zeit in Bezug auf die Größe der Lösung auf Korrektheit zu überprüfen.

Jeder Algorithmus, der in *P* ist, ist auch in *NP*. Hat man die Lösung für ein gegebenes Problem in *P*, kann man diese Lösung auch in einer polynomiellen Anzahl von Schritten verifizieren. Die Frage ist, ob es auch andersrum möglich sei. Die meisten Informatiker denken, nein. Sollte sich aber herausstellen, dass jedes Problem in *NP* auch in *P* ist, wäre das eine wahre Revolution, weil sie die Berechnung vieler Probleme deutlich beschleunigen würde.

Um zu beweisen, dass *P* und *NP* gleich sind, würde es reichen, es für ein einziges *NP*-vollständiges Problem zu beweisen. *NP*-vollständige Probleme stellen eine Teilmenge von *NP* dar, mit der Eigenschaft, dass sie zumindest so schwierig wie jedes andere Problem in NP sind. Ein Problem, für das diese Eigenschaft bewiesen wurde, ist das Erfüllbarkeitsproblem der Aussagenlogik (SAT, Satz von Cook und Levin). Um für ein anderes Problem zu zeigen, dass dieses ebenfalls *NP*-vollständig ist, reicht es aus zu beweisen, dass es in *NP* ist, und dass es zumindest so schwierig wie SAT ist. Letzteres kann erreicht werden, indem man einen Weg findet, SAT zu diesem Problem zu reduzieren, also einen Weg zu finden, wie ein Algorithmus für dieses neue Problem Instanzen von SAT lösen könnte. Bis jetzt wurde noch kein einziger polynomieller Algorithmus für ein *NP*-vollständiges Problem gefunden.

Es scheint realistischer zu sein, einen Beweis dafür zu finden, dass *P* ungleich *NP* ist, aber auch diese Suche gestaltet sich sehr schwierig. Eine Möglichkeit wäre es zu zeigen, dass es für ein bestimmtes *NP*-vollständiges Problem gar keinen Algorithmus mit polynomieller Laufzeit geben kann, aber wie sollte man dies bewerkstelligen? Zu beweisen, dass etwas nicht möglich ist, scheint viel schwieriger zu sein als das Gegenteil zu beweisen, und aus diesem Grund wird das *P*-*NP*-Problem immer noch als ungelöst betrachtet.

Claus D. Volko, cdvolko@gmail.com

Formalisierung des P-NP-Problems

Das *P*-*NP*-Problem lässt sich durch Prädikatenlogik zweiter Ordnung (second order logic) formalisieren.

NP bedeutet, dass eine nichtdeterministische Turingmaschine existiert, die für jede Lösung deren Gültigkeit in polynomieller Zeit nachprüfen kann. Eine nichtdeterministische Turingmaschine ist im Prinzip eine Sammlung von Algorithmen; wenn das Problem in *NP* liegt, gibt es für jede Lösung mindestens einen Algorithmus, der diese Lösung in polynomieller Zeit überprüfen kann. Bei *P* gibt es einen Algorithmus, der jede Lösung in polynomieller Zeit überprüfen kann. Es gilt also:

$NP \equiv \forall x \exists A A(x)$
$P \equiv \exists A \forall x A(x)$

Es lässt sich leicht zeigen, dass $NP(\pi) \rightarrow P(\pi)$, aber nicht $P(\pi) \rightarrow NP(\pi)$. Ein Beispiel dafür ist $\pi \equiv A(x) \wedge B(y) \wedge \neg A(y) \wedge \neg B(x)$.

Auf logischer Ebene kann man also zeigen, dass ein Problem π, definiert durch eine Lösungsmenge $\{x, y\}$ und eine Algorithmenmenge $\{A, B\}$, das Prädikat *NP* erfüllen kann, ohne *P* zu erfüllen. Das eigentliche Problem darin besteht zu zeigen, dass diese Eigenschaft auf einen real existierenden Algorithmus zutrifft.

Claus D. Volko, cdvolko@gmail.com

Zu bisherigen Beweis-Versuchen

Immer wieder werden "Beweise" veröffentlicht, die entweder in die eine oder in die andere Richtung gehen. Von der Fachwelt werden die meisten gar nicht mehr ernst genommen. Das liegt daran, dass es leider auch sehr viel Arbeit ist, einen Beweis logisch zu überprüfen, um eventuelle Fehler zu finden. Interessant finde ich persönlich aber, dass es auch Beweisversuche gibt, die behaupten, das *P*-*NP*-Problem wäre unlösbar. Was ist da dran?

Zunächst ist das *P*-*NP*-Problem nicht unentscheidbar, weil es sich ja nicht um ein Entscheidungsproblem handelt. Paper, die das *P*-*NP*-Problem als unentscheidbar bezeichnen, sind daher schon aus diesem Grund fragwürdig.

Man könnte aber untersuchen, ob ein verwandtes Entscheidungsproblem entscheidbar ist. Dieses Entscheidungsproblem bezeichne ich als die Menge $NP \setminus P$. Es ist mit dem *P*-*NP*-Problem insofern verwandt, als P genau dann gleich NP ist, wenn $NP \setminus P$ gleich der leeren Menge ist. Man kann sich nun Gedanken darüber machen, ob die Menge $NP \setminus P$ entscheidbar ist. Zur Lösung des *P*-*NP*-Problems bringt es meiner Ansicht nach nicht viel. Was würde es denn implizieren, wenn $NP \setminus P$ entscheidbar sein sollte, und was, wenn $NP \setminus P$ unentscheidbar sein sollte? Meiner Meinung nach gar nichts. Auch wenn $NP \setminus P$ unentscheidbar sein sollte, könnte es grundsätzlich möglich sein, das *P*-*NP*-Problem zu lösen. Wenn $NP \setminus P$ entscheidbar wäre, dann würde das alleine auch noch nicht das *P*-*NP*-Problem lösen. So gesehen, ist die Frage der Entscheidbarkeit der Menge $NP \setminus P$ eine reine Übungsaufgabe ohne Relevanz für das eigentliche Problem. Ich lade aber gerne ein, sich an dieser Übungsaufgabe zu versuchen. Die Lösung folgt auf der nächsten Seite.

Claus D. Volko, cdvolko@gmail.com

Lösung der Übungsaufgabe

Ein Turing-Entscheider mag vielleicht in vielen Fällen in der Lage sein zu bestimmen, ob ein Problem in der Menge *NP* liegt, aber nicht in allen Fällen. Damit ein Problem in *NP* sein kann, muss es mindestens einen Algorithmus geben, der die Gültigkeit einer Lösung überprüft und für beliebig große Eingabedaten in polynomieller Zeit abläuft. Da scheint es schon grundsätzlich schwierig zu sein, den Grenzwert dieser Laufzeit zu ermitteln. Wenn man auch feststellen kann, dass sich die Laufzeiten bei relativ großen Instanzen polynomiell verhalten, wie soll man sagen können, dass sie sich auch bei noch größeren Instanzen so verhalten werden? Zudem könnten manche Algorithmen bei allen Eingabedaten ab einer bestimmten Größe in Endlosschleifen geraten. In diesem Fall wäre die Ermittlung der Komplexität auf das Halteproblem reduzierbar, welches bekanntlich unentscheidbar ist.

Claus D. Volko, cdvolko@gmail.com

Könnte man das P-NP-Problem so lösen?

Kurz nach meinem dreißigsten Geburtstag stellte ich die folgende Überlegung an:

Ein Problem ist genau dann in *NP*, wenn es einen polynomiellen Algorithmus (in Bezug auf die Größe der Eingabedaten) gibt, der überprüft, ob eine gegebene Lösung korrekt ist.

Ein Problem ist genau dann in *P*, wenn es in *NP* ist und zusätzlich die Anzahl der Lösungen, die überprüft werden müssen, so dass mit hundertprozentiger Sicherheit die korrekte Lösung darunter ist, polynomiell ist (ebenfalls in Bezug auf die Größe der Eingabedaten).

Ich definiere:

Lösungsraum: ist die Anzahl der Lösungen, die es überhaupt geben kann, egal ob korrekt oder nicht.

Suchraum: ist die Anzahl der Lösungen, die unbedingt durchsucht werden müssen, damit mit hundertprozentiger Sicherheit die korrekte Lösung darunter ist (ohne Raten und ohne vorherige Kenntnis der korrekten Lösung).

Die Frage ist, ob es Probleme gibt, die in *NP*, aber nicht in *P* liegen.

Bekannt ist: Es gibt *NP*-vollständige Probleme; nur wenn ein *NP*-vollständiges Problem in *NP*, aber nicht in *P* liegt, gibt es Probleme, die in *NP*, aber nicht in *P* liegen.

Ein *NP*-vollständiges Problem ist k-*SAT*: Gegeben sind n boolesche Variablen (können die Werte wahr oder falsch annehmen), diese kommen in c Clauses (Disjunktionen) vor, jeweils k Variablen pro Clause; alle Clauses sind miteinander durch Konjunktion überprüft.

Ich möchte zeigen, dass der Suchraum bei k-*SAT* nicht polynomiell und daher k-*SAT* nicht in *P* enthalten ist.

Der Lösungsraum beträgt 2^n, da jede Variable entweder wahr oder falsch sein kann, also genau zwei Werte annehmen kann. Der tatsächliche Suchraum ist kleiner als 2^n, aber im allgemeinen Fall nicht polynomiell. Der Grund: Es gibt keine andere Möglichkeit vorzugehen, als einer Variablen einen Wert zuzuweisen und dann zu schauen, wie sich das auf die anderen Variablen auswirkt. Wenn k-*SAT* in *P* wäre, dann müsste es in jeder Formel die gleiche Anzahl von Variablen geben, deren Wert man festlegen muss, so dass sich die Werte aller anderen Variablen ergeben. Wenn das t Variablen sind, dann ist der Suchraum $\binom{n}{t}$. Wenn t konstant ist, ist das polynomiell. t ist aber nicht konstant. Vielmehr stellt die Anzahl der Clauses eine obere Schranke für t dar. t kann also unter Umständen so groß sein wie die Anzahl der

Clauses. Da die Anzahl der Clauses bei k-*SAT* aber variabel ist, ist der Suchraum nicht polynomiell (bei der Angabe einer oberen Schranke käme n im Exponent vor).

Bei der Diskussion darüber in einem Forum zeigte sich, dass die Schwachstelle an diesen Ausführungen in der Aussage "Es gibt keine andere Möglichkeit vorzugehen, als einer Variablen einen Wert zuzuweisen und dann zu schauen, wie sich das auf die anderen Variablen auswirkt" besteht. Diese Aussage ist unbewiesen. Offenbar ist das aber auch die einzige Schwachstelle - zumindest konnte keine weitere gefunden werden. Dass ein Problem, das in *NP*, aber nicht in *P* liegt, einen nicht-polynomiellen Suchraum haben muss, lässt sich leicht beweisen; es ist eine logische Folgerung aus obigen Definitionen. Die Schwierigkeit besteht bloß darin zu zeigen, dass es nicht möglich ist, den Suchraum zu generieren, indem man einer Anzahl von Variablen einen Wert zuweist, wobei diese Anzahl der Variablen höchstens polynomiell in Bezug auf die Gesamtzahl der Variablen ist, sondern dass dieses Verhältnis nicht-polynomiell ist. Einige Kollegen meinten, ich wäre der Lösung des Problems nicht nähergekommen, sondern hätte es nur auf eine andere Ebene verlagert. Wie dem auch sei, vielleicht ist das Problem auf dieser Ebene leichter lösbar.

In diesem Zusammenhang möchte ich vorschlagen, vielleicht gar nicht über *SAT* zu diskutieren, sondern über ein anderes, anschaulicheres Problem:

Gegeben ist eine Konstante c. Gesucht sind n natürliche Zahlen ungleich 1 (denn 1 ist das neutrale Element der Multiplikation, was das Problem trivial machen würde), die miteinander multipliziert diese Konstante c ergeben.

Dieses Problem scheint mir genauso komplex wie *SAT* zu sein, es scheint mir aber einfacher zu analysieren zu sein. Ich wüsste für dieses Problem auch keine andere Möglichkeit, als nach einem Algorithmus vorzugehen, der für n - 1 Variablen die Werte durchiteriert, wobei sich der Wert der n-ten Zahl ergeben würde. Vielleicht könnte man für dieses Problem leichter als für *SAT* zeigen, dass es gar nicht anders gelöst werden kann. Vorschläge dazu wären sehr willkommen.

Claus D. Volko, cdvolko@gmail.com

Weitere Überlegungen

Alle Probleme in *P* haben die Gemeinsamkeit, dass der Suchraum polynomiell ist, wenn man so vorgeht, dass man alle relevanten Lösungen durchprobiert und mit einem polynomiellen Algorithmus (der ja existiert, weil *P* eine Teilmenge von *NP* ist) überprüft. Das bedeutet aber auch, dass der Suchraum maximal cn^k verschiedene Lösungen enthalten kann, wenn n die Größe der Eingabedaten ist und k und c Konstanten. Sonst wäre er nicht polynomiell. Die Größe der Datenstruktur, die genügt, um diesen Suchraum zu repräsentieren, beträgt daher $k \log n + \log c$ bit.

Ein jedes Problem in *P* erlaubt also, dass alle relevanten Lösungen in einer Anzahl von Bit dargestellt werden können, die sich zur Größe der Eingabedaten logarithmisch verhält, also viel kleiner als die Eingabedaten ist.

Ein Beispiel dafür: Das Problem, ob zwei Zahlen a und b einander teilerfremd sind, ist in *P*, denn ein gemeinsamer Teiler der beiden Zahlen muss zwischen 1 und a liegen, wenn a die kleinere der beiden Zahlen ist. Man muss also nur a verschiedene Lösungen durchprobieren. Das Problem ist polynomiell. Die Zahl a wird durch $1 + \log a$ bit dargestellt.

Ein etwas komplexeres Beispiel: Das Problem 2-COL, das sich damit beschäftigt, ob ein Graph mit nur zwei Farben färbbar ist, so dass zwei miteinander verbundene Knoten zwei verschiedene Farben zugewiesen bekommen, ist in *P*. Die Datenstruktur der Lösung scheint zunächst so beschaffen zu sein, dass es 2^n verschiedene Zahlenwerte gibt, wobei n die Anzahl der Knoten ist. Schließlich liegt es nahe, dass man eine Datenstruktur aus n bit verwendet, wobei jedes Bit entweder 0 oder 1 sein kann, je nachdem, welche Farbe dem Knoten zugewiesen wurde. Tatsächlich aber lässt sich jede Lösung eindeutig darstellen, indem ich einfach nur die Nummer eines Knoten abspeichere, der eine bestimmte Farbe zugewiesen bekommt - denn alle anderen Farbwerte können von einem Algorithmus in polynomieller Zeit ermittelt werden. Es geht sogar noch einfacher: Ich kann auch einen Anfangsknoten festlegen und einfach die Farbe abspeichern, die er zugewiesen bekommt, also 0 oder 1. Eine Instanz dieses Problems, die nicht mit zwei Farben färbbar ist, hat als Lösungsmenge die leere Menge.

Das heißt: Wäre $P = NP$, könnte ich dies beweisen, indem ich für ein *NP*-vollständiges Problem eine Darstellung der Lösungen fände, deren Größe $k \log n + \log c$ bit beträgt. Nehme ich ein *NP*-vollständiges Problem, zum Beispiel *SAT*, her, müsste ich zeigen können, dass ich jede Lösung so komprimieren kann, dass sich die Größe der komprimierten Lösung im Vergleich zur Originallösung asymptotisch betrachtet logarithmisch verhält. Das ist nur dann möglich, wenn ich bei einer Anzahl von n zu ermittelnden Werten bereits durch die Ermittlung von $\log n$ Werten alle anderen Werte eindeutig bestimmen kann. Wenn ich also zwei Variablen in *SAT* habe, muss die Angabe des Wertes einer Variablen genügen, um den Wert der zweiten Variablen eindeutig zu bestimmen; bei drei bis vier Variablen müssen zwei Werte genügen, bei fünf bis acht Variablen drei Werte usw. Wenn ich für eine

einzige konkrete Instanz von SAT beweisen kann, dass sie auf diese Weise nicht lösbar ist, dann ist damit $P \neq NP$ bewiesen. Jedenfalls zumindest sofern meine hier dargelegte "line of reasoning" stimmt.

Claus D. Volko, cdvolko@gmail.com