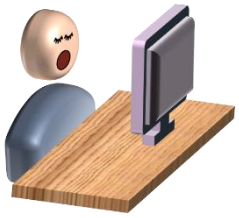
Das P-NP-Problem

Eine wenig bekannte Kardinalfrage

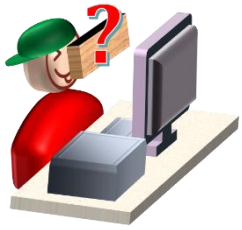
Ausarbeitung zum Vortrag im Rotary-Club Frankfurt am Main Friedensbrücke

Dr. Daniel Korn 12.03.2020

1 Einleitung



Wir alle kennen dieses nervtötende Phänomen fast von Anbeginn unserer ersten Erfahrungen als Computeranwender an bis zum heutigen Tage: quälend langsam rechnende Programme, auf deren Ausgabe man eine scheinbare Ewigkeit warten muss. Meist dauert es in solchen Momenten nicht allzu lange bis einem der Geduldsfaden reißt und man sich die Frage stellt, ob das denn wirklich so lange dauern muss.



Diese in den meisten Fällen eher der subjektiven Befindlichkeit geschuldete Frage hat dabei durchaus einen ernsthaften Hintergrund. Letztlich geht es dabei nämlich um die Grundsatzfrage, ob der konkret mit der Entwicklung der aktuell genutzten Software betraute Programmierer bei seiner Arbeit einfach nur ein Brett vor dem Kopf hatte oder ob es einen objektiven Anhaltspunkt dafür gibt, dass man das betreffende Problem programmiertechnisch nicht effizienter lösen kann. Auch wenn es dem durchschnittlichen IT-Anwender nicht bewusst sein dürfte: von der Beantwortung dieser Frage hängen praktisch die Fundamente der Datensicherheit in unserer gesamten IT-Welt ab. Die Wesentlichen der heute verwendeten Verschlüsselungsverfahren für sichere Datenübertragung basieren nämlich gerade auf der Annahme, dass bestimmte Berechnungen prinzipbedingt „schwer“ sind, so dass die entsprechenden Verschlüsselungen nur mit einem Zeitaufwand geknackt werden können, der weit jenseits jedweder Praktikabilitätsgrenze liegt.

Derartige Grundsatzüberlegungen zur Komplexität von Berechnungsvorgängen fallen innerhalb des Fachgebiets der theoretischen Informatik in den Bereich der sogenannten „Komplexitätstheorie“. Und schon die klassischen Vordenker unserer modernen Informatik wie Kurt Gödel und John von Neumann haben sich daher bereits in den 1950er Jahren mit den Grundlagen der Komplexitätstheorie beschäftigt. Insbesondere dem US-amerikanischen Wissenschaftler Stephen Cook und dem sowjetischen Wissenschaftler Leonid Levin ist es aber auf Basis ihrer in den 1970er Jahren unabhängig voneinander veröffentlichten Arbeiten zu verdanken, dass wir seither über sehr präzise Konzepte verfügen, mit denen sich die Frage nach der objektiven Mindestkomplexität einer automatisierten Berechnung beschreiben und untersuchen lässt: die Theorie der sogenannten „NP-Vollständigkeit“.

Praktisch gesagt ist die „NP-Vollständigkeit“ eines Problems ein Synonym dafür, dass das betreffende Problem in intuitivem Sinne „schwer“ – also nur mit sehr hohem Zeitaufwand – zu berechnen ist. Aber es steckt noch weitaus mehr in dieser formell recht eigenwilligen Charakterisierung der „schweren Probleme“: die Definition von NP-Vollständigkeit sollte nämlich

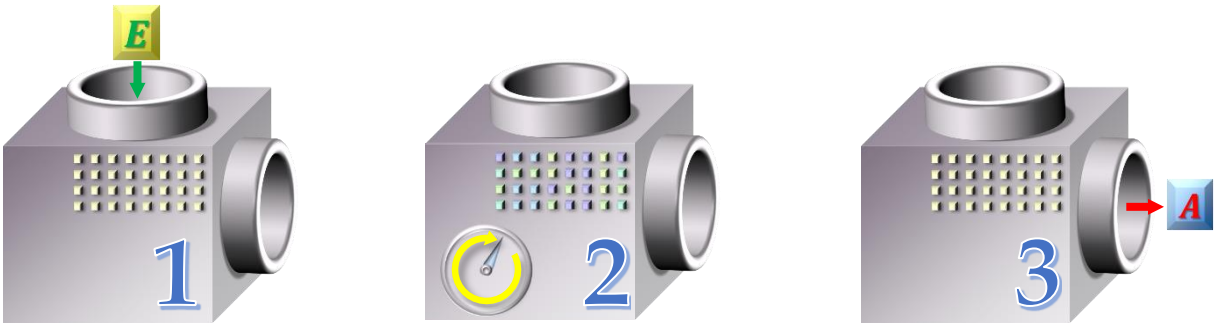
einen relativ leicht zu konstruierenden formalen Beweis dafür ermöglichen, dass die NP-vollständigen Probleme tatsächlich gerade nicht zur Klasse jener Probleme gehören, für die man effiziente Lösungsverfahren kennt – nämlich all jene Probleme, die in der sogenannten Problemklasse „P“ enthalten sind.

Diese ursprüngliche Hoffnung hat sich bis heute leider nicht bestätigt. Die stattdessen bis heute ungelöste Fragestellung, ob die Mengen NP und P gleich oder ungleich sind, ist gemeinhin als das „P-NP“-Problem bekannt geworden und gilt als *die* Kardinalfrage der theoretischen Informatik schlechthin – ähnlich wie einstmals der große Fermat'sche Satz für die Zahlentheorie. Nicht ohne Grund gehört das P-NP-Problem zu den „Millenniumproblemen“, auf deren Lösung das Clay Mathematics Institute einen Preis in Höhe von $ 1.000.000,00 ausgesetzt hat.

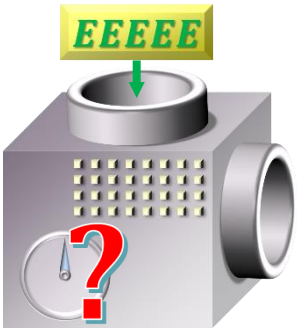
Um die Hintergründe dieses Grundsatzproblems zu beleuchten, soll in diesem Vortrag zunächst veranschaulicht werden, wie der Begriff vom Aufwand eines Berechnungsvorgangs in der Komplexitätstheorie definiert ist. Mit diesem Wissen ausgestattet werden wir dann unterschiedliche Klassen von Berechnungskomplexitäten kennenlernen, aus denen wir insbesondere die bereits oben erwähnte Klasse „P“ der mit „polynomiellem Aufwand“ berechenbaren Probleme herausstellen werden. Anschließend werden wir die Grundgedanken hinter der Klasse „NP“ kurz anreißen und motivieren, warum alle bekannten „schweren Probleme“ zu dieser Klasse gehören. Schließlich werden wir uns mit der Frage befassen, in welchem Verhältnis P – als Klasse der effizient berechenbaren Probleme – zur Klasse NP steht und welche extrem einschneidende Bedeutung die unterschiedlichen Möglichkeiten für unser informationstechnologisch geprägtes Zeitalter haben.

2 Komplexität einer Berechnung

Um sich dem Begriff von der Komplexität einer Berechnung zu nähern, betrachtet man in der Komplexitätstheorie einen Berechnungsvorgang abstrakt als Apparatur, in die man oben eine Eingabe einwirft, sodann die Berechnung auf Basis dieser Eingabe auslöst und schließlich nach einer gewissen Anzahl von Berechnungsschritten eine Ausgabe erhält:



Die wesentliche Frage für das Maß der Komplexität einer Berechnung ist jetzt diejenige nach der Auswirkung, die es auf die Berechnungszeit hat, wenn die Größe der Eingabe um einen bestimmten Betrag wächst:



Was damit gemeint ist, soll am klassischen Lehrbeispiel eines Sortierprogrammes demonstriert werden. Die Eingabe eines solchen Programms ist klassischer Weise eine unsortierte Liste vergleichbarer Elemente – in unserem Fall schlichtweg die Zahlen 1 bis 7:

Die Berechnung der Sortierung beginnt nun, indem man das linksäußerste Element der Liste – in unserem Fall die 7 – nimmt, und es so lange um eine Position nach rechts verschiebt, bis keine kleinere Zahl mehr recht von unserer 7 steht:

Hierfür werden also offensichtlich **sechs** Schritte benötigt.

Als nächstes nimmt man wieder das linksäußerste Element – jetzt also die 6 – und verschiebt diese wiederum so lange um eine Position nach rechts, bis keine kleinere Zahl mehr rechts davon steht:

Hierfür werden also offensichtlich **fünf** Schritte benötigt. Insgesamt hat unsere Berechnung bisher also $\mathbf{6 + 5 = 11}$ Schritte benötigt. Es sollte nicht allzu schwer fallen zu erkennen, dass die nun anstehende schrittweise Verschiebung der 5 weitere **vier** Schritte benötigt, diejenige der 4 dann noch weitere **drei** Schritte, das Verschieben der 3 weitere **zwei** Schritte und schließlich das Verschieben der 2 nur noch **einen** weiteren Schritt benötigt. Die 1 steht dann schon links von der 2 und muss daher nicht mehr verschoben werden. Am Ende erhalten wir also insgesamt nach $\mathbf{6 + 5 + 4 + 3 + 2 + 1 = 21}$ Schritten die gewünschte sortierte Liste als Ausgabe unserer Berechnung:

Das Sortieren der Eingabeliste mit 7 Elementen hat also 21 Berechnungsschritte erfordert. Es liegt auf der Hand, dass bei einer Eingabeliste mit 8 Elementen dann noch sieben Schritte dazukommen, so dass wir für eine Liste mit 8 Elementen insgesamt $\mathbf{7 + 6 + 5 + 4 + 3 + 2 + 1 = 7 + 21 = 28}$ Schritte benötigen. Allgemein hat man es also für eine Eingabeliste der Länge $\boldsymbol{n}$ mit der folgenden Anzahl an Berechnungsschritten zu tun:

$$\mathbf{1 + 2 + 3 + \cdots + (n - 2) + (n - 1)}$$

Um die Anzahl der Berechnungsschritte unseres Sortierverfahrens für eine Liste mit $\boldsymbol{n}$ Elementen zu bestimmen, muss man demnach alle Zahlen von **1** bis $\boldsymbol{n - 1}$ zusammenzählen. Dafür gibt es eine bekannte Formel, die auf eine Anekdote aus dem Leben des Mathematikers Carl Friedrich Gauß zurückgeht:

Als Gauß im Jahre 1786 als damals Neunjähriger im Volksschulunterricht saß, der ja seinerzeit für alle vier Grundschulklassen gleichzeitig in einem Raum abgehalten wurde, wollte der Lehrer die dritte Klasse, der Gauß damals angehörte, für eine Zeit lang beschäftigen, damit er die nächste Klasse unterrichten konnte. Die dafür vorgesehene Aufgabe lautete, eben gerade alle Zahlen von **1** bis **100** zusammenzuzählen. Der junge Gauß erkannte sofort, dass er sich das Leben erheblich leichter machen könne, wenn er die Aufgabe wie folgt angeht:

Er zählt zunächst die **1** und die **100** zusammen – macht **101**. Dann zählt er die **2** und die **99** zusammen – macht wieder **101**. Dasselbe gilt für die **3** und die **98** usw. bis er schließlich zur **50** und **51** gelangt. Insgesamt erhält er also **50** mal die **101**, und das ergibt **5050**. Nach wenigen Minuten trat der kleine Gauß damit vor seinen Lehrer, der die außergewöhnliche Begabung des Jungen erkannte und ihn ab dann bei seiner Karriere als angehender Mathematiker förderte.

Die zugehörige allgemeine Formel um alle Zahlen von **1** bis n zusammenzuzählen lautet demnach:

$$(n+1) \times \frac{n}{2}$$

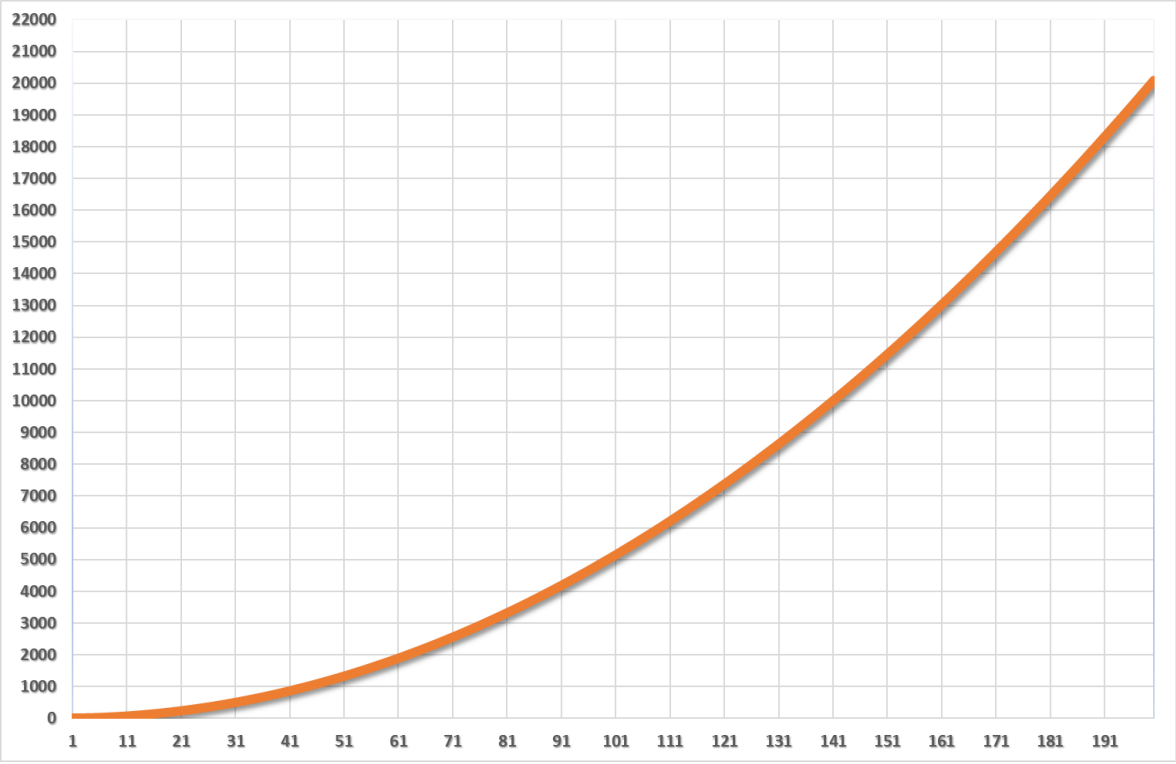
Hier das Beispiel für die ursprüngliche Aufgabe des kleinen Gauß – also $n = 100$:

$$(100+1) \times \frac{100}{2} = 101 \times 50 = 5050$$

Diese, gerne auch als „der kleine Gauß“ bezeichnete Formel lässt sich wie folgt umformen:

$$(n+1) \times \frac{n}{2} = \frac{n^2+n}{2}$$

Die Anzahl der Berechnungsschritte unseres Sortierverfahrens wächst somit im Wesentlichen zum Quadrat der Eingabelänge, denn für wachsende n setzt sich der oben in Rot markierte quadratische Anteil immer stärker durch, was man an folgendem Diagramm gut erkennen kann:



Hier sind unten wachsende n aufgetragen, während die orange Kurve die links abzulesenden Werte repräsentiert, die sich für das jeweilige n aus der „kleinen Gauß“-Formel ergeben. Man

erkennt unschwer, dass es sich um einen nicht-linearen, in diesem Fall also quadratischen Verlauf handelt.

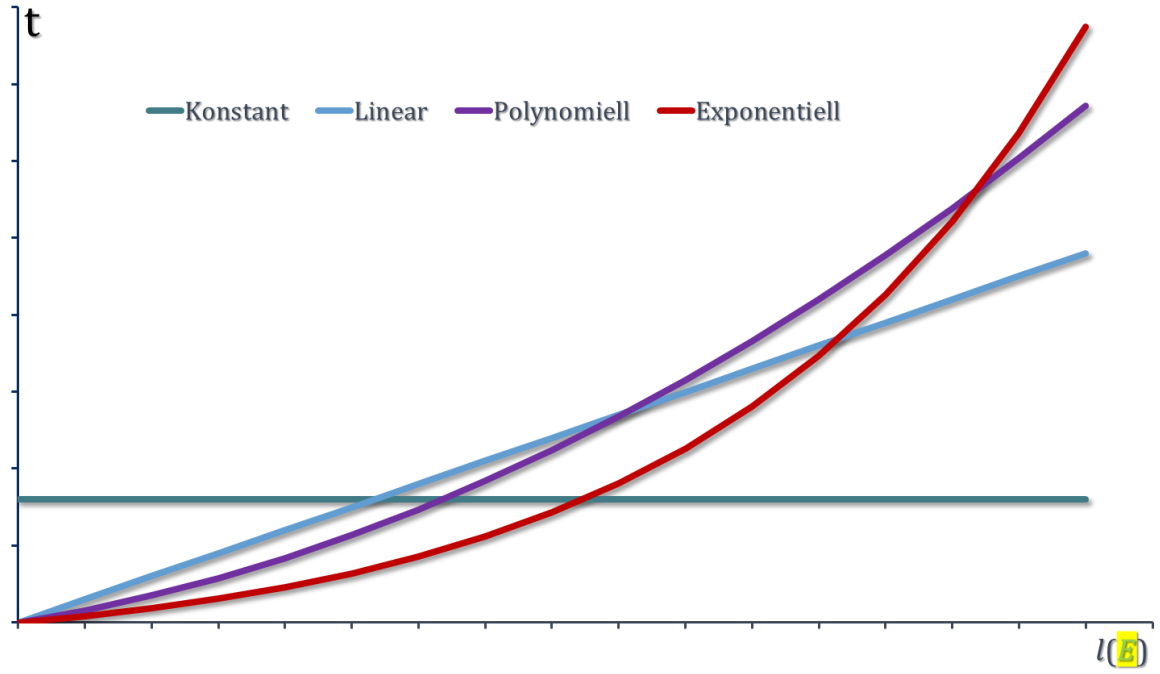
3 Die Problemklasse „*P*“

Das im vorangegangenen Abschnitt beispielhaft betrachtete Sortierproblem hat also eine im Wesentlichen quadratischen Berechnungsaufwand im Verhältnis zur Länge seiner Eingabe. Dies ist natürlich nur eine von sehr vielen verschiedenen Komplexitätsverhältnissen. Andere Probleme könnten Beispielsweise eine konstante Komplexität haben. Das in der Programmierwelt beliebte „Hello World“-Programm, das unabhängig von seiner Eingabe nichts anderes tut, als den Satz „Hello World“ auszugeben, wäre beispielsweise ein solcher Kandidat.

Wiederum andere Programme könnten eine lineare Komplexität haben. Ein Programm, das etwa die Summe einer Liste von Eingabezahlen berechnet, wäre ein solcher Kandidat, denn es hätte im Wesentlichen eine einzelne Addition pro Zahl in der Eingabeliste durchzuführen.

Von „polynomieller Komplexität“ spricht man, wenn bei einer Eingabelänge n im Wesentlichen n^k Berechnungsschritte für ein festes k zur Lösung des betrachteten Problems erforderlich sind. Das oben betrachtete Sortierproblem ist mit $k = 2$ etwa ein solches Problem mit polynomieller Komplexität.

Probleme, deren Lösung im Verhältnis zur Eingabelänge n im Wesentlichen nur mit mindestens k^n Schritten für ein festes k berechnet werden kann, bezeichnet man als Probleme mit „exponentieller Komplexität“. Das folgende Diagramm veranschaulicht diese verschiedenen Komplexitätsarten:



Dabei steht die untere Achse für die Länge der Eingabe und die linke Achse für die Anzahl der Berechnungsschritte. Man sieht deutlich, dass lineare Verläufe über kurz oder lang (also

für genügend große Eingabelängen) aufwändiger als konstante Verläufe sind, ebenso wie polynomielle Verläufe über kurz oder lang aufwändiger als lineare Verläufe sind. Exponentielle Verläufe hingegen, sind für genügend große Eingabelängen immer aufwändiger als polynomielle Verläufe.

Probleme, für die es eine Lösung mit höchstens polynomiellen Komplexitätsverläufen gibt, werden gemeinhin gerade noch als effizient lösbare Probleme betrachtet. Alles, was aufwändiger ist als das, wird gemeinhin als „schwer lösbares" Problem betrachtet.

Daher hat man in der Komplexitätstheorie die Klasse ***„P"*** eingeführt. Sie enthält alle Probleme, für die ein Berechnungsverfahren mit höchstens polynomieller Komplexität bekannt ist. Anders ausgedrückt, enthält ***P*** all jene Probleme, für die ein effizient berechenbares Lösungsverfahren bekannt ist.

Basierend auf dieser Definition der effizient berechenbaren Probleme haben sich die Komplexitätstheoretiker nun Gedanken darüber gemacht, wie man nachweisen kann, dass ein bestimmtes gegebenes Problem ***p*** eben gerade nicht in ***P*** enthalten ist, was soviel bedeutet, wie dass es kein Berechnungsverfahren für ***p*** mit polynomieller Komplexität und damit kein effizient berechenbares Lösungsverfahren für ***p*** geben kann.

Diese Frage, die zunächst wie ein Hirngespinst aus dem praxisenthobenen Elfenbeinturm übereifriger Akademiker erscheinen mag, hat indessen eine enorme Bedeutung für unseren praktischen Alltag: alle gängigen Verschlüsselungsverfahren für die sichere Übertragung digitaler Daten in elektronischen Netzwerken basieren nämlich auf der Annahme, dass die automatisierte Suche nach den für die Dechiffrierung nötigen Schlüsseln nur mit extrem hohem Berechnungsaufwand möglich und somit in der Praxis undurchführbar ist.

Diese Annahme fußt ihrerseits jedoch auf rein empirischen Erkenntnissen: es ist lediglich bisher kein Verfahren bekannt, mit dem man die betreffenden Dechiffrierungsschlüssel effizient (sprich: mit polynomieller Komplexität) berechnen kann. Das schließt aber keineswegs aus, dass ein solches Verfahren eines Tages entwickelt wird, womit ein erheblicher Teil der heute im Einsatz befindlichen Verschlüsselungstechniken wertlos wäre und die Sicherheit unserer täglichen Datenübertragungen damit auf einen Schlag verloren gehen würde!

Daher ist die Suche nach einem formellen Beweis dafür, dass die effiziente Berechnung bestimmter Problemstellungen kategorisch unmöglich ist, essentiell für die Frage, ob unsere heutigen Datenchiffrierungstechniken auch wirklich nachhaltig sicher sind. Wie man sich dieser Beweisführung nähern kann, soll im folgenden Abschnitt illustriert werden.

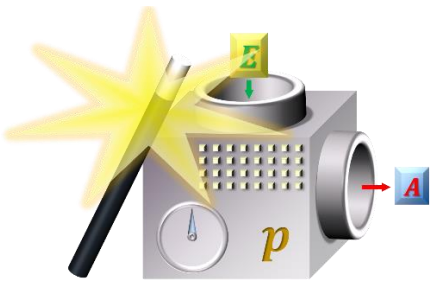
4 Die Problemklasse „*NP*"

Im Sinne der obigen Erläuterungen geht es bei der Frage nach einem Beweis für die kategorische Unmöglichkeit eines effizienten Berechnungsverfahrens für ein gegebenes Problem ***p*** letztlich um die Frage, wie man beweisen kann, dass besagtes ***p*** kategorisch nicht in ***P*** enthal-

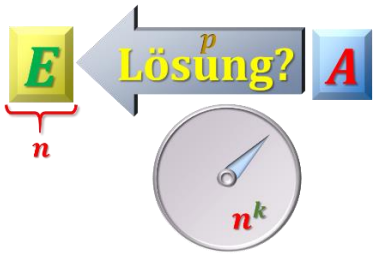
ten sein kann. Das wiederum ist ja gemäß der oben dargelegten Begriffsdefinition gleichbedeutend damit, dass es kein Berechnungsverfahren mit höchstens polynomieller Komplexität für p geben kann.

Auf der Suche nach geeigneten Instrumenten, um so einen Beweis führen zu können, sind die Komplexitätstheoretiker bei einer weiteren Klasse von Problemen fündig geworden: der sogenannten Klasse „NP“. Diese ist im Wesentlichen so definiert:

Anstatt zu einem gegebenen Problem p dessen Lösung explizit zu berechnen, stellt man sich vor, es gäbe eine Art Orakel, das eine mögliche Lösung für p ohne jeglichen Berechnungsaufwand erraten könne.

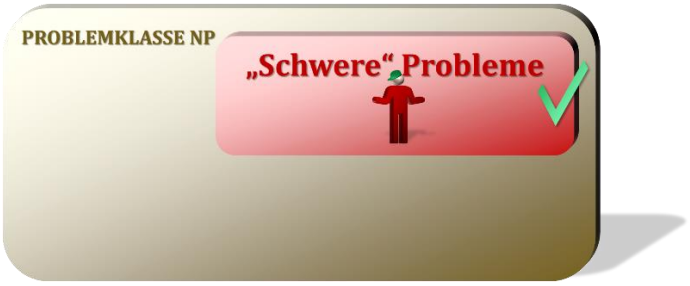


Liegt ein solcher geratener Lösungskandidat vor, betrachtet man jetzt lediglich den Berechnungsaufwand, der anfällt, um zu überprüfen, ob dieser Kandidat auch wirklich eine Lösung für das gegebene Problem p ist. Sofern eine solche Überprüfung mit polynomieller Komplexität möglich ist, gehört das betrachtete Problem p zur Klasse NP.

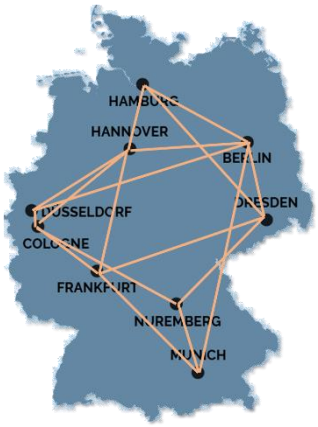


Diese auf den ersten Blick eher etwas okkultistisch anmutende Definition mit magisch erratenen Lösungskandidaten mag befremdend erscheinen. Im Wesentlichen enthält die Klasse NP aber letztlich all jene Probleme, für die sich mit polynomiellen Berechnungsaufwand (und damit auf effiziente Weise) feststellen lässt, ob ein gegebener Lösungskandidat tatsächlich eine Lösung des jeweils betrachteten Problems ist.

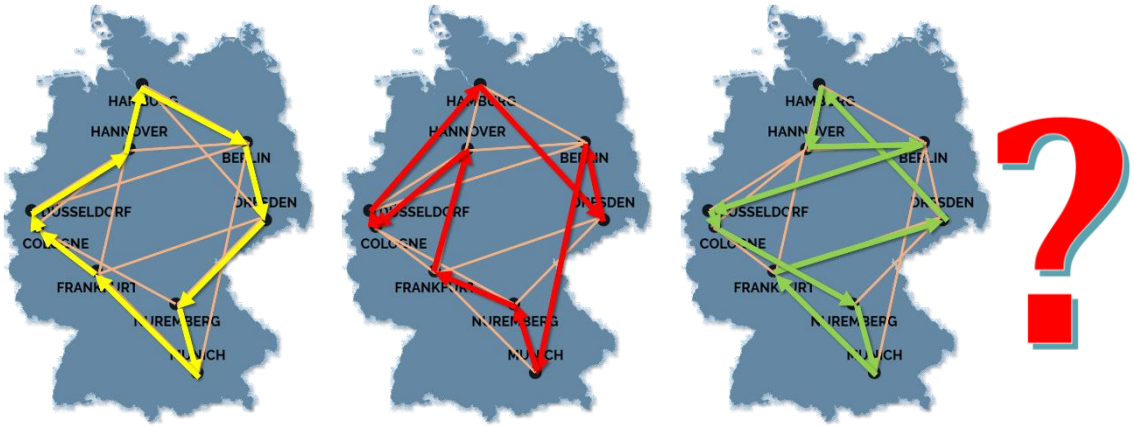
Das eigentlich Bedeutsame an der so definierten Klasse NP ist aber, dass alle bekannten „schweren“ Probleme nachweislich in NP enthalten sind.



Die Details des zugehörigen Nachweises gehen über den Rahmen dieses Vortrags hinaus, sind aber in der einschlägigen Literatur nachzulesen. Im Folgenden wollen wir uns stattdessen etwas genauer mit der Natur dieser „bekannten schweren Probleme“ befassen, um ihre hohe Berechnungskomplexität plausibel zu machen. Tatsächlich handelt es sich praktisch ausschließlich um klassische Optimierungsprobleme, wie etwa den „Travelling Salesman“, bei dem es um die Frage geht, wie man die Kosten für eine Rundreise zwischen einer vorgegebenen Menge von Orten optimieren kann. Neben dieser Menge von Orten sind zudem die möglichen Verbindungen zwischen je zwei Orten und die jeweiligen Kosten einer jeden solchen Verbindung vorgegeben:



Es muss nun vom Ausgangspunkt aus ein Weg entlang dieser Verbindungen gefunden werden, auf dem jeder zu besuchende Ort genau einmal enthalten ist und der am Ende wieder zum Ausgangspunkt führt. Die Summe der Kosten aller dabei verwendeten Einzelverbindungen muss zudem noch minimal sein:



Die hohe Komplexität, die für die Berechnung einer solchen kostenminimierten Rundreise anfällt, ist dabei vor allem dem Umstand geschuldet, dass im Wesentlichen vom Ausgangsort aus jede mögliche Verbindung zu einer Nachbarstadt geprüft werden muss, von wo aus dann wieder jede mögliche Anschlussverbindung geprüft werden muss usw. Geht man jetzt etwa von durchschnittlich **3** möglichen Verbindungen pro Stadt aus, so müssen bei beispielsweise **9** Städten im Wesentlichen $3 \times 3 \times 3 \times 3 \times 3 \times 3 \times 3 \times 3 \times 3$ und damit $\times 3^9$ Verbindungskombinationen betrachtet werden. Diese Art der Suche hätte damit also im Wesentlichen eine exponentielle Berechnungskomplexität.

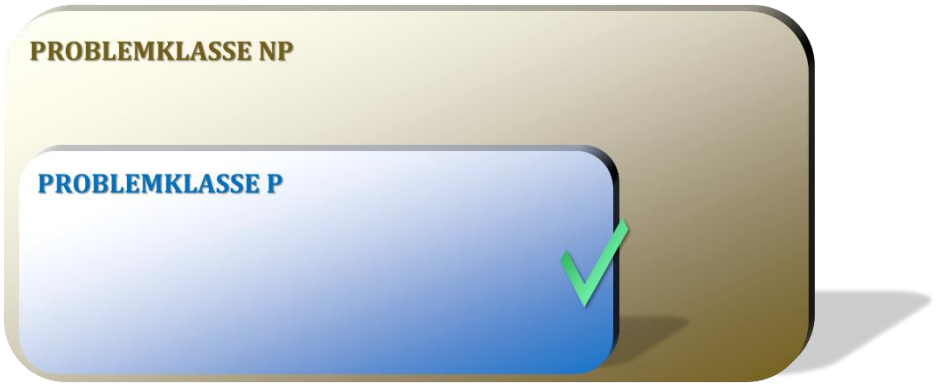
Auch wenn der Travelling Salesman zunächst nur ein akademisches Problem ist, steht er doch für viele praxisrelevante Fragestellungen wie etwa Routenberechnung, Logistikoptimierung,

Mikrochip-Layoutoptimierung oder Genomsequenzierung. Schon vor diesem Hintergrund ist die Frage, inwieweit ein effizientes (=polynomielles) Lösungserfahren für Probleme nach Art des Travelling Salesman kategorisch ausgeschlossen ist, durchaus mehr als nur ein theoretisch interessantes Problem.

5 $P = NP$?

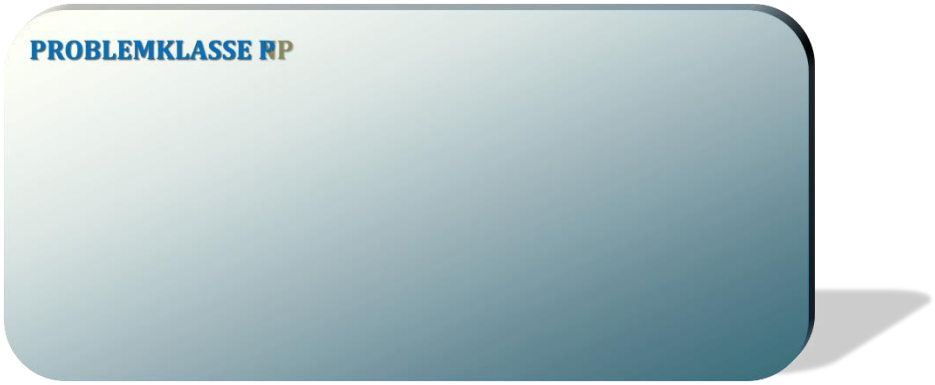
Mit Hilfe der nun eingeführten Klassen P und NP reduziert sich die Frage, ob man die Existenz polynomieller Lösungsverfahren für „schwere Probleme“ kategorisch ausschließen kann, auf die Frage nach dem Verhältnis, in welchem die beiden Klassen P und NP tatsächlich zueinander stehen. Klar ist, dass P eine Teilmenge von NP sein muss.

Um das einzusehen, betrachtet man ein beliebiges Problem p aus P. Nun lassen wir unser gedankliches Orakel eine mögliche Lösung für p erraten. Dann müssen wir nichts weiter tun, als die eigentliche Lösung für p zu berechnen, was wegen dessen angenommener Mitgliedschaft in P per definitionem mit polynomiellem Aufwand möglich ist, und vergleichen diese Lösung dann mit der vom Orakel geratenen Lösung (was sogar mit linearer und damit allemal polynomieller Komplexität möglich ist). Insgesamt haben wir also mit polynomiellen Aufwand überprüfen können, ob ein vom Orakel geratener Lösungskandidat für p tatsächlich eine Lösung für p ist. Dies ist aber genau die Definition für Probleme, die in der Klasse NP enthalten sind, so dass p demnach auch in NP enthalten sein muss. Insgesamt gilt daher, dass P eine Teilmenge von NP ist:



Die eigentliche Kardinalfrage ist jetzt aber diejenige, ob es Probleme in NP gibt, die *nicht* in P enthalten sind. Mit anderen Worten: ist P eine *echte* Teilmenge von NP – also $P \neq NP$ – oder ist $P = NP$?

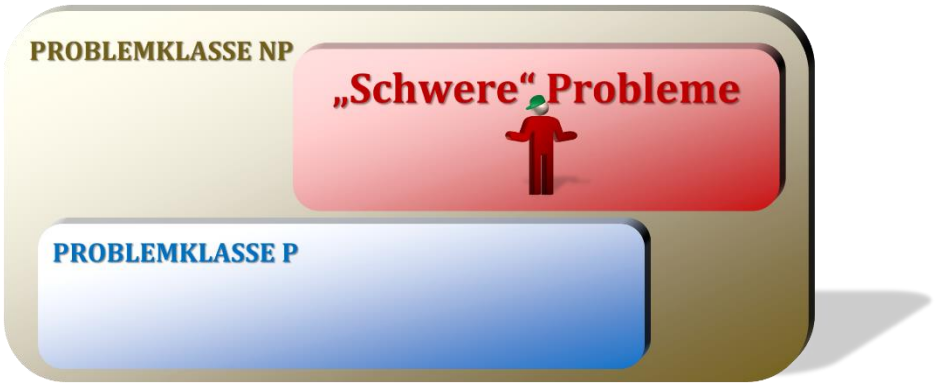
Wie oben dargelegt, sind alle „schweren Probleme“ erwiesenermaßen in NP enthalten. Nehmen wir also an, es wäre tatsächlich $P = NP$:



Dann müssten aber die „schweren Probleme“ ebenfalls in ***P*** enthalten sein, da sie ja erwiesenermaßen in ***NP*** enthalten sind, welches dann dasselbe wie ***P*** wäre:

Das wiederum würde bedeuten, dass es für die „schweren Probleme“ Lösungsverfahren mit polynomieller Berechnungskomplexität geben müsste. Auf die dramatischen Folgen dieses Umstands für die Sicherheit unserer alltäglichen digitalen Datenübertragungen haben wir bereits in Abschnitt 3 hingewiesen: die wesentlichen heute im Einsatz befindlichen Verschlüsselungsverfahren würden auf einen Schlag wertlos, und fast unsere gesamte verschlüsselte Datenkommunikation wäre für jedermann zu knacken!

Tatsache ist aber, dass man bis heute für keines der bekannten „schweren Probleme“ ein Lösungsverfahren kennt, das mit polynomieller Berechnungskomplexität aufwarten würde. Daher muss im Moment – zumindest von einem empirischen Standpunkt aus gesehen – davon ausgegangen werden, dass die „schweren Probleme“ nicht in ***P*** enthalten sind und die Klasse ***NP*** insoweit zumindest um die „schweren Probleme“ größer ist als ***P***:



Tatsache ist aber auch, dass es für diese empirische Beobachtung bis heute keinen formalen Beweis gibt.

Basierend auf der formalen Definition der „schweren Probleme“ (die – wie in der Einleitung erwähnt – im Fachjargon eigentlich „***NP***-vollständige Probleme“ heißen) würde es schon genügen, wenn man nur für *ein einziges* dieser Probleme mit formellen Mitteln beweisen könnte, dass es dafür kein Lösungsverfahren mit polynomieller Komplexität geben kann. Ein solcher Beweis würde dann automatisch auf alle anderen „schweren Probleme“ wirken.

Gerade deshalb ist man ja letztlich beim Versuch, diese Kardinalfrage zu beantworten, den Weg über die ansonsten etwas befremdend anmutende Klasse ***NP*** gegangen. Ebenso würde es genügen, für nur ein einziges der „schweren Probleme“ ein Lösungsverfahren mit polynomieller Berechnungskomplexität zu finden, damit gleich alle „schweren Probleme“ mit polynomiellem Berechnungsaufwand lösbar wären.

Weder das Eine noch das Andere ist aber bisher gelungen, obwohl so mancher renommierte Komplexitätstheoretiker unserer Zeit bisweilen geglaubt hat, eine Antwort auf diese Frage gefunden zu haben, und so bleibt das P-NP-Problem vorläufig auch weiterhin *die* Kardinalfrage der theoretischen Informatik, auf deren Lösung das das Clay Mathematics Institute demnach aus gutem Grund ein Preisgeld von $ 1.000.000,00 ausgelobt hat.

Ob diese Frage jemals gelöst wird, ist Gegenstand vieler Spekulationen. Möglicherweise wird es aussichtsreicher sein, eines Tages den formellen Beweis dafür anzutreten, dass das P-NP-Problem selbst aus denkgesetzlichen Erwägungen heraus kategorisch unlösbar ist. Bis dahin dürfen wir gespannt bleiben und einstweilen noch beruhigt auf die Robustheit unserer gängigen Datenverschlüsselungsverfahren vertrauen.