

Institut für Theoretische Informatik: Theoretische Informatik und das Problem des Handlungsreisenden

Bodo Manthey Jan Arpe Andreas Jakoby Rüdiger Reischuk

Universität zu Lübeck, Institut für Theoretische Informatik
Ratzeburger Allee 160, 23538 Lübeck, Germany
{manthey, arpe, jakoby, reischuk}@tcs.uni-luebeck.de

1 Das Institut für Theoretische Informatik

Eine zentrale Aufgabe der Theoretischen Informatik ist die Untersuchung von algorithmischen Problemen mit Hilfe mathematischer Modelle. Die Ergebnisse sollen allgemein gültig sein, also möglichst unabhängig von speziellen Systemeigenschaften, wie z. B. der Taktrate oder der Speichergröße. Damit bildet die Theoretische Informatik gerade in Zeiten rasanter technologischer Entwicklungen ein stabiles Fundament für die Informatik.

Das Institut für Theoretische Informatik wurde im Jahr 1994 durch die Berufung von Professor Dr. Rüdiger Reischuk gegründet. Es wurde durch eine weitere Professur verstärkt, die Professor Dr. Thomas Zeugmann von April 2000 bis Juni 2004 wahrgenommen hat und die momentan zur Neubesetzung ansteht. Die Forschungsschwerpunkte des Instituts lassen sich in fünf Bereiche einteilen: Algorithmik, Komplexitätstheorie, Algorithmisches Lernen, Kryptographie sowie Parallele und Verteilte Systeme. Wir beschreiben im Folgenden kurz die fünf Schwerpunkte. Weitere Informationen können auf unserer Homepage „<http://www.tcs.uni-luebeck.de/>“ gefunden werden.

Algorithmik Aufgabe der Algorithmik ist die Entwicklung und Untersuchung effizienter Algorithmen. Wir untersuchen zum Einen klassische Fragestellungen aus der Graphentheorie (z. B. Erreichbarkeitsprobleme), zum Anderen aktuelle Probleme aus der Bioinformatik (z. B. Sequenzprobleme). Hierzu entwickeln wir Approximationsverfahren und effiziente parallele Algorithmen [5, 8, 10].

Komplexitätstheorie Die Komplexitätstheorie beschäftigt sich mit der Frage, wie viel Ressourcen, wie z. B. Rechenzeit oder Speicherplatz, notwendig sind, um ein bestimmtes Problem zu lösen. Ziel sind einerseits positive Aussagen, dass ein Problem schnell oder mit wenig Speicherplatz gelöst werden kann (hier überlappen sich Algorithmik und Komplexitätstheorie). Andererseits sind aber auch negative Aussagen von Interesse, etwa in der Form, dass jeder Algorithmus zur Lösung eines Problems mindestens eine bestimmte Zeit lang rechnen muss oder eine bestimmte Menge an Speicherplatz benötigt. Die Komplexitätstheorie gewinnt zunehmend in Anwendungsgebieten an Bedeutung. So

interessiert uns in der Bioinformatik neben der Algorithmik auch die Komplexität von Problemen [11].

Neben der Zeit- und der Platzkomplexität untersuchen wir die Kommunikationskomplexität, d. h. die Anzahl der Nachrichten, die Computer in einem Netzwerk austauschen müssen, um ein Problem zu lösen. Derartige Fragen tauchen bei der Entwicklung von parallelen Algorithmen und in großen Netzwerken wie dem Internet auf [1].

Algorithmisches Lernen Die Algorithmische Lerntheorie bemüht sich um eine formale Definition des Begriffs „Lernen“. Zum Einen untersuchen wir, was ein Computer lernen kann. Zum Anderen entwickeln und analysieren wir effiziente Lernverfahren, die das Lernziel anhand von nur wenigen Beispielen erreichen [2, 3, 7].

Kryptographie Kryptographie beschäftigt sich mit dem Verschlüsseln, Entschlüsseln und der Geheimhaltung von Daten. In diesem Bereich beschäftigen wir uns aktuell mit folgendem Problem: Mehrere Teilnehmer haben jeweils private Daten, die sie keinem anderen mitteilen möchten (z. B. ihr Gehalt). Nun soll von den Teilnehmern ein Wert berechnet werden, der von diesen Daten abhängt (z. B. das Durchschnittseinkommen aller Teilnehmer). Allerdings möchte kein Teilnehmer, dass dabei seine privaten Daten bekannt werden. Wir untersuchen, welche Funktionen in welchen Netzwerken berechnet werden können und wie viele Ressourcen (z. B. Zufallsbits) dazu benötigt werden [4, 9].

Parallele und Verteilte Systeme In verteilten Systemen muss man Aktivitäten verschiedener unabhängiger Komponenten synchronisieren und koordinieren. Z. B. möchte man nicht, dass verschiedene Rechner eines Netzes gleichzeitig auf den Drucker zugreifen. Wir erforschen, welche Synchronisations- und Koordinationsaufgaben in welchen Arten verteilter Systeme grundsätzlich lösbar sind bzw. welche Aufgaben spezielle Hilfsmittel, wie z. B. Randomisierung, zur Lösung erfordern. Dabei interessiert uns vor allem, welche Aufgaben selbst dann gelöst werden können, wenn einige Komponenten ausfallen oder sich sonst irgendwie fehlerhaft verhalten können [12, 13].

2 Das Problem des Handlungsreisenden

Wir diskutieren nun exemplarisch das Problem des Handlungsreisenden (*Traveling Salesman Problem*, TSP), um Konzepte, Methoden und Ergebnisse der Theoretischen Informatik zu erläutern.

Ein Geschäftsmann plant eine Reise, die von seinem Heimatort über verschiedene vorgegebene Städte zurück in seinen Heimatort führen soll. Sein Ziel ist, die Länge dieser Rundreise möglichst kurz zu halten.

2.1 Anwendungen

Das TSP taucht in der Praxis in verschiedensten Anwendungen auf. Als naheliegendes Beispiel sei hier die Routenplanung bei Speditionsunternehmen genannt. Aber auch in anderen Bereichen versteckt sich das TSP. So möchte man beim Verdrahten von Mikrochips möglichst wenig Kabel verwenden, um ein Übersprechen zu vermeiden. Ein weiteres

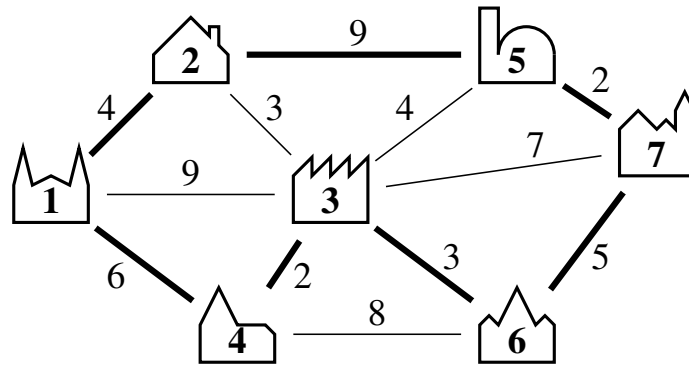


Abbildung 1: Ein Landkarte mit sieben Städten und eine kürzeste Rundreise. Nicht alle Distanzen sind eingezeichnet, um die Karte übersichtlich zu halten.

Beispiel stammt aus der Bioinformatik: Beim *Shotgun Sequencing* werden DNA-Stränge in kurze überlappende Sequenzen aufgeteilt, die dann sequenziert werden. Man kennt aber die Reihenfolge nicht, in der die Teilstücke in der Originalsequenz vorkommen. Daher berechnet man einen so genannten *Shortest Common Superstring* der sequenzierten Teilstücke.

2.2 Abstraktion

Um das TSP mit dem Computer zu lösen, müssen wir es zunächst formalisieren. Die n Städte bezeichnen wir mit 1 bis n , wobei Stadt 1 die Heimatstadt des Handlungsreisenden ist. Die Distanz von der Stadt i zur Stadt j bezeichnen wir mit $D(i, j)$. Ein Beispiel ist in Abb. 1 zu sehen. Die Distanzen $D(i, j)$ können Verschiedenes repräsentieren. Es kann sich um die wirkliche Entfernung handeln (dann suchen wir die kürzeste Rundreise), aber auch z. B. die Reisezeit oder die Reisekosten sein (dann würden wir die schnellste bzw. die billigste Rundreise suchen).

Ziel ist es stets, eine Reihenfolge der Städte zu finden, so dass die Summe der Distanzen aufeinanderfolgender Städte möglichst gering ist. In Abb. 1 ist eine kürzeste Rundreise dick gedruckt.

2.3 Einfache und schwere Probleme

Wir wollen die Schwierigkeit anhand der Zeit messen, die zur Lösung eines Problems benötigt wird. In diesem Sinne ist TSP ein schweres Problem.

Betrachten wir zunächst ein einfaches Problem, das Navigationssysteme und Fahrplanauskünfte täglich lösen müssen: Wir wollen den kürzesten Weg von einer Stadt zu einer anderen auf einer Landkarte mit n Städten berechnen. In Abb. 2 ist ein kürzester Weg von Stadt 1 nach Stadt 7 dick gedruckt.

Alle Wege auszuprobieren, würde eine Laufzeit proportional zu $n!$ erfordern ($n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$). Man kann das Problem jedoch wesentlich schneller lösen, nämlich mit einer Anzahl von Rechenschritten, die proportional zu n^2 ist. Für $n = 12$ ist $n^2 = 144$ und $n! = 479\,001\,600$.

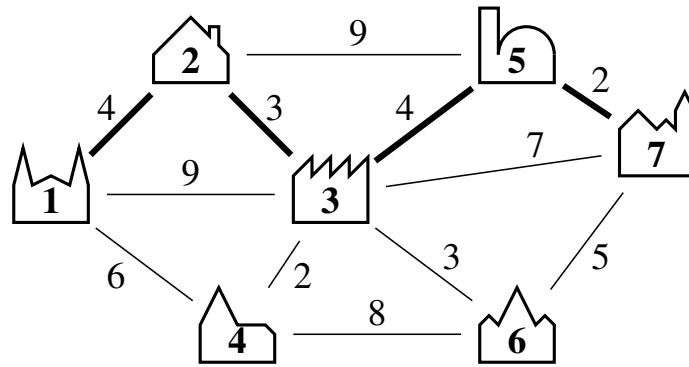


Abbildung 2: Ein kürzester Weg von Stadt 1 nach Stadt 7.

Die Klasse aller Probleme, die in *polynomieller Zeit*, also mit Laufzeit proportional zu n^k , gelöst werden können, wird mit \mathcal{P} bezeichnet. Das Problem, kürzeste Wege zu Berechnen, ist in \mathcal{P} , da es in n^2 Schritten gelöst werden kann. Viele bekannte Probleme gehören zu \mathcal{P} : Die Grundrechenarten, Sortieren, Lösen von Gleichungssystemen und viele mehr.

Man vermutet aber, dass einem für das TSP kaum etwas anderes übrig bleibt, als alle Möglichkeiten auszuprobieren; wir sagen „TSP ist \mathcal{NP} -vollständig“ (\mathcal{NP} steht für *nichtdeterministisch-polynomielle Zeit*). \mathcal{NP} -vollständige Probleme haben besondere Eigenschaften: Die Frage, ob es eine Rundreise von höchstens einer gewissen Länge gibt, ist zwar schwer zu beantworten. Ist jedoch eine konkrete Rundreise gegeben, so kann man sich sehr schnell davon überzeugen, ob sie nicht zu lang ist. Das *Finden* von Lösungen ist schwer, das *Überprüfen* jedoch sehr einfach. Viele Probleme sind \mathcal{NP} -vollständig. Von keinem dieser Probleme weiß man, ob es zu \mathcal{P} gehört oder nicht. Allerdings weiß man Folgendes: Gelänge es, für irgendein \mathcal{NP} -vollständiges Problem einen effizienten Algorithmus anzugeben, so ließen sich alle \mathcal{NP} -vollständigen Probleme effizient lösen (würden also zu \mathcal{P} gehören). Man vermutet jedoch, dass kein \mathcal{NP} -vollständiges Problem zu \mathcal{P} gehört. Dann bleibt einem kaum etwas anderes übrig, als alle Reiserouten durchzuprobieren. Das kann aber sehr lange dauern. Wenn der Handlungsreisende alle 25 EU-Hauptstädte besuchen möchte, dann hat er 15 511 210 043 330 985 984 000 000 Reiserouten zur Auswahl! Wenn er auf einem Supercomputer eine Million Touren pro Sekunde ausprobieren könnte und eine Million solcher Computer zur Verfügung hätte, dann würde er trotzdem ungefähr 491 857 Jahre auf die Antwort warten müssen. Bei 30 Städten würde es sogar wesentlich länger dauern, als das Universum alt ist. Dagegen ist die Berechnung kürzester Wege auch für 10 000 und mehr Städte auf einem handelsüblichen PC in ein paar Sekunden möglich.

2.4 Auswege

Möchte man z. B. Shotgun Sequencing durchführen (was erfolgreich gemacht wird), dann muss man das entsprechende TSP lösen. Wie wir gesehen haben, kann dies aber sehr lange dauern. Im Folgenden wollen wir Verfahren betrachten, die die Schwierigkeit des Problems umgehen.

2.4.1 Approximationsalgorithmen

Im Allgemeinen ist es sehr schwer, eine kürzeste Rundreise zu berechnen. Oftmals genügt es jedoch, eine Rundreise zu berechnen, die nicht allzu viel länger als die kürzeste ist. Diese Überlegung führt zu so genannten *Approximationsalgorithmen*; das sind Algorithmen, die im Allgemeinen nicht die kürzeste Rundreise finden, dafür aber schnell sind (schnell heißt hier polynomielle Laufzeit). Eine *c-Approximation* ist ein Algorithmus, der eine Rundreise berechnet, die höchstens c -mal so lang ist wie die kürzeste Tour. Zunächst eine schlechte Nachricht: Im Allgemeinen ist es kaum möglich, eine gute Approximation effizient zu berechnen.

Aber es gibt auch gute Nachrichten: Schränken wir das Problem ein wenig ein, dann gibt es gute Approximationsalgorithmen. Die zwei wichtigsten Einschränkungen sind *Symmetrie* und *Dreiecksungleichung*.

Symmetrie heißt, dass der Hinweg von Stadt i nach Stadt j genau so weit ist wie der Rückweg, also $D(i, j) = D(j, i)$. Es gibt viele Fälle, in denen die Distanzen symmetrisch sind, dies ist aber nicht immer der Fall. (Man denke an Lübeck und seine Einbahnstraßen.)

Die Dreiecksungleichung besagt, dass der direkte Weg immer der kürzeste ist, also $D(i, j) \leq D(i, k) + D(k, j)$. Für Entfernungen ist die Dreiecksungleichung erfüllt, für Flugpreise jedoch nicht unbedingt.

Sind die Distanzen symmetrisch und gilt die Dreiecksungleichung, so lässt sich immer eine $\frac{3}{2}$ -Approximation berechnen, also eine Tour, die höchstens 50% länger ist als die kürzeste. Dies mag sich zwar sehr viel anhören, heißt aber nicht, dass die berechnete Tour immer 50% länger ist als die kürzeste Tour. In den meisten Fällen wird sie nur wenig länger sein als die optimale Tour. Dass die Länge der berechneten Tour tatsächlich um 50% von der kürzesten abweicht, kommt nur sehr selten vor. Es ist aber bewiesen, dass sie niemals mehr als 50% länger ist.

Sind die Distanzen nicht symmetrisch, ist aber trotzdem die Dreiecksungleichung erfüllt, dann sieht die Situation etwas schlechter aus: Das TSP besitzt dann nur noch eine $0,842 \cdot \log_2 n$ -Approximation, wobei n die Anzahl der Städte ist (\log_2 bezeichnet den Logarithmus zur Basis 2; z. B. ist $\log_2 1024 = 10$, da $2^{10} = 1024$ ist). Es ist unbekannt, ob diese Variante des TSP eine Approximation mit konstanter Güte besitzt.

Ergebnisse unseres Instituts Eine starke Einschränkung des Problems ist, nur 1 und 2 als Distanzen zuzulassen. Selbst diese eingeschränkte Variante ist \mathcal{NP} -vollständig. Es gelang uns aber, eine $\frac{4}{3}$ -Approximation zu entwickeln, die auch für den Fall funktioniert, dass die Distanzen nicht symmetrisch sind [5].

Oft ist der direkte Weg von einer Stadt zu einer anderen wirklich kürzer als der Umweg über eine dritte Stadt. Formal heißt dies $D(i, j) \leq \gamma \cdot (D(i, k) + D(k, j))$ für eine Konstante γ mit $\frac{1}{2} \leq \gamma \leq 1$. Für $\gamma = 1$ erhalten wir die normale Dreiecksungleichung. Für $\gamma = \frac{1}{2}$ ist das Problem trivial: Alle Städte sind gleich weit von einander entfernt, es ist gleichgültig, in welcher Reihenfolge wir die Städte besuchen. Für beliebige Werte von γ gelang es uns, eine $\frac{1+\gamma}{2-\gamma-\gamma^3}$ -Approximation zu entwickeln [6].

2.4.2 Alternative Rechnermodelle

Parallelrechner Bereits für 25 Städte ist es praktisch unmöglich, alle Touren auszuprobieren. Durch geschickte Algorithmen kann man aber viele Touren von vorn herein

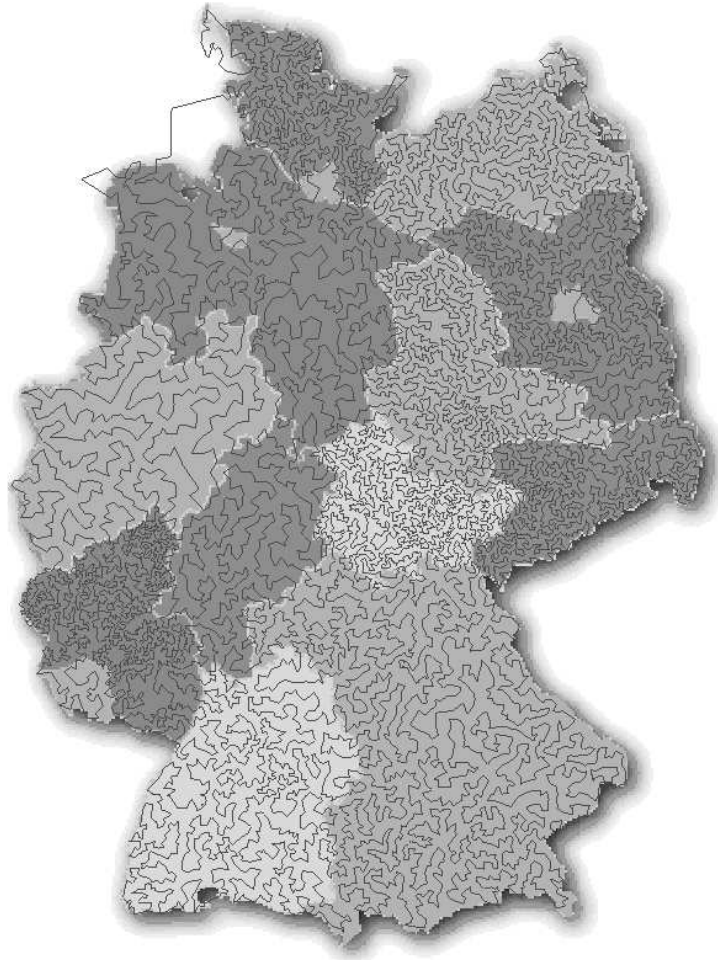


Abbildung 3: Weltrekord: Eine kürzeste Rundreise durch 15 112 deutsche Städte (Quelle: <http://www.math.princeton.edu/tsp/d15sol/>).

ausschließen; z. B. ist eine Rundreise, bei der man von Madrid über Warschau nach Paris reist, sicherlich nicht die kürzeste. Hinzu kommt, dass moderne Parallelrechner ein Vielfaches schneller arbeiten als handelsübliche PCs. Die Kombination aus Beidem erlaubt das Lösen immer größerer Probleme. Der aktuelle Rekord ist eine Landkarte mit 15 112 Städten (siehe Abb. 3).

Am Institut für Theoretische Informatik implementieren wir auf dem Parallelrechner „Mayflower“ ein Lösungsverfahren für ein anderes \mathcal{NP} -vollständiges Problem, das *Multiple Sequence Alignment Problem* aus der Bioinformatik. Ziel ist es dabei, auch Datensätze von DNA-Sequenzen bearbeiten zu können, bei denen das bislang nicht möglich war.

DNA-Computer Eine ganz neue Idee ist, Berechnungen mit Hilfe von Biomolekülen durchzuführen. Hier gelang es, das *Hamilton Path Problem*, ein weiteres \mathcal{NP} -vollständiges Problem und nahe verwandt mit dem TSP, mit Hilfe von DNA-Molekülen zu lösen. Allerdings gelang dies bislang nur für eine sehr kleine Anzahl von Städten.

Quantencomputer Ein weiterer „non-standard“ Ansatz sind *Quantencomputer*. Dabei werden physikalische Quanteneffekte ausgenutzt, um eine große Anzahl Möglichkeiten parallel zu untersuchen. Beispielsweise kann das Problem, eine Zahl in ihre Primfaktoren zu zerlegen, auf einem Quantencomputer in polynomieller Zeit gelöst werden. Es ist aber unbekannt, ob dies auch auf einem herkömmlichen Computer möglich ist oder ob das Problem vielleicht sogar \mathcal{NP} -vollständig ist.

Literatur

- [1] ARPE, JAN, ANDREAS JAKOBY und MACIEJ LIŚKIEWICZ: *One-Way Communication Complexity of Symmetric Boolean Functions*. In: LINGAS, ANDRZEJ und BENGT J. NILSSON (Herausgeber): *Fundamentals of Computation Theory, 14th International Symposium, FCT 2003, Malmö, Sweden, August 2003, Proceedings*, Band 2751 der Reihe *Lecture Notes in Computer Science*, Seiten 158–170. Springer, 2003.
- [2] ARPE, JAN und RÜDIGER REISCHUK: *Robust Inference of Relevant Attributes*. In: GAVALDÀ, RICARD, KLAUS P. JANTKE und EIJI TAKIMOTO (Herausgeber): *Algorithmic Learning Theory, 14th International Conference, ALT 2003, Sapporo, Japan, October 2003, Proceedings*, Band 2842 der Reihe *Lecture Notes in Artificial Intelligence*, Seiten 99–113. Springer, 2003.
- [3] BALBACH, FRANK: *Changing the Inference Type—Keeping the Hypothesis Space*. In: GAVALDÀ, RICARD, KLAUS P. JANTKE und EIJI TAKIMOTO (Herausgeber): *Algorithmic Learning Theory, 14th International Conference, ALT 2003, Sapporo, Japan, October 2003, Proceedings*, Band 2842 der Reihe *Lecture Notes in Artificial Intelligence*, Seiten 84–98. Springer, 2003.
- [4] BLÄSER, MARKUS, ANDREAS JAKOBY, MACIEJ LIŚKIEWICZ und BODO MANTHEY: *Private Computation — k -connected versus 1-connected Graphs*. *Journal of Cryptology*, im Druck.
- [5] BLÄSER, MARKUS und BODO MANTHEY: *Approximating Maximum Weight Cycle Covers in Directed Graphs with Weights Zero and One*. *Algorithmica*, im Druck.
- [6] BLÄSER, MARKUS, BODO MANTHEY und JIŘÍ SGALL: *An Improved Approximation Algorithm for the Asymmetric TSP with Strengthened Triangle Inequality*. KAM-DIMATIA Series 2004-657 and ITI Series 2004-180, Charles University, Prague, Czech Republic, 2004.
- [7] CASE, JOHN, SANJAY JAIN, RÜDIGER REISCHUK, FRANK STEPHAN und THOMAS ZEUGMANN: *Learning a Subclass of Regular Patterns in Polynomial Time*. In: GAVALDÀ, RICARD, KLAUS P. JANTKE und EIJI TAKIMOTO (Herausgeber): *Algorithmic Learning Theory, 14th International Conference, ALT 2003, Sapporo, Japan, October 2003, Proceedings*, Band 2842 der Reihe *Lecture Notes in Artificial Intelligence*, Seiten 232–246. Springer, 2003.

- [8] JAKOBY, ANDREAS und MACIEJ LIŚKIEWICZ: *Paths Problems in Symmetric Logarithmic Space*. In: WIDMAYER, PETER, FRANCISCO TRIGUERO, RAFAEL MORALES, MATTHEW HENNESSY, STEPHAN EIDENBENZ und RICARDO CONEJO (Herausgeber): *Automata, Languages and Programming, 29th International Conference, ICALP 2002, Málaga, Spain, July 2002, Proceedings*, Band 2380 der Reihe *Lecture Notes in Computer Science*, Seiten 269–280. Springer, 2002.
- [9] JAKOBY, ANDREAS, MACIEJ LIŚKIEWICZ und RÜDIGER REISCHUK: *Private Computations in Networks: Topology versus Randomness*. In: ALT, HELMUT und MICHEL HABIB (Herausgeber): *STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February/March 2003, Proceedings*, Band 2607 der Reihe *Lecture Notes in Computer Science*, Seiten 121–132. Springer, 2003.
- [10] JAKOBY, ANDREAS, MACIEJ LIŚKIEWICZ und RÜDIGER REISCHUK: *Space Efficient Algorithms for Directed Series-Parallel Graphs*. *Journal of Algorithms*, im Druck.
- [11] MANTHEY, BODO: *Non-Approximability of Weighted Multiple Sequence Alignment*. *Theoretical Computer Science*, 296(1):179–192, 2003.
- [12] REISCHUK, RÜDIGER: *Zeit und Raum in Rechnernetzen*. In: WEGENER, INGO (Herausgeber): *Highlights aus der Informatik*, Seiten 155–176. Springer, 1996.
- [13] VÖLZER, HAGEN: *On Randomization Versus Synchronization in Distributed Systems*. In: DÍAZ, JOSEP, JUHANI KARHUMÄKI, ARTO LEPISTÖ und DONALD SANNELLA (Herausgeber): *Automata, Languages and Programming, 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004 Proceedings*, Band 3142 der Reihe *Lecture Notes in Computer Science*, Seiten 1214–1226. Springer, 2004.