Was leistet die Komplexitätstheorie für die Praxis?

Heribert Vollmer

*Theoretische Informatik, Universität Würzburg**

Zusammenfassung Ziel des vorliegenden Beitrages ist es, zu zeigen, daß die Komplexitätstheorie als ein wichtiges Teilgebiet der Theoretischen Informatik eine große Relevanz für die informatische Praxis besitzt. Dabei werden Fragen erkenntnistheoretischer Natur bewußt ausgeklammert, aber es wird verdeutlicht, wie die Komplexitätstheorie Impulse sowohl für die Arbeit des einzelnen Software-Ingenieurs wie auch für die Entwicklung der gesamten Informatik gibt.

Schlüsselwörter Theorie und Praxis, Theoretische Informatik, Anwendungsbezug, Komplexitätstheorie

Summary This paper argues, that computational complexity theory, an important field within theoretical computer science, has a great practical relevance. By purpose we do not touch philosophical questions, but point out that computational complexity has impacts on the work of single software engineers as well as on the development of the whole field of informatics.

Keywords theory and practice, theoretical computer science, practical relevance, computational complexity theory

Computing Reviews Classification F.0, F.2, K.2, K.3.2, K.4.3

1. Wozu Theoretische Informatik?

Die Informatik wird, trotz teilweise kontrovers geführter Diskussionen, im allgemeinen als Ingenieurwissenschaft und gleichzeitig Formalwissenschaft („a formal and an engineering science “ [Flo92, p. 19]) verstanden, die aber auch – anders als die klassischen Ingenieurwissenschaften – die sozialen Aspekte des Einsatzes ihrer Produkte untersucht. Unstrittig ist jedoch, daß die Informatik als *Wissenschaft* akzeptiert wird. Dies bedingt, daß Theoriebildung und Systematisierung der erreichten Ergebnisse zu ihren essentiellen Aufgaben gehören, denn ohne sie verliert die Informatik ihren Status als Wissenschaft [Büt95, Abschnitt 6.4]. Dabei muß sich die Informatik als junge Disziplin ihre „eigenen wissenschaftlichen und theoretischen Grundlagen zu einem großen Teil selbst entwickeln“ (nachzulesen in den editorischen Richtlinien des *Informatik Spektrum*, abgedruckt in jeder Ausgabe auf Seite A4); darüber hinaus übernimmt sie, ihrem Charakter als anwendungsbezogene Wissenschaft entsprechend, auch Theoriekonzepte aus verwandten Nachbardisziplinen. Schon früh in der Entwicklung der Informatik ist darauf hingewiesen worden, daß, überläßt die Informatik sich zu sehr reinen Anwendungsfragen, die Gefahr droht, daß sie ihre Einheit verliert und in jedem der Anwendungsbereiche eine eigene Informatik betrieben wird. [Mai79, p. 168]. Diese Gefahr ist heute aktueller denn je.

*Lehrstuhl für Theoretische Informatik, Universität Würzburg, Am Exerzierplatz 3, 97072 Würzburg, e-mail-Adresse: vollmer@informatik.uni-wuerzburg.de.

Für die Entstehung der Disziplin Informatik sind Entwicklungen aus so verschiedenen Gebieten wie Logik, Mathematik, Sprachtheorie, Elektrotechnik und Nachrichtentechnik verantwortlich. Gerade für eine derart breite Disziplin ist eine sichere theoretische Grundlage wichtig. Für die theoretische Fundierung des Gebietes waren dabei Entwicklungen in der Logik von äußerster Wichtigkeit [Dav88, Gan88]. Dies führte dazu, daß manchmal der extreme Standpunkt vertreten wird, Informatik sei „very large scale application of logic (VLSAL)“ [Dij89][1]. Richard Karp, Komplexitätstheoretiker und einer der Väter der Theorie der NP-Vollständigkeit, hat dieser Auffassung jedoch unmittelbar widersprochen und aufgezeigt, daß formale Methoden deutliche Grenzen bei der Anwendbarkeit im Prozeß der Software-Erstellung aufweisen [Kar89]. Genauso kurzsichtig wäre es jedoch, zu leugnen, daß aus der mathematischen Logik stammende Begriffsbildungen (z. B. der des Algorithmus[2]) und Methoden (etwa zur Analyse von Algorithmen) auch heute noch in weiten Gebieten der Informatik große Bedeutung haben[3].

Die Theoretische Informatik bildet den theoretischen Kern[4] der Informatik. Sie umfaßt mathematische Begründungen der Informatik und mathematisch-logisch geprägte Untersuchungen ihres Gegenstandsbereiches; dazu gehören Logik, Formale Sprachen, Berechenbarkeit, Komplexität, Semantik, Spezifikation usw. Durch Abstraktion gelangt sie von Phänomenen, Paradigmen, Vorgehensweisen etc., die in der Praxis auftreten, zu formalen Begriffen. Durch Verwendung von mathematischen Methoden und Ergebnissen kommt sie zu Aussagen über die abstrakten Objekte, die dann ihrerseits wieder Anwendung in der Praxis finden. Die fundamentale Bedeutung der Theoretischen Informatik ist unstrittig. So wird zum Beispiel die Frage nach der Natur effizienter Berechnungen generell als von enormer wissenschaftlicher, ja philosophischer Bedeutung akzeptiert. Auch die Leistungen der Theoretischen Informatik für die „wissenschaftliche Praxis“ werden allgemein anerkannt. C. Papadimitriou hat eindrucksvoll vor Augen geführt, wie der Begriff der NP-Vollständigkeit (siehe Abschnitt 3.1.1.) in ca. 20 wissenschaftlichen Disziplinen erfolgreich bei der Lösung wichtiger Grundsatzfragen geholfen hat [Pap95]. Allein im angelsächsischen Bereich werden derzeit jährlich ungefähr 6000 Artikel veröffentlicht, in denen NP-Vollständigkeit als Schüsselwort genannt wird [GW96]. Die $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$-Fragestellung findet selbst in populärwissenschaftlichen Publikationen und in Nachrichtenmagazinen und Wochenzeitungen Beachtung[5]. Trotz der Akzeptanz der prinzipiellen Bedeutung der Theoretischen Informatik scheint jedoch das Urteil beinahe genauso feststehend zu sein, daß dieses Gebiet für die tägliche Arbeit des Praktikers, aber auch für die technologische Fortentwicklung der Informatik wenig bis keine Bedeutung besitzt.

In diesem Aufsatz soll der Nachweis geführt werden, daß dieses Urteil ungerechtfertigt ist. Anhand des Gebietes der Komplexitätstheorie werden wir zeigen, daß ein theoretisches Gebiet für die gesamte informatische Praxis bedeutungsvolle Impulse geben kann. Dabei fiel unsere Wahl naheliegenderweise auf unser eigenes Arbeitsgebiet; wir sind jedoch davon überzeugt, daß Vertreter anderer Gebiete der Theoretischen Informatik eine ähnliche Rechtfertigung anhand von Beispielen aus ihrer Erfahrung geben können. Wir würden es begrüßen, wenn dieser Beitrag eine Anregung dazu gäbe.

Wir halten Überlegungen, wie sie in dieser Arbeit gegeben werden, in einer Zeit wie der heutigen mit starken Trends zur Anwendungsforschung für sehr sinnvoll, möchten aber betonen, daß wir

[1] Dijkstras Meinung wird auch größtenteils von den Autoren von [DCG+89] geteilt.

[2] Es wird teilweise die Prognose vertreten, das Paradigma des Algorithmus werde sich im kommenden Jahrhundert auflösen [Kor97]; derzeit muß eine derartige Voraussage jedoch als sehr spekulativ angesehen werden. Zweifellos werden Phänomene wie Interaktion eine größere Rolle als heute in der Theoretischen Informatik spielen [Mil93, Weg98], aber dies wird u. E. nicht zu einer Verdrängung des Begriffs des Algorithmus führen.

[3] Eine ausführliche Diskussion der Berechtigung formaler und empirischer Methoden in der Informatik wird in [Büt95, Kapitel 7] gegeben.

[4] In [Kor97] wird erläutert, wie in der Informatik wie auch in jeder anderen wissenschaftlichen Disziplin von einem wissenschaftstheoretischen Standpunkt aus der Kern der Wissenschaft von ihrer Peripherie zu unterscheiden ist. Ersterer umfaßt also die Theoretische Informatik. Teilweise wurde eine allgemeinere „Theorie der Informatik“ gefordert [Coy92a], gekoppelt mit einem sehr weit gefaßten Begriff von Informatik [Nyg86, Coy92b]. Diese Auffassungen sind jedoch nicht unumstritten und werden an anderer Stelle als „zu weit geraten“ kritisiert [Büt95, Abschnitt 6.2].

[5] Wir verweisen hier nur beispielsweise auf den kürzlich in der ZEIT erschienenen Artikel [Beh99].

die Bedeutung der Komplexitätstheorie und der Theoretischen Informatik keineswegs (wie es leider in [AJK$^+$96] etwas anklingt) ausschließlich anhand ihres Einflusses auf die industrielle Praxis messen möchten; eine derartig einseitige Beurteilung verbietet sich hier ebenso wie bei jeder anderen theoretischen Disziplin.

2. Was ist Komplexitätstheorie?

Wie die Ingenieurwissenschaften schafft die Informatik etwas Anwendungsbezogenes, nämlich inbesondere durch Programme beschriebene Algorithmen zur Lösung spezifischer Probleme [Büt95, p. 92]. In der Mathematik und Informatik wurde eine Reihe von Schemata entwickelt, mit deren Hilfe die *Komplexität* eines Problems gemessen werden soll. Seit längerer Zeit werden Probleme aufgrund der *Komplexität der Struktur von Maschinen*, die zu ihrer Lösung benötigt werden [HU79][6], klassifiziert. So wird zum Beispiel das Problem, zu entscheiden, ob ein Wort zu einer regulären Sprachen gehört, als einfacher angesehen, als das gleiche Problem für die kontextfreien Sprachen, da erstere von endlichen Automaten entschieden werden können, wohingegen für die letzteren Kellerautomaten, ein Mechanismus mit komplizierterer Struktur als bei endlichen Automaten, benötigt werden. Weitere Schemata klassifizieren Probleme nach ihrer *deskriptiven Komplexität* (hierbei wird die syntaktische Komplexität eines Spezifikationsformalismus, im allgemeinen logischer Natur, als Maß genommen, siehe [Imm99]), *Kolmogorov-Komplexität* (hier wird die Beschreibungskomplexität endlicher Objekte untersucht, siehe [LV93]) oder *Kommunikationskomplexität* (hier wird die „Menge“ an Information, die zwischen Parteien ausgetauscht werden muß, um gemeinsam ein Berechnungsproblem zu lösen, gemessen, siehe [Hro97]).

In der *Theorie der Berechnungskomplexität* (engl.: *computational complexity theory*[7]) beruht die Angabe der Komplexität eines Problems auf der Laufzeit, dem Speicherbedarf oder anderen Ressourcen, die ein Algorithmus, der das untersuchte Problem löst, benötigt. Das betrachtete Maschinenmodell ist dabei im allgemeinen ein universeller Computer und kein eingeschränktes Automatenmodell wie oben in der Untersuchung der Komplexität der Struktur von Maschinen. Das Hauptaugenmerk der Komplexitätstheorie besteht darin, für ein gegebenes Problem entweder einen effizienten Algorithmus zu finden oder zu zeigen, daß es keinen geben kann [Pap94]. Prototypische Fragestellungen sind daher etwa: Kann Problem Π in Zeit t gelöst werden? Kann man zeigen, daß mindestens t Schritte notwendig sind? Probleme, bei denen diese Fragen zu „ähnlichen“ Antworten führen, werden in *Komplexitätsklassen* zusammengefaßt. Solch eine Klasse wird festgelegt, indem (1) das verwendete abstrakte Rechenmodell, (2) die in diesem Modell zu messenden Ressourcen und (3) Schranken für diese Ressourcen angegeben werden, die von Algorithmen eingehalten werden müssen, damit die von ihnen gelösten Probleme zur definierten Klasse gehören.

„The classes of problems which are respectively known and not known to have good algorithms are of great theoretical interest.“ Diese Feststellung von J. Edmonds aus dem Jahre 1966 (siehe [Pap94]) besitzt auch heute noch uneingeschränkte Gültigkeit und führt zu Fragestellungen von höchstem wissenschaftlichen und erkenntnistheoretischen Interesse, siehe dazu etwa [GW96]. Alleine aufgrund dieser Überlegungen halten wir das Gebiet der Komplexitätstheorie für untersuchenswert; eine entsprechen-

[6] Wir werden im folgenden, falls möglich, keine Referenzen auf Originalliteratur angeben, sondern auf gängige Lehrbücher oder leicht zugängliche Überblicksartikel verweisen. (Diese enthalten dann in der Regel weitergehende Referenzen auf die Originalliteratur.) Wir bemerken, daß inzwischen beinahe alle wichtigen Teilgebiet der Komplexitätstheorie durch Lehrbücher abgedeckt sind.

[7] Der Komplexitätstheorie werden alle die oben angegebenen (sowie weitere) Komplexitätsbetrachtungen zugeordnet. In diesem Aufsatz richten wir unser Augenmerk jedoch primär auf den Bereich, der im Englischen mit *computational complexity* (vielleicht am ehesten zu übersetzen mit „Berechnungskomplexität“) bezeichnet wird. Alle unten angegebenen Beispiele entstammen diesem Gebiet (wobei natürlich eine Trennung nicht immer klar möglich ist).

Im deutschsprachigen Raum werden in neuerer Zeit unglücklicherweise auch hin und wieder Untersuchungen aus der Chaostheorie unter dem Namen „Komplexitätstheorie“ geführt.

de Diskussion soll an dieser Stelle jedoch nicht geführt werden. Wir wollen im folgenden Argumente aufführen, die unsere Überzeugung stützen, daß Edmonds' Fragestellung ebenfalls von enormem praktischen Interesse ist, indem wir aufzeigen, daß die Ergebnisse der Komplexitätstheorie direkte Relevanz für die informatische Praxis und Technologie haben.

Zuvor möchten wir noch auf einen wichtigen Punkt hinweisen: Die Komplexitätstheorie untersucht die Schwierigkeit von Problemen anhand abstrakter Maschinen oder sogar auf eine maschinenunabhängige Art und Weise. Diese Entfernung von der Realität des praktischen Informatikers ist zu einem großen Teil verantwortlich für die Kritik, die an diesem Gebiet geübt worden ist. Die Beispiele, die wir im folgenden ausführen, zeigen jedoch ganz deutlich, wie auch die Komplexitätstheorie, trotz ihrer sehr abstrakten Methodik, unmittelbare konkrete Auswirkungen auf die Praxis ausübt. In vielen Fällen konnte sie, wie wir sehen werden, gerade aufgrund ihrer abstrakten Sichtweise wesentliche Begriffe und Verfahren entwickeln, die sich in der Praxis als äußerst nützlich erwiesen haben.

3. Leistungen der Komplexitätstheorie

3.1. Komplexitätstheorie als „Wegweiser"

Die Komplexitätstheorie stellt Hilfsmittel bereit, die den Informatiker bei der Analyse einzelner Probleme, aber auch bei der Beurteilung ganzer Paradigmen, etwa aus dem Bereich der Rechnerarchitektur oder Programmiermethodologie, unterstützen. Wir werden beide Aspekte in diesem Abschnitt ausführlicher erläutern.

3.1.1. Analyse der Komplexität konkreter Probleme

Unter einem theoretischen Gesichtspunkt werden im allgemeinen die Probleme als *effizient lösbar* angesehen, für die es einen *Polynomialzeitalgorithmus* gibt, der sie löst. Ein Polynomialzeitalgorithmus ist ein Algorithmus, der für Eingaben der Länge n nicht mehr als n^k Schritte (für eine Konstante k) für seine Ausführung auf dem betrachteten Rechner benötigt. Typischerweise sind in der Praxis deutlich stärkere Forderungen anzutreffen. Beispielsweise wird man einen Sortieralgorithmus, der zum Sortieren von n Datensätzen n^2 Vergleiche benötigt, nicht als effizient bezeichnen, obwohl seine Laufzeit durch ein Polynom beschränkt ist. Stattdessen wird man auf einem Algorithmus mit einer Laufzeit von $n \cdot \log n$ (wenigstens im Durchschnitt aller Eingaben) bestehen [CLR90]. Auf der anderen Seite ist es aber sicherlich so, daß ein Problem, für das kein Polynomialzeitalgorithmus existiert, auch und erst recht unter praktischen Gesichtspunkten nicht als effizient lösbar bezeichnet werden kann.

Diese Überlegungen führen direkt zur sog. *P-NP-Theorie* (auch: Theorie der NP-Vollständigkeit). Ohne eine formale Definition des Begriffes zu geben, halten wir fest, daß NP-vollständige Probleme die in gewissem Sinne schwierigsten Probleme in der Klasse NP (der Klasse aller Probleme, die in Polynomialzeit von *nichtdeterministischen* Maschinen gelöst werden können) sind. Gelingt es, von einem dieser schwierigsten Probleme nachzuweisen, daß es einen (deterministischen) Polynomialzeitalgorithmus besitzt, also in der Klasse P enthalten ist, so folgt direkt $P = NP$, das heißt, daß auch alle anderen NP-Probleme dann effiziente Algorithmen besitzen. Hieraus ergibt sich, daß ein Problem, von dem nachgewiesen wurde, daß es NP-vollständig ist, keinen Polynomialzeitalgorithmus besitzt unter der Hypothese $P \neq NP$; diese Annahme wird jedoch von den meisten Forschern aus der Komplexitätstheorie, ja wahrscheinlich aus der gesamten Theoretischen Informatik, geteilt. In der wissenschaftlichen Praxis wird daher der Nachweis der NP-Vollständigkeit eines Problems mit dem Nachweis, daß es keinen Polynomialzeitalgorithmus besitzt, gleichgesetzt. Eines der frühen Standardlehrbücher des Gebietes [GJ79] enthält im Anhang eine Liste vieler zur damaligen Zeit als NP-vollständig erkannten Probleme. D. Johnson, der zweite Autor dieses Buches, aktualisiert ständig diese Sammlung in der fortlaufenden

„NP-completeness column" des *Journal of Algorithms*. Heute weiß man von mehreren tausend Problemen, daß sie zur Klasse der NP-vollständigen Probleme gehören.

Für den Praktiker bedeutet dies folgendes: Stellt sich trotz intensiver Bemühungen heraus, daß für ein zu untersuchendes Problem kein effizienter Algorithmus zu finden ist, so sollte die Möglichkeit in Betracht gezogen werden, daß es sich um ein NP-vollständiges Problem handelt. Ist das Problem (oder ein äquivalentes) nicht in einer der vorhandenen Listen zu finden, so sollte man also versuchen, seine NP-Vollständigkeit zu beweisen. Bemerkenswerterweise gibt es eine Reihe von relativ leicht anzuwendenden Methoden, um dies zu bewerkstelligen. Gelingt der Nachweis der Vollständigkeit, so sollte nicht weiter Zeit, Energie und Geld auf der Suche nach einem effizienten Algorithmus vergeudet werden. Das Problem ist natürlich nach wie vor zu lösen, aber man weiß jetzt, daß man auf andere Weise mit ihm umgehen muß. Der Nachweis der Vollständigkeit ist erst der erste Schritt zu seiner Komplexitätsanalyse. Es ergeben zum Beispiel die folgenden Möglichkeiten des weiteren Vorgehens:

- Der Nachweis der NP-Vollständigkeit zeigt im allgemeinen Gründe für die Schwierigkeit eines Problemes auf. Dies kann dazu genutzt werden, das Problem besser zu spezifizieren. Die Hoffnung bei diesem Vorgehen ist, daß man im günstigsten Falle zu einem leichter lösbaren Teilproblem, das aber für die beabsichtigte Anwendung ausreichend ist, gelangt.

- Findet sich jedoch kein solches Teilproblem, bietet sich als nächstes die Analyse von Heuristiken an, also von „quick-and-dirty"-Algorithmen, die nicht notwendigerweise optimale Lösungen produzieren. Für manche Heuristiken kann sogar garantiert werden, daß die von ihnen produzierten Lösungen nicht weit vom Optimum entfernt sind. Formal spricht man dann von einem *Approximationsalgorithmus*.

- Vielleicht ist aber auch ein Algorithmus, der *im Mittel eine geringe Laufzeit hat*, ausreichend. Dies führt zur sogenannten average-case complexity. Die oben betrachtete worst-case complexity stellt keineswegs in allen Problembereichen automatisch die geeigneteste Art der Analyse dar.

Bei Fragen dieser Art können vorhandene ausführliche Untersuchungen als Richtschnur dienen (siehe z. B. [Hoc97]). Für eine ganze Reihe von Problemen ist die Grenze zwischen effizienter Lösbarkeit und NP-Vollständigkeit genau bekannt. Als Beispiel wollen wir nur nennen, daß in der Graphentheorie einige im allgemeinen schwierige Probleme effiziente Algorithmen für den Spezialfall besitzen, daß nur Graphen betrachtet werden, die in einen metrischen Raum eingebettet sind – eine Forderung, die in praktischen Anwendungssituationen fast immer gegeben ist. In anderen Bereichen sind genau die Fälle identifiziert worden, die zwar keine effiziente korrekte Lösung zulassen, aber eine sehr gute approximative Lösung. Das Problem des Handlungreisenden, bei dem es darum geht, eine kürzeste Tour (Rundreise) durch eine vorgegebene Menge von Städten zu finden, ist von äußerstem Interesse im Operations Research, im VLSI-Design, in der Netzwerkoptimierung (z. B. im Bereich des mobilen Telefonierens), etc. Dieses Problem ist in seiner allgemeinen Formulierung nicht nur NP-vollständig, sondern sogar nur sehr schwer zu approximieren [AL97]; schränkt man sich jedoch auf Euklidische Räume ein, so ist es im Prinzip beliebig gut approximierbar [Aro99]. Dieser Spezialfall ist für die genannten Anwendungen oft ausreichend.

Die Theorie der NP-Vollständigkeit kann also eine Hilfe sein, die konkreten Bemühungen eines Praktikers bei der Lösung eines Problems in eine geeignete Richtung zu lenken. Die Komplexitätstheorie fungiert hier in gewisser Weise als Orientierungshilfe oder „Wegweiser". „To become a good algorithm designer, you must understand the rudiments of the theory of NP-completeness" [CLR90].

Es gibt analoge Theorien in anderen Bereichen des Algorithmendesigns. Es gibt eine Theorie der *P-Vollständigkeit* [GHR95], bei der es darum geht, die Frage zu beantworten, für welche Probleme es effiziente parallele Algorithmen gibt. Gelingt der Nachweis der P-Vollständigkeit eines Problemes, so bedeutet dies, daß es für dieses Problem nach heutigem Kenntnisstand keinen effizienten parallelen

Algorithmus (formal: NC-Algorithmus) geben wird. In diesem Sinne werden die P-vollständigen Probleme oft als *inhärent sequentiell* bezeichnet. Effiziente parallele Algorithmen existieren für viele arithmetische und graphentheoretische Probleme sowie Probleme aus der linearen Algebra [KR90]. Demgegenüber sind die meisten Greedy-Algorithmen inhärent sequentiell [GHR95]. Andere P-vollständige Probleme finden sich unter den Flußproblemen, Schedulingproblemen, etc. Weiterhin gibt es eine Reihe von Vollständigkeitsbegriffen für verschiedene Konzepte der *Approximierbarkeit*. Hat sich von einem zu untersuchenden Problem die NP-Vollständigkeit herausgestellt und wird nun ein Approximationsalgorithmus gesucht, so kann einem diese Theorie als Wegweiser dienen bei der Beurteilung, welche Qualität der Approximation zu erwarten ist [AL97]. So weiß man seit kurzem, daß zum Beispiel das graphentheoretische Cliquenproblem nur schlecht approximierbar ist, wohingegen für viele geometrischen Probleme oder Schedulingprobleme sehr gute Approximationsverfahren bekannt sind [Hoc97]. Ein ständig aktualisiertes Kompendium, das über den Approximationsstatus von Hunderten von Optimierungsproblemen Auskunft gibt, steht im Internet zur Verfügung [CK99].

3.1.2. Berechnungs- und Programmierparadigmen

Die richtungsweisende Funktion der Komplexitätstheorie findet sich auch auf einer Ebene wieder, die nicht so sehr den einzelnen Software-Ingenieur bei seiner konkreten Arbeit betrifft, sondern die prinzipiellerer Natur ist, nämlich dann, wenn es um die Beurteilung *neuer Berechnungsparadigmen* geht. Wir denken da beispielsweise an Berechnungsmodelle wie die in letzter Zeit aufgekommenen Quantencomputer (hier werden Berechnungen mit Hilfe quantenphysikalischer Phänomene wie Superposition und Konvolution durchgeführt, siehe [LSP98, Gru99]) und Molekular-Computer (hier fungieren einzelne DNA-Stränge als elementare Prozessoren, siehe [SM97, Kapitel 9], [Pău98]). In der komplexitätstheoretischen Forschung der vergangenen Jahre wurden beide Modelle ausführlich untersucht, so daß heute eine Reihe von Resultaten über deren Mächtigkeit bekannt sind. Diese Ergebnisse werden natürlich sehr interessant werden, wenn diese Computer erst einmal zur Verfügung stehen. Heute schon können sie bei der Entscheidung helfen, ob die industrielle Entwicklung dieser Rechnertypen vorangetrieben werden soll. Konkret sind zum Beispiel ein effizienter Data-Mining-Algorithmus [Gro96] und ein effizienter Faktorisierungsalgorithmus [Sho97] für Quantencomputer bekannt. Gerade letzterer hätte im Falle der Realisierung von Quantencomputern große Auswirkungen auf die Sicherheit kryptographischer Verfahren (vgl. Abschnitt 3.2.2.).

In den achtziger Jahren wurde in der Komplexitätstheorie ein paralleles Maschinenmodell, die sog. PRAM (parallel random access machine), eingeführt und zugehörige Komplexitätsklassen (die NC-Hierarchie) wurden definiert und untersucht [Rei89, GHR95]. Lange Zeit ist gerade dieses Modell von praktischer Seite aus wegen seiner mangelnden Berücksichtigung technischer Machbarkeit angegriffen worden. Die technologische Entwicklung ist jedoch so rasant verlaufen, daß dieses Modell heute viel näher an praktischen Parallelrechnern liegt, als man vor 15 Jahren voraussehen konnte [GW96]. Damit ist es ein ideales Modell zur Entwicklung und Untersuchung paralleler Rechenverfahren und der dort auftretenden Phänomene (zum Beispiel aus dem Bereich der Datenstrukturen oder der Kommunikation zwischen Prozessen) geworden [Lei92, GS93].

Anhand der gerade geschilderten Entwicklung möchten wir auf zwei wichtige Punkte hinweisen:

- Abstrakte Maschinenmodelle, wegen ihrer Realitätsferne oft angegriffen, abstrahieren von nicht-wesentlichen Implementierungsdetails und sind so das ideale Hilfsmittel für die Wegweiser-Funktion der Komplexitätstheorie. Kann ein Problem nicht auf einer solchen Maschine gelöst werden, dann erst recht nicht auf deren realen Entsprechungen. Oben (Abschnitt 3.1.1.) wurde bereits das Beispiel erwähnt, daß ein Problem, das keinen Polynomialzeitalgorithmus besitzt, erst recht unter praktischen Gesichtspunkten nicht als effizient lösbar angesehen werden kann. Im Kontext hier heißt das: Gibt es für ein Problem keinen effizienten PRAM-Algorithmus, so wird es erst recht keine effiziente parallele Lösung auf der Connection Machine (einem Prozessor-

Hyperwürfel vom Grad 16), Transputer Netzwerken oder anderen realisierten Parallel-Computern geben.

- Dadurch, daß die Entwicklung im Rechnerbau die Theorie eingeholt hat, sind die Aussagen, die die Komplexitätstheorie in den vergangenen 15 Jahren über das vermeintlich unrealistische PRAM-Modell gesammelt hat, heute von großem praktischen Wert. Die Lehre, die wir daraus ziehen sollten, ist die, daß auch eine zunächst von jeglichen praktischen Zwängen losgelöste theoretische Forschung sehr wohl Beiträge zur technologischen Entwicklung liefern kann. Es gibt nicht immer einen unmittelbaren Bezug zwischen innovativer Theorie und praktischen Anwendungen [Büt95, p. 100f].

Die Komplexitätstheorie hilft aber nicht nur weiter bei der Beurteilung von Berechnungsparadigmen, die u. U. immer Fiktion bleiben werden, sondern auch bei der Beurteilung von *Programmierparadigmen*. Vom Standpunkt der Berechenbarkeitstheorie aus betrachtet, sind alle bekannten Programmiersprachen äquivalent, dies gilt jedoch nicht immer, was die Effizienz der Implementation von Algorithmen betrifft. Detaillierte Forschungen an der Nahtstelle zwischen Komplexität und Programmiersprachen wurden am Beispiel logischer, funktionaler, objektorientierter und weiterer Programmierparadigmen angestellt (siehe [RC94, Jon97]). Endlose Debatten sind über die Frage geführt worden, ob funktionale oder logische Programme notwendigerweise weniger effizient als äquivalente imperative Programme sind. Es gibt Untersuchungen über die Effizienz von puren LISP-Programmen im Vergleich zu LISP-Programmen, in denen Seiteneffekte erlaubt sind [Pip97]. Ebenso wurde die Effizienz von verschiedenen Auswertungsmechanismen (lazy vs. eager) in funktionalen Sprachen untersucht. Falls Effizienz eine große Rolle spielt, hat der informierte Programmierer so die Möglichkeit, sich für die für seine Anwendung richtige Sprache zu entscheiden; spielen Komplexitätsfragen keine Rolle oder stehen mehrere gleich-effiziente Kandidaten zur Verfügung, kann derjenige gewählt werden, dessen Sprachmittel größtmögliche Nähe zur Problemstellung aufweisen. In diesem Zusammenhang sollten auch Untersuchungen der Komplexität von Datenbankanfragesprachen nicht unerwähnt bleiben; zu diesem Themenkomplex ist eine reichhaltige Literatur vorhanden (siehe zum Beispiel [CGT90, AHV95, Imm99]).

Anfragesprachen werden nicht nur im Datenbank-Kontext verwendet; sie spielen auch eine große Rolle in vielen Bereichen des Software-Engineering (Reengineering, CASE, etc.). Hier können die auch schon im Compilerbau erfolgreich eingesetzten Klassifikationen formaler Sprachen (z. B. in der Chomsky-Hierarchie [HU79]) und daraus abgeleitete Komplexitätsaussagen (für den Bereich des Parsings oder der Datenflußanalyse) genutzt werden. Die Komplexitätstheorie liefert weiterhin Aussagen über die Mächtigkeit von Anfrageformalismen in Korrespondenz zur Effizienz der Auswertung von formulierten Anfragen. Es ist bekannt, um ein Beispiel zu nennen, daß in SQL die transitive Hülle einer Relation nicht berechnet werden kann [Imm99]. Diese und andere Unzulänglichkeiten von SQL führten zu ausdrucksstärkeren Formalismen, wie etwa DATALOG [CGT90] oder auch verschiedenen graphbasierten Anfragesprachen [KW99]. Letztere erlauben beispielsweise eine effiziente Implementierung des Transitive-Hülle-Operators.

Zusammenfassend stellen wir fest, daß die Komplexitätstheorie als Wegweiser für Software-Ingenieure bei der Lösung eines konkreten Problemes dienen kann, indem dessen effiziente Lösbarkeit, Parallelisierbarkeit, Approximierbarkeit usw. vorausgesagt wird, aber auch für die Entwicklung des gesamten Gebietes zum Beispiel in bezug auf neue Berechnungsparadigmen. Mathematische Aussagen, wie sie die Theorie hier liefert, sind nicht nur richtungsweisend in Entwicklung und Implementierung, sondern auch als Begründungshilfe (etwa in der Diskussion mit Vertretern von Entscheidungsinstanzen) hilfreich.

3.2. Konkrete Aussagen der Komplexitätstheorie

Im vorangegangenen Abschnitt haben wir die Komplexitätstheorie in ihrer Wegweiser-Funktion dargestellt. Ihre Aussagen aus diesem Bereich sind allgemeiner Natur, indem eine Reihe von Methoden entwickelt werden, die Hilfestellung beim Umgang mit schwierigen Problemen geben. Die Komplexitätstheorie ist jedoch mehr als ein Wegweiser. Im folgenden wollen wir Beispiele dafür angeben, wie Aussagen der Theorie explizit zu Algorithmen und Programmen, ja sogar industriellen Produkten, führen können.

3.2.1. Abstrakte Untersuchungen führen zu konkreten Ergebnissen

Es ist eine allgemeine wissenschaftliche Erfahrung, daß die Lösung eines Einzelproblems oft erst ermöglicht wird, nachdem eine Verallgemeinerung und Abstraktion desselben vorgenommen wurde. Dies zeigt sich ganz deutlich in den Anwendungsgebieten der Komplexitätstheorie. Wir geben eine Reihe von Beispielen.

Überall in Rechnernetzen (und beinahe überall sonst beim praktischen Einsatz informationstechnischer Produkte) ist Kommunikation verschiedener Parteien zu beobachten. In der Komplexitätstheorie wird dies abstrakt unter dem Namen *Interaktive Beweissysteme* untersucht. Einführung des Zufalls in dieses Modell führte zur Entwicklung sogenannter „Zero-Knowledge Protokolle“. Dabei handelt es sich um Protokolle, in denen eine Partei A eine zweite Partei B davon überzeugen muß, daß sie ein Geheimnis kennt, jedoch ohne es zu verraten. Dies macht beispielsweise dann Sinn, wenn zu befürchten ist, daß die Kommunikation von einer weiteren Partei C abgehört werden kann, aber auch, wenn man verhindern möchte, daß B sich künftig als A ausgeben kann. Solche Systeme finden klarerweise Anwendung in Authentifikationssystemen, elektronischen Zugangskontrollen, etc. Eine Reihe konkreter „außerordentlich praxisrelevanter Verfahren“ [BSW95] wurde inzwischen entwickelt[8].

Eine neue Charakterisierung der Klasse NP durch sogenannte probabilistisch überprüfbare Beweise (*probabilistically checkable proofs, PCP*) wurde vor wenigen Jahren entwickelt (siehe [Gol99]). Diese Charakterisierung beruht auf einer auf den ersten Blick widersprüchlich erscheinenden Möglichkeit, (lange) Beweise (Passwörter, PINs, PGP-keys, elektronische Unterschriften etc.) zu überprüfen, indem nur an wenigen Stellen (spot checks) in den Beweis gesehen wird[9]. Es ist zu erwarten, daß diese Methode Anwendungen in Kommunikationssytemen aller Art finden wird.

Das eigentliche Ziel bei der Entwicklung der PCP-Theorie war es, ein Hilfsmittel zum Nachweis der Nicht-Approximierbarkeit von NP-Optimierungsproblemen (Wegweiserfunktion der Komplexitätstheorie, Abschnitt 3.1.1.) zu schaffen [MPS98]. Der Approximationsstatus von graphentheoretischen Problemen wie Clique und Färbbarkeit war lange Zeit offen; als Konsequenz aus der PCP-Charakterisierung von NP folgt nun, daß die entsprechenden Approximationsprobleme NP-hart sind [AL97]. Obwohl die PCP-Theorie hauptsächlich zum Nachweis der *Nicht*-Approximierbarkeit dient, ergab sich auch eine „positive“ Konsequenz: S. Arora konnte kürzlich zeigen, daß es für das Problem des Handlungsreisenden in der Euklidischen Ebene beliebig genaue effiziente Approximationsalgorithmen gibt [Aro99]. Dies darf gewiß als bahnbrechendes Resultat in der Theorie der Optimierungsprobleme bezeichnet werden. Das Resultat als solches ist zwar unabhängig von der PCP-Theorie; Arora selbst schreibt jedoch [Aro99], daß er seinen Approximationsalgorithmus erst bei dem Versuch entdeckte, mit den von der Theorie bereitgestellten Methoden gerade die Nicht-Approximierbarkeit nachzuweisen.

Untersuchungen des theoretischen Berechnungsmodells der sogenannten *Branching-Programme* haben weitreichende Anwendungen im VLSI-Design gefunden. Branching-Programme wurden in den

[8] Es ist bekannt, daß alle Sprachen aus NP Zero-Knowledge-Protokolle besitzen (unter einer vernünftigen komplexitätstheoretischen Hypothese, siehe [Gol99]).

[9] Diese Entdeckung veranlaßte die *New York Times* am 7. April 1992 dazu, einen euphorischen Artikel unter der Schlagzeile „New Shortcut Found for Long Math Proofs“ zu veröffentlichen [Kol92].

sechziger Jahren in den Bell Labs zur Repräsentation Boolescher Funktionen verwendet [Lee59]. Inzwischen liegen von komplexitätstheoretischer Seite aus detaillierte Untersuchungen ihrer Berechnungskraft vor. Vor wenigen Jahren stellte sich nun heraus, daß diese Programme zu einer Datenstruktur für Boolesche Funktionen führen, die effiziente Algorithmen für eine Reihe wichtiger Aufgaben aus dem CAD oder VLSI-Design erlaubt [MT98]. Aus dieser Forschung sind inzwischen komplette Programmpakete, sog. OBDD-Packages, die auch industriell eingesetzt werden, hervorgegangen.

Viele Algorithmen aus weiteren Anwendungsgebieten verdanken ihre Entstehung komplexitätstheoretischen oder mathematischen Methoden. Der bekannte Linearzeit-Algorithmus von Knuth, Morris und Pratt zum *pattern matching*, bei dem es darum geht, festzustellen, ob ein Text ein bestimmtes Muster enthält [Sch97], geht ursprünglich auf ein Resultat von S. Cook [Coo72] zurück, das zeigt, daß jeder deterministische 2-Weg-Kellerautomat auf einer Random Access Machine in linearer Zeit simuliert werden kann (siehe [WW86, pp. 234ff]). Unter Verwendung dieser Beziehung ergibt sich der gewünschte Algorithmus auf sehr einfache Weise. Sehr effiziente Verfahren zur Multiplikation von Matrizen, die auf abstrakten algebraischen Resultaten beruhen, sind bekannt. Der Strassen-Algorithmus [Sch97] findet in vielen Programmpaketen praktische Anwendung. (Es sind zwar einige asymptotisch schnellere Verfahren bekannt, deren praktische Einsetzbarkeit jedoch an zu großen Laufzeiten für kleine Matrizen scheitert.) Schon bevor Strassen seinen Matrixmultiplikationsalgorithmus entwickelte, war auch für Multiplikation großer Zahlen eine ähnliche Methode bekannt, die wesentlich effizienter ist, als die „Schulmethode“. Dieser sog. Karatsuba-Algorithmus [Sch97] wird ebenfalls in vielen Implementierungen (auch auf der Hardware-Ebene) verwendet. Darüberhinaus zu nennen sind u.a. Verfahren der linearen Programmierung, Roboterbewegung, Lernverfahren in der künstlichen Intelligenz, fehlertolerante Berechnungen und Übertragungen, etc.

Die in diesem Abschnitt aufgeführten Beispiele zeigen, daß eine Lösung eines anstehenden Problems eventuell erst durch seine Abstraktion, die das Problem auf eine mathematische Ebene hebt, möglich wird. Dadurch werden nämlich (zum Teil altbekannte) mathematische Resultate für dessen Lösung relevant. Gerade in der PCP-Theorie beispielsweise werden tiefliegende Resultate aus einer Reihe verschiedener Teilgebiete der Mathematik und Informatik in bemerkenswerter Weise kombiniert und stehen somit, wie oben erläutert, zur Lösung praktischer Fragestellungen zur Verfügung.

3.2.2. Konsequenzen der Existenz schwieriger Probleme

Ein weiteres Feld positiver Aussagen der Komplexitätstheorie ergibt sich gerade aus der Existenz schwieriger (d. h. nicht effizient-lösbarer) Probleme. Ein typisches Anwendungsgebiet ist hier die Kryptographie.

In der *klassischen Kryptographie* wird die Sicherheit eines Verschlüsselungsschemas informationstheoretisch definiert, indem gefordert wird, daß aus dem Kryptotext (ohne Kenntnis des zugehörigen Schlüssels) keine Information über den Klartext extrahierbar sein soll. Dies impliziert, daß der verwendete Schlüssel mindestens so lang wie der Quelltext sein muß [Gol99]. In der *modernen Kryptographie* wird die Sicherheit eines Schemas mithilfe der Komplexitätstheorie definiert, indem gefordert wird, daß Information über den Klartext aus dem Kryptotext nicht effizient extrahierbar ist. Eine Folge dieses neuen Ansatzes ist die Existenz von *Public-Key-Kryptosystemen*. Dies sind Systeme, die nach folgendem Schema funktionieren: Der Empfänger B einer Nachricht berechnet einen Chiffrierungs- und einen Dechiffrierungsschlüssel und veröffentlicht sodann den Chiffrierungsschlüssel. Der Sender A benutzt diesen, um seinen Klartext zu kodieren, und schickt dann den Kryptotext an den Empfänger. B verwendet nun seinen Dechiffrierungsschlüssel, um den Klartext zu rekonstruieren. Die Schlüssel sind dabei so beschaffen, daß einem potentiellen Angreifer C die Kenntnis des Chiffrierungsschlüssel nicht weiter hilft, denn er ermöglicht es nicht, in effizienter Weise aus dem Kryptotext den Klartext zu rekonstruieren. Daher kann dieser Schlüssel auch über unsichere Kanäle verbreitet werden. Public-Key-Kryptosysteme lösen somit das *Schlüsselverteilungsproblem*, eine bekannte Problematik älterer Kryptosysteme.

Unter der klassischen Auffassung von Sicherheit kann solch ein System klarerweise nicht existieren [Gol99]. Unter dem neuen Paradigma können aber sichere Public-Key-Kryptosysteme mithilfe von Problemen gebaut werden, von denen bekannt ist, daß sie nicht effizient lösbar sind. Beispielsweise beruht das RSA-Verfahren auf der Tatsache, daß kein effizienter Algorithmus bekannt ist, der das Produkt zweier Primzahlen faktorisiert. (Der RSA-Algorithmus wird heute im PGP-System (*pretty good privacy*) verwendet, das weite Verbreitung und Anwendung im Internet, beispielsweise zur Chiffrierung von Dateien, für elektronische Unterschriften etc., gefunden hat [Gar95]. S/MIME, ein neuer Standard zum Verschlüsseln und digitalen Signieren von e-mails, benutzt ebenfalls den RSA-Algorithmus. S/MIME wird u. a. im Modzilla-Browser von Netscape eingesetzt [Net99].) Andere Public-Key-Systeme verwenden andere vermuteterweise schwierige Probleme, zum Beispiel das des sog. diskreten Logarithmus oder NP-vollständige Probleme wie das Rucksackproblem (die NP-Vollständigkeit von Faktorisierung und diskretem Logarithmus ist nicht bekannt). Diese Systeme können ausschließlich deshalb nicht geknackt werden, weil keine schnellen Algorithmen für die betreffenden verwendeten Probleme bekannt sind. Beweisbar sichere kryptographische Verfahren sind nur möglich aufgrund von Resultaten aus der Komplexitätstheorie. Die Theorie beschränkt sich dabei nicht auf Aussagen, daß ein bestimmtes Verschlüsselungssystem schwer zu knacken ist. Es ist vielmehr so, daß zunächst eine Aussage der Art „Problem Π ist nicht effizient lösbar" vorhanden ist. Dieses Problem wird dann zur Konstruktion eines Kryptosystems verwendet, dessen Sicherheit sich aus der Komplexität von Π ergibt (siehe etwa [BSW95, Sti95]). Sollten in Zukunft Quantencomputer gebaut werden könnten, hätte man, wie in Abschnitt 3.1.2. erläutert, ein Mittel zur Hand, Zahlen effizient zu faktorisieren. In diesem Falle böte also der RSA-Algorithmus keine sichere Verschlüsselungsmethode mehr.

Schwierige Probleme können auch verwendet werden, um sogenannte Pseudo-Zufallszahlen-Generatoren [Gol99] zu konstruieren. Dies sind Algorithmen, die Folgen von Zahlen produzieren, die „zufällig aussehen". Diese Generatoren werden in der Kryptographie genutzt. Sie finden jedoch noch eine weitere interessante Anwendung, die wir im folgenden erläutern möchten: In vielen Anwendungsbereichen der Informatik sind effiziente *randomisierte Algorithmen* viel leichter zu konstruieren als effiziente deterministische Verfahren. Oft sind auch nur randomisierte Verfahren bekannt, aber es gibt auch Fälle, in denen randomisierte Algorithmen *de-randomisiert* werden können, also zu deterministischen Lösungswegen führen. Ein Beispiel dafür ist ein paralleler Algorithmus für das Maximal-Independent-Set-Problem [KR90]; der hier durch Derandomisierung gewonnene Algorithmus ist wesentlicher einfacher als alle zuvor bekannten deterministischen Verfahren. Auch der Approximationsalgorithmus für das Euklidische Travelling-Salesperson-Problem, den wir in Abschnitt 3.2.1. erwähnten, entstand durch Derandomisierung. Pseudo-Zufallszahlen-Generatoren können nun benutzt werden, um auf eine systematische Methode randomisierte Algorithmen in deterministische umzuformen. Beispielsweise kann jede randomisierte Berechnung, bei der nur der Speicherplatz die interessante Ressource ist (und nicht die Zeit), derandomisiert werden. So gelangt man zu effizienten deterministischen Algorithmen auf eine Weise, die viel schneller zum Ziel führen kann, als die direkte Konstruktion. Kürzlich wurde sogar gezeigt, daß unter einer nicht unrealistischen schaltkreistheoretischen Hypothese jeder polynomiellzeitbeschränkte randomisierte Algorithmus mit beschränkter Fehlerwahrscheinlichkeit in einen ebenso schnellen deterministischen Algorithmus umgewandelt werden kann [Gol99, Sect. 3.4]. Sollte sich die Hypothese als wahr herausstellen, hätte man also zum Beispiel einen effizienten deterministischen Algorithmus für das Primzahlproblem, für das bislang nur effiziente randomisierte Verfahren bekannt sind.

4. Fazit

In diesem Artikel haben wir gezeigt, daß Ergebnisse der Komplexitätstheorie direkt Auswirkungen auf die praktische Tätigkeit eines Informatikers haben. Die Komplexitätstheorie entwickelt Methoden, die zum Handwerkszeug jedes Software-Ingenieurs gehören sollten.

Darüberhinaus haben wir versucht, deutlich zu machen, wie fundamentale Resultate der Komplexitätstheorie die technische und software-technologische Entwicklung der gesamten Informatik in z. T. revolutionärer Weise beeinflußt haben. Auch weiterhin werden theoretische Entwicklungen aus der Komplexitätstheorie auf den Bereich des industriellen Einsatzes der Informatik einwirken. Derzeit denken wir dabei besonders an Bereiche wie PCP-Theorie, randomisierte Algorithmen oder Quantencomputer.

Danksagung. Wertvolle Impulse zur Entstehung dieser Arbeit erhielt ich aus vielen Diskussionen mit Kollegen, u. a. Christian Glaßer, Frank Puppe, Steffen Reith, Heinz Schmitz und Phuoc Tran-Gia. Insbesondere möchte ich Sven Kosub, Klaus Wagner und Andreas Winter danken.

Literatur

[AHV95] ABITEBOUL, S., R. HULL und V. VIANU: *Foundations of Databases.* Addison-Wesley, Reading, MA, 1995.

[AJK+96] AHO, A. V., D. S. JOHNSON, R. M. KARP, S. R. KOSARAJU, C. C. MCGEOCH, C. H. PAPADIMITRIOUS und P. PEVZNER: *Theory of Computing: Goals and Directions.* Technischer Bericht CSE-96-3-3, University of Washington, 1996. URL: ftp://ftp.cs.washington.edu/tr/1996/03/UW-CSE-96-03-03.ps.gz.

[AL97] ARORA, S. und C. LUND: *Hardness of approximations.* In: HOCHBAUM, D. [Hoc97], Kapitel 10, Seiten 399–446.

[Aro99] ARORA, S.: *Polynomial-time approximation schemes for Euclidian TSP and other geometric problems.* Journal of the ACM, 1999. Erscheint demnächst.

[Beh99] BEHRENDS, E.: *P = NP?* Die Zeit, Seite 43, 4. März 1999.

[BSW95] BEUTELSPACHER, A., J. SCHWENK und K.-D. WOLFENSTETTER: *Moderne Verfahren der Kryptographie – von RSA zu Zero-Knowledge.* Reihe Mathematik. Vieweg, Braunschweig/Wiesbaden, 1995.

[Büt95] BÜTTEMEYER, W.: *Wissenschaftstheorie für Informatiker.* Spektrum Hochschultaschenbuch. Spektrum Akademischer Verlag, Heidelberg Berlin Oxford, 1995.

[CGT90] CERI, S., G. GOTTLOB und L. TANCA: *Logic Programming and Databases.* Surveys in Computer Science. Springer Verlag, Berlin Heidelberg, 1990.

[CK99] CRESCENZI, P. und V. KANN: *A compendium of NP optimization problems.* URL: http://www.nada.kth.se/~viggo/problemlist/compendium.html, 1999.

[CLR90] CORMEN, T. H., C. E. LEISERSON und R. L. RIVEST: *Introduction to Algorithms.* The MIT Electrical Engineering and Computer Science Series. MIT Press, Cambridge, MA, 1990.

[CNP+92] COY, W., F. NAKE, J.-M. PFLÜGER, A. ROLF, J. SEETZEN, D. SIEFKES und R. STRANSFELD (Herausgeber): *Sichtweisen der Informatik.* Reihe Theorie der Informatik. Vieweg, Braunschweig/Wiesbaden, 1992.

[Coo72] COOK, S.: *Linear time simulation of deterministic two-way pushdown automata.* In: *Information Processing 71 – Proc. IFIP World Computer Congress*, Seiten 75–80, Amsterdam, 1972. North Holland.

[Coy92a] COY, W.: *Für eine Theorie der Informatik!* In: COY, W. et al. [CNP+92], Seiten 17–32.

[Coy92b] COY, W.: *Informatik – Eine Disziplin im Umbruch?* In: COY, W. et al. [CNP+92], Seiten 1–9.

[Dav88] DAVIS, M.: *Mathematical logic and the origin of modern computing.* In: HERKEN, R. [Her88], Seiten 149–174.

[DCG+89] DENNING, P. J., D. E. COMER, D. E. GRIES, M. C. MULDER, A. TUCKER, A. J. TURNER und P. R. YOUNG: *Computing as a discipline.* Communications of the ACM, 32:9–23, 1989.

[Dij89] DIJKSTRA, E. W.: *On the cruelty of really teaching computing science.* Communications of the ACM, 32:1398–1404, 1989.

[Flo92] FLOYD, C.: *Human questions in computer science.* In: FLOYD, C., H. ZÜLLIGHOVEN, R. BUDDE und R. KEIL-SLAWIK (Herausgeber): *Software Development and Reality Construction*, Seiten 15–27. Springer, Berlin Heidelberg, 1992.

[Gan88] GANDY, R.: *The confluence of ideas in 1936.* In: HERKEN, R. [Her88], Seiten 55–111.

[Gar95] GARFINKEL, S.: *PGP: Pretty Good Privacy.* O'Reilly & Associates, Sebastopol, CA, 1995.

[GHR95] GREENLAW, R., H. J. HOOVER und W. L. RUZZO: *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, New York, 1995.

[GJ79] GAREY, M. R. und D. S. JOHNSON: *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[Gol99] GOLDREICH, O.: *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Band 17 der Reihe *Algorithms and Combinatorics.* Springer Verlag, Berlin Heidelberg, 1999.

[Gro96] GROVER, L.: *A fast quantum mechanical algorithm for database search.* In: *Proceedings 28th Symposium on Theory of Computing*, Seiten 212–219. ACM Press, 1996.

[Gru99] GRUSKA, J.: *Quantum Computing.* McGraw-Hill, 1999.

[GS93] GIBBONS, A. und P. SPIRAKIS (Herausgeber): *Lectures on Parallel Computation*, Band 4 der Reihe *Cambridge International Series on Parallel Computation.* Cambridge University Press, Cambridge, 1993.

[GW96] GOLDREICH, O. und A. WIGDERSON: *Theory of Computing: A Scientific Perspective.* URL: http://theory.lcs.mit.edu/~oded/toc-sp.html, 1996.

[Her88] HERKEN, R. (Herausgeber): *The Universal Turing Machine – A Half-Century Survey*, Band II der Reihe *Computerkultur.* Kammerer & Unverzagt, Berlin, 1988.

[Hoc97] HOCHBAUM, D. (Herausgeber): *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company, Boston, 1997.

[Hro97] HROMKOVIČ, J.: *Communication Complexity and Parallel Computing.* Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1997.

[HU79] HOPCROFT, J. E. und J. D. ULLMAN: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Series in Computer Science. Addison-Wesley, Reading, MA, 1979.

[Imm99] IMMERMAN, N.: *Descriptive Complexity.* Graduate Texts in Computer Science. Springer Verlag, New York, 1999.

[Jon97] JONES, N. D.: *Computability and Complexity From a Programming Perspective.* Foundations of Computing. MIT Press, Cambridge, MA, 1997.

[Kar89] KARP, R.: *Antwort auf E. W. Dijkstras "On the cruelty of really teaching computing science".* Communications of the ACM, 32:1410–1412, 1989.

[Kol92] KOLATA, G.: *New shortcut found for long math proofs.* New York Times, 7. April 1992.

[Kor97] KORNWACHS, K.: *Um wirklich Informatiker zu sein, genügt es nicht, Informatiker zu sein.* Informatik-Spektrum, 20(2):79–87, 1997.

[KR90] KARP, R. und V. RAMACHANDRAN: *Parallel algorithms for shared-memory machines.* In: LEEUWEN, J. VAN (Herausgeber): *Handbook of Theoretical Computer Science*, Band A, Seiten 869–941. Elsevier, 1990.

[KW99] KULLBACH, B. und A. WINTER: *Querying as an enabling technology in software engineering.* In: *Proceedings of the 3rd Euromicro Conference on Software Maintenance & Reengineering*, Los Alamitos, 1999. IEEE Computer Society Press. Erscheint demnächst.

[Lee59] LEE, C. Y.: *Representation of switching functions by binary decision programs.* Bell Systems Technical Journal, 38:985–999, 1959.

[Lei92] LEIGHTON, F. T.: *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, San Mateo, 1992.

[LSP98] LO, H.-K., T. SPILLER und S. POPESCU (Herausgeber): *Introduction to Quantum Computation and Information.* World Scientific, Singapur, 1998.

[LV93] LI, M. und P. VITÁNYI: *An Introduction to Kolmogorov Complexity and its Applications.* Texts and Monographs in Computer Science. Springer Verlag, New York, 1993.

[Mai79] MAINZER, K.: *Entwicklungsfaktoren der Informatik in der Bundesrepublik Deutschland.* In: DAELE, W. VAN DEN (Herausgeber): *Geplante Forschung – Vergleichende Studien über den Einfluß politischer Programme auf die Wissenschaftsentwicklung*, Band 229 der Reihe *Suhrkamp Taschenbuch Wissenschaft*, Seiten 117–180. Suhrkamp, Frankfurt am Main, 1979.

[Mil93] MILNER, R.: *Elements of interaction.* Communications of the ACM, 36(1):78–89, 1993.

[MPS98] MAYR, E. W., H.-J. PRÖMEL und A. STEGER (Herausgeber): *Lectures on Proof Verification and Approximation Algorithms*, Band 1367 der Reihe *Lecture Notes in Computer Science.* Springer Verlag, Berlin Heidelberg, 1998.

[MT98] MEINEL, C. und T. THEOBALD: *Algorithmen und Datenstrukturen im VLSI-Design: OBDD – Grundlagen und Anwendungen.* Springer Verlag, Berlin Heidelberg, 1998.

[Net99] NETSCAPE: *Security Solutions.* URL: http://home.netscape.com/products/security/index.html, 1999.

[Nyg86] NYGAARD, K.: *Program development as a social activity.* In: KUGLER, H.-G. (Herausgeber): *Information Processing 86 – Proc. 10th IFIP World Computer Congress*, Seiten 189–198, Amsterdam, 1986. North Holland.

[Pap94] PAPADIMITRIOU, C. H.: *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

[Pap95] PAPADIMITRIOU, C. H.: *Vortrag zu Ehren des 60. Geburtstags von Richard Karp, gehalten während eines FOCS Workshops*, 1995.

[Pău98] PĂUN, G.: *Computing with Bio-Molecules.* Springer Series in Discrete Mathematics and Theoretical Computer Science. Springer Verlag, Sinpapore, 1998.

[Pip97] PIPPENGER, N.: *Pure versus impure Lisp.* ACM Transactions on Programming Languages and Systems, 19:223–238, 1997.

[RC94] ROYER, J. S. und J. CASE: *Subrecursive Programming Systems – Complexity & Succinctness.* Progress in Theoretical Computer Science. Birkhäuser, Boston Basel Berlin, 1994.

[Rei89] REISCHUK, R.: *Parallele Maschinenmodelle und Komplexitätsklassen.* Informationstechnik, 31(1):59–77, 1989.

[Sch97] SCHÖNING, U.: *Theoretische Informatik kurz gefasst.* Spektrum Akademischer Verlag, Heidelberg Berlin, 3. Auflage, 1997.

[Sho97] SHOR, P.: *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.* SIAM Journal on Computing, 26:1484–1509, 1997.

[SM97] SETUBAL, J. und J. MEIDANIS: *Introduction to Computational Molecular Biology.* PWS Publishing Company, Boston, 1997.

[Sti95] STINSON, D. R.: *Cryptography – Theory and Practice.* The CRC Press Series on Discrete Mathematics and its Applications. CRC Press, Boca Raton, 1995.

[Weg98] WEGNER, P.: *Interactive foundations of computing.* Theoretical Computer Science, 192:315–351, 1998.

[WW86] WAGNER, K. W. und G. WECHSUNG: *Computational Complexity.* VEB Verlag der Wissenschaften, Berlin, 1986.