Zufall und zufallsgesteuerte Algorithmen

Juraj Hromkovic
Lehrstuhl für Informatik I
RWTH Aachen

Zufallssteuerung ist heutzutage ein fester Bestandteil von Informatiksystemen, und wir verdanken ihr einige der faszinierendsten mathematisch-naturwissenschaftlichen Entdeckungen in der Informatik. Das erste Ziel dieses Artikels ist es, dem Leser an einem einfachen Beispiel zu zeigen, dass zufallsgesteuerte Systeme und Programme eine Leistungsfähigkeit besitzen, die deterministische nie erreichen können. Das zweite Ziel ist es, anhand der "Methode der häufigen Zeugen" zum Verständnis der Berechnungsstärke zufallsgesteuerter Algorithmen beizutragen.

> Das Gewebe dieser Welt ist aus Notwendigkeit und Zufall gebildet;
> die Vernunft des Menschen stellt sich zwischen beide und weiß sie zu beherrschen;
> sie behandelt das Notwendige als den Grund ihres Daseins;
> das Zufällige weiß sie zu lenken, zu leiten und zu nutzen, ...
>
> Johann Wolfgang von Goethe

1. Einleitung

Die meisten Menschen haben zum Zufall eine negative emotionale Einstellung, weil sie Zufall mit Ungewissheit verbinden. Am liebsten hätten sie eine exakte Zukunftsvorhersage, um auf alle möglichen Alternativen vorbereitet zu sein; ersatzweiseschließen sie viele Versicherungen ab, um gegen Eventualitäten gewappnet zu sein. Einige Menschen spielen aber auch gerne mit Zufall, insbesondere wenn die Vision eines Lotteriegewinns bevorsteht.
Die grundlegende Frage ist aber:

Gibt es überhaupt Zufall?

Diese Frage wird schon seit Jahrtausenden zunächst in der Philosophie und später in den Naturwissenschaften intensiv untersucht. Demokrit meinte, dass das *Zufällige das Nichterkannte ist,* und dass *die Natur in ihrer Grundlage determiniert ist.* Dies liegt im Widerspruch zur folgenden Behauptung von Epikur:"Der Zufall ist objektiv, er ist die eigentliche Natur der Erscheinungen". Die heutige Wissenschaft scheint Epikur Recht zu geben.
Die Gesetze der Quantenmechanik sind durch Zufall bestimmt, und die Quantenmechanik ist die Basis aller Prozesse in unserer Welt. Aus dieser Sicht ist es der Zufall, der die Ordnung und Gesetze bestimmt. Aus Sicht der Biologie gibt es keine Evolution ohne Zufall, eine weitere Bestätigung für die Dominanz des Zufalls in der Natur.

Aus diesem und anderen Bedürfnissen hat die Mathematik die Wahrscheinlichkeitstheorie entwickelt, die uns die Modellierung von zufälligen Erscheinungen ermöglicht. Die Basis für diese Modellierung ist sehr einfach. Wir ordnen die Wahrscheinlichkeit 1 der Sicherheit (dass

etwas passieren muss) und die Wahrscheinlichkeit 0 dem Unmöglichen (dass etwas nicht passieren kann) zu. Alle anderen möglichen Erscheinungen können Wahrscheinlichkeiten zwischen 0 und 1 haben. Im Allgemeinen betrachten wir modellierte Realität als ein Zufallsexperiment, bei dem alle möglichen Resultate[1] des Experiments bekannt sind. Eine Wahrscheinlichkeitsverteilung ist eine Zuordnung von Wahrscheinlichkeiten zu allen möglichen Resultaten. Wenn ein Resultat eine Wahrscheinlichkeit p, $0 < p < 1$, hat, bedeutet es nichts anderes, als dass die Häufigkeit des Auftretens des Resultats in einer Folge von unabhängig durchgeführten Experimenten p 100% ist. Weil eines der möglichen Resultate erscheinen muss, fordert man, dass die Summe der Wahrscheinlichkeiten aller möglichen Resultate in einem Experiment 1 ist.

Ein einfaches Beispiel eines Zufallsexperimentes ist das Würfeln. Bei einem idealen Würfel setzt man die gleiche Wahrscheinlichkeit $\frac{1}{6}$ für jedes Resultat, wobei die sechs möglichen Resultate den gewürfelten Zahlen 1, 2, 3, 4, 5 und 6 entsprechen.

Die Physiker waren die Ersten, die auf die Idee kamen, durch zufallsgesteuerte Systeme Naturprozesse zu simulieren. Der große Nutzen von Zufallssteuerung fand aber erst in der Informatik durch den Entwurf von zufallsgesteuerten Algorithmen statt. Was sind aber zufallsgesteuerte Algorithmen? Die Programme (Algorithmen), die wir gewohnt sind, sind deterministisch. "Deterministisch" bedeutet, dass das Programm und die Problemeingabe die Arbeit des Programms vollständig bestimmen. In jedem Augenblick ist in Abhängigkeit von den aktuellen Daten eindeutig klar, was die nächste Aktion des Programms sein wird. Zufallsgesteuerte Programme können mehrere Möglichkeiten haben, ihre Arbeit fortzusetzen, und welcher der Möglichkeiten das Programm folgt, wird zufällig entschieden. Es sieht so aus, als ob ein zufallsgesteuertes Programm von Zeit zu Zeit eine Münze wirft und abhängig davon, ob Kopf oder Zahl gefallen ist, wählt es die entsprechende Strategie auf seiner Suche nach dem richtigen Resultat. Auf diese Weise kann ein zufallsgesteuertes Programm mehrere unterschiedliche Berechnungen für ein und dieselbe Eingabe durchlaufen. Im Unterschied zu deterministischen Programmen, wo immer eine zuverlässige Berechnung des richtigen Resultates gefordert wird, dürfen einige Berechnungen des zufallsgesteuerten Programms auch falsche Resultate liefern. Das Ziel ist jedoch, die Wahrscheinlichkeit einer falschen Berechnung klein zu halten, was in gewissem Sinne bedeutet, den proportionalen Anteil der Berechnungen mit falschem Resultat zu reduzieren.

Auf den ersten Blick sieht ein zufallsgesteuertes Programm unzuverlässig aus im Vergleich zu deterministischen Programmen, und man kann fragen, wozu es gut sein soll, gelegentlich falsche Resultate zu erhalten. Es existieren aber Aufgaben von enormer praktischer Bedeutung, bei denen der schnellste deterministische Algorithmus auf dem schnellsten Rechner mehr Zeit zur Berechnung braucht als die Zeit vom Urknall bis zum heutigen Tag, die Aufgabe also praktisch unlösbar ist. Und da kommt ein "Wunder" – ein zufallsgesteuerter Algorithmus, der die Aufgabe in ein paar Minuten auf einem Standard-PC löst mit einer Fehlerwahrscheinlichkeit von 1 zu Tausend Milliarden. Warum kann man so ein Programm trotzdem als zuverlässig ansehen, obwohl es Fehler macht? Ganz einfach: Ein herkömmliches deterministisches Programm, das diese Aufgabe in einem Tag berechnet, ist viel unzuverlässiger als unser zufallsgesteuertes Programm, weil die Wahrscheinlichkeit des Auftretens eines Hardwarefehlers während des 24-stündigen Programmlaufs viel höher ist als

[1] Diese Resultate nennt man in der Wahrscheinlichkeitstheorie elementare Ereignisse.

die Wahrscheinlichkeit einer fehlerhaften Ausgabe des schnellen zufallsgesteuerten Algorithmus.

Dieser Artikel eröffnet dem Leser einen Blick hinter die Kulissen der Zufallssteuerung. Im nächsten Abschnitt präsentieren wir ein Beispiel eines zufallsgesteuerten Algorithmus, der eine Kommunikationsaufgabe löst, die auf deterministische Weise praktisch unlösbar erscheint. Im dritten Abschnitt nutzen wir dieses Beispiel, um die "Methode der häufigen Zeugen" zu erklären, eine Entwurfsmethode für zufallsgesteuerte Algorithmen. Dabei gewinnen wir ein etwas tieferes Verständnis für die überlegene Leistungsfähigkeit der Zufallssteuerung gegenüber der klassischen deterministischen Steuerung.

2. Ein Kommunikationsprotokoll für den Vergleich von zwei Datenbanken

Betrachten wir die folgende Ausgangssituation. Wir haben zwei Rechner R_I und R_{II}, auf denen ursprünglich eine Datenbank gleichem Inhalts in Original und Kopie abgelegt war. Anschließend hat man jede Änderung oder Ergänzung am Datenbestand auf R_I auch am Datenbestand auf R_{II} vorgenommen, um idealerweise beide Datenbanken identisch zu erhalten. Nach längerer Zeit wollen wir nun überprüfen, ob R_I und R_{II} wirklich noch die gleichen Datenbestände haben. Wir bezeichnen mit n die Größe der Datenbank in Bits. Konkret betrachten wir ein großes $n = 10^{12}$, was für die Datenbank eines Genforschers eine realistische Größe ist. Unser Ziel ist es, einen Kommunikationsalgorithmus (Protokoll) zu entwerfen, der feststellt, ob die Inhalte der Datenbanken von R_I und R_{II} unterschiedlich oder gleich sind. Die Komplexität der Lösung ist bestimmt durch die Anzahl der zwischen R_I und R_{II} ausgetauschten Bits. Man kann mathematisch beweisen, dass jedes Protokoll für diese Aufgabe einen Austausch von n Bits zwischen R_I und R_{II} erfordert. Es gibt also kein Protokoll, das diese Aufgabe zuverlässig löst und mit höchstens $n-1$ Kommunikationsbits auskommt. Wenn man bei dieser auszutauschenden Datenmenge noch sicherstellen soll, dass alle Kommunikationsbits korrekt ankommen, würde man auf den Versuch, die Aufgabe auf diese Weise zu lösen, wahrscheinlich verzichten.

Die Lösung in dieser Situation bietet das folgende zufallsgesteuerte Protokoll. Es basiert auf der grundlegenden Erkenntnis der Zahlentheorie, dass es zwischen den ersten n natürlichen Zahlen $1,2,...,n$ ungefähr $\frac{n}{\ln n}$ Primzahlen gibt. Für jede Folge $x = x_1...x_n$ von Bits sei $Nummer(x) = \sum_{i=1}^{n} x_i 2^{n-i}$ die Zahl mit binärer Darstellung x.

Das zufallsgesteuerte Kommunikationsprotokoll P

Ausgangssituation: Auf R_I liegen die n Bits $x = x_1...x_n$, auf R_{II} die n Bits $y = y_1...y_n$.

Schritt 1: R_I wählt nun zufällig mit einer uniformen Wahrscheinlichkeitsverteilung (also Gleichverteilung) eine von den ungefähr $n^2 / \ln n^2$ Primzahlen kleiner gleich n^2. Dies sei p.

Schritt 2: R_I berechnet $s = Nummer(x) \bmod p$ und schickt die binäre Darstellung von s und p zu R_{II}.

Schritt 3: Nach dem Empfang von s und p berechnet R_{II} $q = Nummer(y) \bmod p$.

Falls $q \neq s$, dann gibt R_{II} die Ausgabe "Datenbanken ungleich".

Falls $q = s$, dann gibt R_{II} die Ausgabe "Datenbanken gleich".

Jetzt analysieren wir die Arbeit von P. Zuerst bestimmen wir die Komplexität gemessen als die Anzahl der Kommunikationsbits, und dann analysieren wir die Zuverlässigkeit (Fehlerwahrscheinlichkeit) von P.

Die einzige Kommunikation besteht darin, dass R_I die Zahlen s und p an R_{II} schickt. Weil $s \leq p \leq n^2$, ist die Länge der binären Nachricht $2 \lceil \log_2 n^2 \rceil \leq 4 \lceil \log_2 n \rceil$. Für $n = 10^{12}$ sind es $4 \cdot 28 = 112$ Bits. Es ist also eine sehr kurze Nachricht, die man problemlos zuverlässig übertragen kann.

Bei der Analyse der Fehlerwahrscheinlichkeit unterscheiden wir zwei Möglichkeiten.

(i) Angenommen $x = y$. Dann gilt $Nummer(x) \bmod p = Nummer(y) \bmod p$ für alle Primzahlen p. Also gibt R_{II} die korrekte Antwort "Datenbanken gleich" mit Sicherheit. In diesem Fall ist also die Fehlerwahrscheinlichkeit 0. Erhält man also die Antwort "Datenbanken ungleich", so kann man sicher sein, dass diese Aussage korrekt ist.

(ii) Angenommen $x \neq y$. Wir bekommen die falsche Antwort "Datenbanken gleich" nur dann, wenn R_I eine zufällige Primzahl p gewählt hat, die die Eigenschaft hat, dass $z = Nummer(x) \bmod p = Nummer(y) \bmod p$ gilt.

In anderer Form geschrieben:
$Nummer(x) = x_1 \cdot p + z$ und $Nummer(y) = y_1 \cdot p + z$ für zwei natürliche Zahlen x_1 und y_1.

Daraus folgt, dass $Nummer(x) - Nummer(y) = x_1 \cdot p - y_1 \cdot p = (x_1 - y_1) \cdot p$, dass also p ein Teiler von $Nummer(x) - Nummer(y)$ ist.

Also gibt unser Protokoll P nur dann eine falsche Antwort, wenn die gewählte Primzahl p ein Teiler der Zahl $Nummer(x) - Nummer(y)$ ist. Wir wissen, dass p aus ungefähr $n^2 / \ln n^2$ Primzahlen von $\{2,3,...,n^2\}$ nach der uniformen Wahrscheinlichkeitsverteilung gewählt wurde. Wir müssen also ermitteln, wie viele von diesen $n^2 / \ln n^2$ Primzahlen die Zahl $Nummer(x) - Nummer(y)$ teilen können. Weil x und y in Binärdarstellung die Länge n haben, gilt $w = |Nummer(x) - Nummer(y)| < 2^n$.

Sei $w = p_1^{i_1} p_2^{i_2} ... p_k^{i_k}$, wobei $p_1 < p_2 < ... < p_k$ Primzahlen und $i_1, i_2, ..., i_k$ positive ganze Zahlen sind. Wir wissen, dass jede Zahl solch eine eindeutige Faktorisierung hat. Unser Ziel ist zu beweisen, dass $k < n - 1$.

Wir beweisen es indirekt. Angenommen $k \geq n$.

Dann gilt $w = p_1^{i_1} p_2^{i_2} ... p_k^{i_k} > p_1 \cdot p_2 \cdot ... \cdot p_n > 1 \cdot 2 \cdot 3 \cdot ... \cdot n = n! > 2^n$.

Das widerspricht aber der obigen Erkenntnis, dass $w < 2^n$. Also kann w höchstens $n - 1$ unterschiedliche Primfaktoren haben. Weil jede Primzahl aus $\{2,3,...,n^2\}$ die gleiche Wahrscheinlichkeit hat, gewählt zu werden, ist die Wahrscheinlichkeit, ein p zu wählen, das w teilt, höchstens $\frac{n-1}{n^2 / \ln n^2} < \frac{\ln n^2}{n}$.

Also ist die Fehlerwahrscheinlichkeit von P für unterschiedliche Inhalte x und y höchstens $\ln n^2 / n$, was für $n = 10^{12}$ höchstens $0{,}277\ 10^{-10}$ ist.

So eine kleine Fehlerwahrscheinlichkeit ist kein ernsthaftes Risiko mehr. Für den Fall jedoch, dass man eine noch kleinere Fehlerwahrscheinlichkeit benötigt, kann man das Protokoll P 10-mal nacheinander mit 10 unabhängigen Wahlen einer Primzahl wie folgt laufen lassen.

Protokoll P_{10}

Anfangssituation: R_I hat n Bits $x = x_1...x_n$ und R_{II} hat n Bits $y = y_1...y_n$.

Schritt 1: R_I wählt zufällig bezüglich uniformer Wahrscheinlichkeitsverteilung 10 Primzahlen $p_1, p_2,..., p_{10}$ aus $\{2,3,...,n^2\}$.

Schritt 2: R_I berechnet $s_i = Nummer(x) \bmod p_i$ für $i = 1,2,...,10$ und schickt die binären Darstellungen von $p_1, p_2,..., p_{10}, s_1, s_2,..., s_{10}$ zu R_{II}.

Schritt 3: Nach dem Empfang von $p_1, p_2,..., p_{10}, s_1, s_2,..., s_{10}$ berechnet R_{II} $q_i = Nummer(y) \bmod p_i$ für $i = 1,2,...,10$.

Falls ein i $\{1,2,...,10\}$ existiert, so dass $q_i \neq s_i$, dann gibt R_{II} die Ausgabe "Datenbanken ungleich".

Falls $q_j = s_j$ für alle j $\{1,2,...,10\}$, dann gibt R_{II} die Ausgabe "Datenbanken gleich".

Wir sehen, dass die Kommunikationskomplexität von P_{10} zehn mal größer ist als die Komplexität von P ist. In unserem Fall $n = 10^{12}$ sind es aber höchstens 1120 Bits, was kein technisches Problem darstellt. Wie ändert sich aber die Fehlerwahrscheinlichkeit? Falls $x = y$ wird P_{10} wieder keinen Fehler machen und gibt die richtige Antwort "Datenbanken gleich" mit Sicherheit. Falls $x \neq y$ wird P_{10} nur eine falsche Antwort liefern, wenn alle 10 zufällig gewählten Primzahlen zu den höchstens n-1 Primzahlen, die $Nummer(x) - Nummer(y)$ teilen, gehören. Weil die 10 Primzahlen in 10 unabhängigen Experimenten gewählt worden sind, ist die Fehlerwahrscheinlichkeit höchstens

$$\left(\frac{n-1}{n^2 / \ln n^2}\right)^{10} < \left(\frac{\ln n^2}{n}\right)^{10} = \frac{2^{10}\ (\ln n)^{10}}{n^{10}}\ .$$

Für $n = 10^{12}$ ist es höchstens $\frac{0{,}266}{10^{102}}$. Wenn wir bedenken, dass die Anzahl der Mikrosekunden seit dem Urknall bis zum heutigen Tag eine 24-stellige Zahl ist und die Anzahl von Protonen im bekannten Universum eine 79-stellige Zahl ist, kann man eine Fehlerwahrscheinlichkeit unter 10^{-102} leichten Herzens in Kauf nehmen. Auch wenn ein deterministisches Protokoll mit Kommunikationskomplexität 10^{12} Bits praktisch realisierbar wäre, ist es klar, dass man aus Kostengründen das zufallsgesteuerte Protokoll implementieren würde.

Die Konstruktion von P_{10} aus P liefert uns eine wichtige Erkenntnis. Wir können die Fehlerwahrscheinlichkeit von zufallsgesteuerten Algorithmen durch mehrfach wiederholtes Durchlaufen des Algorithmus beliebig reduzieren. Bei einigen Algorithmen, wie unserem Protokoll P, führen wenige Wiederholungen zu einem extremen Rückgang der Fehlerwahrscheinlichkeit.

3. Die Methode der häufigen Zeugen

Zum Schluß dieses Artikels wollen wir noch nach den Gründen suchen, warum unser zufallsgesteuertes Protokoll P unvergleichbar effizienter ist als jedes deterministische Protokoll für die gestellte Aufgabe ist. Das Protokoll P haben wir durch eine einfache Anwendung der Methode von häufigen Zeugen gewonnen.

Im Allgemeinen betrachtet man ein sogenanntes Entscheidungsproblem, bei dem man entscheiden soll, ob eine gegebene Eingabe eine gewisse Eigenschaft besitzt oder nicht. Setzen wir noch voraus, dass wir keinen effizienten deterministischen Algorithmus für die Aufgabe gefunden haben (oder sogar, dass kein effizienter Algorithmus für die Aufgabe existiert). Bei einer Anwendung der Methode der häufigen Zeugen fängt man jetzt mit der Suche nach passender Definition von Zeugen an. Ein Zeuge sollte eine Zusatzinformation zur Eingabe sein, mit deren Hilfe man effizient deterministisch beweisen kann, dass die Eingabe die gewünschte Eigenschaft hat (oder dass die Eingabe die Eigenschaft nicht hat). In unserem Beispiel war eine Primzahl p der Zeuge des Unterschiedes zwischen x und y (also Zeuge von $x \neq y$) falls $Nummer(x) \bmod p \neq Nummer(y) \bmod p$.

Wenn man also solch ein p geschenkt bekommt, kann man effizient die Tatsache "x ist unterschiedlich von y" beweisen. In der Realität können wir auf ein solches Geschenk nicht hoffen und schlimmer noch, wir können den Zeugen nicht effizient deterministisch berechnen (sonst hätten wir ja schon einen effizienten deterministischen Algorithmus für die Aufgabe). Um einen effizienten Algorithmus zu entwerfen, brauchen wir für jede Eingabe eine Menge von Zeugenkandidaten, von denen dann ausreichend viele wirklich Zeugen sind. In unserem Beispiel sind Kandidaten für Zeugen alle Primzahlen kleiner gleich n^2, also ungefähr $n^2 / \ln n^2$. Von diesen Kandidaten sind mindestens $\frac{n^2}{\ln n^2} - (n-1)$ tatsächlich Zeugen, so dass die Wahrscheinlichkeit, einen Zeugen aus der Kandidatenmenge zufällig zu ziehen, mindestens

$$\frac{\frac{n^2}{\ln n^2} - (n-1)}{\frac{n^2}{\ln n^2}} \geq 1 - \frac{\ln n^2}{n} \text{ ist.}$$

Dieser Wert liegt sehr nahe bei 1, ist also sehr günstig, so dass fast jeder Kandidat auch Zeuge ist. Aber auch wenn die Wahrscheinlichkeit, einen Zeugen zu ziehen, nur $\frac{1}{2}$ wäre, wären die Zeugen noch immer ausreichend häufig. Uns reicht es, einfach mehrere Zufallsversuche zu machen. Dadurch wächst die Wahrscheinlichkeit schnell, in mindestens einem der Versuche einen Zeugen zu bekommen, nach zwei Versuchen ist die Wahrscheinlichkeit $\frac{3}{4}$, nach drei Versuchen $\frac{7}{8}$ usw.

Man kann sich jetzt noch fragen, warum wir einen Zeugen nicht auch deterministisch schnell finden können, wenn schon nahezu jeder Zeugenkandidat auch Zeuge ist. Man könnte doch z.B. systematisch der Reihe nach auf geschickte Weise alle Kandidaten überprüfen, so dass man nach kurzer Zeit einen Zeugen findet. Das Problem ist aber, dass die Zeugen für jede Eingabe unterschiedlich zwischen den Zeugenkandidaten verteilt sind. Wenn man sich also auf eine Durchsuchungsstrategie festlegt, kann man immer Eingaben finden, bei denen die Strategie versagt.

Betrachten wir unser Beispiel. Hier kann man sogar beweisen, dass keine Strategie existiert, die für jede Eingabe (x, y) effizient einen Zeugen findet. Um das zu veranschaulichen, nehmen wir die einfache Strategie, alle Primzahlen bei der kleinsten beginnend der Reihe nach

auszuprobieren. Es ist klar, dass spätestens nach n Proben ein Zeuge gefunden werden muss, weil höchstens $n-1$ Nicht-Zeugen zwischen den Kandidaten sind. Leider bedeuten n Proben eine Kommunikationskomplexität von $n \cdot \log_2 n$, was wir uns nicht leisten können. Und warum gelingt es nicht, schon nach ein paar Proben einen Zeugen zu finden? Bei Eingaben (x, y) mit $Nummer(x) - Nummer(y) = p_1 \cdot p_2 \cdot \ldots \cdot p_k$, wobei $k = n/(2(\log_2 n)^2)$ und $p_1 < p_2 < \ldots < p_k$ die kleinsten Primzahlen sind, braucht unsere Strategie $k+1$ Proben, um einen Zeugen zu finden. Man kann sich leicht vorstellen, dass man bei jeder anderen Aufzählung der Primzahlen spezifische Eingaben findet, für die viele Proben notwendig werden, um einen Zeugen zu finden.

Die Methode der häufigen Zeugen ist ein leistungsfähiges Verfahren zum Entwurf von zufallsgesteuerten Algorithmen. Der effiziente zufallsgesteuerte Primzahltest basiert auf dieser Methode und gehört damit zu den wichtigsten Anwendungen von großer praktischer Bedeutung in der Kryptographie. Die besten bekannten deterministischen Algorithmen würden für den Primzahltest großer Zahlen mehr Zeit benötigen als die Lebensdauer des Universums. Wir können hier nicht darauf eingehen, wie man beim Primzahltest die Zeugen definiert und für den Algorithmus verwendet. In diesem Falle hat auch noch niemand bewiesen, dass man einen Zeugen nicht effizient deterministisch finden kann. Man weiß sogar, dass die Gültigkeit der Riemann'schen Vermutung eine solche Methode liefern würde. Aber für die Praxis ist es im Augenblick auch nicht wichtig, ob schnelle deterministische Algorithmen für die Aufgaben existieren oder nicht. Das Einzige was zählt, ist, dass die Lösungen von Problemen durch zufallsgesteuerte Algorithmen effizient und hinreichend zuverlässig sind. Der Unterschied zwischen der Effizienz der bestmöglichen deterministischen Algorithmen und der bestmöglichen zufallsgesteuerten Algorithmen zu bestimmen, bleibt eine der zentralen Forschungsaufgaben der theoretischen Informatik.

Prof. Dr. Juraj Hromkovic
Lehrstuhl für Informatik I
RWTH Aachen
52056 Aachen
Email: jh@cs.rwth-aachen.de
Internet: http://www-i1.informatik.rwth-aachen.de

Literatur

Hromkovic, J.: Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Springer-Verlag 2001.

Hromkovic, J.: Communication Protocols – an Exemplary Study of the Power of Randomness. In: Pardalos, P., Rajasekaran, S., Reif, J., Rolim, J. (Eds.): Handbook of Randomized Computing. Kluwer Publ. 2001.

Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press 1995