Von der Turingmaschine zum Quantencomputer – ein Gang durch die Geschichte der Komplexitätstheorie

Johannes Köbler, Olaf Beyersdorff

Humboldt-Universität zu Berlin
koebler,beyersdo@informatik.hu-berlin.de

Zusammenfassung. Die Komplexitätstheorie beschäftigt sich mit der Abschätzung des Aufwandes, welcher zur Lösung algorithmischer Probleme nötig ist. In diesem Aufsatz verfolgen wir die spannende Entwicklung dieses Teilgebiets der Theoretischen Informatik von ihren Wurzeln in den 30er Jahren des 20. Jahrhunderts bis in die heutige Zeit.

Die Informatik als eine den Anwendungen verpflichtete Wissenschaft sieht sich vor die Aufgabe gestellt, für praktisch auftretende Probleme möglichst gute Algorithmen zu finden. Anwendungsszenarien aus völlig verschiedenen Bereichen führen dabei auf einer abstrakteren Ebene häufig zu derselben Problemstellung. Oftmals lassen sich solche Probleme mit Methoden der Logik, Graphen oder anderen kombinatorischen Werkzeugen modellieren. Viele dieser Probleme sind seit Jahrzehnten intensiv untersucht worden, und für viele hat man gute Algorithmen gefunden: diese Algorithmen sind schnell und gehen sparsam mit dem Speicherplatz um. Eine große Klasse praktisch überaus relevanter Probleme jedoch hat sich einer befriedigenden algorithmischen Lösung bislang hartnäckig widersetzt. Trotz der stetig steigenden Leistungsfähigkeit moderner Rechner werden selbst für einfache Instanzen dieser Probleme derart immense Rechenkapazitäten benötigt, dass diese Probleme nach derzeitigem Wissensstand als praktisch unlösbar angesehen werden müssen. Woran liegt das? Warum sind manche Probleme relativ einfach und andere, oft ganz ähnliche Probleme, anscheinend algorithmisch viel komplizierter?

Antworten hierauf sucht die Komplexitätstheorie. Die Komplexitätstheorie klassifiziert Probleme anhand des Aufwands, der zu ihrer algorithmischen Lösung nötig ist. Die wichtigsten Aufwandsparameter sind die Rechenzeit und der Speicherplatzbedarf. Zwei Klassen von Problemen haben wir bereits grob unterschieden: solche mit guten Algorithmen und solche ohne. Aber auch die algorithmische Welt lässt sich nicht hinreichend durch eine Einteilung in schwarz und weiß erklären, und so ist in den letzten 40 Jahren eine beinahe schon unübersichtliche Vielzahl von Komplexitätsklassen definiert worden:

durch Kombinationen aus Zeit- und Platzschranken, aber auch durch die Einbeziehung anderer Ressourcen wie Zufall, Nichtuniformität, Kommunikationsaufwand oder Orakelanfragen. Einige dieser Klassen werden wir in unserem historischen Spaziergang durch die Komplexitätstheorie genauer vorstellen. Auf diesem Spaziergang werden uns typische Probleme aus der Zahlentheorie, der Logik und der Graphentheorie begleiten, anhand derer wir den stürmischen Fortschritt dieses Gebietes zu illustrieren versuchen.

1 Die 30er Jahre: die Anfänge, Turings Maschinenmodell und die Rekursionstheorie

Die Wurzeln der Komplexitätstheorie liegen in der Rekursionstheorie. Die Rekursionstheorie lotet die Grenzen des prinzipiell algorithmisch Machbaren, des Berechenbaren aus, ohne sich jedoch um den Zeit- und Platzbedarf von Algorithmen zu kümmern. Zentral ist hier zunächst die Präzisierung des Algorithmenbegriffs.

1.1 Die Formung des Algorithmenbegriffs

Das Wort Algorithmus geht auf den usbekischen Mathematiker Muhammad ibn Musa al-Chorezmi zurück, der im 9. Jahrhundert verschiedene Schriften u.a. zur Algebra und Arithmetik verfasste. Aus dem Namenszusatz al-Chorezmi, seine vermutliche Geburtsstadt Choresm bezeichnend, formte sich der Begriff Algorithmus. Diesen Begriff präzise zu definieren bestand lange Zeit keine Notwendigkeit. Man verstand unter einem Algorithmus einfach eine Vorschrift zum Lösen eines mathematischen Problems, etwa zum Auflösen von Gleichungen oder für geometrische Konstruktionen.

Diese Situation veränderte sich zu Beginn des 20. Jahrhunderts im Zusammenhang mit den Bemühungen um die Fundierung der Mathematik und der so genannten Grundlagenkrise. Auf dem 2. Internationalen Mathematikerkongress im Jahre 1900 in Paris stellte David Hilbert eine Liste von 23 ungelösten mathematischen Problemen vor, deren Bearbeitung er für die weitere Entwicklung der Mathematik große Bedeutung beimaß. In der Tat lieferte die Beschäftigung mit vielen dieser Probleme den Anstoß für ganz neue mathematische Disziplinen. Einige der Hilbertschen Probleme sind noch heute offen. Das zehnte Hilbertsche Problem betrifft die Frage nach der algorithmischen Lösbarkeit Diophantischer Gleichungen. Diophantische Gleichungen, benannt nach dem griechischen Mathematiker Diophantos (3. Jahrhundert), sind Polynomgleichungen in mehreren Variablen mit ganzzahligen Koeffizienten. Das zehnte Hilbertsche Problem fragt nun nach der Existenz eines Algorithmus, der entscheidet, ob eine vorgelegte Diophantische Gleichung ganzzahlige Lösungen besitzt oder nicht. In positiver Weise hätte diese Frage durch die Angabe eines Algorithmus beantwortet werden können. Um jedoch nachzuweisen, dass für dieses Problem kein Algorithmus existiert, muss zuvor natürlich


David Hilbert
(1862–1943)


Alonzo Church
(1903–1995)


Kurt Gödel
(1906–1978)


Alan M. Turing
(1912–1954)

geklärt werden, was ein Algorithmus im präzisen Sinne ist. In der Tat konnte schließlich 1970 Juri Matijasevič [66] unter Benutzung von Vorarbeiten von Julia Robinson, Martin Davis und Hilary Putnam [25] die algorithmische Unlösbarkeit des zehnten Hilbertschen Problems nachweisen.

Für die mathematische Formalisierung des Algorithmenbegriffs scheint in den 30er Jahren des 20. Jahrhunderts die Zeit reif gewesen zu sein: unabhängig und in zeitlich dichter Folge wurden mehrere, auf den ersten Blick völlig unterschiedliche Konzepte zur Berechenbarkeit vorgeschlagen. Stephen Cole Kleene stellte 1936 die allgemeinrekursiven Funktionen als Verallgemeinerung der von Kurt Gödel 1931 untersuchten primitiv rekursiven Funktionen vor. Zur selben Zeit entwarf Alonzo Church den so genannten λ-Kalkül. Die Ansätze von Kleene und Church beschreiben Operationen zur Bildung berechenbarer Funktionen aus einfachen Basisfunktionen. Maschinenorientierte Rechenmodelle lieferten 1936 unabhängig voneinander Alan Turing und Emil Post. Natürlich stellte man sich sofort die Frage, in welcher Beziehung diese verschiedenen Algorithmenmodelle zueinander stehen, und erstaunlicherweise erwiesen sich alle Ansätze als äquivalent, d.h. sie beschreiben dieselbe Klasse berechenbarer Funktionen. Daher formulierte Church 1936 die nach ihm benannte These, dass durch die genannten Präzisierungen des Algorithmenbegriffs genau die intuitive Vorstellung von Effektivität eingefangen wird, d.h. dass sich die Klasse aller durch Turingmaschinen berechenbaren Funktionen mit der Klasse aller im intuitiven Sinne berechenbaren Funktionen deckt. Natürlich lässt sich eine solche These nicht beweisen, da sich der intuitive Berechenbarkeitsbegriff durch seine Verschwommenheit einer mathematischen Analyse entzieht. Empirisch aber haben die seither verstrichenen 70 Jahre die Churchsche These bestätigt. Trotz des enormen Fortschritts auf dem Gebiet der Rechentechnik haben sich grundsätzlich keine Erweiterungen des Berechenbarkeitsbegriffs gezeigt, nicht einmal durch so neuartige Ansätze wie etwa die Verwendung von Quantenrechnern.

1.2 Turings Maschinenmodell

Da die Turingmaschine das zentrale Maschinenmodell der Komplexitätstheorie darstellt, wollen wir es genauer beschreiben. Wie bereits erwähnt wurde diese 1936 von Turing in der Arbeit „On computable numbers with an application to the Entscheidungsproblem“ [94] entworfen. Mit dem Entscheidungsproblem ist übrigens die von Alonzo Church ebenfalls 1936 gezeigte Unentscheidbarkeit der Prädikatenlogik erster Stufe gemeint [19].

Die Turingmaschine besteht aus einer Steuereinheit und einem Arbeitsband, auf welches die Steuereinheit mittels eines Schreib- und Lesekopfes zugreifen kann. Das Arbeitsband ist dabei in einzelne Felder unterteilt, die jeweils einen Buchstaben des Arbeitsalphabets enthalten, meist besteht dies einfach aus dem Binäralphabet $\{0, 1\}$, erweitert um ein spezielles Blanksymbol für leere Felder. Die Turingmaschine arbeitet taktweise. Die Arbeitsweise wird von der Steuereinheit bestimmt, die sich in verschiedenen, aber insgesamt nur endlich vielen Zuständen befinden kann. Pro Takt liest der Schreib- und Lesekopf ein Zeichen vom Band und ersetzt dieses in Abhängigkeit vom gelesenen Zeichen und dem Zustand der Steuereinheit. Sodann wird der Kopf um ein Feld nach links oder rechts bewegt und der Zustand der Steuereinheit neu bestimmt, wiederum geschieht dies abhängig vom gelesenen Zeichen und dem alten Zustand.

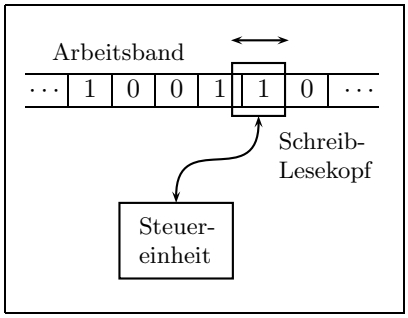


Abb. 1: Die Turingmaschine

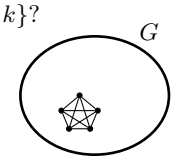
Wie können nun mit solchen Maschinen konkrete Probleme gelöst werden? In der Komplexitätstheorie betrachtet man meist Entscheidungsprobleme, bei denen die Eingabeinstanz durch eine ja/nein-Antwort entschieden wird. Alle positiven Instanzen eines Problems werden dann zu einer Sprache zusammengefasst. Einige typische Entscheidungsprobleme, auf die wir auch im folgenden oft zurückkommen werden, sind in Abb. 2 zusammengefasst. Um diese Probleme durch Turingmaschinen zu lösen, werden zunächst die Eingabeinstanzen, also natürliche Zahlen, aussagenlogische Formeln oder Graphen geeignet binär kodiert. Zu Beginn der Turingmaschinenrechnung steht die Eingabe kodiert auf dem Band. Sodann liest die Maschine die Eingabe, modifiziert diese eventuell oder macht auf dem Band Zwischenrechnungen und begibt sich zum Ende der Rechnung in einen Zustand, welcher signalisiert, ob die Eingabe akzeptiert oder verworfen wird.

Viele praktische Probleme treten auch als Optimierungsprobleme oder Berechnungsprobleme auf, bei denen eine Ausgabe abhängig von der Eingabe berechnet werden soll. In diesem Fall schreibt die Turingmaschine das Ergebnis am Ende der Rechnung in kodierter Form auf das Band. Oftmals lassen sich

Das Primzahlproblem PRIMES**:**
Gegeben: Eine natürliche Zahl n.
Gefragt: Ist n eine Primzahl?

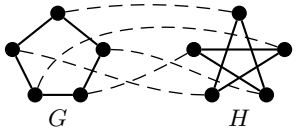
Das Faktorisierungsproblem FACTORIZE**:**
Gegeben: Zwei natürliche Zahlen n, k.
Gefragt: Besitzt n einen Faktor in der Menge $\{2, \ldots, k\}$?

Das Cliquenproblem CLIQUE**:**
Gegeben: Ein Graph G und eine natürliche Zahl k.
Gefragt: Existiert in G eine Clique der Größe k?



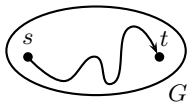
Das aussagenlogische Erfüllbarkeitsproblem SAT**:**
Gegeben: Eine aussagenlogische Formel F.
Gefragt: Ist F erfüllbar?

Das Graphenisomorphieproblem GI**:**
Gegeben: Zwei Graphen G und H.
Gefragt: Sind G und H isomorph?



Das Flussproblem MAXFLOW**:**
Gegeben: Ein Netzwerk N mit zwei Knoten s, t und eine Zahl k.
Gefragt: Existiert in N ein Fluss der Größe k von s nach t?

Das Erreichbarkeitsproblem GAP**:**
Gegeben: Ein gerichteter Graph G und zwei Knoten s, t.
Gefragt: Existiert in G ein gerichteter Pfad von s nach t?



Das Erreichbarkeitsproblem UGAP in ungerichteten Graphen ist analog zu GAP definiert.

Abb. 2: Algorithmische Problemstellungen

aber solche funktionalen Probleme in Entscheidungsprobleme vergleichbarer Komplexität übersetzen, so dass die Fokussierung auf letztere gerechtfertigt ist (wie etwa beim Faktorisierungsproblem FACTORIZE, siehe Abb. 2).

Berechtigterweise erhebt sich an dieser Stelle vielleicht die Frage, warum wir ein solch eingeschränktes und aus praktischer Sicht eher untaugliches Rechenmodell als Ausgangspunkt wählen. Turing formulierte seine Ideen zu einer Zeit, als die praktische Realisierung von Computern noch ausstand (übrigens widmete sich Turing dann in den 40er und 50er Jahren selbst intensiv dem Bau von Rechenanlagen). Später wurden auch, inspiriert von der Architektur dann schon verfügbarer Rechner, praxisnähere Modellierungen vorgeschlagen, so etwa 1963 die Registermaschine, auch Random Access Machine genannt, von John Shepherdson und Howard Sturgis [86]. Aber auch diese Modelle sind bezüglich ihrer Rechenkraft äquivalent zu Turings Ansatz – ein weiterer Beleg für die Gültigkeit der Churchschen These. Dass sich die Turingmaschine trotzdem durchsetzte, liegt vor allem an ihrer Einfachheit, die elegante Beweise erlaubt. Natürlich entwirft niemand Algorithmen durch Angabe von

Turingmaschinen bestehen nur aus endlich vielen Anweisungen und können daher binär kodiert werden. Bezeichne M_c die durch c kodierte Maschine. Das Halteproblem ist

$$H = \{c\#x \mid M_c(x)\downarrow\},$$

wobei „$M_c(x)\downarrow$“ bedeutet, dass M_c bei Eingabe x hält. Unter der Annahme, dass H entscheidbar ist, können wir eine Turingmaschine $\hat{M}$ konstruieren, die bei Eingabe c genau dann hält, wenn $c\#c \notin H$ ist (d.h. $\hat{M}$ verhält sich komplementär zur Diagonalen der Matrix A, deren Eintrag in Zeile c und Spalte x angibt, ob $M_c(x)$ hält oder nicht). Für die Kodierung $\hat{c}$ von $\hat{M}$ folgt dann

$$\hat{c}\#\hat{c} \in H \;\Leftrightarrow\; M_{\hat{c}}(\hat{c})\downarrow \Leftrightarrow\; \hat{M}(\hat{c})\downarrow \Leftrightarrow\; \hat{c}\#\hat{c} \notin H \;\lightning$$

A	x_1	x_2	x_3	x_4	$\cdots$
c_1	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\cdots$
c_2	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\cdots$
c_3	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\cdots$
c_4	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
$\hat{c}$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\cdots$

Abb. 3: Die Unentscheidbarkeit des Halteproblems

Programmcodes für Turingmaschinen. Will man aber nachweisen, dass für ein konkretes Problem keine Algorithmen gewisser Güte existieren, so ist es von Vorteil, diesen Unmöglichkeitsbeweis, der ja gegen alle möglichen Algorithmen argumentieren muss, anhand eines restriktiven Modells zu führen.

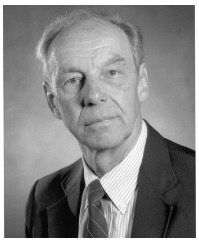
1.3 Unentscheidbare Probleme

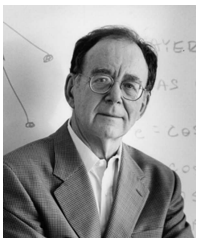
Probleme, die prinzipiell nicht algorithmisch lösbar sind, d.h. für die es nicht gelingt, ein Turingmaschinenprogramm zu entwerfen, heißen unentscheidbar. Bereits 1936 zeigte Turing die Unentscheidbarkeit des Halteproblems, welches zu einem gegebenen Turingmaschinenprogramm und einer zugehörigen Eingabe bestimmt, ob die Turingmaschine bei dieser Eingabe hält oder aber unendlich lange läuft. Der Beweis benutzt die auf Georg Cantor (1845–1918) zurückgehende Diagonalisierungstechnik (siehe Abb. 3). Dieses Resultat hat durchaus praktische Konsequenzen. In moderner Terminologie könnten wir es etwa so formulieren: es gibt keinen Algorithmus, der für ein gegebenes C-Programm entscheidet, ob das Programm bei einer bestimmten Eingabe nach endlicher Zeit terminiert oder in eine Endlosschleife gerät. Dem Fortschritt bei der automatischen Programmverifikation sind also Grenzen gesetzt.

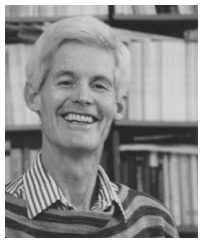
In gewisser Weise das allgemeinste Resultat dieser Art bewies 1953 Henry Rice [77]. Der Satz von Rice besagt, dass es unmöglich ist, anhand des Programmcodes interessante Eigenschaften der vom Programm akzeptierten Sprache zu entscheiden.

1.4 Effektivität versus Effizienz

In der Rekursionstheorie stehen die unentscheidbaren Sprachen im Mittelpunkt. Entscheidbare Sprachen gelten aus dieser Perspektive als trivial, da


Juris Hartmanis
(geb. 1928)


Richard Karp
(geb. 1935)


Stephen Cook
(geb. 1939)


Leonid Levin
(geb. 1948)

man sie ja algorithmisch lösen kann. Algorithmische Lösungen, bei denen der Aufwand nicht in Betracht gezogen wird, heißen effektiv.

Für die Praxis ist es aber natürlich wichtig zu wissen, wieviel Rechenressourcen zur Lösung dieser Probleme nötig sind. Ein Beispiel mag das verdeutlichen. Das Erfüllbarkeitsproblem SAT der Aussagenlogik besteht darin, zu einer aussagenlogischen Formel zu entscheiden, ob diese erfüllbar ist. Natürlich gibt es hierfür einen Algorithmus: man überprüft einfach den Wahrheitswert der Formel unter allen Belegungen. Für eine Formel mit n Variablen benötigt man so etwa 2^n Schritte. Hat die Formel also 100 Variablen, und solche Formeln treten häufig bei automatisch generierten Prozessen auf, so ist der Zeitbedarf etwa 2^{100} Schritte. Angenommen, ein Computer kann eine Milliarde solcher Operationen pro Sekunde ausführen, so ergibt sich für 2^{100} Operationen immer noch eine Rechenzeit von mehr als 10^{13} Jahren. Dieser Algorithmus für SAT ist also praktisch nicht durchführbar. Wesentlich bessere Algorithmen, d.h. mit subexponentieller Laufzeit, sind bislang nicht bekannt. Gerade für SAT-Solver gibt es einen aktiven Wettbewerb um die schnellsten Verfahren. Die beste Laufzeit für 3-SAT (mittels eines probabilistischen Algorithmus) liegt derzeit bei 1.32216^n [79]. Somit ist SAT mit heutigen Methoden zwar effektiv, aber nicht effizient lösbar.

2 Die 60er Jahre: Effizienz

Die Entscheidbarkeit eines Problems ist nicht genug, um es auch praktisch befriedigend lösen zu können, wie das Beispiel des letzten Abschnitts gezeigt hat. Welches aber ist die richtige Bedingung für Effizienz? Mit der Beantwortung dieser Frage beginnt die eigentliche Geschichte der Komplexitätstheorie.

Die wichtigsten Parameter sind der Bedarf an Rechenzeit und Speicherplatz, gemessen bei Turingmaschinen in der Anzahl der Schritte bzw. der Anzahl der insgesamt besuchten Bandfelder. Erste Überlegungen zu effizienten Algorithmen äußerte 1956 Kurt Gödel in einem Brief an Johann von Neumann [38]. Gödels Ideen, von denen erst in den 80er Jahren ein größerer Kreis Kenntnis erhielt, wurden jedoch zunächst nicht weiter verfolgt.

1964 zeigte Alan Cobham [20], dass viele wichtige Probleme in Polynomialzeit lösbar sind, d.h. die Laufzeit ist durch ein Polynom der Form $n^k + k$ in der Länge n der Eingabe beschränkt. Etwa zur gleichen Zeit schlug Jack Edmonds [28] Polynomialzeitberechnungen als Formalisierung von berechnungsmäßiger Effizienz vor. Die Komplexitätsklasse P enthält alle Entscheidungsprobleme, die mit Turingmaschinen in polynomieller Laufzeit lösbar sind. Ähnlich wie bei der Churchschen These zeigten Cobham [20] und Edmonds [28], dass diese Klasse nicht vom verwendeten Maschinenmodell abhängt.

Die systematische Untersuchung von Zeit- und Platzbeschränkungen für Turingmaschinen begann 1965 mit Juris Hartmanis und Richard Stearns [44]. In dieser sehr einflussreichen Arbeit formalisierten Hartmanis und Stearns Zeit- und Platzbedarf für Turingmaschinen und bewiesen für diese Komplexitätsmaße Hierarchiesätze, die besagen, dass bei wachsenden Zeit- bzw. Platzschranken die Anzahl der lösbaren Probleme echt zunimmt. Einfacher ist der Platzhierarchiesatz: sind $s_1(n)$ und $s_2(n)$ zwei hinreichend einfach zu berechnende Funktionen, für die der Quotient $s_1(n)/s_2(n)$ gegen 0 strebt, so existieren Probleme, die zwar mit Platzbedarf $s_2(n)$, aber nicht mit $s_1(n)$ entschieden werden können. Den bislang besten Zeithierarchiesatz bewiesen 1966 Hennie und Stearns [48]: falls $t_2(n)$ asymptotisch schneller als $t_1(n) \log t_1(n)$ wächst, garantiert dies die Existenz von Problemen, die in Zeit $t_2(n)$, aber nicht in $t_1(n)$ lösbar sind. Auch die Beweise der Hierarchiesätze benutzen das Diagonalisierungsverfahren von Cantor, derzeit eigentlich die einzige verfügbare Technik zur Trennung von Komplexitätsklassen.

3 Die 70er Jahre: das P/NP-Problem und die NP-Vollständigkeit

Anfang der 70er Jahre wurde man auf eine wachsende Menge praktisch sehr relevanter Probleme aufmerksam, für die trotz intensiver Bemühungen keine effizienten Algorithmen angegeben werden konnten. Vielfach waren das Optimierungsprobleme, wie etwa das Problem des Handlungsreisenden, oder deren Entscheidungsvarianten, zu denen auch die schon erwähnten Probleme Primes, Sat und Clique gehörten. Verschiedene Hypothesen bezüglich des Status dieser Probleme wurden geäußert: meinten manche, mit der Zeit werde man die algorithmische Behandlung dieser Probleme Schritt für Schritt verbessern können, so gab es auch weniger optimistische Stimmen, die deren Entscheidbarkeit in Polynomialzeit bezweifelten.

Um Hilberts Optimismus bezüglich einer positiven Antwort auf sein zehntes Problem zu widerlegen, musste zu Beginn des Jahrhunderts die Rekursionstheorie geschaffen werden. Welche theoretische Rechtfertigung gab es aber nun gegen die Existenz effizienter Algorithmen für Primes, Sat oder Clique? Und zum zweiten: sind alle diese Probleme gleich schwer und aus dem gleichen Grund? Die Antwort fällt differenziert aus: für das Primzahlproblem

Turing-Award
1976: M. Rabin, D. Scott
1982: S. Cook
1985: R. Karp
1993: J. Hartmanis, R. Stearns
1995: M. Blum
2000: A. Yao
2002: R. Rivest, A. Shamir, L. Adleman

Gödel-Preis	
1993: L. Babai, S. Goldwasser, S. Micali, S. Moran, C. Rackoff	1998: S. Toda
1994: J. Håstad	1999: P. Shor
1995: N. Immerman, R. Szelepcsényi	2001: S. Arora, U. Feige, S. Goldwasser, C. Lund, L. Lovász, R. Motwani, S. Safra, M. Sudan, M. Szegedy

Abb. 4: Turing-Award- und Gödel-Preisträger aus der Komplexitätstheorie. Der Turing-Award wird seit 1966 jährlich vergeben und gilt als wichtigster Preis für Informatiker, da es für die Informatik keinen Nobelpreis gibt. Für herausragende Arbeiten im Bereich der Theoretischen Informatik wird seit 1993 jährlich der Gödel-Preis verliehen.

Primes wurde schließlich im Jahr 2002 ein Polynomialzeitalgorithmus entworfen (siehe Abschnitt 6.1). Für die anderen Probleme ist dies bis heute nicht gelungen und wird auch allgemein für die Zukunft nicht erwartet. Die theoretische Rechtfertigung hierfür lieferte Stephen Cook zu Beginn der 70er Jahre mit der Theorie der NP-Vollständigkeit, mit der die Erfolgsgeschichte der Komplexitätstheorie beginnt.

3.1 Nichtdeterminismus

Den erwähnten Problemen Sat und Clique ist gemeinsam, dass ihrer Lösung exponentiell große Suchräume zugrunde liegen. Bei Sat sind dies die Menge aller Belegungen, unter denen eine erfüllende gesucht wird. Bei Clique suchen wir unter allen Knotenmengen der Größe k eine solche, bei der alle Knoten untereinander verbunden sind. Bei dem naiven Ansatz, der so genannten Brute-Force-Methode, wird dieser Suchraum systematisch durchsucht, dazu braucht man aber im allgemeinen exponentiell viel Zeit in der Länge der Eingabe. Andererseits sind die Lösungen, im Beispiel erfüllende Belegungen oder Cliquen, einfach zu verifizieren, wenn sie vorgelegt werden. Dies ist typisch für nichtdeterministische Berechnungen, ein von Michael Rabin und Dana Scott [73] 1959 in die Informatik eingeführtes Konzept. Rabin und Scott wurden hierfür 1976 mit dem Turing-Award ausgezeichnet (Abb. 4).

Eine nichtdeterministische Turingmaschine darf in jedem Schritt zwischen mehreren Alternativen beliebig wählen. Zu einer Eingabe gibt es damit nicht nur eine, sondern mehrere Turingmaschinenrechnungen, von denen durchaus manche akzeptierend und andere verwerfend sein können. Das Akzeptanzverhalten ist nun so definiert, dass es nur einer akzeptierenden Rechnung bedarf, damit die Eingabe zur von der Maschine akzeptierten Sprache gehört. Bei Eingaben außerhalb der Sprache müssen hingegen alle Rechnungen verwerfend

sein. Man kann sich das auch so vorstellen, dass die Turingmaschine zunächst nichtdeterministisch ein Element des zur Eingabe gehörenden Suchraums rät und dann deterministisch die Korrektheit dieses Zeugen überprüft. In der Komplexitätsklasse NP werden nun alle Sprachen zusammengefasst, die durch nichtdeterministische Turingmaschinen in Polynomialzeit akzeptiert werden. Der NP-Algorithmus für SAT etwa sieht so aus: bei Eingabe einer aussagenlogischen Formel wird zunächst eine Belegung geraten und dann überprüft, ob diese die Formel erfüllt. Dies ist in Polynomialzeit möglich.

3.2 NP-Vollständigkeit

1971 führte Stephen Cook den Begriff der NP-Vollständigkeit ein, der wie viele komplexitätstheoretische Definitionen aus der Rekursionstheorie in den Kontext effizienter Berechnungen übertragen wurde. Zugrunde gelegt wird hierbei ein Reduktionsbegriff, mittels dessen Sprachen bezüglich ihrer Schwierigkeit verglichen werden können. Informal ausgedrückt ist ein Problem A auf ein Problem B reduzierbar, falls A berechnungsmäßig einfacher ist als B. Cook benutzt den relativ starken Begriff der Turing-Reduktion zur Definition der NP-Vollständigkeit (siehe Abschnitt 3.5). Die NP-vollständigen Probleme bilden die schwersten Probleme innerhalb der Klasse NP, d.h. alle Probleme aus NP sind auf sie reduzierbar. In [21] zeigte Stephen Cook die NP-Vollständigkeit von SAT und einer eingeschränkten Variante 3-SAT.

Die Tragweite dieser Resultate wurde als erstes von Richard Karp verstanden, der 1972 in der sehr einflussreichen Arbeit [53] die NP-Vollständigkeit von 20 weiteren natürlichen Problemen nachwies, unter ihnen auch das Cliquenproblem CLIQUE. Hierbei benutzte Karp einen feineren Reduktionsbegriff, die many-one-Reduktionen für Polynomialzeitberechnungen, und führte die Klassenbezeichnungen P (**p**olynomial time) und NP (**n**ondeterministic **p**olynomial time) ein.

In der Folge wurde die NP-Vollständigkeit für eine große, bis heute wachsende Anzahl praktisch wichtiger Probleme nachgewiesen. Einen vorläufigen Höhepunkt dieser Entwicklung markierte 1979 das klassische Lehrbuch zur Komplexitätstheorie von Michael Garey und David Johnson [36], welches im Anhang eine Beschreibung von rund 300 NP-vollständigen Problemen enthält.

Unabhängig von Cook und Karp untersuchte Leonid Levin [63] 1973 „universelle Suchprobleme“, einen der NP-Vollständigkeit verwandten Begriff, und zeigte diese Eigenschaft für SAT sowie fünf weitere Probleme. Auch andere Ergebnisse zu Polynomialzeitberechnungen waren unabhängig in den 60er Jahren in der Sowjetunion, insbesondere von Boris Trachtenbrot, entwickelt worden, jedoch waren diese auf russisch publizierten Resultate in der westlichen Welt bis in die 70er Jahre hinein unbekannt.

3.3 Das P/NP-Problem

Mit dem Nachweis der NP-Vollständigkeit von SAT, CLIQUE und vielen weiteren Problemen verlagert sich die Frage nach der Existenz effizienter Algo-

rithmen für diese Probleme auf eine abstraktere Ebene: die der Trennung der Klassen P und NP. Gilt nämlich $\mathsf{P} \neq \mathsf{NP}$, so gibt es für kein NP-vollständiges Problem Algorithmen mit polynomieller Laufzeit. Gelingt es andererseits, für ein NP-vollständiges Problem einen effizienten Algorithmus zu finden, so ist $\mathsf{P} = \mathsf{NP}$, und somit kann man auch alle anderen NP-Probleme effizient lösen.

Trotz intensiver Bemühungen ist der Status des P/NP-Problems, d.h. die Frage, ob $\mathsf{P} \neq \mathsf{NP}$ gilt, weiterhin offen. In Anlehnung an die von Hilbert 1900 in Paris vorgestellten Probleme wurden zur Jahrtausendwende, wiederum in Paris, sieben „Millennium Prize Problems“ präsentiert, von denen für die Mathematik des 21. Jahrhunderts wichtige Impulse erwartet werden. Für die Lösung eines jeden dieser Probleme, und hierunter befindet sich auch das P/NP-Problem, hat das Clay Mathematics Institute ein Preisgeld von einer Million Dollar festgesetzt.

Wie nah wir derzeit der Lösung des P/NP-Problems sind ist unklar. In einer Diskussion über Perspektiven der Logik im 21. Jahrhundert benennt Samuel Buss den Zeitraum 2010 ± 10 Jahre zur Lösung des Problems [18]. Die meisten Forscher sind jedoch weitaus pessimistischer.

3.4 Komplemente von NP-Mengen

Das P/NP-Problem ist bei weitem nicht die einzige Ungewissheit bezüglich der Klasse NP. Eine Verschärfung des P/NP-Problems besteht in der Frage nach dem Komplementabschluss von NP. Alle Komplemente von NP-Mengen bilden die Klasse coNP. Falls NP nicht unter Komplementbildung abgeschlossen ist, d.h. $\mathsf{NP} \neq \mathsf{coNP}$, so folgt auch $\mathsf{P} \neq \mathsf{NP}$. Die umgekehrte Implikation konnte aber bislang nicht bewiesen werden. Aus Cooks NP-Vollständigkeitsresultat für SAT ergibt sich leicht die coNP-Vollständigkeit des aussagenlogischen Tautologienproblems TAUT. Die Frage nach der Existenz von NP-Algorithmen für TAUT führt in die aussagenlogische Beweiskomplexität, einem maßgeblich von Stephen Cook mitbegründeten Forschungsgebiet (vergl. Abschnitt 5.3).

3.5 Orakelberechnungen und die Polynomialzeithierarchie

Orakelberechnungen, ein ebenfalls aus der Rekursionstheorie entlehntes Konzept, erlauben einer Turingmaschine Zugriff auf eine Sprache möglicherweise hoher Komplexität, das so genannte Orakel. Die Maschine darf dabei während ihrer Rechnung beliebige Anfragen an die Orakelsprache stellen und erhält hierauf sofort die Antwort in einem speziellen Zustand. Der Maschine wird also gratis Information zur Verfügung gestellt, die sie selbst im allgemeinen nicht berechnen könnte. Hierdurch ergeben sich die von Cook für seine Vollständigkeitsresultate benutzten

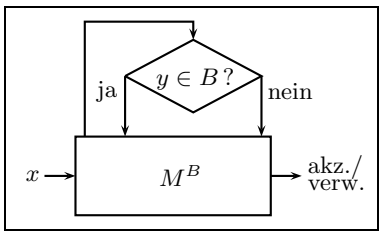
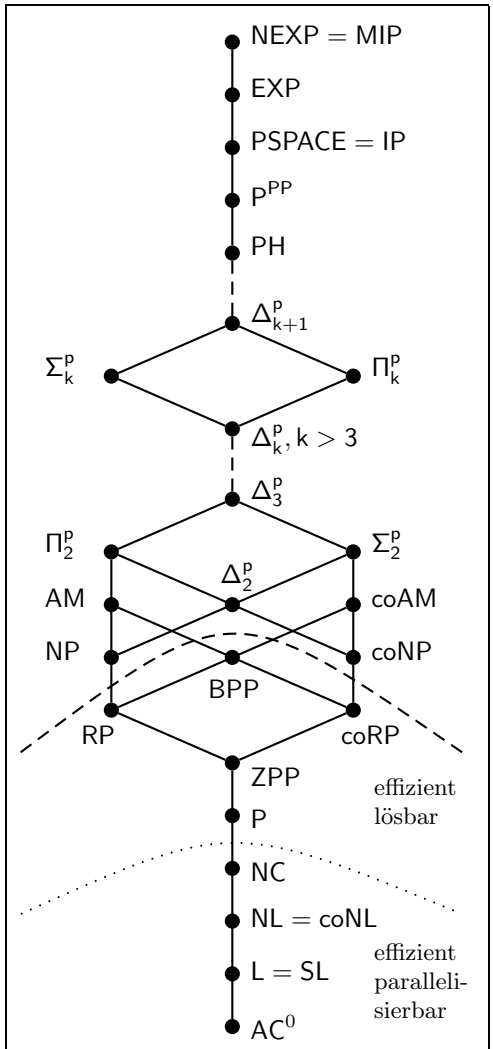


Abb. 5: Die Orakelmaschine

Turing-Reduktionen: wird A von einer Orakelturingmaschine in Polynomialzeit mit Hilfe des Orakels B entschieden, so heißt A Turing-reduzierbar auf B. Auch Komplexitätsklassen können relativ zu einem Orakel definiert werden. So wird zum Beispiel die Klasse aller in Polynomialzeit unter Zugriff auf ein Orakel B entscheidbaren Probleme mit P^B bezeichnet.

Eines der interessantesten und historisch auch das erste Orakelresultat wurde 1975 von Theodore Baker, John Gill und Robert Solovay [11] betreffs des P/NP-Problems nachgewiesen. Die Autoren konstruieren zwei Orakel A und B, bezüglich derer das P/NP-Problem verschiedene Antworten erfährt. Unter dem ersten Orakel gilt $\mathsf{P}^A = \mathsf{NP}^A$, relativ zum zweiten jedoch $\mathsf{P}^B \neq \mathsf{NP}^B$. Im Jahr 1981 zeigten Charles Bennet und John Gill [14] sogar, dass man bei zufälliger Wahl des Orakels A mit Wahrscheinlichkeit 1 $\mathsf{P}^A \neq \mathsf{NP}^A \neq \mathsf{coNP}^A$ erhält. Unter fast allen Orakeln sind also die Klassen P, NP und coNP verschieden. Über den eigentlichen Status des P/NP-Problems sagt dies wenig, mehr allerdings über die gegenwärtigen Schwierigkeiten bei der Lösung des Problems. Die meisten bisher bekannten Beweistechniken, insbesondere das Diagonalisierungsverfahren, sind nämlich relativierbar, d.h. die erzielten Resultate gelten bezüglich beliebiger Orakel. Nach den Ergebnissen von Baker, Gill und Solovay lässt sich aber das P/NP-Problem nicht mit relativierbaren Methoden bewältigen.

Eine wichtige Verallgemeinerung der Klassen P, NP und coNP bildet die Polynomialzeithierarchie PH, eingeführt 1976 von Albert Meyer und Larry Stockmeyer [67, 90]. Ähnlich wie bei der in



Probleme unterhalb der gestrichelten Linie haben polynomielle Schaltkreiskomplexität.

Abb. 6: Die wichtigsten Zeit- und Platzkomplexitätsklassen

der Rekursionstheorie 1943 von Kleene definierten arithmetischen Hierarchie wird hier eine unendliche Hierarchie von Komplexitätsklassen gebildet, wobei jede Stufe auf die darunter liegende als Orakel zugreift. Beginnend mit $\Sigma_0^P = P$ und $\Sigma_1^P = NP$ werden die Stufen von PH mit $\Delta_k^P = P^{\Sigma_{k-1}^P}$, $\Sigma_k^P = NP^{\Sigma_{k-1}^P}$ und $\Pi_k^P = co\Sigma_k^P$ bezeichnet. Eine starke, jedoch heute generell akzeptierte Verallgemeinerung von $P \neq NP$ ist die Hypothese, dass alle Stufen von PH verschieden sind.

3.6 Weitere klassische Komplexitätsklassen

Nicht alle interessanten Problemstellungen liegen in NP oder selbst PH. Manche benötigen zu ihrer Lösung weitaus größere Ressourcen. Diejenigen Probleme, die mit polynomiellem Platzaufwand entscheidbar sind, bilden die Klasse PSPACE. Auch für PSPACE wurden Vollständigkeitsresultate erzielt, etwa für die Erfüllbarkeit von quantifizierten aussagenlogischen Formeln 1973 von Stockmeyer und Meyer [91].

Weitere typische PSPACE-vollständige Probleme stammen von Erweiterungen klassischer Brettspiele wie Dame oder Go für beliebig große Bretter (Fraenkel et al. [33] 1978, Lichtenstein und Sipser [64] 1980). Schachspielen ist sogar noch komplizierter und erfordert Exponentialzeit (Fraenkel und Lichtenstein [34] 1981). Diese bereits sehr mächtige Problemklasse wird mit EXP bezeichnet. Die spieltheoretischen Charakterisierungen von PSPACE sind kein Zufall. PSPACE kann nämlich als Klasse aller Gewinnstrategien für 2-Personenspiele mit effizient auswertbaren Spielregeln und polynomieller Rundenzahl aufgefasst werden. Beschränkt man sich dagegen auf konstant viele Runden, so gelangt man zur Polynomialzeithierarchie PH, genauer gesagt ergibt die Frage, ob der erste Spieler eine Gewinnstrategie mit k Zügen besitzt, ein Σ_k^P-vollständiges Problem.

Einen Überblick über die Lagebeziehungen zwischen den wichtigsten Komplexitätsklassen vermittelt Abb. 6. Interessant ist, dass aus dem Zeithierarchiesatz $P \neq EXP$ folgt. Mithin muss wenigstens eine der Beziehungen $P \neq NP$ oder $NP \neq EXP$ gelten, jedoch wissen wir gegenwärtig nicht, welche. Es wird aber angenommen, dass alle drei Klassen verschieden sind. Die einzigen weiteren bekannten Separierungen zwischen den in Abb. 6 angegebenen Klassen sind $NP \neq NEXP$, $NL \neq PSPACE$ und $AC^0 \neq L$ (vergl. Abschnitt 4.2).

3.7 Determinismus und Nichtdeterminismus für Platzklassen

Die Frage, ob Nichtdeterminismus als Berechnungsmodell stärker ist als Determinismus, ist nicht nur in Form des P/NP-Problems für Polynomialzeitberechnungen interessant, sondern zieht sich als roter Faden durch die gesamte Komplexitätstheorie. Für Platzklassen konnte Walter Savitch [81] bereits 1970 zeigen, dass Nichtdeterminismus höchstens einen geringen Vorteil bedeutet:

jedes nichtdeterministisch in Platz $s(n)$ berechenbare Problem kann deterministisch mit Platz $s(n)^2$ gelöst werden. Damit fällt zum Beispiel die Klasse PSPACE mit ihrem nichtdetermistischen Analogon zusammen.

Ein weiteres, aufsehenerregendes Resultat, mit dem wir jetzt bereits die 80er Jahre betreten, erzielten 1987 unabhängig voneinander Neil Immerman [49] und Róbert Szelepcsényi [92]. Mittels einer neuen Technik, dem „inductive counting“, wiesen sie nach, dass nichtdeterministische Platzklassen unter Komplementbildung abgeschlossen sind. Für nichtdeterministische Zeitklassen ist dies offen, wird aber im allgemeinen nicht erwartet.

Der Satz von Immerman und Szelepcsényi ist auch ein gutes Beispiel dafür, dass die allgemeine Intuition bezüglich berühmter ungelöster Probleme durchaus irren kann. Die Frage, ob nichtdeterministische Platzklassen unter Komplement abgeschlossen sind, wurde nämlich bereits seit den 60er Jahren als zweites LBA-Problem diskutiert, und gemeinhin wurde eine negative Antwort erwartet. Um so überraschender war fast 30 Jahre später die positive Lösung, und dazu noch zeitgleich in zwei Beiträgen, die in der Beweisführung zwar trickreich, aber dennoch elementar waren.

Das zweite LBA-Problem ist somit geklärt. Das erste hingegen, bestehend in der Frage, ob Nichtdeterminismus für linear beschränkte Automaten (LBA) mächtiger als Determinismus ist, wartet noch immer auf seine Lösung.

4 Die 80er Jahre: Randomisierung, Nichtuniformität und interaktive Beweise

Neben dem Ausbau der strukturellen Komplexitätstheorie, die sich vornehmlich den im letzten Kapitel besprochenen Komplexitätsklassen widmet, rückten in den 80er Jahren zunehmend weitere Berechnungsparadigmen in den Blickpunkt komplexitätstheoretischer Forschung. Als wichtigstes hiervon kann Randomisierung gelten, die sich des Zufalls als Berechnungsressource bedient.

4.1 Probabilistische Algorithmen

1977 entwarfen Robert Solovay und Volker Strassen [89] einen neuen, zufallsbasierten Algorithmus zum Primzahltest. Der Algorithmus läuft in polynomieller Zeit, benutzt aber Münzwürfe bei der Berechnung. Diese Münzwürfe machen natürlich auch das Ergebnis zu einer Zufallsvariablen, d.h. manchmal gibt der Algorithmus die falsche Antwort. Im Fall des Solovay-Strassen-Algorithmus liegt ein einseitiger Fehler vor: Primzahlen werden immer als solche erkannt, zusammengesetzte Zahlen dagegen nur mit Wahrscheinlichkeit $1 - \varepsilon$. Anfangs wurden solche „unzuverlässigen“ Algorithmen teilweise recht kontrovers diskutiert. Man kann jedoch durch eine etwas höhere, aber immer noch polynomielle Laufzeit den Fehler ε vernachlässigbar klein machen, sogar so klein, dass zum Beispiel die Ausfallwahrscheinlichkeit des Computers während der Rechnung über der Fehlerschranke des Algorithmus liegt.

Entscheidungsprobleme, für die randomisierte Polynomialzeitalgorithmen mit einseitigem Fehler existieren (genauer: die ihre Eingabe nie fälschlicherweise akzeptieren), werden zur Klasse RP (**r**andomized **p**olynomial time), eingeführt 1977 von Leonard Adleman und Kenneth Manders [2], zusammengefasst. Erlaubt man bei polynomieller Laufzeit einen beidseitigen Fehler, gelangt man zur Klasse BPP von John Gill [37], der bereits 1972 in seiner Dissertation die Grundlagen für randomisierte Turingmaschinen und randomisierte Komplexitätsklassen legte. 1983 zeigte Michael Sipser [88], dass BPP in PH enthalten ist. Kurz darauf lokalisierten Peter Gács (ebenfalls in [88]) und Clemens Lautemann [61] BPP genauer in der zweiten Stufe Σ_2^{P} der Polynomialzeithierarchie.

Da Probleme in BPP immer noch eine praktisch befriedigende Lösung erfahren, hat man die These „Effizienz=Polynomialzeit“ aus den 60er Jahren zu „Effizienz=randomisierte Polynomialzeit“ erweitert. Ob allerdings in BPP wirklich mehr Probleme liegen als in P ist unklar. Derandomisierungsresultate der 90er Jahre deuten eher auf $\mathsf{P} = \mathsf{BPP}$ hin (vergl. Abschnitt 5.2).

Es gibt aber auch randomisierte Polynomialzeitalgorithmen, die nicht mehr als effizient gelten können: Algorithmen, bei denen die Wahrscheinlichkeit, die richtige Antwort zu erhalten, nur minimal größer ist als $\frac{1}{2}$, führen zur Klasse PP (**p**robabilistic **p**olynomial time) von Gill. Im Gegensatz zu BPP-Algorithmen, bei denen die Antwort mit hoher Wahrscheinlichkeit richtig ist, lassen sich hier falsche von richtigen Antworten kaum noch unterscheiden. Ein Hinweis dafür, dass die Klasse PP vermutlich wesentlich stärker ist als BPP, liefert bereits die Inklusion $\mathsf{NP} \subseteq \mathsf{PP}$. Die Beziehung zwischen NP und BPP ist hingegen ungeklärt. Für viele überraschend gelang Seinosuke Toda [93] 1991 der Nachweis, dass sogar die gesamte Polynomialzeithierarchie fast in PP (genauer: in ihrem Turing-Abschluss P^{PP}) enthalten ist.

Im Gegensatz zu den randomisierten Algorithmen mit ein- oder beidseitigem Fehler, auch Monte-Carlo- bzw. Atlantic-City-Algorithmen genannt, geben Las-Vegas-Algorithmen immer die richtige Antwort. Dafür muss man aber in Kauf nehmen, dass nun die Laufzeit vom Zufall abhängt, im Extremfall muss man also sehr lange auf die Antwort warten. Die ebenfalls von Gill eingeführte Klasse ZPP (**z**ero error **p**robabilistic **p**olynomial time) vereinigt alle Probleme, die Las-Vegas-Algorithmen mit erwarteter polynomieller Laufzeit besitzen. Es ist leicht zu sehen, dass $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$ ist.

1987 gelang Leonard Adleman und Ming-Deh Huang [1] der Nachweis, dass der Primzahltest mit solchen Las-Vegas-Algorithmen ausführbar ist, d.h. PRIMES $\in$ ZPP. Der in der Praxis am häufigsten verwendete Algorithmus für PRIMES ist jedoch ein Monte-Carlo-Algorithmus, entworfen 1980 von Michael Rabin [72] unter Verwendung von Vorarbeiten von Gary Miller [68].

4.2 Boolesche Schaltkreise

Ein anderes interessantes Berechnungsmodell bilden Boolesche Schaltkreise. Turingmaschinen liefern Algorithmen, die alle Instanzen eines gegebenen Pro-

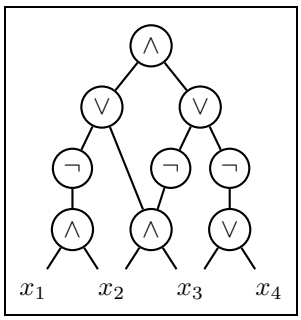


Abb. 7: Ein Schaltkreis

blems entscheiden. Dieses Modell wird deshalb auch als uniform bezeichnet. Wenn wir für jede Eingabelänge einen anderen Algorithmus erlauben, gelangen wir zu einem weitaus mächtigeren, nichtuniformen Modell. Hier werden alle Eingaben derselben Länge durch einen Schaltkreis mit UND-, ODER- sowie Negationsgattern entschieden, für verschiedene Eingabenlängen können jedoch völlig andere Schaltkreise benutzt werden. Die Ausgabe liefern die Schaltkreise binär in einem speziellen Ausgabegatter. Als Rechenmodell mögen nichtuniforme Schaltkreisfamilien unrealistisch erscheinen, denn wie sollte man mit einer unendlichen Anzahl verschiedener Algorithmen zurechtkommen? Gerade aber wenn es wie in der Kryptografie auf die Sicherheit von Verfahren ankommt, sollte man auch gegen möglicherweise sehr starke Gegner gewappnet sein. Bei der Sicherheitsanalyse kryptografischer Algorithmen werden daher meist nichtuniforme Gegner betrachtet.

Ein wichtiges Komplexitätsmaß bei Schaltkreisen ist deren Größe, gemessen in der Anzahl der Gatter. Die Mächtigkeit des Modells zeigt sich bereits darin, dass schon mit Schaltkreisfamilien konstanter Größe unentscheidbare Probleme berechnet werden können. Andererseits gibt es längst nicht für alle Probleme kleine Schaltkreise: mit einem relativ einfachen Abzählargument wies bereits Mitte der 40er Jahre Claude Shannon [85] nach, dass fast alle Booleschen Funktionen Schaltkreise exponentieller Größe benötigen.

Probleme, für die Schaltkreisfamilien polynomieller Größe existieren, werden zur Klasse $\mathsf{P/poly}$ zusammengefasst. 1972 bewies John Savage [80], dass P in $\mathsf{P/poly}$ enthalten ist (die Notation $\mathsf{P/poly}$ wurde allerdings erst 1980 von Karp und Lipton [54] eingeführt). 1981 bemerkten Charles Bennett und John Gill [14], dass sich dieses Resultat ebenfalls auf BPP überträgt (siehe auch [82]). Ob es jedoch auch für die Klasse NP gilt, ist ein berühmtes offenes Problem. Gelingt es nämlich, für ein NP-Problem superpolynomielle untere Schranken für die Schaltkreisgröße zu zeigen, so folgt mit dem Resultat von Savage $\mathsf{P} \neq \mathsf{NP}$. Einen Hinweis darauf, dass zumindest NP-vollständige Probleme keine Schaltkreise polynomieller Größe haben, liefert der 1980 von Richard Karp und Richard Lipton [54] bewiesene und später vielfach verbesserte Satz: NP ist nicht in $\mathsf{P/poly}$ enthalten, außer wenn die Polynomialzeithierarchie auf ihre zweite Stufe kollabiert. Dies jedoch gilt als unwahrscheinlich.

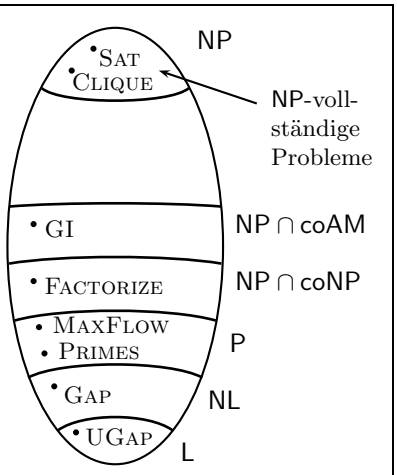
Die Klasse $\mathsf{ZPP}^{\mathsf{NP}}$ enthält zwar für jedes Polynom p Probleme, die keine Schaltkreise der Größe p haben [56]. Für konkrete Probleme gestaltet sich der Nachweis unterer Schranken für die Schaltkreiskomplexität aber äußerst schwierig. Derzeit sind lediglich lineare untere Schranken bekannt.

Bedeutende Resultate wurden jedoch für eingeschränkte Schaltkreisklassen erzielt. So zeigten zu Beginn der 80er Jahre unabhängig voneinander Merrick Furst, James Saxe und Michael Sipser [35] sowie Miklós Ajtai [5], dass die

Paritätsfunktion nicht mit Schaltkreisen konstanter Tiefe, so genannten AC^0-Schaltkreisen, berechnet werden kann. Johan Håstad [45] konnte dieses Ergebnis 1989 noch einmal wesentlich verbessern und die verwendete Technik, das so genannte „switching lemma“, auch für viele weitere Anwendungen nutzbar machen. Für monotone Schaltkreise, die auf Negationsgatter verzichten, zeigte Alexander Razborov [74] 1985, dass für das Cliquenproblem exponentiell große monotone Schaltkreise notwendig sind. Diese Resultate auf uneingeschränkte Schaltkreise zu übertragen ist jedoch bislang nicht gelungen. Vielmehr kam der in den 80er Jahren enthusiastisch begrüßte Fortschritt in der Schaltkreistheorie in den 90er Jahren nachhaltig ins Stocken: Alexander Razborov und Steven Rudich [75] zeigten 1994 unter kryptografischen Annahmen, dass eine große Klasse kombinatorischer Beweisverfahren, so genannte „natural proofs“, prinzipiell nicht zum Nachweis superpolynomieller unterer Schranken für die Schaltkreisgröße von NP-Problemen geeignet ist.

4.3 Parallele Algorithmen

Ein wenig paradox erscheint es vielleicht: obwohl die Rechentechnik in Riesenschritten voranschreitet, interessieren sich die theoretischen Informatiker für immer eingeschränktere Rechenmodelle. Befasste man sich in den 30er Jahren unbekümmert mit Turingmaschinen in ihrer vollen Mächtigkeit und schränkte sich dann in den 60er Jahren auf Polynomialzeitberechnungen ein, so rückten später verstärkt noch restriktivere Rechenmodelle, die Komplexitätsklassen innerhalb von P definieren, in den Blickpunkt. Gerade aber durch wachsende technische Möglichkeiten gewinnt die Feinstruktur der effizienten Welt zunehmend an Bedeutung.



Da SAT und CLIQUE NP-vollständig sind, sind sie nicht in einer kleineren Komplexitätsklasse C enthalten (außer wenn NP = C ist). Entsprechendes gilt für die in ihrer jeweiligen Klasse vollständigen Probleme MAXFLOW, GAP und UGAP.

Abb. 8: Komplexität von NP-Problemen

Eine solche Klasse innerhalb von P, eigentlich sogar eine ganze Hierarchie von Komplexitätsklassen, bildet die von Nick Pippenger [71] 1979 eingeführte Schaltkreisklasse NC. Besondere Bedeutung kommt NC deshalb zu, weil sie genau diejenigen Probleme enthält, die sich effizient parallelisieren lassen, wie Stephen Cook Anfang der 80er argumentierte [22]. Dies bedeutet, dass sich die Rechenzeit durch den Einsatz einer polynomiellen Anzahl von Prozessoren von polynomiell auf polylogarithmisch (d.h. $(\log n)^k + k$) beschleunigen lässt.

Viele Probleme, insbesondere viele Spezialfälle schwieriger Probleme, befinden sich in NC. So wiesen Richard Karp und Avi Wigderson [52] 1985 nach, dass sich maximale, d.h. nicht erweiterbare Cliquen effizient parallel finden lassen. Hingegen ist das allgemeine Cliquenproblem, welches nach der größten Clique eines Graphen fragt, wie schon bemerkt, NP-vollständig und damit wahrscheinlich wesentlich schwieriger (siehe Abb. 8).

Eine weitere wichtige Teilklasse von P ist L, die alle in logarithmischem Platz entscheidbaren Probleme umfasst. Zunächst sieht diese Definition wahrscheinlich etwas problematisch aus, da man mit weniger Platz, als die Eingabe ohnehin schon verbraucht, sicherlich nicht zurechtkommt. Um aber auch solche Algorithmen, die im Platzverbrauch extrem sparsam sind, besser analysieren zu können, müssen wir das Maschinenmodell etwas modifizieren. Bei der Offline-Turingmaschine steht die Eingabe auf einem speziellen Eingabeband, auf das nur lesend, nicht aber schreibend, zugegriffen werden darf. Zusätzlich steht für Zwischenrechnungen ein Arbeitsband zur Verfügung, und nur der Platzverbrauch auf diesem Band, auf welches natürlich auch geschrieben werden darf, wird gewertet. Diese Definition befindet sich im Einklang mit der Architektur tatsächlicher Computer: auch hier ist es ja häufig so, dass sich die Eingabe, etwa eine große Datenbank, auf einem externen Speichermedium befindet, während die eigentliche Rechnung im viel kleineren Arbeitsspeicher abläuft. Eine Reihe interessanter Probleme können mit logarithmischem Platz gelöst werden (siehe z.B. [23]).

Das nichtdeterministische Gegenstück zu L ist die Klasse NL aller in logarithmischem Platz durch nichtdeterministische Offline-Maschinen entscheidbaren Probleme. Analog zum P/NP-Problem ist die Trennung von L und NL eine große, bislang nicht bewältigte Herausforderung. Im Gegensatz zur offenen Frage nach der Gültigkeit von $\mathsf{NP} \neq \mathsf{coNP}$ impliziert das bereits erwähnte Resultat von Immerman und Szelepcsényi aus dem Jahr 1988 jedoch $\mathsf{NL} = \mathsf{coNL}$. Wie für NP wurden auch für NL eine Reihe von Vollständigkeitsresultaten erzielt. Das bekannteste NL-vollständige Problem ist das Erreichbarkeitsproblem GAP in gerichteten Graphen, dessen Vollständigkeit bereits 1975 von Neil Jones [51] nachgewiesen wurde.

Ein für die Praxis sehr wichtiges Problem besteht in der Berechnung maximaler Flüsse in Netzwerken. Thomas Lengauer und Klaus Wagner [62] zeigten im Jahr 1990, dass dieses Problem MAXFLOW vollständig für die Klasse P ist. MAXFLOW ist also vermutlich nicht effizient parallelisierbar.

4.4 Interaktive Beweissysteme

Das Grundszenario interaktiver Beweise wurde 1985 unabhängig von László Babai [8] und Shafi Goldwasser, Silvio Micali und Charles Rackoff [40] entworfen: ein Beweiser (prover) und ein Verifizierer (verifier) führen gemeinsam ein in Runden ablaufendes Protokoll aus, um eine vorgegebene Behauptung zu überprüfen. Der berechnungsmäßig mächtige Beweiser hat dabei die Aufgabe, den berechnungsmäßig eingeschränkten Verifizierer durch die Beantwortung

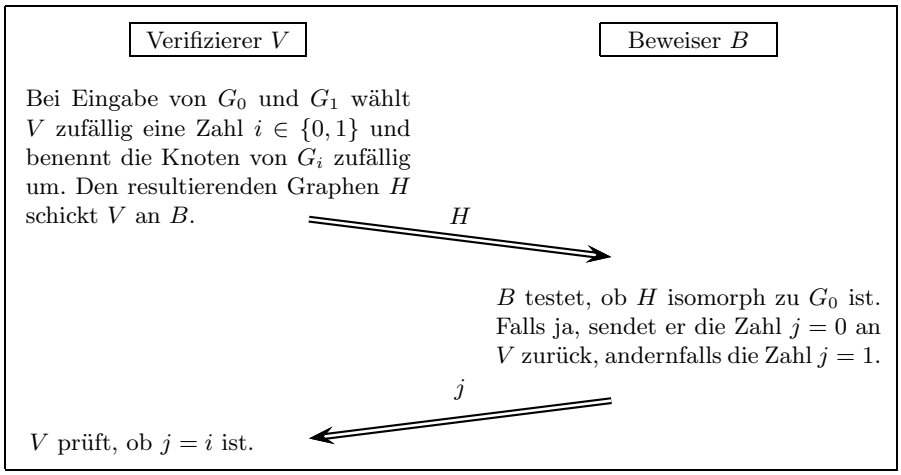


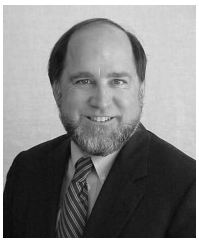
Abb. 9: Ein $\mathsf{IP}[2]$-Protokoll für $\overline{\mathrm{GI}}$ mit Zero-Knowledge-Eigenschaft

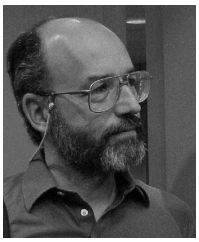
von Fragen von der Gültigkeit der Behauptung zu überzeugen. Verifizierer und Beweiser arbeiten dabei randomisiert, und auch der Verifizierer muss nur mit hoher Wahrscheinlichkeit richtige von falschen Beweisen unterscheiden können.

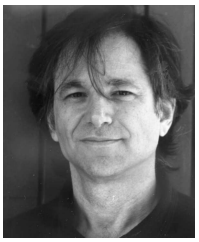
Die von Babai sowie von Goldwasser, Micali und Rackoff untersuchten interaktiven Beweissysteme unterscheiden sich darin, ob der Beweiser und der Verifizierer die benutzten Zufallsbits gegenseitig einsehen dürfen. Bei dem Modell von Goldwasser, Micali und Rackhoff, bezeichnet mit $\mathsf{IP}[k]$ (**i**nteractive **p**roofs, k Runden), bleiben die Zufallsbits geheim. Dagegen werden bei Babais Klasse $\mathsf{AM}[k]$, benannt nach dem englischen König Artus, der als Verifizierer die Beweise des Zauberers Merlin überprüfen muss, die Zufallsbits offengelegt.

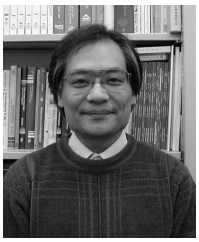
Babai [8] zeigte bereits 1985, dass bei seiner Version interaktiver Beweise eine beliebige konstante Rundenzahl durch ein Beweissystem mit nur zwei Runden ersetzt werden kann. Damit folgt $\mathsf{AM}[k] = \mathsf{AM}[2]$ für konstantes $k \geq 2$. Bei einem $\mathsf{AM}[2]$-Protokoll (auch kurz AM-Protokoll genannt) schickt also Artus in der ersten Runde eine Anfrage an Merlin, Merlin antwortet, und zum Schluss ist es an Artus, Merlins Beweis zu akzeptieren oder zu verwerfen. Oded Goldreich und Michael Sipser [39] konnten 1989 nachweisen, dass es überraschenderweise egal ist, ob die Zufallsbits geheim gehalten werden: es gilt $\mathsf{IP}[k] \subseteq \mathsf{AM}[k+2]$ und somit $\mathsf{IP}[k] = \mathsf{AM}$ für konstantes $k \geq 2$.

Ein interessantes Problem in der Klasse AM ist das Komplement $\overline{\mathrm{GI}}$ des Graphenisomorphieproblems, bestehend in der Frage nach der Nichtisomorphie zweier gegebener Graphen. Das $\mathsf{IP}[2]$-Protokoll für $\overline{\mathrm{GI}}$ von Goldreich, Micali und Wigderson [41] ist in Abb. 9 dargestellt. Hieraus folgt auch, dass GI vermutlich nicht NP-vollständig ist. Ravi Boppana, Johan Håstad und Stathis Zachos [17] haben nämlich 1987 gezeigt, dass die Klasse AM keine Komplemente NP-vollständiger Probleme enthält, es sei denn, die Polyno-


Ronald L. Rivest
(geb. 1947)


Adi Shamir
(geb. 1952)


Leonard Adleman
(geb. 1945)


Seinosuke Toda
(geb. 1959)

mialzeithierarchie kollabiert auf ihre zweite Stufe (siehe auch [83]). Dies wird jedoch nicht angenommen. Da andererseits trotz intensiver Suche bislang kein effizienter Algorithmus für GI bekannt ist, gilt GI als Kandidat für ein NP-Problem, welches weder in P liegt, noch NP-vollständig ist. Zwar hatte schon Richard Ladner [60] 1975 gezeigt, dass es solche Probleme geben muss (falls $\mathsf{P} \neq \mathsf{NP}$ ist), konkrete Kandidaten hierfür sind jedoch nur wenige bekannt.

Eine vor allem für kryptografische Anwendungen wichtige Spielart interaktiver Beweise bilden Zero-Knowledge-Protokolle, ebenfalls eingeführt 1989 von Goldwasser, Micali und Rackoff [40]. Solche interaktiven Beweise, aus denen der Verifizierer nichts außer der Gültigkeit des Beweises lernt, besitzt neben $\overline{\mathsf{GI}}$ auch GI, wie Goldreich, Micali und Wigderson [41] 1991 nachwiesen. Obiges Protokoll für $\overline{\mathsf{GI}}$ hat zwar die Zero-Knowledge-Eigenschaft, wenn der Verifizierer das Protokoll befolgt, nicht jedoch, wenn er hiervon abweicht.

Wie bereits erwähnt ändert sich die Ausdrucksstärke interaktiver Beweise nicht, falls wir statt zwei eine beliebige andere konstante Rundenzahl zulassen. Darf der Verifizierer hingegen polynomiell viele Runden mit dem Beweiser kommunizieren, bevor er seine Entscheidung trifft, kommen wir zur Klasse $\mathsf{IP} = \mathsf{IP}[\mathsf{poly}]$. Hierdurch erhalten wir eine sehr mächtige Komplexitätsklasse, es gilt nämlich $\mathsf{IP} = \mathsf{PSPACE}$. Dieses bedeutsame Resultat wurde 1989 in recht kurzer Zeit über verschiedene Zwischenergebnisse (u.a. [65]) von einem via E-Mail kommunizierenden Forscherkreis gewissermaßen interaktiv bewiesen (siehe [9]). Den letzten Baustein erbrachte 1990 Adi Shamir [84]. Die Charakterisierung $\mathsf{IP} = \mathsf{PSPACE}$ verdient auch deswegen Interesse, weil sie eines der wenigen komplexitätstheoretischen Resultate darstellt, die nicht relativieren, d.h. sich nicht auf beliebige Orakel übertragen [32].

Eine interessante Erweiterung des Modells ergibt sich, wenn der Verifizierer statt mit einem mit mehreren unabhängigen Beweisern kommunizieren darf. Dieses Szenario wurde erstmals 1988 von Ben-Or, Goldwasser, Kilian und Wigderson [12] betrachtet. Dadurch, dass der Verifizierer die verschiedenen Beweiser gewissermaßen gegeneinander ausspielen kann, ergibt sich eine wesentlich größere Ausdrucksstärke: 1990 bewiesen Babai, Fortnow und Lund [10], dass die Multi-Prover-Klasse MIP nichtdeterministischer Exponentialzeit NEXP entspricht. Dieses Resultat wurde vor allem als Zwischenschritt

zum PCP-Theorem wichtig, womit wir allerdings ein neues Jahrzehnt betreten.

5 Die 90er Jahre: algebraische, logische und physikalische Paradigmen

Die in den 80er Jahren begonnene Entwicklung, die Komplexitätstheorie mit immer weiteren mathematischen Teilgebieten und Anwendungsfeldern zu vernetzen, setzte sich in den 90er Jahren in verstärktem Maße fort. Die vier Teilgebiete, die wir für dieses Jahrzehnt ausgewählt haben, reichen zwar auch in frühere Jahre zurück, sind aber typisch für den gegenwärtigen Trend, algebraische Ansätze, Methoden der Logik und ganz neue physikalische Paradigmen fruchtbar im Zusammenspiel mit bewährten kombinatorischen Techniken einzusetzen.

5.1 Das PCP-Theorem

Die Geschichte des PCP-Theorems, das zu Beginn der 90er Jahre für viel Aufsehen sorgte, knüpft an die im letzten Abschnitt besprochenen interaktiven Beweissysteme an. Auch die Klasse NP kann als ein Beweissystem aufgefasst werden: der mächtige Beweiser erstellt einen Beweis, den der Verifizierer in Polynomialzeit überprüft. Ist es dem Verifizierer aber auch möglich, den Beweis zu überprüfen, ohne ihn vollständig gelesen zu haben? Auf den ersten Blick scheint dies intuitiv schwierig, andererseits hat aber sicher auch schon mancher Literaturkritiker ein Buch ohne vollständige Lektüre rezensiert.

Im Jahr 1992 bewiesen Arora, Lund, Motwani, Sudan und Szegedy [7] das erstaunliche Resultat, dass es für jede Sprache aus NP Beweise gibt, zu deren Überprüfung der Verifizierer unter Benutzung von logarithmisch vielen Zufallsbits nur konstant viele Bits des Beweises lesen muss. Diese Charakterisierung von NP wird als PCP-Theorem (**p**robabilistically **c**heckable **p**roofs) bezeichnet.

Das PCP-Theorem hat immense Auswirkungen auf die Approximation schwieriger Probleme. Christos Papadimitriou und Mihalis Yannakakis [70] führten 1991 die Klasse MAXSNP approximierbarer Optimierungsprobleme ein. Hier geht es nicht darum, die Lösung exakt zu bestimmen, sondern möglichst gute Näherungen zu finden. Beim Cliquenproblem etwa sucht man möglichst große Cliquen, und beim Erfüllbarkeitsproblem MaxSat interessiert man sich für Belegungen, die nicht unbedingt die gesamte Formel, aber doch viele Klauseln erfüllen. Arora et al. zeigten jedoch, dass unter der Annahme $\mathsf{P} \neq \mathsf{NP}$ kein MAXSNP-vollständiges Problem, wozu z.B. auch MaxSat gehört, gut approximiert werden kann. Für MaxSat bedeutet dies z.B. die Existenz einer Konstante $\delta > 1$, so dass das Verhältnis der Anzahl maximal erfüllbarer Klauseln zur Zahl der vom Algorithmus erfüllten Klauseln für keinen effizienten Approximationsalgorithmus garantiert kleiner als δ wird.

Die genaue Bestimmung dieser maximalen Approximationsgüte δ ist in vielen Fällen noch offen. Für 3-SAT und einige weitere Probleme erzielte Johan Håstad [46] 1997 exakte Schranken.

5.2 Pseudozufallsgeneratoren und Derandomisierung

Randomisierte Algorithmen spielen für die Praxis eine immer größere Rolle. Oft sind diese konzeptionell einfacher als ihre deterministischen Gegenstücke, manchmal sind auch gar keine effizienten deterministischen Algorithmen bekannt. Reale Computer sind aber nicht randomisiert sondern deterministisch. Eine kleine Anzahl zufälliger Bits lässt sich meist aus der Systemzeit, zufälligen Mausbewegungen oder ähnlichem gewinnen, echter Zufall bleibt aber eine kostbare Ressource. Der Grundgedanke bei Pseudozufallsgeneratoren besteht darin, aus einer kleinen Anzahl echter Zufallsbits effizient eine große Anzahl pseudozufälliger Bits zu erzeugen. Pseudozufällig bedeutet hierbei, dass die Bits von echten Zufallsbits nicht effizient unterscheidbar sind.

Die erste Arbeit zu Pseudozufallsgeneratoren stammt von Manuel Blum und Silvio Micali [16] aus dem Jahr 1982, die Pseudozufallsgeneratoren aus harten kryptografischen Funktionen gewinnen. Kurz darauf leistete Andrew Chi-Chih Yao [95] einen wichtigen Beitrag, in dem grundlegende Konstruktionen für Pseudozufallsgeneratoren beschrieben werden. Insbesondere zeigte Yao, dass die Existenz von Einwegpermutationen die Existenz von Pseudozufallsgeneratoren nach sich zieht. Dieses Resultat wurde vielfach verallgemeinert. Den Höhepunkt dieser Entwicklung markiert die Arbeit von Håstad, Impagliazzo, Levin und Luby [47], die 1999 nachwiesen, dass Pseudozufallsgeneratoren genau dann existieren, wenn es einfach zu berechnende, aber schwer zu invertierende Einwegfunktionen gibt. Die Frage nach der Existenz von Pseudozufallsgeneratoren bleibt damit offen, ist jedoch eng mit zentralen kryptografischen und komplexitätstheoretischen Fragen verknüpft.

Einen anderen Weg beschritten 1994 Noam Nisan und Avi Wigderson [69], die Pseudozufallsgeneratoren aus Funktionen konstruieren, die hohe nichtuniforme Komplexität besitzen. Solche Pseudozufallsgeneratoren erlauben die Derandomisierung probabilistischer Komplexitätsklassen. Die Grundidee besteht darin, die für die probabilistischen Algorithmen benötigten Zufallsbits durch Pseudozufallsgeneratoren zu erzeugen. Haben die Pseudozufallsgeneratoren hinreichend gute Expansionseigenschaften, so können auch die zur Initialisierung der Generatoren nötigen Zufallsbits eliminiert werden. Die so gewonnenen Algorithmen sind also deterministisch. Auf diese Weise gelang Impagliazzo und Wigderson [50] 1997 der Nachweis, dass die Komplexitätsklassen P und BPP zusammenfallen, falls es Sprachen in EXP gibt, die nicht mit Schaltkreisen subexponentieller Größe berechnet werden können.

5.3 Aussagenlogische Beweiskomplexität

Aussagenlogische Beweissysteme spielen für Anwendungen wie das automatische Theorembeweisen oder die künstliche Intelligenz eine wichtige Rolle.

In der Beweiskomplexität stehen die Beweislängen im Mittelpunkt: wie lang müssen aussagenlogische Beweise für konkrete Formeln in einem gegebenen System mindestens sein? Diese Frage wurde bereits 1956 von Gödel in dem schon erwähnten Brief an von Neumann gestellt [38].

Eine der ersten und für das Gebiet grundlegenden Arbeiten stammt aus dem Jahr 1979 von Stephen Cook und Robert Reckhow [24], in der die Autoren eine sehr allgemeine komplexitätstheoretische Formalisierung aussagenlogischer Beweissysteme angeben, welche alle in der Praxis verwendeten Beweissysteme einschließt. Cook und Reckhow zeigen auch den Zusammenhang zwischen Beweislängen und zentralen Fragen der Komplexitätstheorie: gibt es kein Beweissystem mit polynomiell langen Beweisen für alle Tautologien, so ist die Klasse NP nicht unter Komplementbildung abgeschlossen, und damit ist $\mathsf{P} \neq \mathsf{NP}$. Daraus leitet sich das so genannte Cook-Reckhow-Programm ab, das darin besteht, schrittweise für leistungsfähigere Beweissysteme untere superpolynomielle Schranken für die Beweislänge nachzuweisen.

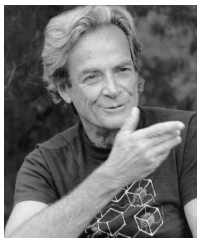
Das erste große Resultat auf diesem Weg erbrachte 1985 Amin Haken [43] für den Resolutionskalkül, das in der Praxis am häufigsten eingesetzte Beweissystem. Haken bewies, dass die aus dem Dirichletschen Schubfachschluss entstehenden Tautologienfolgen exponentiell lange Resolutionsbeweise erfordern. Seitdem wurden untere Schranken für eine Vielzahl weiterer Beweissysteme nachgewiesen, die auf verschiedenen algebraischen, geometrischen oder kombinatorischen Prinzipien beruhen.

Sehr starke Beweissysteme, für die gegenwärtig keine interessanten unteren Schranken bekannt sind, bilden aussagenlogische Hilbert-Kalküle, im Kontext der Beweiskomplexität meist Frege-Systeme genannt. Für eingeschränkte Frege-Systeme konnte Ende der 80er Jahre Miklós Ajtai [6] mit modelltheoretischen Methoden superpolynomielle untere Schranken nachweisen. Zusammen mit nachfolgenden Verbesserungen ist dies das bislang stärkste Resultat.

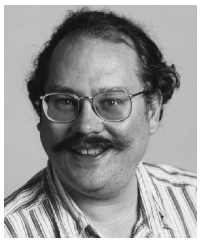
Eine allgemeine Technik zum Nachweis unterer Schranken lieferte 1997 Jan Krajíček [57] mit der Methode der effizienten Interpolation, die für viele schwache Beweissysteme erfolgreiche Anwendung fand. Eine weitere neue Technik, die den Zusammenhang zwischen minimalen Formelgrößen in Beweisen und minimalen Beweislängen ausnutzt, wurde 1999 von Eli Ben-Sasson und Avi Wigderson [13] vorgestellt. Für starke Beweissysteme wie Frege-Systeme greifen diese Techniken leider nicht, wie Jan Krajíček und Pavel Pudlák [59] 1998 nachweisen konnten. Ein vielversprechender Ansatz zur Verwendung der im letzten Abschnitt vorgestellten Pseudozufallsgeneratoren in der Beweistheorie wurde Ende der 90er Jahre von Krajíček [58] vorgeschlagen.

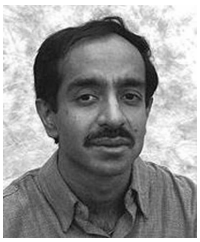
5.4 Quantencomputer

Ein ganz neues Forschungsgebiet, welches die Informatik in bisher unbekannter Weise mit der Physik verbindet, hat in der letzten Zeit vermehrt für Schlagzeilen gesorgt: die Quantenrechner. Der Physiker Richard Feynman [29] stellte


Richard Feynman
(1918–1988)


David Deutsch
(geb. 1952)


Peter W. Shor
(geb. 1959)


Lov Grover
(geb. 1960)

1982 fest, dass klassische Computer nicht in der Lage sind, quantenmechanische Systeme effizient zu simulieren, und erörterte in diesem Zusammenhang erstmals die Möglichkeit, Computer auf der Grundlage quantenmechanischer Prinzipien zu bauen. Die Frage nach der Einsetzbarkeit der Quantenmechanik für den Rechnerbau motiviert sich außerdem durch die Erwartung, dass bei fortschreitender Miniaturisierung elektronischer Bauteile ohnehin quantenmechanische Effekte in den Komponenten auftreten werden. Setzt sich die gegenwärtige Entwicklung ungebremst fort, wäre dies ungefähr 2020 der Fall, wie Robert Keyes [55] bereits 1988 prognostizierte.

1985 führte David Deutsch [26] ein theoretisches Modell für Quantenrechner, die Quanten-Turingmaschine, ein. Deutsch demonstrierte auch, dass es Probleme gibt, die mittels Quanten-Turingmaschinen effizienter lösbar sind als mit klassischen Turingmaschinen. Zwar haben die von Deutsch 1985 und auch 1992 in Zusammenarbeit mit Richard Jozsa [27] untersuchten Probleme keine praktische Relevanz. Sie zeigen jedoch die prinzipielle Überlegenheit von Quantenrechnern gegenüber klassischen und selbst randomisierten Verfahren.

Das wohl wichtigste Resultat zu Quantenalgorithmen stammt von Peter Shor [87] aus dem Jahr 1993. Shor entwarf einen Quantenalgorithmus, der das Faktorisierungsproblem FACTORIZE in Polynomialzeit löst. Die besten bekannten klassischen Algorithmen benötigen hingegen fast exponentielle Laufzeit. Dieses Ergebnis sorgte deswegen für Furore, weil die Sicherheit vielfach eingesetzter kryptografischer Verfahren wie des RSA [78] auf der Härte des Faktorisierungsproblems beruht. Gelänge der Bau von Quantenrechnern, würden nahezu alle derzeit benutzten Public-Key-Verfahren unsicher.

Ein weiteres bemerkenswertes Resultat lieferte Lov Grover [42] 1996 in Form eines Quantenalgorithmus, der Suchoperationen in einer unstrukturierten Datenbank mit n Elementen in $\sqrt{n}$ Schritten realisiert. Klassische Verfahren können das nicht in weniger als n Schritten durch vollständiges Durchsuchen der Datenbank leisten, randomisierte Verfahren benötigen im Mittel immerhin noch $\frac{n}{2}$ Vergleiche. Solche Suchprobleme sind typisch für NP-vollständige Probleme. Die Brute-Force-Suche in Zeit 2^n könnte also mittels Quantenrechner auf $2^{n/2}$ beschleunigt werden.

1997 führten Ethan Bernstein und Umesh Vazirani [15] die Klasse BQP ein, die alle in Polynomialzeit mit Quantenalgorithmen lösbaren Probleme enthält. Die Frage, ob NP in BQP enthalten ist, wird durch Grovers Resultat nicht beantwortet. Überhaupt ist das Verhältnis der Klasse BQP zu den klassischen Komplexitätsklassen noch relativ ungeklärt. Bernstein und Vazirani zeigten $\mathsf{BQP} \subseteq \mathsf{PSPACE}$. Leonard Adleman, Jonathan DeMarrais und Ming-Deh Huang [3] verbesserten dies kurz darauf zu $\mathsf{BQP} \subseteq \mathsf{PP}$. Lance Fortnow und John Rogers [31] konnten sogar zeigen, dass BQP-Orakel für PP-Berechnungen nutzlos sind, was für NP-Orakel als unwahrscheinlich gilt. Dies ist ein Indiz, dass BQP keine NP-vollständigen Probleme enthält.

Über dem Gebiet der Quantenalgorithmen steht die große Frage, ob der Bau praktisch einsetzbarer Quantencomputer technisch jemals möglich sein wird. Zur Zeit bereitet vor allem die Realisierung von Quantenrechnern mit einer größeren Anzahl von Quantenbits Probleme. Im Jahr 2001 wurde in einem IBM-Labor die Zahl 15 mit Shors Verfahren auf einem Quantenrechner faktorisiert, der über sieben Quantenbits verfügte. Ob aber Quantenrechner tatsächlich eines Tages die Informatik revolutionieren werden, kann heute niemand mit Gewissheit voraussagen.

6 Das neue Jahrhundert

Ein noch recht junges und schnell wachsendes Forschungsgebiet wie die Komplexitätstheorie umfassend zu würdigen, ist kein einfaches Unterfangen. Gerade Ergebnisse der jüngsten Zeit entziehen sich oft einer objektiven geschichtlichen Darstellung. Deshalb verzichten wir auf eine ausführlichere Besprechung der erst zur Hälfte abgelaufenen Dekade und erwähnen exemplarisch zwei herausragende Resultate des neuen Jahrhunderts.

6.1 Zwei herausragende Ergebnisse aus jüngster Zeit

Das erste Ergebnis betrifft die seit über 30 Jahren diskutierte Frage, ob das Primzahlproblem PRIMES in Polynomialzeit gelöst werden kann. Bereits 1976 bewies Gary Miller [68], dass diese Frage eine positive Antwort erfährt, wenn man die Gültigkeit der Riemannschen Vermutung, eines der großen ungelösten Probleme der Analysis, voraussetzt. Für die praktische Lösung des Primzahlproblems verwendete man die schon erwähnten randomisierten Algorithmen. Im Jahr 2002 nun gelang es drei indischen Forschern, Manindra Agrawal, Neeraj Kayal und Nitin Saxena [4], einen Polynomialzeitalgorithmus für PRIMES zu entwerfen. Der Algorithmus, der auch außerhalb der Informatik für großes Aufsehen sorgte, macht wesentlich von zahlentheoretischen Ergebnissen Gebrauch – ein weiterer Beleg für den erfolgreichen Einsatz algebraischer Methoden in der Komplexitätstheorie.

Das zweite Resultat stammt von Omer Reingold [76] aus dem Jahr 2004. Bei dem Erreichbarkeitsproblem UGAP gilt es herauszufinden, ob in einem


Omer Reingold
(geb. 1969)


Nitin Saxena
(geb. 1981)

Neeraj Kayal
(geb. 1979)

Manindra Agrawal
(geb. 1966)

gegebenen ungerichteten Graphen ein Weg zwischen zwei ausgezeichneten Knoten existiert. Für gerichtete Graphen ist das Problem GAP, wie schon erwähnt, NL-vollständig. Für ungerichtete Graphen ist die Frage aber vermutlich einfacher. Mit einem trickreichen Algorithmus zeigte Reingold, dass UGAP deterministisch mit logarithmischem Platz lösbar ist. Dadurch gibt es jetzt sogar eine Komplexitätsklasse weniger: bislang wurde nämlich UGAP mit der Klasse SL assoziiert. Aus Reingolds Resultat folgt $\mathsf{L} = \mathsf{SL}$.

Diese beiden Algorithmen sind auch gute Beispiele für die Derandomisierung konkreter Probleme, da sowohl für PRIMES als auch für UGAP zuvor randomisierte Algorithmen bekannt waren.

6.2 Ausblick

Damit sind wir am Ende unseres historischen Spazierganges angelangt. Wichtige Teilbereiche haben wir dabei unerwähnt gelassen. Zählklassen etwa, deskriptive und parametrisierte Komplexitätstheorie, Kolmogoroff- und Kommunikationskomplexität und vieles weitere haben keine Aufnahme gefunden. Für ergänzende Lektüre und weiterführende Literaturempfehlungen verweisen wir auf die geschichtliche Darstellung der Komplexitätstheorie von Lance Fortnow und Steve Homer [30].

Naturgemäß steht am Ende einer solchen Übersicht die Frage, wie es weitergeht. Darüber kann nur spekuliert werden. Zu hoffen und zu erwarten ist jedoch, dass die Komplexitätstheorie auch im neuen Jahrhundert nichts von ihrer Faszination und Vitalität einbüßen wird. Neue Anwendungsfelder gilt es zu erschließen, und die großen Fragen des letzten Jahrhunderts harren ihrer Lösung.

Literaturverzeichnis

1. L. Adleman, M. Huang *Recognizing primes in random polynomial time.* in: Proc. 19th ACM Symposium on Theory of Computing, ACM Press, 1987, S. 462–469
2. L. Adleman, K. Manders *Reducibility, randomness, and intractibility.* in: Proc. 9th ACM Symposium on Theory of Computing, ACM Press, 1977, S. 151–163
3. L.M. Adleman, J. DeMarrais, M.-D.A. Huang *Quantum computability.* SIAM Journal on Computing, 1997, 26(5):1524–1540
4. M. Agrawal, N. Kayal, N. Saxena *PRIMES is in P.* Annals of Mathematics, 2004, 160(2):781–793
5. M. Ajtai σ_1^1*-formulae on finite structures.* Annals of Pure and Applied Logic, 1983, 24(1):1–48
6. M. Ajtai *The complexity of the pigeonhole-principle.* Combinatorica, 1994, 14(4):417–433
7. S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy *Proof verification and the hardness of approximation problems.* Journal of the ACM, 1998, 45(3):501–555
8. L. Babai *Trading group theory for randomness.* Proc. 17th ACM Symposium on Theory of Computing, ACM Press, 1985, S. 421–429
9. L. Babai *E-mail and the unexpected power of interaction.* Proc. 5th Structure in Complexity Theory Conference, 1990, S. 30–44
10. L. Babai, L. Fortnow, C. Lund *Non-deterministic exponential time has two-prover interactive protocols.* Computational Complexity, 1991, 1:1–40
11. T. Baker, J. Gill, R. Solovay *Relativizations of the P=?NP question.* SIAM Journal on Computing, 1975, 4:431–442
12. M. Ben-Or, S. Goldwasser, J. Kilian, A. Wigderson *Multiprover interactive proofs: How to remove intractability assumptions.* Proc. 20th ACM Symposium on Theory of Computing, 1988, S. 113–131
13. E. Ben-Sasson, A. Wigderson *Short proofs are narrow – resolution made simple.* Journal of the ACM, 2001, 48(2):149–169
14. C.H. Bennett, J. Gill *Relative to a random oracle A,* $P^A \neq NP^A \neq co\text{-}NP^A$ *with probability 1.* SIAM Journal on Computing, 1981, 10:96–113
15. E. Bernstein, U. Vazirani *Quantum complexity theory.* SIAM Journal on Computing, 1997, 26(5):1411–1473
16. M. Blum, S. Micali *How to generate cryptographically strong sequences of pseudorandom bits.* SIAM Journal on Computing, 1984, 13(4):850–864
17. R. Boppana, J. Håstad, S. Zachos *Does co-NP have short interactive proofs?* Information Processing Letters, 1987, 25(2):27–32
18. S.R. Buss, A. Kechris, A. Pillay, R.A. Shore *The prospects for mathematical logic in the twenty-first century.* Bulletin of Symbolic Logic, 2001, 7(2):169–196
19. A. Church *A note on the Entscheidungsproblem.* The Journal of Symbolic Logic, 1936, 1:345–363
20. A. Cobham *The intrinsic computational difficulty of functions.* Proc. of the 1964 International Congress for Logic, Methodology, and Philosophy of Science, 1964, S. 24–30
21. S.A. Cook *The complexity of theorem proving procedures.* Proc. 3rd Annual ACM Symposium on Theory of Computing, 1971, S. 151–158
22. S.A. Cook *A taxonomy of problems with fast parallel algorithms.* Information and Control, 1985, 64:2–22

23. S.A. Cook, P. McKenzie *Problems complete for deterministic logarithmic space.* Journal of Algorithms, 1987, 8:385–394
24. S.A. Cook, R.A. Reckhow *The relative efficiency of propositional proof systems.* The Journal of Symbolic Logic, 1979, 44:36–50
25. M. Davis, H. Putnam, J. Robertson *The decision problem for exponential Diophantine equations.* Ann. Math., 1961, 74:425–436
26. D. Deutsch *Quantum theory, the Church-Turing principle and the universal quantum computer.* Proc. of the Royal Society, 1985, 400:97–117
27. D. Deutsch, R. Jozsa *Rapid solutions of problems by quantum computations.* Proc. of the Royal Society, 1992, 439:553–558
28. J. Edmonds *Paths, trees, and flowers.* Canadian Journal of Mathematics, 1965, 17(3):449–467
29. R. Feynman *Simulating physics with computers.* International Journal of Theoretical Physics, 1982, 21:467–488
30. L. Fortnow, S. Homer *A short history of computational complexity.* Bulletin of the EATCS, 2003, 80:95–133
31. L. Fortnow, J. Rogers *Complexity limitations on quantum computation.* Journal of Computer and System Sciences, 1999, 59(2):240–252
32. L. Fortnow, M. Sipser *Are there interactive protocols for co-NP languages.* Information Processing Letters, 1988, 28:249–251
33. A.S. Fraenkel, M.R. Garey, D.S. Johnson, T. Schäfer, Y. Yesha *The complexity of checkers on an n x n board – preliminary report.* Proc. 19th IEEE Symposium on the Foundations of Computer Science, 1978, S. 55–64
34. A.S. Fraenkel, D. Lichtenstein *Computing a perfect strategy for n x n chess requires time exponential in n.* J. Comb. Theory, Ser. A, 1981, 31(2):199–214
35. M.L. Furst, J.B. Saxe, M. Sipser *Parity, circuits, and the polynomial-time hierarchy.* Mathematical Systems Theory, 1984, 17(1):13–27
36. M. Garey, D. Johnson *Computers and Intractability – A Guide to the Theory of NP-Completeness.* Freeman and Company, 1979
37. J. Gill *Computational complexity of probabilistic complexity classes.* SIAM Journal on Computing, 1977, 6:675–695
38. K. Gödel *Ein Brief an Johann von Neumann, 20. März, 1956.* in: P. Clote, J. Krajíček (Hrsg.) Arithmetic, Proof Theory, and Computational Complexity, Oxford University Press, 1993, S. 7–9
39. A. Goldberg, M. Sipser *Private coins versus public coins in interactive proof systems.* in: S. Micali (Hrsg.) Randomness and Computation, Advances in Computing Reasearch, Vol 5, JAI Press, 1989, S. 73–90
40. O. Goldreich, S. Micali, C. Rackoff *The knowledge complexity of interactive proof systems.* SIAM Journal on Computing, 1989, 18(2):186–208
41. O. Goldreich, S. Micali, A. Wigderson *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems.* Journal of the ACM, 1991, 38:691–729
42. L. Grover *A fast quantum mechanical algorithm for database search.* Proc. 28th ACM Symposium on Theory of Computing, 1996, S. 212–219
43. A. Haken *The intractability of resolution.* Theoretical Computer Science, 1985, 39:297–308
44. J. Hartmanis, R.E. Stearns *On the computational complexity of algorithms.* Transactions of the American Mathematical Society, 1965, 117:285–306

45. J. Håstad *Almost optimal lower bounds for small depth circuits.* in: S. Micali (Hrsg.) Randomness and Computation, Advances in Computing Reasearch, Vol 5, JAI Press, 1989, S. 143–170
46. J. Håstad *Some optimal inapproximability results.* Journal of the ACM, 2001, 48(4):798–859
47. J. Håstad, R. Impagliazzo, L.A. Levin, M. Luby *Construction of a pseudo-random generator from any one-way function.* SIAM Journal on Computing, 1999, 28(4):1364–1396
48. F.C. Hennie, R.E. Stearns *Two-tape simulation of multitape turing machines.* Journal of the ACM, 1966, 13(4):533–546
49. N. Immerman *Nondeterministic space is closed under complementation.* SIAM Journal on Computing, 1988, 17(5):935–938
50. R. Impagliazzo, A. Wigderson *P=BPP unless E has sub-exponential circuits: derandomizing the XOR lemma.* Proc. 29th ACM Symposium on Theory of Computing, ACM Press, 1997, S. 220–229
51. N.D. Jones *Space-bounded reducibility among combinatorial problems.* Journal of Computer and System Sciences, 1975, 11:68–85
52. R. Karp, A. Wigderson *A fast parallel algorithm for the maximal independent set problem.* Journal of the ACM, 1985, 32:762–773
53. R.M. Karp *Reducibility among combinatorial problems.* in R.E. Miller, J.W. Thatcher (Hrsg.) Complexity of Computer Computations, Plenum Press, 1972, S. 85–103
54. R.M. Karp, R.J. Lipton *Some connections between nonuniform and uniform complexity classes.* Proc. 12th ACM Symposium on Theory of Computing, ACM Press, 1980, S. 302–309
55. R.W. Keyes *Miniaturization of electronics and its limits.* IBM Journal of Research and Development, 1988, 32:24–28
56. J. Köbler, O. Watanabe *New collapse consequences of NP having small circuits.* SIAM Journal on Computing, 1998, 28(1):311–324
57. J. Krajíček *Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic.* The Journal of Symbolic Logic, 1997, 62(2):457–486
58. J. Krajíček *Tautologies from pseudo-random generators.* Bulletin of Symbolic Logic, 2001, 7(2):197–212
59. J. Krajíček, P. Pudlák *Some consequences of cryptographical conjectures for S_2^1 and EF.* Information and Computation, 1998, 140(1):82–94
60. R.E. Ladner *On the structure of polynomial-time reducibility.* Journal of the ACM, 1975, 22:155–171
61. C. Lautemann *BPP and the polynomial hierarchy.* Information Processing Letters, 1983, 17:215–217
62. T. Lengauer, K.W. Wagner *The binary network flow problem is logspace complete for P.* Theoretical Computer Science, 1990, 75(3):357–363
63. L. Levin *Universal sequential search problems.* Problems of Information Transmission, 1975, 9(3):265–266, Englische Übersetzung des 1973 in russischer Sprache veröffentlichten Originalartikels
64. D. Lichtenstein, M. Sipser *GO is polynomial-space hard.* Journal of the ACM, 1980, 27(2):393–401
65. C. Lund, L. Fortnow, H. Karloff, N. Nisan *Algebraic methods for interactive proof systems.* Journal of the ACM, 1992, 39(4):859–868

66. J.V. Matijasevič *Enumerable sets are Diophantine.* Dokl. Akad. Nauk, 1970, 191:27–282 (russisch).
67. A.R. Meyer, L.J. Stockmeyer *The equivalence problem for regular expressions with squaring requires exponential space.* Proc. 13th IEEE Symposium on Switching and Automata Theory, 1972, S. 125–129
68. G.L. Miller *Riemann's hypothesis and tests for primality.* Journal of Computer and System Sciences, 1976, 13:300–317
69. N. Nisan, A. Wigderson *Hardness vs randomness.* Journal of Computer and System Sciences, 1994, 49(2):149–167
70. C.H. Papadimitriou, M. Yannakakis *Optimization, approximation, and complexity classes.* Journal of Computer and System Sciences, 1991, 43(3):425–440
71. N. Pippenger *On simultaneous resource bounds.* Proc. 20th IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press, 1979, S. 307–311
72. M.O. Rabin *Probabilistic algorithm for testing primality.* Journal of Number Theory, 1980, 12(1):128–138
73. M.O. Rabin, D. Scott *Finite automata and their decision problems.* IBM Journal of Research and Development, 1959, 3:114–125
74. A.A. Razborov *Lower bounds on the monotone complexity of boolean functions.* Doklady Akademii Nauk SSSR, 1985, 282:1033–1037, Englische Übersetzung in: Soviet Math. Doklady, 31, S. 354–357.
75. A.A. Razborov, S. Rudich *Natural proofs.* Proc. 26th ACM Symposium on Theory of Computing, 1994, S. 204–213
76. O. Reingold *Undirected st-connectivity in log-space.* Proc. 37th ACM Symposium on Theory of Computing, ACM Press, 2005, S. 376–385
77. H.G. Rice *Classes of recursively enumerable sets and their decision problems.* Trans. Am. Math. Soc., 1953, 74:358–366
78. R.L. Rivest, A. Shamir, L.M. Adleman *A method for obtaining digital signatures and public-key cryptosystems.* Communications of the ACM, Feb. 1978, 21(2):120–126
79. D. Rolf *Improved bound for the PPSZ/Schöning-algorithm for 3-SAT.* Technical Report TR05-159, Electronic Colloquium on Computational Complexity, 2005.
80. J.E. Savage *Computational work and time of finite machines.* Journal of the ACM, 1972, 19:660–674
81. W. Savitch *Relationships between nondeterministic and deterministic tape complexities.* Journal of Computer and System Sciences, 1970, 4(2):177–192
82. U. Schöning *Complexity and Structure*, Band 211 von *Lecture Notes in Computer Science.* Berlin Heidelberg, Springer-Verlag, 1986.
83. U. Schöning *Graph isomorphism is in the low hierarchy.* Journal of Computer and System Sciences, 1988, 37:312–323
84. A. Shamir *IP=PSPACE.* Journal of the ACM, 1992, 39(4):869–877
85. C. Shannon *Communication theory of secrecy systems.* Bell Systems Technical Journal, 1949, 28:657–715
86. J.C. Sheperdson, H.E. Sturgis *Computability of recursive functions.* Journal Ass. Comp. Mach., 1963, 10:217–255
87. P. Shor *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.* SIAM Journal on Computing, 1997, 26(5):1484–1509
88. M. Sipser *A complexity theoretic approach to randomness.* Proc. 15th ACM Symposium on Theory of Computing, ACM Press, 1983, S. 330–335

89. R. Solovay, V. Strassen *A fast Monte-Carlo test for primality.* SIAM Journal on Computing, 1977, 6:84–85
90. L.J. Stockmeyer *Arithmetic versus Boolean operations in idealized register machines.* Technical Report RC 5954, Math. Dept., IBM Thomas J. Watson Research Center, 1976.
91. L.J. Stockmeyer, A.R. Meyer *Word problems requiring exponential time.* Proc. 5th ACM Symposium on Theory of Computing, 1973, S. 1–9
92. R. Szelepcsényi *The method of forced enumeration for nondeterministic automata.* Acta Informatica, 1988, 26(3):279–284
93. S. Toda *PP is as hard as the polynomial-time hierarchy.* SIAM Journal on Computing, 1991, 20:865–877
94. A.M. Turing *On computable numbers, with an application to the Entscheidungsproblem.* Proc. Lond. Math. Soc., 1936, 42:230–265
95. A.C. Yao *Theory and applications of trapdoor functions.* Proc. 23rd IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press, 1982, S. 80–91