Randomisierte Algorithmen

Thomas Worsch
Fakultät für Informatik
Karlsruher Institut für Karlsruhe

Vorlesungsunterlagen
Wintersemester 2019/2020

Vorläufiges Inhaltsverzeichnis

1 Einleitung **5**
- 1.1 Grundsätzliches . . . 5
- 1.2 Einige Schlagwörter . . . 6
- 1.3 Geschichtliches . . . 7
- 1.4 Einige Literaturhinweise . . . 8

2 Erste Beispiele **12**
- 2.1 Ein randomisierter Identitätstest . . . 12
- 2.2 Vergleich von Wörtern . . . 14
- 2.3 Ein randomisierter Quicksortalgorithmus . . . 17

3 Probabilistische Komplexitätsklassen **21**
- 3.1 Probabilistische Turingmaschinen . . . 21
- 3.2 Komplexitätsklassen . . . 22
- 3.3 Beziehungen zwischen Komplexitätsklassen . . . 26

4 Routing in Hyperwürfeln **29**
- 4.1 Das Problem und ein deterministischer Algorithmus . . . 29
- 4.2 Markov- und Chebyshev-Ungleichung . . . 32
- 4.3 Chernoff-Schranken . . . 32
- 4.4 Erster randomisierter Algorithmus . . . 36
- 4.5 Die probabilistische Methode . . . 40
- 4.6 Zweiter randomisierter Algorithmus . . . 40

5 Zwei spieltheoretische Aspekte **44**
- 5.1 Und-Oder-Bäume und ihre deterministische Auswertung . . . 44
- 5.2 Analyse eines randomisierten Algorithmus für die Auswertung von UOB . . . 45
- 5.3 Zwei-Personen-Nullsummen-Spiele . . . 47
- 5.4 Untere Schranken für randomisierte Algorithmen . . . 48

6 Graph-Algorithmen **51**
- 6.1 Minimale Schnitte . . . 51
- 6.2 Minimale spannende Bäume . . . 58
  - 6.2.1 Ein deterministischer Algorithmus für MST . . . 58
  - 6.2.2 F-leichte und F-schwere Kanten . . . 60
  - 6.2.3 Ein randomisierter MSF-Algorithmus . . . 60

7 Random Walks **64**
- 7.1 Ein randomisierter Algorithmus für 2-SAT . . . 64
- 7.2 Random Walks . . . 65
- 7.3 Widerstandsnetzwerke . . . 66
- 7.4 Randomisierte Algorithmen für Zusammenhangstests . . . 69

8 Markov-Ketten **72**
8.1 Grundlegendes zu Markov-Ketten . . . 72
8.2 Irreduzible und ergodische Markov-Ketten . . . 74

9 Metropolis-Hastings-Algorithmus **79**
9.1 Einleitung . . . 79
9.2 Der Metropolis-Hastings-Algorithmus . . . 79
9.3 Simulated Annealing . . . 80

10 Schnell mischende Markov-Ketten **82**
10.1 Anmerkungen zu Eigenwerten . . . 82
10.2 Konvergenzverhalten ergodischer Markov-Ketten . . . 82
10.3 Schnell mischende Markov-Ketten . . . 85

11 Randomisiertes Approximieren **90**
11.1 Polynomielle Approximationsschemata . . . 90
11.2 Zählen von Lösungen von Formeln in DNF . . . 91

12 Online-Algorithmen am Beispiel des Seitenwechselproblems **96**
12.1 Das Seitenwechselproblem und deterministische Algorithmen . . . 96
12.2 Randomisierte Online-Algorithmen und Widersacher . . . 98
12.3 Seitenwechsel gegen einen unwissenden Widersacher . . . 99
12.4 Einschub: Amortisierte Analyse . . . 103
12.5 Seitenwechsel gegen adaptive Widersacher . . . 104

13 Hashing **111**
13.1 Grundlagen . . . 111
13.2 Universelle Familien von Hashfunktionen . . . 111
13.3 Zwei 2-universelle Familien von Hashfunktionen . . . 113
13.4 Das dynamische Wörterbuchproblem . . . 114
13.5 Das statische Wörterbuchproblem . . . 116
13.6 Derandomisierung . . . 117

14 Pseudo-Zufallszahlen **122**
14.1 Allgemeines . . . 122
14.2 Generatoren für Pseudo-Zufallszahlen . . . 123
14.2.1 Middle-square- und Muddle-square-Generator . . . 124
14.2.2 Lineare Kongruenzgeneratoren . . . 124
14.2.3 Mehrfach rekursive Generatoren . . . 125
14.2.4 Inverse Kongruenzgeneratoren . . . 125
14.2.5 Mersenne-Twister . . . 126
14.2.6 Zellularautomaten als Generatoren . . . 126
14.2.7 Kombination mehrerer Generatoren . . . 127
14.3 Tests für Pseudo-Zufallszahlen . . . 128
14.3.1 Grundsätzliches: χ^2- und KS-Test . . . 128
14.3.2 Einfache empirische Tests . . . 129
14.3.3 Weitere Tests . . . 131

A Mathematische Grundlagen **135**
A.1 Allgemeines 135
A.2 Zufallsvariablen 136
A.3 Kleinigkeiten 138
A.3.1 Exponentialfunktion 138

1 Einleitung

1.1 Grundsätzliches

1.1 *Randomisierte* (oder auch *probabilistische*) *Algorithmen* sind Rechenverfahren, bei denen man den klassischen Algorithmenbegriff aufweicht.

Eine mögliche Sichtweise ist die, dass als ein möglicher algorithmischer Elementarschritt ein *zufälliger Wert* „irgendwie“ zur Verfügung gestellt wird.

Eine andere Sichtweise ist die, dass nicht mehr verlangt wird, dass stets *ein*deutig festgelegt ist, welches der als nächstes auszuführende Elementarschritt ist. Im Unterschied zu nichtdeterministischen Verfahren ist aber quantifiziert, welche Fortsetzung mit welcher Wahrscheinlichkeit gewählt wird.

Eine dritte Sichtweise ist die eines deterministischen Algorithmus, der aber neben der „eigentlichen“ Eingabe auch noch als „uneigentliche“ Eingabe einen Zufallswert (etwa eine Reihe von Zufallsbits) bekommt. Auf diese Weise wird sozusagen aus einer Vielzahl von deterministischen Algorithmen jeweils einer ausgewählt.

1.2 Allen diesen Sichtweisen gemeinsam ist die Tatsache, dass sich bei mehreren Ausführungen des randomisierten Algorithmus *für die gleiche Eingabe*[1] *verschiedene Berechnungen* ergeben können.

1.3 Hieraus folgt zum Beispiel, dass im allgemeinen selbst für eine einzelne festgehaltene Eingabe im allgemeinen *das berechnete Ergebnis, die Laufzeit und der Speicherplatzbedarf nicht präzise anzugeben* sind. Vielmehr handelt es sich dabei um *Zufallsvariablen*.

Man wird dann etwa daran interessiert sein, die Erwartungswerte dieser Größen herauszufinden. Unter Umständen sind auch weitergehende Aussagen interessant, zum Beispiel die, dass (in einem später noch zu definierenden mathematisch präzisen Sinne) „mit hoher Wahrscheinlichkeit“ Abweichungen vom Erwartungswert „sehr klein“ sind. Die Erwartungswerte etc. möchte man wie üblich in Abhängigkeit nur von der Größe der Eingabe bestimmen.

1.4 Man geht sogar so weit, von einem randomisierten Algorithmus für ein Problem zu sprechen, wenn die Ausgaben manchmal, aber eben „hinreichend selten“ im Sinne der Problemspezifikation *falsch* sind.

Je nach bisher genossener Informatikausbildung mag dies im ersten Moment äußerst befremdlich erscheinen. Ohne jeden Zweifel kommt hier ein (im Vergleich zu deterministischen Algorithmen) neuer Standpunkt ins Spiel. Dass der aber nicht völlig aus der Luft gegriffen ist, mag durch die folgende Überlegung verdeutlicht werden (Niedermeier 1997):

Nimmt man an, dass jedes Jahrtausend mindestens einmal durch einen Meteoriteneinschlag auf der Erde eine Fläche von 100 m^2 verwüstet wird, dann wird ein Rechner während einer Mikrosekunde seiner Arbeit mit einer Wahrscheinlichkeit von mindestens 2^{-100} zerstört. Andererseits gibt es randomisierte Algorithmen, die auf einem Rechner in Sekundenschnelle die

[1] Damit sind hier natürlich „eigentliche“ Eingaben gemeint.

Ausgabe produzieren: „Die größte Primzahl kleiner als 2^{400} ist $2^{400} - 593$.“. Und wenn diese Aussage geliefert wird, ist sie nur mit Wahrscheinlichkeit 2^{-100} falsch.

1.5 Unter Umständen etwas weniger irritierend ist die Situation bei *Optimierungsalgorithmen*. Hier ist es häufig so, dass die Berechnung von Lösungen, die in einem gewissen Sinne optimal sind, (zumindest mit allen bekannten Verfahren) zu lange dauern würde, und man daher mit Algorithmen zufrieden ist, die zwar suboptimale aber immerhin noch akzeptable Lösungen liefern.

Auch hierzu können randomisierte Algorithmen eingesetzt werden. Eine Frage, die dann vermutlich eine Rolle spielt, ist, ob man eventuell zeigen kann, dass Lösungen einer gewissen Qualität mit einer gewissen Wahrscheinlichkeit gefunden werden.

1.6 Weshalb ist man an randomisierten Algorithmen interessiert? Manchmal gibt es randomisierte Algorithmen zur Lösung eines Problemes, die genauso gut sind wie die besten deterministischen, aber weitaus leichter zu formulieren und implementieren. Manchmal kennt man randomisierte Algorithmen, die sogar „besser“ sind als die besten deterministischen Algorithmen. In einigen Fällen ist es sogar so, dass es für ein Problem zwar keinen deterministischen Algorithmus gibt, der es löst, aber einen randomisierten, der (in gewissem Sinne) das Gewünschte leistet.

Es sei aber noch einmal erwähnt, dass man dafür häufig einen Preis bezahlt, dass nämlich Ausgaben falsch sein können. Mit anderen Worten wird der Begriff der Lösung eines Problems unter Umständen nicht mehr mit der gleichen Strenge wie bei deterministischen Algorithmen benutzt. Es gibt aber auch randomisierte Algorithmen, die immer das richtige Ergebnis liefern.

1.7 Man beachte, dass bei randomisierten Algorithmen zum Beispiel eine Aussage über den Erwartungswert der Laufzeit *für alle Eingaben* korrekt zu sein hat.

Dies ist im Unterschied etwa zur *probabilistischen Analyse deterministischer Algorithmen* zu sehen. Dort wird eine gewisse Wahrscheinlichkeitsverteilung für die Eingaben zu Grunde gelegt, und zum Beispiel daraus und aus den Laufzeiten für die einzelnen Eingaben ein (anderer!) Erwartungswert für die Laufzeiten ermittelt („average-case complexity“).

1.2 Einige Schlagwörter

Wie kann „Zufall“ vorteilhaft in einem randomisierten Algorithmus eingesetzt werden? Im Laufe der Zeit hat sich Reihe von „Standardmethoden“ gefunden, um Zufallsbits anzuwenden. Einige der in der folgenden Liste aufgeführten werden im Laufe der Vorlesung noch in Beispielalgorithmen auftreten und analysiert werden. Andere seien hier der Vollständigkeit halber aus der Arbeit von Karp (1991) mit aufgeführt:

Fingerabdrücke: Hierunter hat man sich kompakte „Repräsentationen“ großer Daten vorzustellen. Durch Benutzung einer zufälligen Komponente bei der Berechnung kann man erreichen, dass (in einem gewissen Rahmen) identische Fingerabdrücke tatsächlich auf identische Ausgangsdaten hinweisen.

Zufälliges Auswählen und Umordnen: Durch die Benutzung zufällig ausgewählter Teilmengen, zufälliger Umordnungen, o.ä. kann man mitunter mit großer Wahrscheinlichkeit Eingaben, die im deterministischen Fall zu schlechten Ausreißern z. B. bei der Laufzeit führen, „unschädlich machen“.

Überfluß an Zeugen: Mitunter gibt es für eine gewisse Eigenschaft von Eingabedaten „Zeugen“. Mit einem deterministischen Algorithmus einen von ihnen zu finden ist unter Umständen sehr schwer, obwohl es „sehr viele“ gibt. Deshalb kann die zufällige Wahl irgendeines Wertes schnell einen Zeugen liefern. Wenn dann auch noch die Zeugeneigenschaft leicht nachzuweisen ist ...

Vereitelung gegnerischer Angriffe: Im Zusammenhang mit der Komplexität eines Problems kann man den Wert eines Zwei-Personen-Nullsummen-Spieles des Algorithmenentwerfers gegen einen „Bösewicht“ betrachten, der versucht, durch die Auswahl „schlechter“ Eingaben ineffiziente Berechnungen des Algorithmus zu erzwingen. Durch zufällige Entscheidungen zwischen verschiedenen Algorithmen (siehe die dritte Sichtweise in Punkt 1.1) hat es der Bösewicht unter Umständen schwerer.

Lastbalancierung: In parallelen Algorithmen kann randomisierte Lastverteilung gute Dienste leisten.

Schnell mischende Markovketten: Bei der Konstruktion eines „zufälligen Objektes“ muß manchmal sichergestellt werden, dass man schnell genug eines mit den gewünschten Eigenschaften findet. Hierzu bedient man sich der Theorie der Markovketten und zeigt, dass sie die dem Algorithmus zu Grunde liegende „schöne Eigenschaften“ hat und schnell gegen die stationäre Verteilung konvergiert.

Isolierung und Symmetriebrechung: Wenn in einer verteilten Umgebung alle nach dem gleichen Algorithmus vorgehen und nicht durch eindeutige Namen bzw. Nummern unterschieden sind, sich aber trotzdem Unterschiede ergeben sollen, läßt sich dies z. B. durch Randomisierung erreichen.

Weitere Aspekte, die im Zusammenhang mit randomisierten Algorithmen eine Rolle spielen und auf die wir zum Teil auch eingehen werden, sind unter anderem:

probabilistische Methode: Das ist ein Werkzeug, um die Existenz gewisser Objekte nachzuweisen, indem man zeigt, dass die Wahrscheinlichkeit, bei einem bestimmten randomisierten Algorithmus auf ein Objekt der gewünschten Art zu stoßen, echt größer Null ist.

randomisierte Komplexitätsklassen: So wie im Nicht-/Deterministischen Komplexitätsklassen wie **P** oder **NP** eine wichtige Rolle spielen, haben sich für komplexitätstheoretische Untersuchungen randomisierter Algorithmen unter anderem die Modelle probabilistischer Turingmaschinen und Schaltkreise bewährt und mit **BPP**, **ZPP**, **RP** oder **RNC** bezeichnete Komplexitätsklassen Bedeutung erlangt.

Zufall als Berechnungsressource: Woher bekommt man Zufallsbits? Sind Zufallsbits eigentlich „umsonst“?

Elimination von Zufall: Falls nicht, gibt es Möglichkeiten, ihren Gebrauch einzuschränken, ohne algorithmische Nachteile in Kauf nehmen zu müssen?

1.3 Geschichtliches

Wie so häufig gibt es nicht genau einen exakt bestimmbaren Punkt, an dem zum ersten Mal das Konzept eines randomisierten Algorithmus eingeführt und verwendet wurde.

Shallit 1992 hat die Informatik darauf hingewiesen, dass in wenigstens zwei Kulturen, die als „primitiv“ zu bezeichnen man sich immer noch gelegentlich herablässt, in Verfahren zur Bestimmung eines bestimmten zukünftigen Verhaltens (z. B. „Wo soll gejagt werden?“) Ansätze vorhanden sind, denen zumindest eine Verwandtschaft zu randomisierten Algorithmen und den dort angewendeten Methoden nicht abgesprochen werden kann.

Als erstes Beispiel eines randomisierten Algorithmus betrachten wir im Folgenden ein Verfahren des zentralafrikanischen Stammes der Azande. Sie befragen ein „Orakel“, indem einem Huhn eine giftige rote Paste verabreicht wird. Üblicherweise wird das Huhn dann eine Zeit lang an heftigen Krämpfen leiden. Manchmal wird es sogar sterben, manchmal aber auch überhaupt keine Reaktion zeigen.

Die Tatsache, dass ein Huhn überlebt bzw. nicht, wird als Antwort des Orakels auf eine vorher gestellte Frage interpretiert. Man geht davon aus, dass das Orakel nie irrt, sofern man es korrekt befragt, d. h. die richtige Menge des Giftes verabreicht hat. Es kann nicht ausgeschlossen werden, dass einmal „aus Versehen“ eine zu kleine oder zu große Menge Giftes benutzt wurde. Jede Orakelbefragung beinhaltet also sozusagen ein zufälliges Element. Um damit fertig zu werden, wenden die Azande folgendes Verfahren an:

> Sie formulieren die Anfrage zweimal. Einmal so, dass die Bestätigung durch das Orakel durch den Tod eines Huhns zum Ausdruck käme, und einmal so, dass die Bestätigung durch das Orakel durch das Überleben eines Huhns zum Ausdruck käme. Nur wenn bei den beiden Orakelbefragungen in genau einem Fall ein Huhn stirbt, wird die Antwort als gültig akzeptiert. Sterben beide oder kein Huhn, wird die Antwort des Orakels verworfen.

Da die Hühner eine wertvolle Ressource für den Stamm darstellen, wendet er darüberhinaus Methoden an, um damit sparsam umzugehen. So wird z. B. auf die zweite Befragung verzichtet, wenn das Huhn bei der ersten sehr schnell stirbt. Manchmal werden zunächst die ersten Orakelantworten für zwei verschiedene Fragen ermittelt. Falls bei beiden ein Huhn stirbt, wird nur ein (drittes) Huhn benutzt, um beide bestätigen zu lassen. Schließlich merkt Moore 1957 in einer Fußnote einer Arbeit zu diesem Thema an: „Incidentally, their manner of framing questions—they use complex conditionals—so as to obtain as many definitive answers while sacrificing as few fowls as possible, would do credit to a logician.“ Was damit genau gemeint ist, ist unklar. Shallit (1992) mutmaßt, dabei könne es sich um so etwas wie binäre Suche handeln, indem z. B. drei Hühner geopfert werden, um eine von acht Möglichkeiten auszuwählen.

Die anscheinend erste Arbeit über probabilistische Maschinen stammt von Leeuw u. a. 1955. Weitere grundlegende automatentheoretische Arbeiten sind die von Rabin (1963) über probabilistische endliche Automaten und die von Gill (1977) über probabilistische Turingmaschinen. Auf dieses Modell werden wir im Zusammenhang mit komplexitätstheoretischen Betrachtungen zurückkommen.

Einige der ersten randomisierten Algorithmen überprüfen (in einem gewissen Sinne) zahlentheoretische Eigenschaften. Solovay und Strassen (1977), Solovay und Strassen (1978) und Rabin (1976) haben Verfahren angegeben, die eine eingegebene Zahl auf Zusammengesetztheit bzw. Primheit überprüfen. Wir werden später darauf zurückkommen.

1.4 Einige Literaturhinweise

Das Standardbuch über randomisierte Algorithmen ist

- Rajeev Motwani und Prabhakar Raghavan (1995). *Randomized Algorithms*. Cambridge University Press. ISBN: 0-521-47465-5.

Außerdem sind die folgenden Bücher zumindest für einen Teil der Vorlesung relevant und empfehlenswert:

- Allan Borodin und Ran El-Yaniv (1998). *Online Computation and Competitive Analysis*. Cambridge Unviersity Press. ISBN: 0-521-56392-5.
- Juraj Hromkovič (2004). *Randomisierte Algorithmen: Methoden zum Entwurf von zufallsgesteuerten Systemen für Einsteiger*. Teubner-Verlag. ISBN: 3-519-00470-4.

Abschnitte bzw. Kapitel über komplexitätstheoretische Aspekte finden sich zum Beispiel auch in den folgenden Büchern:

- Giorgio Ausiello u. a. (1999). *Complexity and Approximation*. Berlin: Springer. ISBN: 3-540-65431-3.
- Daniel Pierre Bovet und Pierluigi Crescenzi (1994). *Introduction to the Theory of Complexity*. New York: Prentice Hall.
- Jozef Gruska (1997). *Foundations of Computing*. International Thomson Computer Press.
- Christos H. Papadimitriou (1994). *Computational Complexity*. Addison-Wesley.

Um erste Eindrücke von dem Gebiet zu bekommen, sind die folgenden Übersichtsartikel hilfreich:

- Richard M. Karp (1991). „An introduction to randomized algorithms“. In: *Discrete Applied Mathematics* 34, S. 165–201.
- Rajiv Gupta, Scott A. Smolka und Shaji Bhaskar (1994). „On Randomization in Sequential and Distributed Algorithms“. In: *ACM Computing Surveys* 26.1, S. 7–86.
- Rajeev Motwani und Prabhakar Raghavan (1997). „Randomized Algorithms“. In: *The Computer Science and Engineering Handbook*. Hrsg. von A. B. Tucker Jr. CRC Press. Kap. 7, S. 141–161.

Schließlich finden sich im WWW eine Reihe von Vorlesungsskripten über randomisierte Algorithmen. Zum Beispiel:

- Josep Diaz (1997). *Probabilistic Algorithms*. Im WWW zugänglich unter `http://www.lsi.upc.es/~diaz/ralcom.ps.gz`.
- Michel X. Goemans (1994). *Randomized Algorithms*. Im WWW zugänglich unter `ftp://ftp.theory.lcs.mit.edu/pub/classes/18.415/notes-random.ps`.
- Seffi Naor (1993). *Probabilistic Methods in Computer Science*. Im WWW zugänglich unter `http://www.uni-paderborn.de/fachbereich/AG/agmadh/Scripts/GENERAL/naor.notes.ps.gz`.
- Rolf Niedermeier (1997). *Randomisierte Algorithmen*. Im WWW zugänglich unter `http://www-fs.informatik.uni-tuebingen.de/~niedermr/teaching/ra-script.ps.Z`.

Einige dieser Skripte sind auch über `http://www.uni-paderborn.de/fachbereich/AG/agmadh/WWW/scripts.html` zugänglich und als Kopie von dort schneller zu erhalten.

Literatur

Ausiello, Giorgio u. a. (1999). *Complexity and Approximation*. Berlin: Springer. ISBN: 3-540-65431-3 (siehe S. 9).

Borodin, Allan und Ran El-Yaniv (1998). *Online Computation and Competitive Analysis*. Cambridge Unviersity Press. ISBN: 0-521-56392-5 (siehe S. 9).

Bovet, Daniel Pierre und Pierluigi Crescenzi (1994). *Introduction to the Theory of Complexity*. New York: Prentice Hall (siehe S. 9).

Diaz, Josep (1997). *Probabilistic Algorithms*. Im WWW zugänglich unter `http://www.lsi.upc.es/~diaz/ralcom.ps.gz` (siehe S. 9).

Gill, John (1977). „Computational complexity of probabilistic Turing machines“. In: *SIAM Journal on Computing* 6.4, S. 675–695 (siehe S. 8).

Goemans, Michel X. (1994). *Randomized Algorithms*. Im WWW zugänglich unter `ftp://ftp.theory.lcs.mit.edu/pub/classes/18.415/notes-random.ps` (siehe S. 9).

Gruska, Jozef (1997). *Foundations of Computing*. International Thomson Computer Press (siehe S. 9).

Gupta, Rajiv, Scott A. Smolka und Shaji Bhaskar (1994). „On Randomization in Sequential and Distributed Algorithms“. In: *ACM Computing Surveys* 26.1, S. 7–86 (siehe S. 9).

Hromkovič, Juraj (2004). *Randomisierte Algorithmen: Methoden zum Entwurf von zufallsgesteuerten Systemen für Einsteiger*. Teubner-Verlag. ISBN: 3-519-00470-4 (siehe S. 9).

Karp, Richard M. (1991). „An introduction to randomized algorithms“. In: *Discrete Applied Mathematics* 34, S. 165–201 (siehe S. 6, 9).

Leeuw, K. de u. a. (1955). „Computability by probabilistic machines“. In: *Automata Studies*. Hrsg. von C. E. Shannon und J. McCarthy. Princeton Unviersity Press, S. 183–212 (siehe S. 8).

Moore, Omar Khayyam (1957). „Divination—a new perspective“. In: *American Anthropologist* 59, S. 69–74 (siehe S. 8).

Motwani, Rajeev und Prabhakar Raghavan (1995). *Randomized Algorithms*. Cambridge University Press. ISBN: 0-521-47465-5 (siehe S. 9).

— (1997). „Randomized Algorithms“. In: *The Computer Science and Engineering Handbook*. Hrsg. von A. B. Tucker Jr. CRC Press. Kap. 7, S. 141–161 (siehe S. 9).

Naor, Seffi (1993). *Probabilistic Methods in Computer Science*. Im WWW zugänglich unter `http://www.uni-paderborn.de/fachbereich/AG/agmadh/Scripts/GENERAL/naor.notes.ps.gz` (siehe S. 9).

Niedermeier, Rolf (1997). *Randomisierte Algorithmen*. Im WWW zugänglich unter `http://www-fs.informatik.uni-tuebingen.de/~niedermr/teaching/ra-script.ps.Z` (siehe S. 5, 9).

Papadimitriou, Christos H. (1994). *Computational Complexity*. Addison-Wesley (siehe S. 9).

Rabin, Michael O. (1963). „Probabilistic Automata“. In: *Information and Control* 6, S. 230–245 (siehe S. 8).

— (1976). „Probabilistic Algorithms“. In: *Algorithms and Complexity*. Hrsg. von J. F. Traub. Academic Press, S. 21–39 (siehe S. 8).

Shallit, Jeffrey (1992). „Randomized Algorithms in “Primitive” Cultures or What is the Oracle Complexity of a Dead Chicken“. In: *ACM SIGACT News* 23.4, S. 77–80 (siehe S. 8).

Solovay, R. und V. Strassen (1977). „A Fast Monte-Carlo Test for Primality“. In: *SIAM Journal on Computing* 6.1, S. 84–85 (siehe S. 8).

Solovay, R. und V. Strassen (1978). „Erratum: A Fast Monte-Carlo Test for Primality“. In: *SIAM Journal on Computing* 7.1, S. 118 (siehe S. 8).

