Advanced Topics in Algorithmics

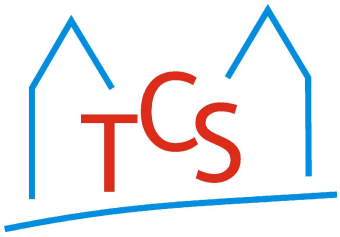
Unterlagen zum Lehrmodul CS4008

Prof. Dr. R. Reischuk

Institut für Theoretische Informatik

Universität zu Lübeck

WS 2009/10



Version 8.3.2010

Inhaltsverzeichnis

1 Einleitung

1.1 Überblick

Ziel des Lehrmoduls, Vorgehensweise

Lehrinhalte, Methodiken

1.2 Literatur

M. Atallah (Ed.), Algorithms and Theory of Computation Handbook, Chapman & Hall / CRC, 1999.

W. Kocay, D. Kreher, Graphs, Algorithms and Optimization, Chapman & Hall / CRC, 2005.

D. Hochbaum (Ed.), Approximation Algorithms for NP-Hard Problems, Int. Thompson Publishing, 1997.

B. Chazelle, The Discrepancy Method, Cambridge Univ. Press, 2000.

G. Gutin, A. Punnen (Eds.), The Traveling Salesman Problem and Its Variations, Kluwer, 2002.

M. Mitzenmacher, E. Upfal, Probability and Computing, Cambridge Univ. Press, 2005.

[BDI05] Y. Boufkhad, O. Dubois, Y. Interian, B. Selman, Regular Random k-SAT: Properties of Balanced Formulas, J. Automated Reasonig 35, 2005, 181-200.

[KKK98] L. Kirousis, E. Kranakis, D. Krizanc, Y. Stamatiou, Approximating the Unsatisfiability Threshold of Random Formulas, Random Struct. Alg. 12, 1998, 253-269.

[HSS07] T. Hofmeister, U. Schöning, R. Schuler, O. Watanabe, Randomized Algorithms for 3-SAT, Theory of Comput. Systems 40, 2007, 249-262.

[S07] U. Schöning, Principles of Stochastic Local Search, Proc. 6. Int. Conf. Unconventional Computation 2007, Springer LNCS 4618, 178-187

2 Erfüllbarkeit von 3SAT-Formeln

Aufgabe 2.1 *Gegeben sei die folgende Aufgabe* **schwere Instanzen für Erfüllbarkeit***: Finde eine konkrete 3SAT-Formel, die mindestens eine erfüllende Belegung besitzt, für die es aber schwer ist, eine erfüllende Belegung zu finden.*

Es sei r die Anzahl der Variablen und m die Anzahl der Klauseln.

2.1 Lösungsvorschläge für schwere SAT-Instanzen 1. Runde

1. Überlegungen:

   (a) es gibt insgesamt 560 potentielle Klauseln,

   (b) mindestens 8 Klauseln sind notwendig, damit die Formel unerfüllbar wird,

   (c) es sollte möglichst wenige erfüllende Belegung geben, am Besten nur eine,

   (d) alle Variablen sollten mindestens 2 mal benutzt werden, sonst können sie einfach gesetzt werden.

   Das alles führte aber zu extrem langen Formeln. Daher habe ich mich auf nur 5 Variablen beschränkt und folgende Formel entworfen.

$(-1,-3,-5)$	$(2,3,4)$
$(3,4,-5)$	$(-2,-4,-5)$
$(-1,-2,-4)$	$(-2,3,-5)$
$(1,2,-5)$	$(-1,2,-3)$
$(2,-3,-5)$	$(1,3,-4)$
$(2,-3,-4)$	$(-1,-2,3)$
$(1,3,5)$	$(1,2,-4)$
$(-2,-3,4)$	$(-2,4,5)$
$(-3,4,5)$	

   Fazit: 5 Variable sind zu wenig, um zu verhindern, daß ein Mensch durch einfaches Testen eine Lösung schnell finden kann.

2. Überlegungen:

   (a) Komplexität durch Unübersichtlichkeit ist langweilig. Es ist zwar richtig, dass so eine Formel aufwändiger zu untersuchen ist, wenn sie länger ist, allerdings ist hier zu unterscheiden zwischen Aufwand, der entsteht, weil man die Formel nicht im Kopf behalten kann und Aufwand, der entsteht, weil die Formel in sich trickreich angeordnet ist. Der erstgenannte Aufwand ist immer etwas langweilig, weil man ihn durch Papier und Stift (und mehr Zeit) ausgleichen kann. Deswegen wurde hier auch verzichtet, eine wesentlich längere Formel zu konstruieren.

(b) Unvorhergesehen Konsequenzen und Seiteneffekte machen Systeme allgemein kompliziert, und so auch Formeln. Deswegen war das Ziel prinzipiell, eine Formel zu konstruieren, bei der man durch Festlegung eines Wertes praktisch in eine noch nicht offensichtliche ,,Falle" läuft, d.h., die Formel wird dadurch unerfüllbar. Nach einigem Nachdenken habe ich dann auch gefunden, wie man derartige Formeln gut konstruieren kann: Es wird beispielsweise ausgegangen von ((A oder B) und (nicht A)). Daraus folgt direkt, dass A eben 0 sein muss, andernfalls ist die Formel unerfüllbar, da eben (nicht A) dann 0 wird. Davon ausgehend kann man dann via Distributivgesetzen und umgekehrten Absorptionsgesetzen (A == A oder Falsum) die Formel auf 3-CNF umgeformt werden. Damit ist es dann auch möglich, weitere solche Fallen in eine Formel einzubauen oder auch weitere solche Fallen zu suchen, um eine Formel dann komplizierter zu gestalten.

$(-1, 5, 6)$	$(-7, -6, -16)$
$(-2, 7, -6)$	$(1, 5, 6)$
$(6, 8, 12)$	$(-7, -6, 15)$
$(6, 8, -12)$	$(2, 7, -6)$
$(-7, -6, 15)$	

Fazit: *Fallen* sind eine gute Idee, aber die Formel selber ist zu klein, um nicht schnell per Hand durchgetestet werden zu können. Wenn Variable wie x_{15} nur in einer Form – hier nur positiv – vorkommen, helfen sie wenig, um das Finden einer Lösung zu erschweren.

3. Überlegungen:

   (a) kein Literal darf nur positiv oder negativ vorkommen,

   (b) kein Literal darf nur einmal vorkommen,

   (c) keine zwei Klauseln dürfen einen (offensichtlichen) Widerspruch darstellen,

   (d) die Formelanzahl sollte (für Menschen) möglichst groß sein,

   (e) keine Formel sollte sich wiederholen,

   (f) die Veränderung eines Literals sollte möglichst viele Änderungen anderer Literale nach sich ziehen,

   (g) gleiche Paarungen von Literalen sollten vermieden werden.

$(1, 4, 5)$	$(-6, 9, -1)$
$(7, -4, -9)$	$(-13, -16, 15)$
$(14, -8, -7)$	$(-2, -7, -8)$
$(-5, -14, -15)$	$(3, 2, 6)$
$(13, 10, -3)$	$(11, -12, -10)$
$(-2, 5, 12)$	$(-11, 7, -14)$

4. Eine meine möglichst schwer zu testende"3-SAT-Formel mit 16 Variablen und 49 Klauseln:

$(-4, 5, -1)$	$(10, -12, -7)$	$(-1, 3, 16)$
$(14, 5, -3)$	$(2, 3, -3)$	$(-3, 15, -9)$
$(-4, -2, 12)$	$(9, 3, 9)$	$(4, 15, -13)$
$(-5, 2, -4)$	$(-6, -15, -2)$	$(13, 13, 13)$
$(-4, -11, -2)$	$(-1, 7, 14)$	$(6, 16, 4)$
$(16, -15, 15)$	$(10, 16, -10)$	$(12, -4, 8)$
$(11, 11, -16)$	$(16, -12, -14)$	$(-16, 11, -11)$
$(16, -15, 15)$	$(5, 8, -13)$	$(-10, 7, -10)$
$(-6, -6, -7)$	$(-9, -14, -8)$	$(-3, -2, -7)$
$(15, 4, 8)$	$(10, 6, 8)$	$(5, -7, -6)$
$(15, -7, -6)$	$(-8, -7, -3)$	$(4, -11, -16)$
$(-13, 11, -10)$	$(-1, 9, 13)$	$(14, 3, -1)$
$(-8, 2, 12)$	$(-15, -9, 12)$	$(-11, -1, 9)$
$(-13, -6, -12)$	$(2, -14, -13)$	$(-14, 2, -1)$
$(-9, 9, -14)$	$(-5, -10, 12)$	$(1, 6, -10)$
$(16, 14, 11)$	$(-15, 5, -7)$	$(5, 5, -12)$
$(8, 8, 1)$		

   Meine Strategie war folgende:
   Ich habe mich für eine nicht erfüllbare Formel entschieden, weil solche Formeln mit "Brute Forc" und "Random Walk" bereits maximale Zeit benötigen und es mir auch mit anderen Algorithmen intuitiv schwerer erscheint, solche Formeln zu entscheiden.
   Um es besonders schwer zu machen, habe ich mir gedacht, dass es gut sei eine möglichst kurze Formel zu finden (weil längere Formeln mit höherer Wahrscheinlichkeit unerfüllbar sind und dies dabei leichter herauszufinden ist) und gleichzeitig die Anzahl der wiederkehrenden Variablen-Paare in verschiedenen Klauseln zu minimieren und damit die Abhängigkeiten zwischen den Klauseln zu minimieren, die ausgenutzt werden könnten.
   Also habe ich ein kleines Programm geschrieben, das folgenden Algorithmus ausführt:

   (a) generiere eine 3-CNF-Formel aus einer Permutation von Literalen, bei der jede Variable so wenig wie möglich, aber mindestens vier Mal vorkommt, weil erst dann die Möglichkeit besteht, dass die Formel unerfüllbar ist. (Aus Berechnungszeitgründen ist dies im Programm neun Mal)

   (b) prüfe mit einem externen SAT-Solver, ob die Formel unerfüllbar ist

   (c) wenn ja, berechne ein Rating für die Formel, das davon abhängt, wie oft gleiche Variablen-Paare auftauchen

   (d) gib nach einer festgelegten Anzahl von Tests die Formel mit dem niedrigsten Rating aus.

   Das Python3-Programm habe ich angehängt. Angegeben ist die beste innerhalb von 10000 Durchläufen gefundene Formel mit 49 Klauseln.

Aufgabe 2.2 *ℓ-3SAT sei die Einschränkung auf Formeln, bei denen jedes Literal höchstens ℓ-mal vorkommt. Gesucht ist eine Reduktion von 3SAT auf ℓ-3SAT mit einem möglichst kleinen ℓ:*

Antwort 1:
Fall 1: Die 3-Sat-Formel ist unerfüllbar.
Dann benötigt man mind. 8 Klauseln, um alle 8 Belegungen einer Variablensubmenge der Größe 3 auszuschließen. In diesem Fall ist also k = 8.
Fall 2: Die 3-Sat-Formel ist erfüllbar.
Im schlimmsten Fall besitzt die Formel genau eine erfüllende Belegung. Dann muss jede Variablensubmenge der Größe 3 genau eine erfüllende Belegung besitzen. Dies kann erzwungen werden, indem für jede Variablensubmenge der Größe 3 sieben Klauseln erstellt werden, in der alle Kombinationen vorkommen bis auf das Gegenteil der gewollten Belegung. Es lässt sich sagen, dass das Vorkommen der Variablen in jeder erfüllbaren Formel auf 7 erniedrigt werden kann.

Problem bei dieser Antwort: die Erfüllbarkeit der gegebenen Formel muß erst entschieden werden, wozu uns keine effizienten Algorithmen zur Verfügung stehen. Damit ist dies keine Polynomialzeit-Reduktion.

Lösung: man kann $\ell = 9$ erreichen, indem man jedes Vorkommen eines Literals z durch eine neue Variable z_i ersetzt und die Bedingungen $z_1 \equiv z_2\,,\ z_2 \equiv z_3\,,\ \ldots$ hinzufügt, d.h. $z_1 \vee \bar{z_2}\ \wedge\ \bar{z_1} \vee z_2$ usw. und die neuen Klauseln durch weitere Variable y auf Länge 3 bringt. Jedes solche z_i kommt damit genau $1 + 2 * 2$ oft vor, jede Füllvariable y wird nur zweimal benötigt.

2.2 Balanzierte Formeln

Frage: wie erzeugt man zufällige k-SAT Formeln für eine vorgegebenen Zahl von Variablen und Klauseln?

Eine einfache Möglichkeit, m Klauseln zu generieren, ist es, m-mal aus der Menge aller zulässigen Klauseln über der gegebenen Variablenmenge eine Klausel zufällig und uniform zu ziehen. Ob mit oder ohne Zurücklegen ist dabei relativ unbedeutend, so lange m klein im Verhältnis zu der Anzahl aller möglichen Klauseln ist. Bei dieser Wahl tritt jedoch das Problem auf, daß die Literale nicht gleichmäßig häufig auftreten. Bei einer balanzierten Formel wird dies jedoch durch einen anderen Auswahlprozeß sichergestellt.

Resultate hierzu, insbesondere über die sogenannte **Erfüllbarkeits-Threshold** finden sich in [BDI05], siehe auch [KKK98].

2.3 Lösungsvorschläge schwere SAT-Instanzen 2. Runde

Zur Erinnerung: Gesucht ist eine erfüllbare 3-Sat-Formel über 12 Variablen, die von Menschen auf Erfüllbarkeit möglichst schwer zu überprüfen ist.

1. Ich habe ein Programm geschrieben, dass eine 3-Sat-Formel für eine gegebene Belegung generiert, so dass die Belegung eine erfüllende ist. Die Klauseln sind zufällig und begrenzt auf eine Anzahl von 60, so dass jede Variable im Schnitt 10 mal vorkommt.

$(-5, 6, 7)$	$(4, -6, -12)$	$(-1, 5, 9)$
$(3, -8, 12)$	$(5, -9, 12)$	$(-7, -10, -11)$
$(3, -6, 10)$	$(-2, 5, 6)$	$(3, -8, -9)$
$(-6, 8, 11)$	$(-1, 2, 7)$	$(3, 9, -10)$
$(2, 7, 10)$	$(3, 7, -9)$	$(-5, 9, -12)$
$(8, -10, -12)$	$(-3, 4, 8)$	$(8, -9, 12)$
$(-5, -8, -11)$	$(2, 3, 5)$	$(1, 6, -11)$
$(-1, 5, 11)$	$(5, -7, 12)$	$(2, -3, -9)$
$(4, 8, -12)$	$(2, 4, -9)$	$(-4, 8, 10)$
$(-4, -5, -8)$	$(5, 10, 12)$	$(-7, 8, 10)$
$(1, 2, -12)$	$(6, -7, 8)$	$(-1, -5, 7)$
$(-3, 9, -11)$	$(-5, 7, 10)$	$(5, 8, 9)$
$(-6, -11, -12)$	$(2, -6, 10)$	$(6, 9, 10)$
$(2, -6, -7)$	$(-2, -5, -10)$	$(3, 11, -12)$
$(1, -5, -12)$	$(1, 3, 10)$	$(-4, -7, -12)$
$(-1, -4, -10)$	$(6, -8, 9)$	$(-2, -5, -12)$
$(6, -8, -10)$	$(-7, -9, -12)$	$(1, 4, 9)$
$(-6, -7, 10)$	$(-1, -7, -8)$	$(-5, -10, 11)$
$(3, 4, 6)$	$(3, -8, 10)$	$(5, 6, 12)$
$(4, 5, -7)$	$(-1, 2, -12)$	$(-1, 2, -11)$

Anzahl erfüllende Belegungen 7 :	001101010100
	011010111000
	011101010001
	01110101010x
	011101011x01

2. Das m, also die Klauselzahl, ergab sich durch folgende Überlegung: Bei den ersten Versuchen wurden Variablen anhand der Vorkomnisse gewichtet, d.h., die Anzahl der Vorkomnisse einer Variablen enthielt relativ viel Information. Daher wurde hier darauf geachtet, dass jede Variable genau 2x positiv und 2x negativ vorkommt. Damit ergeben sich 2*2*12 = 48 Literale und somit 48/3 = 16 Klauseln.

Ansonsten wurde mit einer randomisierten Belegung angefangen und dann eine Art lokale Suche gestartet, d.h., es wurden 2 Literale heuristisch getauscht, eine Belegung gesucht und wenn in der Belegung weniger Variablenwerte egal waren, wurde die neue Formel akzeptiert, bis die Anzahl Belegungen akzeptabel klein war.

$(7, 2, 5)$	$(-2, -1, -11)$	$(3, 12, 8)$
$(4, -3, -5)$	$(-5, 1, -2)$	$(6, -12, -9)$
$(-7, -6, -4)$	$(8, 6, -10)$	$(9, 11, 4)$
$(10, 1, 2)$	$(11, 10, 12)$	$(12, -3, 5)$
$(3, -4, -7)$	$(-3, 7, 1)$	$(-1, -6, -9)$
$(-8, -10, -11)$		

Anzahl erfüllende Belegungen: 256 , **Beispiel**: 1 1 1 1 1 0 1 1 0 1 0 x

3. **3-Sat als Spiel zwischen Erzeuger und Löser**
Wenn der Algorithmus A_{erzeug} bekannt ist, der eine erfüllbare 3-Sat-Formel ohne Zufallselemente erzeugt, so kann ein Algorithmus A_{loes} möglicherweise Informationen über A_{erzeug} nutzen, so dass die Formel leichter evaluiert werden kann.
Wenn andersrum A_{loes} bekannt ist, so kann man A_{erzeug} so modifizieren, dass dies für A_{loes} schwierige Lösungen erzeugt.

Schwierige Formeln
Wenn man den Belegungsbaum betrachtet der zu einer möglichst schwierigen 3-Sat-Formel gehört, so sollte dieser möglichst wenige erfüllende Belegungen haben, da dann die Wahrscheinlichkeit sinkt, die richtige Belegung zu finden. Das die Formel für andere Belegungen nicht erfüllt ist, sollte erst spät oder gar in den Blattknoten ersichtlich sein, so dass der Baum prinzipiell sehr tief/vollständig durchsucht werden muss. Ist ein Literal in vielen Klauseln enthalten, so nimmt die Höhe des Teilbaums für den Ast der erfüllenden Belegung des Literals ab.

Finden einer Formel
Mit Computer für $r = 12, m = 30$: Um eine Formel zu finden, die möglichst wenige erfüllende Belegungen hat, habe ich ein Java-Programm geschrieben, was zufällig Formeln generiert und nach erfüllenden Belegungen sucht.

Das Ergebnis ist:

$(-8, 3, 4)$	$(-8, 1, -3)$	$(-9, -2, -6)$
$(3, 8, 6)$	$(5, -12, 1)$	$(-6, 12, -5)$
$(5, -1, 2)$	$(-2, -5, -6)$	$(5, -10, -11)$
$(12, 7, -2)$	$(12, 9, -6)$	$(-10, -8, 9)$
$(-1, -8, -4)$	$(-3, 11, 2)$	$(-3, 8, -1)$
$(11, -4, 5)$	$(8, -2, 1)$	$(-9, -11, 1)$
$(-7, 12, -3)$	$(-7, 11, -1)$	$(11, -10, -9)$
$(1, 7, -9)$	$(4, -5, 3)$	$(-11, -1, -3)$
$(2, 1, -7)$	$(-12, -3, 11)$	$(9, 3, 2)$
$(-11, 10, -12)$	$(7, 12, -2)$	$(1, -3, 8)$

Anzahl erfüllende Belegungen 12 : $0101x0110010$
010110010001
$01011011000x$
$01011011100x$
100111001001
$100111x01111$
$110001000x01$

Überlegung für $r = 12$, beliebiges m
Nimmt man alle Kombinationen von Literalen und mischt diese, so hat man eine komplexe, aber nicht erfüllbare Formel. Man wäle sich eine Belegung und streiche alle nicht erfüllten Klauseln. Die verbleibende Formel ist nur für die eine gewählte Belegung erfüllt, sonst nicht.

Es gibt dann $m = \underbrace{\binom{12}{3}}_{Kombinationen} \cdot \underbrace{2^3}_{Vorzeichen} - \underbrace{\binom{12}{3}}_{Unerfüllte} = 1540$ Klauseln.

Beispiel für den Fall *1=2=3=4=5=6=7=8=9=10=11=12=true*:
[hier nicht wiedergegeben, da sehr lang]
In den Klauseln kommt jedes der Literale mindestens einmal nicht negiert vor.

4.

$(-4,-12,-9)$	$(4,3,8)$	$(-11,5,-7)$
$(-7,6,9)$	$(-8,7,-3)$	$(-9,-8,-11)$
$(10,2,-12)$	$(12,-4,-1)$	$(3,6,-10)$
$(12,-6,7)$	$(-5,-6,4)$	$(-3,5,8)$
$(11,2,9)$	$(-1,-5,-2)$	$(-2,10,1)$
$(-10,11,1)$		

Anzahl erfüllende Belegungen 199 : **Beispiele** : $000000x11000$
$111000111x0x$
$11100111xx0x$
$111101110x01$

Die Formel enthält 16 Klauseln und die Begründung dafür ist folgende: Es ist umso schwieriger, eine Belegung für eine Variable zu finden, je ausgeglichener sie als Literal positiv und negativ auftritt. Also sollte die Anzahl der Vorkommnisse jeder Variable gerade sein.

Bei nur zwei Vorkomnissen wäre die Belegung zu einfach zu erraten, deshalb habe ich vier gewählt, da dies schwer genug erscheint. Es gibt 12 Variablen, die jeweils viermal vorkommen, also gibt es 48 Literale = 16 Klauseln.

Bei der Konstruktion der Formel wurde darauf geachtet, dass jede Variable genau zweimal positiv und zweimal negativ vorkommt.

Weiterhin, um die transitiven Beziehungenßweier Variable möglichst kompliziert zu machen, habe ich darauf geachtet, dass jedes Variablenpaar innerhalb einer Klausel in der Formel möglichst selten vorkommt. Dadurch ist es schwer, einen Wiederspruch bei einer Variablenbelegung zu erkennen. Ganz ließen sich Wiederholungen leider nicht vermeiden (z.B. für 2,10 und 9,10).

5.

$(1, 5, -9)$	$(-8, 12, 7)$	$(-2, 9, 10)$
$(3, -16, 11)$	$(-5, -10, -11)$	$(2, 7, 11)$
$(12, 13, 14)$	$(4, -9, 13)$	$(9, -10, -16)$
$(1, 5, 9)$	$(7, -11, -14)$	$(3, -7, 9)$
$(-2, 8, -9)$	$(-1, -9, -14)$	$(5, -12, 16)$
$(-3, -5, 8)$	$(4, 8, -15)$	$(3, 7, -10)$
$(5, 6, 9)$	$(1, 8, -16)$	$(4, 7, 10)$
$(10, -13, 16)$	$(-7, 9, 15)$	$(-6, 8, -10)$
$(-1, 8, -11)$	$(9, -13, -15)$	$(-2, -6, 9)$
$(-3, -4, -10)$	$(-3, -9, 10)$	$(6, -10, -14)$

Das ist meine formel. sie hat wieder 30 klauseln; ich wollte die formel groß halten damit sie möglichst schwer auf den ersten blick zu lösen ist. jede variable kommt zwischen 6 und 8mal vor und ihc habe drauf geachtet dass
$|\#$ negative vorkommen $-\#$ positive vorkommen $| \leq 1$ ist.

Desweiteren kommt keine paarung von 2 variablen in einer klausel mehr als 2mal vor, aber jede variable *sieht* jede andere mindestens einmal.

16 Variablen mit 882 erfüllenden Belegungen

Beispiele :

$0000101110101 0x1$
$0000101110101 1x1$
$0000101110111 0x1$
$0000101110111 1x1$
$0000111110101 0x1$
$000011111010 1xx1$
$000011111011 1xx1$
$000011111100 1xx0$
$000011111101 1xx0$
$0000101111 0x10x0$

6. Die folgende Formel ist sehr ausgewogen bzgl. der Vorkommnisse jeder Variablen und ihrer Negation – jedes kommt genau 5 mal vor – und besitzt nur genau 1 Lösung. Dennoch kann man diese schnell finden, da jede Klausel mindestens ein

positive Literal besitzt. Setzt man alle Variablen auf `true`, dann wird offensichtlich jede Klausel erfüllt.

$(1,4,5)$ $(1,-2,-7)$ $(8,11,12)$
$(2,3,6)$ $(4,-3,-9)$ $(7,9,10)$
$(5,-6,-10)$ $(8,-2,-10)$ $(11,-3,-9)$
$(12,-6,-7)$ $(2,-4,-5)$ $(3,-5-8)$
$(6,-8,-11)$ $(7,-11-12)$ $(9,-12,-1)$
$(10,-1,-4)$ $(9,5,-7)$ $(10,8,-9)$
$(1,-8,-3)$ $(4,-11,-6)$ $(5,-12,-7)$
$(8,-1,-9)$ $(11,-4,-10)$ $(12,-5,-2)$
$(2,-4,-7)$ $(3,-5,-9)$ $(6,-8,-10)$
$(7,-11,-2)$ $(9,-12,-3)$ $(10,-1,-6)$
$(1,6,-4)$ $(4,9,-5)$ $(5,2,-8)$
$(8,10,-11)$ $(11,3,-12)$ $(12,7,-1)$
$(2,11,-10)$ $(3,12,-2)$ $(6,1,-3)$
$(7,4,-6)$

Einzige Lösung: 1 1 1 1 1 1 1 1 1 1 1 1

7. Es sollte also immer Klauseln geben, die nur negative Literale enthalten, und ebenso solche mir nur positiven Literalen. Wir können dies für die obige Formel dadurch erreichen, daß wir einige Variable durch ihre Negation ersetzen, so daß nun die eindeutige erfüllende Belegung sowohl `true-` als auch `false`-Werte besitzt.

$(1,4,5)$ $(1,2,7)$ $(8,11,12)$
$(-2,-3,-6)$ $(4,3,9)$ $(-7,-9,-10)$
$(5,6,10)$ $(8,2,10)$ $(11,3,9)$
$(12,6,7)$ $(-2,-4,-5)$ $(-3,-5-8)$
$(-6,-8,-11)$ $(-7,-11-12)$ $(-9,-12,-1)$
$(-10,-1,-4)$ $(-9,5,7)$ $(-10,8,9)$
$(1,-8,3)$ $(4,-11,6)$ $(5,-12,7)$
$(8,-1,9)$ $(11,-4,10)$ $(12,-5,2)$
$(-2,-4,7)$ $(-3,-5,9)$ $(-6,-8,10)$
$(-7,-11,2)$ $(-9,-12,3)$ $(-10,-1,6)$
$(1,-6,-4)$ $(4,-9,-5)$ $(5,-2,-8)$
$(8,-10,-11)$ $(11,-3,-12)$ $(12,-7,-1)$
$(-2,11,10)$ $(-3,12,2)$ $(-6,1,3)$
$(-7,4,6)$

Einzige Lösung: 1 0 0 1 1 0 0 1 0 0 1 1

8. Permutiert man zusätzlich eine Indizes, um die Negationen zu verschleiern, so kann

man beispielsweise die folgende Formel erhalten:

$(1, 2, 9)$ $(-1, -9, 12)$ $(8, 11, 12)$
$(5, 6, 7)$ $(-2, -3, -6)$ $(-7, -9, -10)$
$(3, 4, 10)$ $(2, 10, 11)$ $(6, 7, 8)$
$(3, 9, 12)$ $(-2, -4, -5)$ $(-4, -6 - 11)$
$(-3, -8, -11)$ $(-8, -9 - 12)$ $(-1, -7, -12,)$
$(-1, -5 - 10)$ $(4, -7, 9)$ $(7, -10, 11)$
$(1, 6, -11)$ $(3, 5, -8)$ $(4, 9, -12)$
$(-1, 7, 11)$ $(-5, 8, 10)$ $(2, -4, 12)$
$(-2, -5, 9)$ $(-4, -6, 7)$ $(-3, 10, -11)$
$(2, 8 - 9)$ $(6, -7, -12)$ $(-1, 3 - 10)$
$(1, -3, -5)$ $(-4, 5, -7)$ $(-2, 4, -11)$
$(-8, -10, 11)$ $(-6, 8, -12)$ $(1, 4, 5)$
$(-2, 8, 10)$ $(2, -6, 12)$ $(1, -3, 6)$
$(3, 5, -9)$

2.4 Randomisierte Algorithmen für 3-SAT

Um ein erfüllende Belegung zu finden, kann man stochastische Suchverfahren verwenden, die mit Hilfe von lokalen Kriterien Variablenbelegungen abändern, um so eine erfüllende Belegung zu finden. Die Vorgehensweise wird näher in [HSS07] und [S07] beschrieben.

3 Das Traveling-Salesman-Problem TSP

Definition 3.1 *Das TSP-Problem: lineare – quadratische Repräsentation*

Reduktion TSP – S-TSP (= symmetrisches TSP auf ungerichteten Graphen)

3.1 Das Schnürband-Problem

TSP in der 2-dimensionalen Ebene mit einer besonderen Konfiguration der Knoten: Ösen eines Schuhs.

3.2 TSP-Varianten

MAX-TSP

Gesucht ist eine Roudntour, deren Kosten maximal ist:
Reduktion auf TSP durch Negierung der Gewichte.

Bottleneck-TSP

Gesucht ist eine Rundtour, deren maximale Kosten einer Kante minimal ist:
Reduktion auf TSP durch exponentielles Aufblähen der Kosten

TSP mit Mehrfachbesuch von Knoten

Eine Tour darf Knoten mehr als einmal besuchen:
ersetze Kosten einer Kante (v_i, v_j) durch die Kosten eines kürzesten Weges von v_i nach v_j

TSP-Pfad: TSPP

Verallgemeinertes TSP: GTSP

Multiple-TSP: M-TSP

m Vertreter, die alle im Startknoten v_1 starten, teilen sich die Knoten auf, jeder Vertreter besucht seine Knotenmenge auf einer Rundtour und kehrt dann zum Startknoten zurück; Ziel: minimiere die Summe der Kosten der einzelnen Touren

Mannschafts-TSP

Die Knoten sind aufgeteilt in eine Menge von zwingenden Knoten und eine Menge von Aufteilknoten. Jeder der m Vertreter muß alle zwingenden Knoten besuchen in beliebiger Reihenfolge [alternativ in fest vorgegebener Reihenfolge], die übrigen Knoten teilen sich die Vertreter auf.

Zeitabhängiges TSP

Traveling-Repairman-Problem TRP

Black-and-White-TSP

Winkel-TSP

3.3 Anwendungen von TSP

Job-Scheduling

Flow-Shop-Problem

Definition 3.2 *Gegeben sind n Jobs $J_1, \ldots, J_n$,von denen jeder nacheinander ohne Wartezeit auf einer Folge von m Maschinen $M_1, \ldots, M_m$ bearbeitet werden muß. Es sei $p_{i,r}$ die Ausführungszeit von J_i auf M_r .*
Aufgabe: minimiere die totale Ausführungszeit!

Dies Problem ist $\mathcal{NP}$ -hart im strengen Sinn, aber in $\mathcal{P}$ für $m = 2$.

3.4 IP-Formulierung von TSP

Minimiere $\sum_{e=1}^{m} c_e x_e$ unter der Bedingung $X \in Polytop(\mathcal{T})$,
wobei m die Anzahl der Kanten sei und $X \in \{0,1\}^m$ eine zulässige Tour $\tau \in \mathcal{T}$ beschreibt als Auswahl der dabei benutzten Kanten. $Polytop(\mathcal{T})$ sei das m-dimensionale Polytop, das durch die Menge aller zulässigen Touren gebildet wird. Schwierigkeit, $Polytop(\mathcal{T})$ vollständig zu beschreiben.

Das Assignment-Problem
minimiere $\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$ unter den Bedingungen $\sum_{i=1}^{n} x_{ij} = 1$ für $j \in [1..n]$ und $\sum_{j=1}^{n} x_{ij} = 1$ für $i \in [1..n]$ sowie $x_{ij} \in \{0,1\}$. In diesem Fall beschreibt die Variable x_{ij} die Kante vom Knoten v_i nach v_j .

Eine Lösung des Assignment-Problems beschreibt eine vollständige Tour oder eine Menge von Subtouren, die jeden Knoten enthalten. Subtour-Elimination: zusätzliche Bedingungen, die Subtouren ausschließen.

exponentiell viele Bedingungen (natürliche Formulierungen): Clique-Packung-Constraints Kreis-Packung-Constraints

Polynomiell viele Bedingungen (kompakte Formulierungen):
Beispiel **MTZ-Constraints:**

$$(n-1)x_{ij} + u_i - u_j \leq n-2 \qquad \text{für alle } i,j \in [2..n]\,,$$

wobei u_i für $i \in [2..n]$ neue reelwertige Variable ohne Beschränkungen sind.

Lemma 3.1 *Beschreibt $X \in \{0,1\}^{n^2}$ eine vollständige Rundtour, dann gibt es Belegungen der u_i, so daß die MTZ-Constraints erfüllt sind.*
Zerfällt X dagegen in Subtouren, so sind diese Constraints nicht erfüllbar.

Beweis: Wähle $u_i =$ Zeitpunkt des Besuchs von v_i durch die Tour, wobei der Knoten v_1 als Startpunkt definiert wird. Dann gilt für alle Kanten x_{ij} der Tour: $u_j = u_i + 1$, d.h. die Ungleichung für i,j ist erfüllt. Für alle anderen Kanten gilt $x_{ij} = 0$ und $u_i - u_j \leq n-2$.

Zerfällt X in Subtouren, so sei $v_{i_1}, \ldots, v_{i_\ell}$ eine Subtour, die den Knoten v_1 nicht enthält. Addiert man alle zugehörigen Ungleichungen für die Paare $(i_1,i_2), \ldots, (i_\ell, i_1)$, so heben sich die u_i gerade weg. Wegen $\sum x_{i_k,i_{k+1}} = \ell$ ergibt sich die nicht zu erfüllende Ungleichung $(n-1)\cdot\ell \leq (n-2)\cdot\ell$.

3.5 RLP und GLP: rationale und ganzzahlige lineare Programmierung

grafische Repräsentation

Formulierung der Ungleichungen als Gleichungssystem mit Hilfe von Slack und Surplus-Variable

Relaxation: zunächst Lösung als ein RLP-Problem, anschließend durch Runden eine ganzzahlige Lösung erzeugen, die weiterhin die Nebenbedingungen erfüllt;
Beispiel, daß optimale Lösungen von GLP weit von optimalen Lösungen von RLP entfernt liegen können

4 Backtracking bei diskreten Optimierungsproblemen

Ein diskretes Optimierungsproblem Q wird beschrieben durch eine Menge $\mathcal{X}$ potentieller Lösungen sowie eine Funktion $\texttt{val} : \mathcal{I} \times \mathcal{X} \to \Re$, wobei $\mathcal{I}$ die Menge der möglichen Probleminstanzen (Eingaben) von Q ist. $\texttt{val}(I, X)$ ist der Wert des Optimierungsproblems bei Eingabe I und Lösungskandidat X. Die algorithmische Aufgabe besteht darin, für eine gegebene Probleminstanz I möglichst effizient ein oder alle X zu bestimmen, für die $\texttt{val}(I, X)$ maximal ist (bzw. minimal bei einem Minimierungsproblem). Manchmal ist man auch nur an dem maximalen Wert $\texttt{val}(I, X)$ interessiert, ohne X explizit zu kennen. Wächst der Lösungsraum $\mathcal{X}$ exponentiell in der Größe der Probleminstanz I, so ist ein vollständiges Durchsuchen extrem zeitaufwendig und für größere Instanzen praktisch nicht mehr durchführbar.

4.1 Das Rucksackproblem

Problemspezifikation:
Gegeben n Gegenstände mit Gewichten $w_0, \ldots, w_{n-1}$ und Nutzen $v_0, \ldots, v_{n-1}$ sowie eine Kapazitätsschranke M.
Aufgabe: bestimme eine Auswahl $X = (x_0, \ldots, x_{n-1}) \in \{0, 1\}^n$,
so daß $\texttt{val}(X) := \sum_i x_i \cdot v_i$ maximal wird unter der Nebenbedingung $\sum_i x_i \cdot w_i \leq M$.

Bei einem Lösungskandidaten X bedeutet $x_i = 1$, daß der i-te Gegenstand in den Rucksack aufgenommen wird. Dieser darf insgesamt mit nicht mehr als Gewicht M belastet werden. Die potentielle Lösungsmenge $\mathcal{X}$ umfäst also alle Teilmengen der n-elementigen Gütermenge. Ein Lösungskandidat ist zulässig, wenn seine Güter die Gewichtsgrenze nicht überschreiten. Eine Lösung des Rucksackproblems ist ein zulässiger Lösungskandidat X, dessen Bewertung $\texttt{val}(X)$ maximal ist unter allen zulässigen Kandidaten.

Algorithmus 1: KNAPSACK1 (ℓ)
global $X, \texttt{OptX}, \texttt{OptV}$;
if $\ell = n$
  then
    if $\sum_{i=0}^{n-1} w_i x_i \leq M$ **then**
      $\texttt{CurV} := \sum_{i=0}^{n-1} v_i x_i$ #[aktuelle Bewertung]#
      if $\texttt{CurV} > \texttt{OptV}$ **then**
        $\texttt{OptV} := \texttt{CurV}$;
        $\texttt{OptX} := [x_0, \ldots, x_{n-1}]$
  else
    $x_\ell := 1$;
    KNAPSACK1 $(\ell + 1)$;
    $x_l := 0$;
    KNAPSACK1 $(\ell + 1)$.

Algorithmus 1 berechnet eine optimale Lösung durch vollständiges Durchsuchen des Lösungsraumes exponentieller Größe (*exhaustive search*). Die Implementation geschieht durch durch eine rekursive Strategie, wobei

X die aktuelle partielle Lösung,

`OptX` die bislang beste gefundene Lösung,

`OptV` die Bewertung der bislang besten Lösung

bezeichnen. KNAPSACK1 (0) gibt dann in `OptX` eine global optimale Lösung zurück, wobei die globalen Variablen $X, \texttt{OptX}, \texttt{OptV}$ am Anfang mit 0 initialisiert werden.

4.2 Die allgemeine Backtracking-Strategie

Die allgemeine Strategie beim Backtracking läßt sich folgendermaßen beschreiben. Voraussetzung an die Problemstellung ist dabei, daß der Lösungsraum eine Darstellung als kartesisches Produkt $\mathcal{X}_0 \times \mathcal{X}_1 \times \ldots$ jeweils endlich vieler Alternativen $\mathcal{X}_\ell$ besitzt. $C_\ell \subseteq \mathcal{X}_\ell$ bezeichnet eine Auswahlmenge zu betrachtender Alternativen, die möglichst klein gehalten werden soll.

Algorithmus 2: BACKTRACK (ℓ)

global $X, C_\ell \quad \ell = 0, 1, \ldots;$
  #[C_ℓ mögliche Erweiterungen einer partiellen Lösung auf Stufe ℓ]#
if $[x_0, x_1, \ldots, x_{\ell-1}]$ ist zulässige partielle Lösung
  then process it;
  compute C_ℓ;
  for each $x \in C_\ell$ **do**
    $x_\ell := x$;
    BACKTRACK ($\ell + 1$).

Die Anwendung dieser Vorgehensweise auf das Rucksackproblem mit einer Beschränkung der Auswahlmenge durch die Bedingung, daß das Gewicht einer partiellen Lösung die Kapazitätsobergrenze nicht überschreiten darf, ergibt dann folgendes Verfahren, wobei die Variable `CurW` das Gewicht der aktuellen partiellen Lösung X beschreibt.

Durch den Test $\texttt{CurW} + w_\ell \leq M$ werden nur partielle Lösungen weiterverfolgt, die zu einer zulässigen Lösung führen können. Auf diese Weise kann der Suchraum unter Umständen massiv verkleinert werden.

Algorithmus 3: KNAPSACK2 $(\ell, \texttt{CurW})$

global $X, \texttt{OptX}, \texttt{OptV}, C_\ell \;\; (\ell = 0, 1, \ldots)$;
if $\ell = n$
    then
        $C_\ell := \emptyset$
        if $\sum_{i=0}^{n-1} v_i x_i > \texttt{OptV}$ **then**
            $\texttt{OptV} := \sum_{i=o}^{n-1} v_i x_i$;
            $\texttt{OptX} := [x_0, \ldots, x_{n-1}]$;
    else
        if $\texttt{CurW} + w_\ell \leq M$
            then $C_\ell := \{1, 0\}$
            else $C_\ell := \{0\}$;
for each $x \in C_\ell$ **do**
    $x_\ell := x$;
    KNAPSACK2 $(\ell + 1, \texttt{CurW} + w_\ell x_\ell)$.

4.2.1 Das Clique-Problem

Als ein weiteres Beispiel betrachten wir die Aufgabe, in einem Graphen $G = (V, E)$ alle vollständigen Subgrahen, auch **Cliques** genannt, zu finden.

Für einen Knoten $v \in V$ bezeichne

$$\mathbf{\Gamma(v)} := \{u \in V \mid \{v, u\} \in E\}$$

die Menge seiner direkten Nachbarn. Die Knoten in V seien durch eine Relation $\prec$ vollständig geordnet und $B(v) := \{u \in V \mid v \prec u\}$ die Menge der Knoten, die bzgl. dieser Ordnung größer als v sind.

Als Auswahlmenge C_ℓ wählen wir in diesem Fall also $C_0 := V$ und mit $C_\ell := C_{\ell-1} \cap \Gamma(x_{\ell-1}) \cap B(x_{\ell-1})$ die Teilmenge aller Knoten in $C_{\ell-1}$, die zu dem letzten ausgewählten Knoten $x_{\ell-1}$ benachbart sind und bzgl. der Ordnung größer als $x_{\ell-1}$.

Die Menge $N_\ell = \bigcup_{0 \leq j < \ell} \Gamma(x_j)$ enthält genau die verbleibenden Knoten aus V, die zu allen Knoten $x_0, \ldots, x_{\ell-1}$ benachbart sind und damit Kandidaten für eine Erweiterung der Clique bestehend aus den Knoten $x_0, \ldots, x_{\ell-1}$. Diese Clique ist daher genau dann maximal, d.h. nicht erweiterbar, wenn N_ℓ leer ist.

Algorithmus 4: ALLCLIQUES (ℓ)

global $X, C_\ell \;\; (\ell = 0, 1, \ldots, n-1)$;
#[every X generated by the algorithm is a clique]#
if $\ell = 0$
  then output [ ]
  else output $[x_0, \ldots, x_{\ell-1}]$;
if $\ell = 0$
  then $N_\ell := V$
  else $N_\ell := \Gamma(x_{\ell-1}) \cap N_{\ell-1}$;
if $N_\ell = \emptyset$ **then** $\{x_0, \ldots, x_{\ell-1}\}$ is a maximal clique;
if $\ell = 0$
  then $C_\ell := V$
  else $C_\ell := C_{\ell-1} \cap \Gamma(x_{\ell-1}) \cap B(x_{\ell-1})$;
for each $x \in C_\ell$ **do**
    $x_\ell := x$;
    ALLCLIQUES $(\ell + 1)$.

4.2.2 Das Problem EXACT-COVER

Gegeben eine Menge $\mathcal{S} = S_1, S_2, \ldots$ von Teilmengen über einer Grundmenge U , so heißt eine Auswahl $S_{i_1}, S_{i_2}, \ldots$ von Menge aus $\mathcal{S}$ eine **exakte Überdeckung**, wenn jedes Element aus U in genau einem der S_{i_j} enthalten ist. Ein Spezialfall ist die Verallgemeinerung von `3-DIM-MATCHING` auf das Problem X3C, wo jede Teilmenge S_i genau 3 Elemente enthält.

Dies Problem kann mit Hilfe des Clique-Problems gelöst werden. Betrachte dazu den Graphen $G = (V, E)$, dessen Knoten gerade die Mengen in $\mathcal{S}$ sind und zwei Knoten S_i und S_j aus G genau dann durch eine Kante verbunden sind, wenn $S_i \cap S_j = \emptyset$. Die maximalen Cliques in G sind dann gerade potentielle Lösungen von EXACT-COVER. Solch eine Clique ist eine zulässige Lösung, wenn jedes Element von U in einem Knoten der Clique vorkommt.

Die folgende Backtracking-Strategie berechnet alle exakten Überdeckungen. Dabei sei $U = \{0, \ldots, n-1\}$.

Algorithmus 5: EXACTCOVER (n, S)

global $X, C_\ell, C'_\ell \ \ (\ell = 0, 1, \ldots)$;

procedure EXACTCOVER-Backtrack (ℓ, r');
  if $\ell = 0$ **then**
    $U_0 := \{0, \ldots, n-1\}$;
    $r := 0$
  else
    $U_\ell := U_{\ell-1} \backslash S_{x_{\ell-1}}$;
    $r := r'$;
    while $r \notin U_\ell$ **and** $r < n$ **do** $r := r + 1$;
  if $r = n$ **then output** $[x_0, \ldots, x_{\ell-1}]$;
  if $\ell = 0$
    then $C'_0 := \{0, 1, \ldots, m-1\}$
    else $C'_\ell := \Gamma(x_{\ell-1}) \cap B(x_{\ell-1}) \cap C'_{\ell-1}$;
  $C_\ell := C'_\ell \cap H(r)$;
  for each $x \in C'_\ell$ **do**
    $x_\ell := x$;
    EXACTCOVER-Backtrack $(\ell + 1, r)$;

main
  $m := |S|$;
  sort S in decreasing lexicographic order;
  for $i := 0$ **to** $m-1$ **do** $\Gamma(i) := \{j \mid S_i \cap S_j = \emptyset\}$;
  for $i := 0$ **to** $m-1$ **do** $B(i) := \{i+1, i+2, \ldots, m-1\}$;
  for $i := 0$ **to** $n-1$ **do** $H(i) := \{j \mid S_j \cap \{0, \ldots, i\} = \{i\}\}$;
  $H(n) := \emptyset$;
  EXACTCOVER-Backtrack $(0, 0)$.

4.3 Backtracking mit einer Bounding-Funktion

Für ein gegebenes Optimierungsproblem Q heißt eine Funktion $\texttt{Bound}(Y)$ definiert auf den partiellen Lösungen Y von Q eine **Bounding-Funktion**, wenn für alle Probleminstanzen I und alle partiellen Lösungen Y gilt:

$$\texttt{val}(I, X) \ \leq \ \texttt{Bound}(Y) \qquad \text{für alle Erweiterungen } X \text{ von } Y\,.$$

Mit anderen Worten, in dem Teilbaum $T(Y)$, der alle möglichen Lösungskandidaten enthält, die Erweiterungen einer partiellen Lösung Y sind, kann kein größerer Wert als $\texttt{Bound}(Y)$ erzielt werden (bei einem Minimierungsproblem entsprechend kein kleinerer). Liegt diese Schranke unter dem aktuellen Optimum `OptV`, so können wir auf das Durchsuchen von $T(Y)$ verzichten. Dies führt zu dem folgenden beschleunigten Verfahren:

Algorithmus 6: BOUNDING (ℓ)

external `Value( ), Bound( )` ;
global X, `OptX`, `OptV`, C_ℓ $(\ell = 0, 1, \ldots)$;

if $[x_0, \ldots, x_{\ell-1}]$ ist zulässige Teillösung **then**
    $V :=$ `Value`$([x_0, \ldots, x_{\ell-1}])$;
    if $V >$ `OptV` **then**
        `OptV` $:= V$;
        `OptX` $:= [x_0, \ldots, x_{\ell-1}]$;
compute C_ℓ ;
for each $x \in C_l$ **do**
    $B :=$ `Bound`$([x_0, \ldots, x_{\ell-1}, x])$;
    if $B >$ `OptV` **then**
        $x_\ell := x$;
        BOUNDING $(\ell + 1)$.

4.3.1 Bounding-Funktion für das Rucksackproblem

Durch eine rationale Variante des Rucksackproblems erhalten wir eine Bounding-Funktion, die sich effizient berechnet läßt. Für den Fall, daß jedes Gut geteilt werden kann, d.h. $0 \leq x_\ell \leq 1$, kann das Optimum nämlich durch ein einfaches Greedy-Verfahren berechnet werden.

Algorithmus 7: RKNAP $(v_0, v_1, \ldots, v_{n-1}, w_0, w_1, \ldots, w_{n-1}, M)$

permutiere die Indizes so daß $v_{i-1}/w_{i-1} \geq v_i/w_i$ für alle $i < n$;
initialisiere $i := 0$; $V := 0$; $W := 0$;
for $j := 0$ **to** $n - 1$
  do $x_j := 0$;
while $W < M$ and $i < n$ **do**
    if $W + w_i \leq M$
      then
        $x_i := 1$;
        $W := W + w_i$;
        $V := V + v_i$;
        $i := i + 1$
      else
        $x_i := (M - W)/w_i$;
        $W := M$;
        $V := V + x_i \cdot v_i$;
        $i := i + 1$;
return (V) .

Verwendet man den optimalen Lösungswert des rationalen Rucksackproblems als Boundung-Funktion für das ganzzahlige Rucksackproblem, so ergibt sich das folgende Verfahren:

Algorithmus 8: KNAPSACK3 $(\ell, \texttt{CurW})$

external RKNAP( );
global $X, \texttt{OptX}, \texttt{OptV}, C_\ell \quad (\ell = 0, 1, \ldots)$
if $\ell = n$
  then
    $C_\ell := \emptyset$
    if $\sum_{i=0}^{n-1} v_i x_i > \texttt{OptV}$ **then**
      $\texttt{OptV} := \sum_{i=0}^{n-1} v_i x_i$:
      $\texttt{OptX} := [x_0, \ldots, x_{n-1}]$;
  else
    if $\texttt{CurW} + w_\ell \leq M$
      then $C_\ell := \{1, 0\}$
      else $C_\ell := \{0\}$;
for each $x \in C_\ell$ **do**
  $x_\ell := x$;
  $B := \sum_{i=0}^{\ell} v_i x_i + \text{RKNAP}(v_{\ell+1}, \ldots, v_{n-1}, w_{\ell+1}, \ldots, w_{n-1}, M - \texttt{CurW} - w_\ell x_\ell)$;
  if $B > \texttt{OptV}$ **then**
    KNAPSACK3 $(\ell + 1, \texttt{CurW} + w_\ell x_\ell)$.

Aufgabe 4.1 *Man konstruiere eine Probleminstanz, für die RKNAP keine optimale Lösung generiert.*

4.4 Backtracking für das Traveling Salesman Problem (TSP)

Gegeben sei ein Graph $G = (V, E)$ mit n Knoten und einer Kostenmatrix M für die Kanten, wobei $m[i, j] = \infty$, falls $\{i, j\} \notin E$.
Für einen Hamiltonschen Kreis in G mit Knotenfolge $x_0, x_1, \ldots, x_{n-1}, x_n = x_0$ sei

$$\texttt{Cost}([x_0, \ldots, x_{n-1}]) \;:=\; \sum_{i=0}^{n-1} m[x_i, x_{i+1}] \;.$$

Sucht man eine Knotenfolge mit minimalen Kosten, so kann diese durch das folgende Backtracking-Verfahren berechnet werden.

Algorithmus 9: TSP1 (ℓ)

global $C_\ell \;\; (\ell = 0, 1, \ldots, n-1)$;

case $\ell = n$ **then**
    `val` := `Cost`$([x_0, \ldots, x_{n-1}])$;
    if `val` $<$ `OptV` **then**
        `OptV` := `val`;
        `OptX` := $[x_0. \ldots, x_{n-1}]$
case $\ell = 0$ **then** $C_\ell := \{0\}$
case $\ell = 1$ **then** $C_\ell := \{1, \ldots, n-1\}$
    else $C_\ell := C_{\ell-1} \backslash \{x_{\ell-1}\}$;

for each $x \in C_\ell$ **do**
    $x_\ell := x$;
    TSP1 $(\ell + 1)$.

Für eine Teilmenge $W \subseteq V$ der Knoten und $x \in V$ sei

$$b(x, W) \;:=\; \min\{m[x, w] \mid w \in W\}$$

die kostenminimal Verbindung von x in die Knotenmenge W. Für einen Weg $X = [x_0, \ldots, x_{\ell-1}]$ in G definieren wir

$$\texttt{MINCOSTBOUND}(X) \;:=$$

$$\begin{cases} \displaystyle\sum_{i=0}^{\ell-2} m[x_i, x_{i+1}] \;+\; b(x_{\ell-1}, V \setminus X) \;+\; \sum_{v \in V \setminus X} b(v, \{x_0\} \cup V \setminus X) \;, & \text{falls } \ell < n \\ \displaystyle\sum_{i=0}^{\ell-2} m[x_i, x_{i+1}] \;+\; m[x_{n-1}, x_0] \;, & \text{falls } \ell = n \;. \end{cases}$$

Es ist leicht einzusehen, daß $\texttt{MINCOSTBOUND}(X)$ eine Bounding-Funktion darstellt. Daher können wir das Backtracking-Verfahren verbessern zu:

Algorithmus 10: TSP2 (ℓ)

```
external B( ) ;
global C_ℓ  (ℓ = 0, 1, ..., n − 1]) ;
if ℓ = n then
      C := Cost([x_0, ..., x_{n−1}])
      if C < OptV then
         OptV := C ;
         OptX := [x_0, ..., x_{n−1}] ;
case ℓ = 0 then C_ℓ := {0}
case ℓ = 1 then C_ℓ := {1, ..., n − 1}
           else C_ℓ := C_{ℓ−1} \ {x_{ℓ−1}} ;
for each x ∈ C_ℓ do
      B := MINCOSTBOUND([x_0, ..., x_{ℓ−1}, x])
      if B ≥ OptV then return ;
      x_ℓ := x ;
      TSP2 (ℓ + 1) .
```

4.4.1 Bounding-Funktion mit Hilfe reduzierter Kostenmatrizen

Eine andere Bunding-Funktion kann man auch folgende Weise erhalten. Wir definieren dazu eine **Reduktion einer Kostenmatrix** M. Die Einträge der Matrix werden zeilen- und spaltenweise um bestimmte Werte verringert, so daß jede Zeile und jede Spalte mindestens einen 0-Eintrag erhält, alle Einträge jedoch weiterhin nichtnegativ sind.
Der **Wert der Reduktion** `redval`(M) ist die Summe der Zahlen, um die die Zeilen- und Spalteneinträge verringert werden.

Algorithmus 11: REDUCE (M)

```
#[ M is a k × k matrix ]#
   redval := 0 ;
   for i := 0 to k − 1 do
      minrow := m[i, 0] ;
      for j := 1 to k − 1 do  minrow := min{minrow, m[i, j]} ;
      for j := 0 to k − 1 do  m[i, j] := m[i, j] − minrow ;
      redval := redval + minrow ;
   for j := 0 to k − 1 do
      mincol := m[0, j] ;
      for i := 1 to k − 1 do  mincol := min{mincol, m[i, j]} ;
      for i := 0 to k − 1 do  m[i, j] := m[i, j] − mincol ;
      redval := redval + mincol ;
   return redval .
```

Lemma 4.1 Jeder Hamiltonsche Kreis in einem Graphen G mit Kostenmatrix M hat Kosten mindestens $\texttt{redval}(M)$.

Beweis: Markiert man für einen Hamiltonschen Kreis in der Kostenmatrix genau die Kosten der Kanten des Kreises, so wird in jeder Zeile und in jeder Spalte genau ein Eintrag $m[i,j]$ markiert. Man sieht leicht, daß die Summe dieser Werte mindestens so groß sein muß wie $\texttt{redval}(M)$. ∎

Der folgende Algorithmus berechnet für eine Teillösung X, d.h. einen Weg in G eine Kostenfunktion für den Weg im Restgraphen und reduziert dann die sich daraus ergebende Kostenmatrix.

Algorithmus 12: REDUCEBOUND (X)

```
external Cost( ), REDUCE( );
global M, V;
#[ M ist die Kostenmatrix eines Graphen G mit Knotenmenge V der Größe n ]#
#[der Vektor X = [x_0, ..., x_{m-1}] ist ein Weg in G ]#
if m = n then return Cost(X);
#[Konstruktion einer neuen Matrix M', die die Kosten definiert,
      um X zu einem Kreis in G zu verlängern ]#
M'[0,0] := ∞;
j := 1;
for each y ∈ V\{x_0, x_1, ..., x_{m-1}} do
    M'[0,j] := M[x_{m-1}, y];
    j := j + 1;
i := 1;
for each x ∈ V\{x_0, x_1, ..., x_{m-1}} do
    M'[i,0] := M[x, x_0];
    i := i + 1;
i := 1;
for each x ∈ V\{x_0, x_1, ..., x_{m-1}} do
      j := 1;
      for each y ∈ V\{x_0, x_1, ..., x_{m-1}} do
            M'[i,j] := M[x,y];
            j := j + 1;
      i := i + 1;
redbou := REDUCE(M');
for i := 1 to m - 1 do
   redbou := redbou + M[x_{i-1}, x_i];
return redbou.
```

4.5 Das MAXCLIQUE-Problem

Wir wollen nun die maximale Größe einer Clique eines gegebenen Graphen berechnen. Dies Problem ist ebenfalls $\mathcal{NP}$ -hart. Die Suche soll wiederum mit Hilfe einer Bounding-Funktion beschleunigt werden.

Algorithmus 13: MAXCLIQUE1 (ℓ)

```
global C_ℓ  (ℓ = 0, 1, ..., n − 1);
if ℓ > OptV then
      OptV := ℓ + 1;
      OptX := [x_0, ..., x_{ℓ−1}];
if ℓ = 0
   then C_ℓ := V
   else C_ℓ := Γ(x_{ℓ−1}) ∩ B(x_{ℓ−1}) ∩ C_{ℓ−1};
for each x ∈ C_ℓ do
      x_ℓ := x;
      MAXCLIQUE1 (ℓ + 1);
main
   OptV := 0;
   MAXCLIQUE1 (0);
   output (OptX).
```

Das Abschätzen der maximalen Cliquen-Größe in einem Restgraphen geschieht mit Hilfe einer Knotenfärbung, d.h. jeder Knoten des Graphen bekommt eine Farbe, wobei benachbarte Knoten unterschiedlich gefärbt werden müssen. Ziel dabei ist es, möglichst wenig Farben zu verwenden. Kann man nämlich einen Graphen mit k Farben färben, so besitzt er offensichtlich keine Clique mit mehr als k Knoten.

Da das Färbungsproblem für Graphen im allgemeinen $\mathcal{NP}$ -hart ist, können wir eine optimale Färbung mit der Minimalzahl an Farben im allgemeinen nicht effizient berechnen. Stattdessen wird durch ein einfaches Greedy-Verfahren, welches die Färbungszahl approximiert. Der berühmte **Vierfarbensatz**, der als erstes mathematisches Theorem mit Hilfe eines Computerprogrammes bewiesen worden ist, besagt, daß man jeden planaren Graphen mit 4 Farben färben kann. Ob ein beliebiger Graph 2-färbbar ist, kann auf einfache Weise entschieden werden. Die Frage, ob 3 Farben ausreichen, ist dagegen bereits $\mathcal{NP}$ -vollständig.

Es sei G ein Graph mit Knotenmenge $V = \{v_0, \ldots, v_{n-1}\}$. Zur Färbung verwenden wir die Farben $0, 1, \ldots$. Die Menge `ColorClass`$[h]$ enthält alle Knoten v_i, die mit Farbe h gefärbt werden. Dies wird auch repräsentiert durch den Wert der Variablen `Color`$[v_i] = h$.

Algorithmus 14: GREEDYCOLOR (G)

```
global Color;
k := 0;
for i := 0 to n − 1 do
    h := 0;
    while h < k and Γ(v_i) ∩ ColorClass[h] ≠ ∅ do h := h + 1;
    if h = k then
                k := k + 1;
                ColorClass[h] := ∅;
        ColorClass[h] := ColorClass[h] ∪ {v_i};
        Color[v_i] := h;
return (k).
```

Bei Terminierung von GREEDYCOLOR mißt der Wert der Variablen k gerade die Zahl der benutzten Farben. Man findet leicht Beispiele, die zeigen, daß diese Zahl nicht immer optimal ist. Andererseits zeigt der Algorithmus, daß für einen Graphen mit maximalem Grad d, d.h. $|\Gamma(v_i)| \leq d$ für alle v_i, stets $d+1$ Farben ausreichen.

Wie oben bereits diskutiert, liefert die Farbzahl k eine Bounding-Funktion für die maximale Cliquengröße. Man kann nun entweder mit Hilfe von GREEDYCOLOR eine Färbung für den gesamten Graphen berechnen und dann bei einem Restgraphen zählen, wie viele verschiedene Farben in diesem Restgraphen vorkommen. Eine schärfere Bounding-Funktion ergibt sich, wenn man für den Restgraphen selber ein Färbung berechnet, denn diese wird in vielen Fällen mit weniger Farben auskommen.

4.6 Branch-and-Bound-Verfahren

Das Traversieren des Baumes der Lösungskandidaten kann bei einer gegebenen Bounding-Funktion noch weiter verbessert werden. Bislang haben wir die Söhne eines Knotens in beliebiger Reihenfolge betrachtet. Die **Branch-and-Bound-Strategie** für ein Maximierungsproblem dagegen untersucht die Subbäume eines Knotens in fallender Ordnung der Bounding-Werte. Daher kann unter Umständen das Betrachten von Subbäumen mit schlechteren Bounding-Werten entfallen, nämlich genau wenn ein Bruder mit einem besseren Bounding-Wert tatsächlich auch einen besseren Wert für die Zielfunktion liefert.

Algorithmus 15: BRANCHANDBOUND (ℓ)

```
external B( ), profit( );
global C_ℓ  (ℓ = 0, 1, ...);
if [x_0, ..., x_{ℓ-1}] is a complete solution then
        V := profit([x_0, ..., x_{ℓ-1}]);
        if V > OptV then
            OptV := V;
            OptX := [x_0, ..., x_{ℓ-1}];
compute C_ℓ;
count := 0;
for each x ∈ C_ℓ do
      nextchoice[count] := x;
      nextbound[count] := B([x_0, ..., x_{ℓ-1}, x]);
      count := count + 1;
simultaneously permute nextchoice and nextbound
   so that nextbound is in decreasing order;
for i := 0 to count − 1 do
      if nextbound[i] ≤ OptV then return ;
      x_ℓ := nextchoice[i];
      BRANCHANDBOUND (ℓ + 1).
```

Wendet man diese Vorgehensweise auf des Traveling-Salesman-Problem an – in diesem Fall ein Minimierungsproblem, so ergibt sich der folgende Algorithmus:

Algorithmus 16: TSP3 (ℓ)

```
external Sort ( ) , Cost ( ) ;
global C_ℓ  (ℓ = 0, 1, ..., n − 1) ;
if ℓ = n then
        C := Cost([x_0, ..., x_{n−1}]) ;
        if C < OptV then
            OptV := C ;
            OptX := [x_0, ..., x_{n−1}] ;
case ℓ = 0 then C_ℓ := {0}
case ℓ = 1 then C_ℓ := {1, ..., n − 1}
           else C_ℓ := C_{ℓ−1} \ {x_{ℓ−1}} ;
count := 0 ;
for each x ∈ C_ℓ do
      nextchoice[count] := x ;
      nextbound[count] := B([x_0, ..., x_{ℓ−1}, x]) ;
      count := count + 1 ;
simultaneously permute nextchoice and nextbound
   so that nextbound is in increasing order;
for i := 0 to count − 1 do
      if nextbound[i] ≥ OptV then return ;
      x_ℓ := nextchoice[i] ;
      TSP3 (ℓ + 1) .
```

4.7 Experimentelle Daten

Ausführung der verschiedenen Bachtracking-Verfahren auf zufällig generierten Probleminstanzen; gemessen wird die Größe des entsprechenden Suchraumes.
Beim Rucksackproblem werden die Gewichte und Nutzen der n Gegenstände in einem engen Bereich mit leichten Schwankungen zufällig generiert.

Rucksackproblem	Problemgröße		
Lösungsverfahren	Anzahl Gegenstände		
	8	16	24
A1: vollständige Suche	511	$1{,}3 \cdot 10^5$	$3{,}3 \cdot 10^7$
A3: Überprüfung der Kapazitätsschranke	312 - 333	$7,6 - 7,8 \cdot 10^4$	$1,7 - 1,9 \cdot 10^7$
A8: Bounding Funktion RKNAP	52 -78	58 - 601	180 - 755

Für das Traveling-Salesman-Problem werden die Kosten der einzelnen Kanten zufällig gewählt.

Traveling-Salesman	Anzahl Städte			
Lösungsverfahren	5	10	15	20
A12: Bounding mit				
MINCOST BOUND	45	5200	$1{,}5 \cdot 10^6$	$6{,}4 \cdot 10^7$
REDUCE BOUND	18	1287	$5{,}3 \cdot 10^4$	$1{,}3 \cdot 10^6$
A16: Branch-and-Bound mit				
MINCOST BOUND	25	490	$1{,}2 \cdot 10^5$	$6{,}1 \cdot 10^6$
DEDUCE BOUND	9	102	5078	$3{,}9 \cdot 10^4$

Beim Cliquen-Problem wird ein Graph G mit n Knoten zufällig generiert; die Wahrscheinlichkeit, daß zwei Knoten von G durch eine Kante verbunden werden, beträgt 0,75 und ist unabhängig von der Existenz anderer Kanten.

Clique	Grapheigenschaften		
Anzahl Knoten	25	50	100
Anzahl Kanten	236	959	3720
Maximale Clique-Größe	11	14	17
Bounding Funktion			
keine	$2{,}5 \cdot 10^4$	$2 \cdot 10^6$	$1{,}8 \cdot 10^8$
Größenschranke	1840	$9{,}1 \cdot 10^4$	$5{,}3 \cdot 10^6$
GREEDY COLOR	91	2843	$7 \cdot 10^4$

5 Randomisierung und probabilistische Analyse

5.1 Randomisierte Algorithmen

Wir beginnen mit einem Routing-Problem. Gegeben sei ein Kommunikationsnetzwerk G, bei dem jeder Knoten v in einem Zeittakt maximal ein Datenpaket von einem seiner Nachbarn entgegennehmen und dann an einen anderen Nachbarn weitersenden kann oder es behalten, wenn es für ihn bestimmt ist. Treffen mehrere Pakete gleichzeitig ein, so werden sie zwischengepuffert, bis der Knoten diese Schritt für Schritt verarbeiten kann.

Definition 5.1 Packet-Routing in Netzwerken
Jeder Knoten v will ein Paket an einen anderen Knoten schicken. Ziel ist es, alle Pakete möglichst schnell zuzustellen, d.h. die maximale Laufzeit eines Paketes soll minimiert werden. Die einzelnen Wege der Pakete sollten daher so gewählt werden, daß sie sich möglichst wenig gegenseitig behindern.

Um Trivialfälle von Überlastungen auszuschließen, nehmen wir an, daß die Zuordnung Absender–Empfänger der Pakete eine Bijektion auf der Knotenmenge bildet, ansonsten könnte bereits ein einzelner Knoten sehr viel Zeit benötigen, wenn eine hohe Anzahl an Paketen ihm zugesendet werden.

Die Entscheidung, welcher Wege für ein Pakete gewählt wird, soll aber nicht zentral vorherberechnet werden, sondern lokal in den einzelnen Knoten nur anhand der Empfängeradresse und somit unabhängig von den Zieladressen anderer Pakete. Eine derartige distributive Routing-Strategie heißt **oblivious**.

Wichtige Parameter für die benötigte Zeit, um dies Routing-Problem zu lösen, sind offensichtlich der Durchmesser des Netzwerkes – die maximale Entfernung zwischen zwei Knoten – sowie die maximale Belastung einer Kante e, die Zahl der Pakete, die die Kante e benutzen.

Man kann eine worst-case untere Schranke $\Omega(\sqrt{n/d})$ zeigen für jede deterministische oblivious Routing-Strategie in Netzwerken der Größe n mit Knotengrad maximal d.

Durch Zufallsentscheidungen bei der Wahl der Wege für die Pakete wollen wir versuchen, solch ein worst-case Szenario mit hoher WS auszuschließen. Betrachten wir dazu das konkrete Problem des Routing in einem $\boldsymbol{d}$**-dimensionalen Hyperwürfel** $\boldsymbol{\mathcal{H}_d}$. Jeder Knoten hat als Adresse einen Binärstring der Länge d und Knoten, die sich an genau einer Bitposition unterscheiden, sind durch eine Kante verbunden. $\mathcal{H}_d$ besitzt 2^d Knoten mit gleichem Grad d und somit $2^d \cdot d/2$ Kanten.

Die **Bit-Fixing-Routing-Strategie** geht in $\mathcal{H}_d$ wie folgt vor. Der Reihe nach wird für die Bitpositionen, an denen sich Absender- und Empfängeradresse unterscheiden, das Paket über die Kante dieser Dimension geschickt. Diese Strategie ist offensichtlich oblivious,

und man kann leicht Beispiele konstruieren, bei denen ihre Laufzeit die untere Schranke $\Omega(\sqrt{n/d}) = \Omega(\sqrt{n/\log n})$ erreicht.

Valiants 2-Phasen Routing an eine zufällig gewählte Zwischenadresse

Aufgabe 5.1 *Konstruiere eine Permutation, so daß die Bit-Fixing-Strategie möglichst viel Zeit benötigt.*

5.2 Probabilistische Auswahlen

Wir präsentieren zunächst einige elementare Ergebnisse über probabilistische Prozesse.

Definition 5.2 *Coupon-Collector Problem Es gebe n verschiedene Arten von Coupons und von jeder Art unbeschränkt viele. Ein Zufallsexperiment generiert eine Folge von Coupons $c_1, c_2, \ldots$, wobei bei jedem c_j die Art zufällig und unabhängig von den anderen c_ℓ gewählt wird. Es stellt sich die Frage, wieviele Coupons man ziehen muß, bis man von jeder Art mindestens einen Coupon besitzt.*

Es bezeichne $H_n := 1 + 1/2 + 1/3 + \ldots 1/n$ die n-te harmonische Zahl. Es gilt

$$H_n = \ln n + 0,577\ldots + O(1/n) \ .$$

Theorem 5.1 *Die erwartete Anzahl an Versuchen, bis man Coupons von allen n Arten besitzt, beträgt*

$$n \cdot H_n = n \ln n + 0,577\, n + O(1) \ .$$

Die Varianz ist beschränkt durch $2\, n^2$.

Beweis: Es bezeichne T_i die Anzahl der Schritte, um die Zahl der verschiedenen Coupons von $i-1$ auf i zu erhöhen, und T die Gesamtzahl der Schritte, bis alle n Arten vertreten sind. Dann gilt $T = \sum_{i=1}^{n} T_i$. Der Erwartungswert von T_i ist $n/(n-i+1)$, da die Wahrscheinlichkeit p_i , daß der nächste Coupon nicht zu den bereits $i-1$ erhaltenen Arten gehört, gerade $(n-i+1)/n$ ist. Somit gilt

$$\mathrm{E}[T] = \sum_{i=1}^{n} \mathrm{E}[T_i] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \cdot H_n \ .$$

Für die Varianz ergibt sich

$$\mathrm{Var}[T] = \sum_{i=1}^{n} \mathrm{Var}[T_i] = \sum_{i=1}^{n} \frac{1-p_i}{p_i^2} \leq n^2 \sum_{i=1}^{n} \frac{1}{(n-i+1)^2} \leq \frac{\pi^2}{6}\, n^2 \leq 2\, n^2 \ .$$

∎

Wie groß ist die WS, daß die ZV T stark von ihrem Erwartungswert abweicht? Aus der Chebyshevschen Ungleichung ergibt sich für $c \geq 1$

$$\Pr[|T - n\ H_n| \geq c\ n] \ \leq \ \frac{2}{c^2}\ .$$

Eine bessere Abschätzung kann man wie folgt herleiten. Sei Y_j^t das Ereignis, daß ein Coupon vom Typ j nicht unter den ersten t Coupons vorkommt. Es gilt

$$\Pr[Y_j^t] \ = \ (1 - 1/n)^t \ \leq \ e^{-t/n}\ .$$

Für $t = \delta\ n \ln n$ ergibt sich

$$\Pr[T > \delta\ n \ln n] \ \leq \ \Pr[\bigcup_j Y_j^t] \ \leq \ n \cdot \Pr[Y_1^t] \ \leq \ n \cdot e^{-\delta\ n \ln n/n} \ = \ n^{-\delta+1}\ .$$

kurz besprochen: Birthday-Paradoxon

Aufgabe 5.2 *Tennisturnier A versus BCB oder CBC*

Definition 5.3 *Ball-into-Bins-Problem*

Theorem 5.2 *Die WS, daß maximale Füllung eines Bins $k = 3 \ln n / \ln \ln n$ übersteigt, ist beschränkt durch $1/n$ für n groß genug.*

Beweis: ∎

Wir wollen dies Ergebnis anwenden auf die Analyse von Bucket-Sort.

5.3 Programming Challenges (Einschub)

Aufgaben des NWERC2009, Nürnberg November 2009
siehe Wettbewerbsunterlagen

1. Einschätzung der Schwierigkeiten der einzelnen Aufgaben
2. Besprechung von Lösungsansätzen für die Aufgaben nach Schwierigkeit geordnet

5.4 Suffix Trees und Arrays (Einschub)

Definition und Anwendungen dieser Datenstrukturen
Quelle: S. Aluru, Suffix Trees and Suffix Arrays, in Handbook of Data Structures and Algorithms, Kap. 29, D. Mehta, S. Sahni (Ed.), CRC 2005

5.5 Die probabilistische Methode

5.5.1 Ramsey-Probleme

Problem: färbe den K_{1000} mit 2 Farben, so daß es keinen monochromatischen Subgraphen der Größe m gibt. Wie groß kann m maximal sein?
Ramsey-Zahlen: Wie groß muß ein Graph mindestens sein, sodaß jede k-Färbung garantiert ein monochromatisches Objekt aus einer gewissen vorgegebenen Menge enthält?

Theorem 5.3 *Der vollständige Graph K_n kann mit 2 Farben gefärbt werden ohne einen monochromatischen Subgraphen der Größe m, falls $\binom{n}{2} < 2^{\binom{m}{2}-1}$.*

5.5.2 MAXIMUM-CUT

Problem: gegeben einen ungerichteten Graph $G = (V, E)$ finde eine Zerlegung $V = V_1 \cup V_2$ mit einem möglichst großen Schnitt zwischen den beiden Teilen. Dies Problem ist $\mathcal{NP}$-hart.

Theorem 5.4 *Ein ungerichteter Graph mit m Kanten besitzt einen Schnitt der Größe mindestens $m/2$.*

Die Wahrscheinlichkeit, daß ein zufällig gewählter Schnitt mindestens $m/2$ Kanten besitzt, ist mindestens $(1 + m/2)^{-1}$.

5.5.3 Derandomisierung

Sei $v_1, v_2, \ldots$ eine Aufzählung der Knoten von V in beliebiger Reihenholge. Betrachte $\mathrm{E}[\gamma(V_1, V_2) \mid x_1, \ldots, x_i]$, die erwartete Größe des Cuts, wenn der Knoten v_j bereits der Teilmengen x_j zugeordnet ist für $j \in [1..i]$.

Lemma 5.1 *Für jede Folge $x_1, \ldots, x_i$ gibt es einen Wert x_{i+1}, so daß*

$$\mathrm{E}[\gamma(V_1, V_2) \mid x_1, \ldots, x_i] \;\leq\; \mathrm{E}[\gamma(V_1, V_2) \mid x_1, \ldots, x_i, x_{i+1}] \; .$$

x_{i+1} ist so zu wählen, daß v_{i+1} in der Menge landet, wo weniger Nachbarn sind. Dies führt auf einen einfachen Greedy-Algorithmus für das Problem MAXIMUM-CUT.

5.5.4 Modifikation einer zufälligen Auswahl

Theorem 5.5 *Ein ungerichteter Graph mit n Knoten und m Kanten besitzt eine unabhängige Menge der Größe mindestens $n^2/4m$.*

Beweis: Sei $d = 2m/n$ der durchschnittliche Grad eines Knotens. Folglender randomisierter Algorithmus konstruiert eine unabhängige Menge:

1. lösche jeden Knoten aus G mit Wahrscheinlichkeit $1-1/d$ unabhängig voneinander;
2. für jede verbleibende Kante lösche die Kante und einen ihrer Endpunkte.

Die ZV X und Y messen die Zahl der Knoten, bzw. Kanten, die den ersten Schritt überleben. Es gilt

$$\mathrm{E}[x] = \frac{n}{d} \quad \text{und} \quad \mathrm{E}[Y] \;=\; \frac{n\,d}{2}\,\frac{1}{d^2} \;=\; \frac{n}{2d}\;.$$

Da pro Kante maxiaml ein Knoten im 2. Schritt gelöscht wird, verbleiben mindestens

$$\mathrm{E}[X-Y] \;=\; \frac{n}{d} - \frac{n}{2d} \;=\; \frac{n}{2d} \;=\; \frac{n^2}{4m}$$

viele Knoten. ∎

Für bestimmte Anwendungen ist es wichtig, daß ein Netzwerk keine kleinen Kreise besitzt. Wir definieren daher:

Definition 5.4 *Der* **Girth** *eines ungerichteten Graphen G sei die minimale Länge eines Kreises in G.*

Bäume haben Girth unendlich, während man bei dichten Graphen – Graphen mit vielen Kanten – erwartet, daß auch kleine Kreise auftreten. Mit Hilfe eines probabilistischen Arguments wollen wir nun zeigen, daß man derartiges auch bei relativ dichten Graphen vermeiden kann.

Theorem 5.6 *Für jede Zahl $k \geq 3$ und $n \in \mathbb{N}$ gibt es einen Graphen mit n Knoten, Girth mindestens k und $\frac{1}{4}n^{1+1/k}$ Kanten.*

Beweis: Wähle einen zufälligen Graph $G \in \mathcal{G}_{n,p}$ für $p = n^{1/k-1}$, d.h. für jede der $\binom{n}{2}$ Kanten wird unabhängig mit WS p entschieden, ob diese vorhanden ist. Sei X die Anzahl der Kanten von G und Y die Anzahl der Kreise der Länge höchstens $k-1$. Es gilt

$$\mathrm{E}[X] \;=\; p\,\binom{n}{2} \;=\; \frac{1}{2}\,(1-\frac{1}{n})\,n^{1+1/k}\;.$$

Eine Folge von j Knoten bildet einen Kreis in G mit WS p^j. Es gibt $\binom{n}{j} \frac{(j-1)!}{2}$ verschiedene Möglichkeiten für Kreise der Länge j. Daher läßt sich die erwartete Anzahl an Kreise der Länge kleiner als k abschätzen durch

$$\mathrm{E}[Y] \;=\; \sum_{j=3}^{k-1} \binom{n}{j} \frac{(j-1)!}{2}\, p^j \;\leq\; \sum_{j=3}^{k-1} n^j p^j \;=\; \sum_{j=3}^{k-1} n^{j/k} < k\, n^{(k-1)/k}\;.$$

Im 2. Schritt zerstören wir nun alle diese kleinen Kreise, indem jeweils eine Kante entfernt wird. Es verbleiben mindestens

$$\mathrm{E}[X-Y] \;\geq\; \frac{1}{2}\,(1-\frac{1}{n})\; n^{1+1/k} - k\, n^{(k-1)/k} \;\geq\; \frac{1}{4} n^{1+1/k}$$

Kanten. ∎

Mit Hilfe der Ungleichung von Tschebychev läßt sich leicht zeigen

Lemma 5.2 *Für eine ZV X mit Werten in $\mathbb{N}$ gilt*

$$\Pr[X=0] \;\leq\; \frac{\mathrm{Var}[X]}{(\mathrm{E}[X])^2}\;.$$

Beweis:

$$\Pr[X=0] \;\leq\; \Pr[|X-E(X)| \;\geq\; E[X]] \;\leq\; \frac{\mathrm{Var}[X]}{(\mathrm{E}[X])^2}\;.$$ ∎

Für Bernoulli-ZV läßt sich diese WS auf nach unten abschätzen.

Lemma 5.3 *Sei $X = \sum_{i=1}^n X_i$, wobei die X_i binäre ZV sind. Dann gilt:*

$$\Pr[X>0] \;\geq\; \sum_{i=1}^{n} \frac{\Pr[X_i=1]}{\mathrm{E}[X \mid X_i=1]}\;.$$

Beweis: Sei $Y = 1/X$ falls $X>0$ und $Y=0$ andernfalls. Dann gilt

$$\begin{aligned}
\Pr[X>0] \;&=\; \mathrm{E}[X\cdot Y] \;=\; \mathrm{E}\left[\sum_i X_i \cdot Y\right] \;=\; \sum_i \mathrm{E}[X_i \cdot Y] \\
&=\; \sum_i \mathrm{E}[X_i\cdot Y \mid X_i=1]\cdot\Pr[X_i=1] + \mathrm{E}[X_i\cdot Y \mid X_i=0]\cdot\Pr[X_i=0] \\
&=\; \sum_i \mathrm{E}[Y \mid X_i=1]\cdot\Pr[X_i=1] \;=\; \sum_i \mathrm{E}[1/X \mid X_i=1]\cdot\Pr[X_i=1] \\
&\geq\; \sum_i \frac{\Pr[X_i=1]}{\mathrm{E}[X\mid X_i=1]}\;,
\end{aligned}$$

wobei wir im letzten Schritt Jensens Ungleichung auf die konvexe Funktion $1/x$ angewandt haben. ∎

Neben dem Girth hängen auch andere Grapheigenschaft sehr stark von der Dichte der Graphen ab. Oftmals ist der Übergang sehr abrupt in Form eines Phasenübergangs. Es gibt dann eine Kantenwahrscheinlichkeit $\tau = \tau(n)$, so daß zufällige Graphen aus $\mathcal{G}_{n,p}$ diese Eigenschaft besitzen mit Wahrscheinlichkeit fast 0, wenn $p < \tau(n)$, und mit Wahrscheinlichkeit fast 1, wenn $p <> \tau(n)$.

Theorem 5.7 *Sei K_4 die Eigenschaft, eine Clique der Größe mindestens 4 zu besitzen. Für $\tau(n) = n^{-2/3}$ gilt:*
Falls $p \in o(\tau)$, dann besitzen die Graphen in $\mathcal{G}_{n,p}$ die Eigenschaft K_4 mit Wahrscheinlichkeit höchstens ϵ, wobei $\epsilon > 0$ beliebig klein gewählt werden kann und die Behauptung für alle n ab einem gewissen $n_0(\epsilon)$ gilt.
Für $p \in \omega(\tau)$ hingegen erfüllen die Graphen in $\mathcal{G}_{n,p}$ die Eigenschaft K_4 mit Wahrscheinlichkeit mindestens $1 - \epsilon$.

Beweis: Für jeweils eine Menge C_i von 4 Knoten in G definieren wir eine binäre ZV X_i, die genau dann 1 ist, wenn diese Knoten einen K_4 aufspannen. Es gilt $\Pr[X_i = 1] = p^6$, da alle 6 Kanten zwischen diesen Knoten vorhanden sein müssen. Dann ist

$$X = \sum_{i=1}^{\binom{n}{4}} X_i$$

die Anzahl verschiedener 4-Cliquen und

$$\mathrm{E}[X] \;=\; \mathrm{E}[\sum_{i=1}^{\binom{n}{4}} X_i] \;=\; \sum_{i=1}^{\binom{n}{4}} \mathrm{E}[X_i] \;=\; \binom{n}{4} \cdot p^6 \;.$$

Für $p \in o(n^{-2/3})$ konvergiert dieser Wert gegen 0. Für $p \in \omega(n^{-2/3})$ dagegen strebt der Erwartungswert gegen unendlich, damit ist aber noch nicht garantiert, daß fast alle Graphen mindestens einen K_4 besitzen. Dazu bedarf es einer genaueren Abschätzung.

$$\mathrm{E}[X|X_j = 1] \;=\; \sum_{i=1}^{\binom{n}{4}} \mathrm{E}[X_i \mid X_j = 1] \;=\; \sum_{i=1}^{\binom{n}{4}} \Pr[X_i = 1 \mid X_j = 1] \;.$$

Wir müssen nun die möglichen Abhängigkeiten zwischen X_i und X_j betrachten. Es gibt $\binom{n-4}{4}$ Mengen C_i, die disjunkt zu C_j sind. In diesem Fall sind X_i, X_j unabhängig und $\Pr[X_i = 1 \mid X_j = 1] \;=\; \Pr[X_i = 1] = p^6$. Das gleiche gilt für die $4\binom{n-4}{3}$ Teilmengen C_i, die mit C_j genau 1 Knoten gemeinsam haben. Es verbleiben noch die $\binom{4}{2} \cdot \binom{n-4}{2}$ Fälle, wo C_i und C_j genau 2 gemeinsame Knoten besitzen und damit eine gemeinsame Kante, sowie die $4\,(n-4)$ Fälle mit 3 gemeinsamen Knoten und demzufolge 3 gemeinsamen Kanten. Im

ersten Fall gilt $\Pr[X_i = 1 \mid X_j = 1] = p^5$, im zweiten Fall $\Pr[X_i = 1 \mid X_j = 1] = p^3$. Somit gilt

$$\mathrm{E}[X|X_j = 1] = 1 + \binom{n-4}{4} p^6 + 4 \binom{n-4}{3} p^6 + 6 \binom{n-4}{2} p^5 + 4 \binom{n-4}{1} p^3 .$$

Mit Hilfe des vorherigen Lemmas können wir nun schließen

$$\Pr[X > 0] \geq \frac{\binom{n}{4} p^6}{1 + \binom{n-4}{4} p^6 + 4 \binom{n-4}{3} p^6 + 6 \binom{n-4}{2} p^5 + 4 \binom{n-4}{1} p^3} ,$$

was für $p \in \omega(n^{-2/3})$, d.h. $p = n^{-2/3} \cdot \omega(1)$ gegen 1 konvergiert. ∎

5.6 Lovasz Local Lemma

Wir wollen nun die folgende Situation betrachten. Es gebe eine gewisse Menge von Ereignissen A_i, die wir vermeiden wollen. Jedes A_i habe eine WS $\Pr[A_i] < 1$. Wie groß ist dann die WS, daß keines der Ereignisse A_i eintritt. Gesucht ist also $\Pr[\bigcap \bar{A}_i]$. Sind die A_i unabhängig, so gilt

$$\Pr[\bigcap \bar{A}_i] = \prod (1 - p(A_i)) > 0 ,$$

d.h. es gibt Situationen, in denen kein Ereignis A_i eintritt. In der Regel sind die A_i jedoch nicht unabhängig, so daß diese Schlußfolgerung nicht mehr korrekt ist. Ein einfaches Gegenbeispiel wäre $A_2 = \bar{A}_1$ mit $\Pr[A_1] = 1/2$. Im folgenden soll nun gezeigt werden, daß es dennoch möglich ist, alle A_i zu vermeiden, wenn zwischen den Ereignissen nur eine beschränkte Abhängigkeit besteht.

Definition 5.5 *Es seien $\mathcal{A} = A_1, \ldots, A_n$ eine Folge von Ereignissen in einem W-Raum. Ein Ereignis B heißt* **unabhängig** *von $A_1, \ldots, A_n$, falls für jede Teilmenge $I \subseteq [1..n]$ gilt:*

$$\Pr[B \mid \bigcap_{j \in I} A_j] = \Pr[B] .$$

Der **Abhängigkeitsgraph $\Gamma(\mathcal{A})$** *von $\mathcal{A}$ besteht aus den Knoten $A_1, \ldots, A_n$ und Kanten zwischen allen Paaren A_i, A_j, die nicht unabhängig voneinander sind.*

Theorem 5.8 *Es sei $\mathcal{A} = A_1, \ldots, A_n$ eine Folge von Ereignissen in einem W-Raum. Es gebe eine Abbildung $g : \mathcal{A} \to (0, 1)$ mit der Eigenschaft*

$$\forall\, A \in \mathcal{A} \quad \Pr[A] \leq g(A) \prod_{\{A,B\} \in \Gamma(\mathcal{A})} (1 - g(B)) .$$

Dann gilt

$$\Pr[\bigcap \bar{A}_i] \geq \prod (1 - g(A_i)) .$$

Bevor wir den Beweis vorstellen, betrachten wir einen Spezialfall.

- Der Knotengrad im Abhängigkeitsgraph $\Gamma(\mathcal{A})$ sei beschränkt durch eine Zahl $d \in \mathbb{N}$, d.h. jedes Ereignis ist abhängig von maximal d anderen, und
- es gelte $\Pr[A_i] \leq (e\,(d+1))^{-1}$ für alle i, wobei $e = 2,7\ldots$ die Eulersche Zahl bezeichnet.

Dann können wir die Abbildung $g \equiv 1/(d+1)$ wählen, denn es gilt

$$g(A) \prod_{\{A,B\}\in\Gamma(\mathcal{A})} (1-g(B)) \;\geq\; \frac{1}{d+1}\,\left(1-\frac{1}{d+1}\right)^d \;\geq\; \frac{1}{d+1}\,e^{-1} \;\geq\; \Pr[A_i]\;.$$

Dann impliziert das Theorem

$$\Pr[\textstyle\bigcap \bar{A}_i] \;\geq\; \prod(1-g(A_i)) \;\geq\; \left(1-\frac{1}{d+1}\right)^n \;>\; 0\;.$$

Halten wir also fest:

Korollar 5.1 *Für eine Folge von Ereignissen $\mathcal{A} = A_1,\ldots,A_n$ erfülle jedes A_i die Bedingung $\Pr[A_i] \leq (e\,(\Gamma(\mathcal{A})+1))^{-1}$. Dann gilt $\Pr[\bigcap \bar{A}_i] > 0$.*

Diese Abschätzung können wir beispielsweise auf das Erfüllbarkeitsproblem für k-CNF-Formeln anwenden. Besteht solch eine Formel aus weniger als 2^k Klauseln, dann ist sie trivialerweise erfüllbar.
Sei nun eine Formel $F = F(x_1,\ldots,x_r) = K_1 \wedge \ldots K_m$ in r Variablen gegeben, wobei jede Klausel K_i genau k verschiedenen Variable, positiv oder negiert, enthalte. Jede Variable komme insgesamt in F nicht mehr als ℓ-mal vor. A_i bezeichne das Ereignis, daß bei einer zufälligen Wahl der Belegung aller Variablen die Klausel K_i nicht erfüllt wird. Es gilt

$$\Pr[A_i] \;=\; \Pr[\text{keines der } k \text{ Literale in } K_i \text{ ist wahr}] \;=\; 2^{-k}\;.$$

Der Abhängigkeitsgraph $\Gamma(\mathcal{A})$ verbindet zwei Ereignisse A_i, A_j genau dann, wenn die Klauseln K_i, K_j gemeinsame Variable besitzen. Klauseln mit disjunkten Variablen sind offensichtlich bezüglich Erfüllbarkeit unabhängig. Jede der k Variablen in K_i kann nach Voraussetzung in maximal $\ell - 1$ weiteren Klauseln vorkommen, also ist der Grad von $\Gamma(\mathcal{A})$ durch $k\,(\ell-1)$ beschränkt. Aus dem Local Lemma können wir schließen, daß für

$$2^{-k} \leq \frac{1}{e\,(k\,(\ell-1)+1)}\;, \quad \text{d.h.} \quad \ell \;\leq\; \frac{2^k}{e\,k} + 1 - \frac{1}{k}$$

eine erfüllende Belegung existiert, da $\Pr[\bigcap \bar{A}_i] > 0$.

Die Schranke für ℓ ist erstaunlich scharf, denn für $\ell = 2^k - 1$ ist die Schlußfolgerung nicht mehr zulässig. Für $k = 10$ erfüllt $\ell = 38$ die Voraussetzung, d.h. solange keine Variable mehr als 38-mal in einer beliebigen 10-CNF-Formel vorkommt, ist diese garantiert

erfüllbar. Für den Basisfall $k = 3$ verlangt die Abschätzung allerdings nur die triviale Schranke $\ell = 1$, für $k = 4$ erhalten wir allerdings die Schranke $\ell = 2$. Wir können also schlußfolgern, daß jede 4-CNF-Formel, in der jede Variable entweder nur positiv oder nur negativ oder genau einmal positiv und einmal negativ vorkommt, erfüllbar ist.

Beweis des Local Lemmas:
Sei $S \subseteq [1..n]$. Per Induktion über $s = |S|$ zeigen, wir daß für alle S gilt

$$\Pr[A_i \mid \bigcap_{j \in S} \bar{A}_j] \;\leq\; g(A_i) \;.$$

Damit kann man dann schlußfolgern

$$\begin{aligned} \Pr[\textstyle\bigcap \bar{A}_i] &= \prod_{i=1}^{n} \Pr[\bar{A}_i \mid \bar{A}_1, \ldots, \overline{A_{i-1}}] \\ &= \prod_{i=1}^{n} (1 - \Pr[A_i \mid \bar{A}_1, \ldots, \overline{A_{i-1}}] \;\geq\; \prod_{i=1}^{n} (1 - g(A_i)) \;>\; 0 \;. \end{aligned}$$

∎

Bislang haben wir lediglich die Existenz eines Elementes in $\bigcap \bar{A}_i$ nachgewiesen. Wir wollen jetzt der Frage nachgehen, ob man dies auch effizient konstruieren kann. Um diese Suche algorithmisch durchführen zu können, müssen wir gewisse formale Voraussetzungen an den W-Raum stellen.

Es sei $\mathcal{Q}$ eine endliche Menge von paarweise unabhängigen ZV in einem W-Raum Ω. Wir betrachten Ereignisse A_i, die jeweils durch eine Teilmenge Q_i von Q determiniert werden. Mit anderen Worten, es gibt eine binäre Funktion f_i definiert auf dem Produkt der ZV in Q_i, so daß $A_i = f^{-1}(1)$. Gilt für eine Wertevektor q dieser ZV $f_i(q) = 1$, dann sagen wir q **generiert** A_i.

Algorithmus SEQUENTIAL-LLL $(\mathcal{Q}, \mathcal{A})$

Algorithmus PARALLEL-LLL $(\mathcal{Q}, \mathcal{A})$

Algorithmus Derandomized-LLL $(\mathcal{Q}, \mathcal{A})$

5.7 Erweiterungen

azyklische Kantenfärbung, Graph-Färbung

MAX- k -SAT

5.8 Wahrscheinlichkeitstheoretische Grundlagen

Zunächst wollen wir kurz die notwendige Notation für eine probabilistische Analyse einführen und wichtige wahrscheinlichkeitstheoretische Abschätzungen auflisten.

Definition 5.6 *Ist A ein* **Ereignis** *eines Wahrscheinlichkeitsraumes $\mathcal{U}$, so bezeichne* $\mathbf{Pr}[\boldsymbol{A}]$ *die* **Wahrscheinlichkeit** *des Auftretens von A. Für eine* **Zufallsvariable** *T mit Werten in $\mathbb{N}$ verwenden wir für ihren* **Erwartungswert** *und ihre* **Varianz** *die Notation*

$$\begin{aligned} \mathbf{E}[\boldsymbol{T}] &:= \sum_{m \in \mathbf{N}} \Pr[T = m] \cdot m \ , \\ \mathbf{Var}[\boldsymbol{T}] &:= \mathrm{E}[\ (T - \mathrm{E}[T])^2] \ . \end{aligned}$$

Durch unabhängige Wiederholungen eines probabilistischen Experimentes kann die Genauigkeit und Korrektheit verbessert werden. Dies elementare Vorgehen modellieren wir durch eine Folge $Y_1, \ldots, Y_k$ unabhängiger, identisch verteilter Zufallsvariablen. Von besonderem Interesse sind binäre Experimente, d.h. als Ergebnis treten nur die Werte 0 und 1 auf. Es sei $\mu \in [0,1]$ die Wahrscheinlichkeit des Wertes 1, und damit

$$\Pr[Y_i = 1] \ = \ \mu \quad \text{und} \quad \Pr[Y_i = 0] \ = \ 1 - \mu \qquad \text{für alle } i \, .$$

Dann gilt für den Erwartungswert $\mathrm{E}[\ Y_i\ ] = \mu$ und

$$\mathrm{E}\left[\ \sum_{i=1}^{k} Y_i \ \right] \ = \ \sum_{i=1}^{k} \mathrm{E}[\ Y_i\ ] \ = \ k \cdot \mu \ .$$

Im Fall $\mu < 1/2$ sollte der Wert 0 als Ergebnis häufiger auftreten als die 1. Wie groß ist die Wahrscheinlichkeit, daß dies nicht eintrifft?

Lemma 5.4 *$Y_1, \ldots, Y_k$ seien unabhängige, identisch verteilte, binäre Zufallsvariable mit Erwartungswert $\mu < 1/2$. Dann gilt:*

$$\epsilon_k \ := \ \Pr\left[\ \sum_{i=1}^{k} Y_i \ \geq \ \frac{k}{2} \ \right] \ \leq \ \left(2\sqrt{\mu\,(1-\mu)}\right)^{\frac{k}{2}} \ .$$

Beweis: Durch elementare Umformungen können wir unter Verwendung von $\mu/(1-\mu) < 1$ schließen:

$$\begin{aligned} \epsilon_k \ &= \ \Pr\Big[\ \sum_{i=1}^{k} Y_i \ \geq \ \frac{k}{2} \ \Big] \ = \ \sum_{l = \lceil k/2 \rceil}^{k} \binom{k}{l} \mu^l \, (1-\mu)^{k-l} \\ &= \ \mu^{\lceil k/2 \rceil} \ (1-\mu)^{k - \lceil k/2 \rceil} \ \sum_{l = \lceil k/2 \rceil}^{k} \binom{k}{l} \ \left(\frac{\mu}{1-\mu}\right)^{l - \left\lceil \frac{k}{2} \right\rceil} \ \leq \ (\mu\,(1-\mu))^{k/2} \cdot 2^k \\ &= \ \left(2\sqrt{\mu\,(1-\mu)}\right)^{k} \ . \end{aligned}$$

∎

Man beachte, daß die Wahrscheinlichkeit ϵ_k mit k exponentiell schnell gegen 0 konvergiert. Wir hätten ϵ_k auch mit Hilfe der Chebyshevschen Ungleichung abschätzen können; dies liefert allerdings nur eine Konvergenz gegen 0 linear in k. Das folgende Resultat von *Chernoff* liefert eine allgemeine Schranke für Abweichungen vom Erwartungswert.

Lemma 5.5 Chernoff-Schranke *$Y_1, \ldots, Y_k$ seien unabhängige, identisch verteilte binäre Zufallsvariable mit Erwartungswert μ. Dann läßt sich die Wahrscheinlichkeit einer Abweichung vom Erwartungswert $\mu \cdot k$ um mindestens einen Faktor α beschränken durch*

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{k} Y_i \;\leq\; (1-\alpha)\;\mu\;k\right] &\leq \exp\left(-\frac{\log\, e}{2}\;\alpha^2\;\mu\;k\right) \qquad \textit{für } 0 \leq \alpha \leq 1,\\
\Pr\left[\sum_{i=1}^{k} Y_i \;\geq\; (1+\alpha)\;\mu\;k\right] &\leq \left(\frac{1}{1+\alpha}\;\left(\frac{e}{1+\alpha}\right)^{\alpha}\right)^{\mu\;k}\\
&= \exp\left(-\Theta(\alpha\;\log\,\alpha\;\cdot\;\mu\;k)\right) \qquad \textit{für } 0 \leq \alpha,\\
\Pr\left[\sum_{i=1}^{k} Y_i \;\geq\; (1+\alpha)\;\mu\;k\right] &\leq \exp\left(-\frac{\log\, e}{3}\;\alpha^2\;\mu\;k\right) \qquad \textit{für } 0 \leq \alpha \leq 1.
\end{aligned}$$

Auch diese Schranken konvergieren exponentiell schnell in k gegen 0. Bei der zweiten Ungleichung gilt das gleiche bezüglich des Faktors α. Sei $Y := \sum_{i=1}^{k} Y_i$. Fassen wir die erste und letzte Abschätzung zusammen, so ergibt sich für $\alpha \in [0,1]$ die Schranke

$$\Pr\Big[|Y - \mathrm{E}[Y]| \geq \alpha\;\mathrm{E}[Y]\Big] \;\leq\; 2\;\exp\left(-\frac{\log\, e}{3}\;\alpha^2\;\mu\;k\right)\;.$$

5.9 Las-Vegas-Algorithmen

Wir wollen den Leser zunächst anhand einer Beispiele mit verschiedenen probabilistischen Lösungsstrategien vertraut machen. Die randomisierte Version von Quicksort wie oben ausgeführt ist ein typisches Beispiel. Seine Laufzeit ist gewissen Schwankungen unterworfen, wobei diese im Mittel jedoch recht klein sind. Die Laufzeitunterschiede beruhen auf internen Zufallsentscheidungen, durch die aus verschiedenen Möglichkeiten, das Problem zu lösen, ausgewählt wird. Ein derartiges Vorgehen nennt man einen **Las-Vegas-Algorithmus**. Um die Zeitkomplexität solch eines Algorithmus M zu definieren, betrachten wir für eine Eingabe X seine erwartete Laufzeit bezüglich aller möglichen Zufallsentscheidungen:

$$\boldsymbol{time}_{\boldsymbol{M}}^{\mathbf{erw}}\boldsymbol{(X)} \;:=\; E[time_M(X)]\;,$$

wobei $time_M(X)$ nun eine Zufallsvariable ist. Die Zeitkomplexität von M ist dann gegeben durch die Funktion

$$\boldsymbol{Time}_{\boldsymbol{M}}^{\mathbf{erw}}(\boldsymbol{n}) \;:=\; \max_{|X|=n} time_M^{\mathrm{erw}}(X) \;.$$

Bei einem Las-Vegas-Algorithmus wird gefordert, daß er unabhängig von den internen Zufallsentscheidungen immer ein korrektes Ergebnis liefert. Um jedoch zu verhindern, daß bei unglücklichen Zufallsentscheidungen sehr viel mehr Zeit aufgewandt wird als der Erwartungswert – was auf Grund der obigen Ungleichungen nur selten vorkommt – kann man das Modell dahingehend erweitern, daß eine Berechnung ohne ein Ergebnis enden darf. Allerdings wird die Existenz einer festen positiven Zahl γ verlangt, so daß für jede Eingabe X die Wahrscheinlichkeit, daß der Algorithmus auf X eine Lösung ausgibt, mindestens γ beträgt. Wählt man für Eingaben der Länge n etwa als Stopzeit $2 \cdot Time_M^{\mathrm{erw}}(n)$, so kann man für γ den Wert $1/2$ wählen. Durch hinreichend viele Wiederholungen kann so die Wahrscheinlichkeit, daß keine Lösung gefunden wird, beliebig klein gemacht werden, denn bei k unabhängigen Versuchen beträgt die Erfolgswahrscheinlichkeit mindestens $1-(1-\gamma)^k$. Man beachte, daß dadurch die Mißerfolgswahrscheinlichkeit $(1-\gamma)^k$ exponentiell schnell in k gegen 0 konvergiert, während die Abschätzung über den Erwartungswert nur eine lineare Konvergenz liefert

$$\Pr\Big[time_M(X) \;\geq\; k \cdot Time_M^{\mathrm{erw}}(|X|)\Big] \;\leq\; \frac{1}{k} \;.$$

In vielen Fällen kann man die Effizienz von Lösungsstrategien durch Randomisierung noch weiter steigern, wenn man bereit ist, eine gewisse Fehlerwahrscheinlichkeit in Kauf zu nehmen. Diese Verallgemeinerung von Las-Vegas-Algorithmen nennt man **Monte-Carlo-Algorithmen**.[1]

5.10 Monte-Carlo-Test für Primzahlen

Beim nichtdeterministischen Raten, beispielsweise einer Faktorisierung $a \cdot b = p$ einer Nichtprimzahl p, kann, je nach Wahl von p, die Erfolgswahrscheinlichkeit sehr klein sein. Man betrachte etwa den Fall, daß p das Produkt zweier großer Primzahlen ist. Bei einer zufälligen Wahl von a und b sind daher die Aussichten, die Faktoren zu finden, äußerst gering. Bei einem Monte-Carlo-Verfahren wird verlangt, daß die Chancen erheblich günstiger sind.

Definition 5.7 *Es sei A eine Teilmenge einer festen Menge U. Ein* **Zeugensystem** *für A ist eine Menge Z und ein Prädikat $\pi : U \times Z \to \{$ `true` , `false` $\}$. Gilt $\pi(x,z)$,*

[1] Durch diese Namensgebung sei jedoch keine Aussage über die entsprechenden Zentren des Glücksspiels verbunden.

so nennt man z *einen Zeugen für* x*. Für* $\epsilon \geq 0$ *heißt ein Zeugensystem* (Z, π) *für* A **ϵ-fehlerbeschränkt**, *falls*

$$\begin{aligned} \forall\, x \in A \quad & \Pr[\textit{ ein zufällig gewähltes } z \in Z \textit{ ist ein Zeuge für } x] \;\geq\; 1 - \epsilon\ , \\ \forall\, x \notin A \quad & \Pr[\textit{ ein zufällig gewähltes } z \in Z \textit{ ist ein Zeuge für } x] \;\leq\; \epsilon\ . \end{aligned}$$

Gilt für Elemente $x \notin A$ *die schärfere Bedingung, daß sie überhaupt keinen Zeugen besitzen, mit anderen Worten*

$$\forall\, x \in U \quad \forall\, z \in Z \qquad \pi(x, z) = \texttt{true} \qquad \Longrightarrow \qquad x \in A\ .$$

so nennen wir das Zeugensystem **widerspruchsfrei**.

Für ein nichtwiderspruchsfreies Zeugensystem machen offensichtlich nur Werte $\epsilon < 1/2$ Sinn, denn durch das Werfen einer fairen Münze kann man die Frage "$x \in A$" trivialerweise mit Fehlerwahrscheinlichkeit $1/2$ entscheiden. Bei einem widerspruchsfreien Zeugensystem ist dagegen jeder Wert $\epsilon < 1$ von Interesse, denn man kann das folgende Experiment durchführen.

```
auf Eingabe x wähle zufällig ein z ∈ Z
berechne π(x, z)
if π(x, z) = true then stop und akzeptiere x
```

Für $k \in \mathbb{N}$ sei ein **probabilistischer Test** der Länge k basierend auf (Z, π) eine Hintereinanderreihung von k solchen Experimenten. Falls x in keinem der Experimente akzeptiert wird, so wird x verworfen.

Die folgende Tabelle zeigt die Wahrscheinlichkeiten der verschiedenen Ausgänge eines Experimentes und die resultierende Fehlerwahrscheinlichkeiten, wenn der Test auf einem widerspruchsfreien Zeugensystem mit Fehlerschranke ϵ basiert.

	Wahrscheinlichkeit		Fehlerwahrscheinlichkeit	
	akzeptiert	akzeptiert nicht	1 Experiment	Test der Länge k
$x \in A$	$\geq 1 - \epsilon$	$\leq \epsilon$	$\leq \epsilon$	$\leq \epsilon^k$
$x \notin A$	0	1	0	0

Tabelle 1: *Wahrscheinlichkeitsverteilung bei einem widerspruchsfreien Zeugensystem*

Wir halten fest, daß bei einem Test, der auf einem widerspruchsfreien Zeugensystem beruht, nur sogenannte **einseitige Fehler** auftreten; nur für $x \in A$ liefert der Test eventuell eine falsche Antwort.

Ist das Zeugensystem nicht widerspruchsfrei, so hat ein Test der Länge k die folgende Gestalt

```
auf Eingabe x wähle zufällig und unabhängig k Zeugen z ∈ Z
berechne π(x,z) für alle z
if für mindestens die Hälfte aller z  π(x,z) = true
      then akzeptiere x
```

In diesem Fall läßt sich die Fehlerwahrscheinlichkeit für $\epsilon < 1/2$ folgendermaßen abschätzen. Der Erwartungswerte für die Anzahl Zeugen bei k Versuchen ist mindestens $(1-\epsilon)\cdot k$ für $x \in A$ und höchstens $\epsilon \cdot k$ für $x \notin A$. Eine falsche Entscheidung nach dem Mehrheitsprinzip, d.h. dem Schwellwert $k/2$, setzt also eine Abweichung um einen Faktor $1/2-\epsilon$ vom Erwartungswert voraus. Die Wahrscheinlichkeit dafür ist gemäß der Chernoff-Schranke maximal $2\cdot \exp\left(-(\frac{1}{2}-\epsilon)^2\cdot(1-\epsilon)/4\cdot k\right)$ bzw. $2\cdot \exp\left(-(\frac{1}{2}-\epsilon)^2\cdot\epsilon/4\cdot k\right)$. Definieren wir $\zeta(\epsilon) := (\frac{1}{2}-\epsilon)^2\cdot\epsilon/4$ als den kleineren der beiden Faktoren vor dem Parameter k, so können wir die folgende Tabelle aufstellen:

	Wahrscheinlichkeit		Fehlerwahrscheinlichkeit	
	akzeptiert	akzeptiert nicht	1 Experiment	Test der Länge k
$x \in A$	$\geq 1-\epsilon$	$\leq \epsilon$	$\leq \epsilon$	$\leq 2\exp\left(-\zeta(\epsilon)\cdot k\right)$
$x \notin A$	$\leq \epsilon$	$\geq 1-\epsilon$	$\leq \epsilon$	$\leq 2\exp\left(-\zeta(\epsilon)\cdot k\right)$

Tabelle 2: *Wahrscheinlichkeitsverteilung bei nichtwiderspruchsfreiem Zeugensystemen*

Man sieht also, daß durch genügend viele Wiederholungen eines Experimentes die Fehlerwahrscheinlichkeit beliebig klein gemacht werden kann, wobei diese exponentiell schnell gegen 0 konvergiert.

Definition 5.8 *Existiert für einen randomisierten Algorithmus ein $\epsilon < 1/2$, so daß für jede Eingabe X eine Fehlerwahrscheinlichkeit von maximal ϵ garantiert werden kann, so nennen wir diesen einen* **Monte-Carlo-Algorithmus**.

Wir wollen nun einen Monte-Carlo-Test für die Menge der Primzahlen bzw. für ihr Komplement vorstellen. Nach dem kleinen Fermat'schen Satz gilt für eine Primzahl p:

$$a^{p-1} \equiv 1 \mod p \quad \forall a \in [1, p-1] \; .$$

Leider ist diese Eigenschaft nicht hinreichend, um die Primzahlen von den Nichtprimzahlen zu unterscheiden. Es gibt nämlich zusammengesetzte Zahlen, sogenannte **Carmichael-Zahlen**, die diese Kongruenzen ebenfalls erfüllen. Die kleinste Zahl dieser Art ist $561 = 3\cdot 11\cdot 17$. Der **Miller-Rabin-Primzahltest** benutzt eine weitere Eigenschaft und erhält so die folgenden Prädikate als Zeugensysteme. Sei $p-1 = 2^u\cdot v$ mit v ungerade, wobei wir $p > 2$ voraussetzen.

$$\pi_1(p,a) := \left[a^{p-1} \not\equiv 1 \mod p \quad \vee \quad \exists\, 0 \leq i < u : \quad 1 < \mathtt{ggT}(a^{2^i\cdot v} - 1, p) < p\right] ,$$

$$\pi_2(p,a) := \left[a^{v} \not\equiv 1 \mod p \quad \wedge \quad \forall\, 0 \leq i < u : \quad a^{2^i\cdot v} \not\equiv -1 \mod p\right] .$$

Beide Prädikate ergeben widerspruchsfreie Zeugensysteme für die Menge der Nichtprimzahlen. Für π_1 folgt dies unmittelbar aus dem Fermatschen Satz und der Beobachtung, daß, falls die 2. Bedingung in der $\vee$-Verknüpfung zutrifft, der `ggT` einen nichttrivialen Teiler von p liefert.

Lemma 5.6 *Sei $p > 4$ ungerade. Es gilt dann für alle $p \in \mathbb{N}$ und alle $a \in [1, p-1]$:*

1.) $\pi_1(p,a) =$ `true` $\implies$ *p ist eine zusammengesetzte Zahl ,*

2.) $\pi_1(p,a) = \pi_2(p,a)$.

Ist p eine zusammengesetzte Zahl, dann gilt für höchstens $1/4$ aller Zahlen $a \in [1, p-1]$ $\pi_i(p,a) =$ `false` , d.h. a ist kein Zeuge dafür, daß p eine Nichtprimzahl ist.

Die Äquivalenz von π_1 und π_2 nachzuweisen, erfordert einigen zahlentheoretischen Aufwand. Entscheidend ist nun die Zuverlässigkeitseigenschaft dieser Zeugensysteme, eine zahlentheoretische Eigenschaft, deren Beweis wir hier ebenfalls nicht ausführen wollen. Daher garantieren beide Zeugensysteme eine Fehlerschranke $\epsilon = 1/4$. Benutzt man das Zeugensystem π_2 und berechnet die Potenzen $a^{2^i \cdot v} \mod p$ in Folge, so genügen $O(\log^3 p)$ Schritte zur Berechnung von $\pi_2(p,a)$ (siehe die Laufzeitanalyse des nichtdeterministischen Primzahltests).

Theorem 5.9 *Die Sprache `PRIMES` kann durch einen Monte-Carlo-Algorithmus in Zeit $O(k \cdot n^3)$ mit Fehlerwahrscheinlichkeit höchstens 2^{-2k} erkannt werden. Fehler treten dabei nur bei Nichtprimzahlen auf.*

5.11 Verifikation algebraischer Identitäten

Als nächstes betrachten wir die Matrix-Multiplikation als Entscheidungs- oder Verifikationsproblem. Gegeben seien drei $(n \times n)$–Matrizen A, B, C über einem Ring R (beispielsweise $\mathbb{Z}$ oder $\mathbb{R}$). Aufgabe ist es zu überprüfen, ob $A \cdot B = C$.

Lemma 5.7 *Die Menge $Z = \{0,1\}^n$ und das Prädikat*

$$\pi(A, B, C; z) \;:=\; \Big[A \cdot (B \cdot z) \neq C \cdot z\Big]$$

mit $z \in Z$ bilden ein widerspruchsfreies $(1/2)$–fehlerbeschränktes Zeugensystem für die Entscheidung, ob $A \cdot B = C$.

Beweis: Unterscheidet sich C von dem Produkt von A und B, so ist $D := A \cdot B - C$ nicht die Nullmatrix. D besitzt also mindestens eine Zeile D_i, die nicht nur Nullen enthält. Sei $d_{i,j} \neq 0$ ein solch Eintrag in D_i. Ein Zeuge $z \in Z$ ist fehlerhaft, wenn $D \cdot z$ der

Nullvektor ist, d.h. insbesondere $0 = D_i \cdot z = \sum_j d_{i,j} \cdot z_j$. Es bezeichne $z^{[j]}$ den zu z an der j-ten Position komplementären Vektor, d.h. z_j wird durch $1 - z_j$ ersetzt. Dann gilt offensichtlich

$$D_i \cdot z^{[j]} = D_i \cdot z + (1 - 2z_j)d_{i,j} = \pm d_{i,j} \neq 0 \ .$$

Zu jedem falschen Zeugen erhalten wir also durch Komplementierung des j-ten Bits einen korrekten Zeugen. Daher sind mindestens die Hälfte aller Zeugen korrekt. ∎

Mißt man den Zeitaufwand in der Anzahl der Ring-Operationen, so wächst dieser für das Prädikat π quadratisch in n, während das offensichtliche deterministische Verfahren, das Produkt von A und B zu berechnen und dann mit C zu vergleichen, Aufwand $O(n^3)$ bzw. $O(n^\tau)$ besitzt ($\tau < 2,37$ bezeichnet hier den **Matrix-Multiplikations-Exponenten**. Die Beschleunigung um den Exponenten $\tau - 2 \approx 0,4$ ist nicht überwältigend. Betrachtet man jedoch das noch simplere Problem, die Gleichheit zweier $(n \times n)$–Matrizen A, A' zu verifizieren, so muß ein deterministischer Algorithmus alle n^2 Einträge vergleichen. Wählt man dagegen zufällig eine kleine Teilmenge von Einträgen und beschränkt den Gleichheitstest auf diese, etwa nur $o(n^2)$ viele, so ist die Fehlerwahrscheinlichkeit sehr hoch, wenn beide Matrizen fast identisch sind.

Der Monte-Carlo-Test hingegen, der die Werte $A \cdot z$ und $A' \cdot z$ für zufällig gewähltes $z \in Z$ auf Gleichheit untersucht, benötigt nur n Vergleiche, bzw. $k \cdot n$ bei k Wiederholungen und garantiert eine Fehlerschranke von maximal 2^{-k}. Sind die beiden Matrizen räumlich weit voneinander getrennt und ist der Datenaustausch aufwendig und nicht immer fehlerfrei – etwa auf entfernten Rechnern im Internet oder in einer Raumsonde und ihrer Bodenstation verbunden über eine störungsanfällige Funkverbindung – so wird durch das probabilistische Verfahren der Kommunikationsaufwand drastisch reduziert, ohne daß die leicht erhöhte Fehlerwahrscheinlichkeit ins Gewicht fällt.

Mit einer ähnlichen Idee läßt sich feststellen, ob zwei vorgegebene Polynome $p(x_1, \ldots, x_n)$ und $q(x_1, \ldots, x_n)$ im Polynomring $R[x_1, \ldots, x_n]$ verschieden sind. Liegen beide Polynome explizit durch ihre Koeffizienten $a_{i_1,\ldots,i_n}$ vor, d.h.

$$p(x_1, \ldots, x_n) = \sum_{(i_1,\ldots,i_n)|in\mathbb{N}^n} a_{i_1,\ldots,i_n} \cdot x_1^{i_1} \ldots x_n^{i_n}$$

mit $a_{i_1,\ldots,i_n} \in R$, wobei nur endlich viele dieser Koeffizienten nicht 0 sind, so genügt ein einfacher Koeffizientenvergleich. Denn zwei Polynome sind genau dann gleich, wenn sie in allen Koeffizienten übereinstimmen. Kennt man die Koeffizienten nicht, wird die Aufgabe deutlich schwieriger. Wir nehmen dann, daß ein Polynom p durch eine **Blackbox** B_p gegeben ist, die auf einen Eingabevektor $(x_1, \ldots, x_n)$ den Wert $p(x_1, \ldots, x_n)$ zurückliefert. Man kann somit das Polynom nur punktweise testen.

Die Problemstellung läßt sich zunächst auf die Frage reduzieren, ob das Polynom

$$r(x_1, \ldots, x_n) := p(x_1, \ldots, x_n) - q(x_1, \ldots, x_n)$$

gleich dem Nullpolynom ist, d.h. für alle Eingaben den Wert 0 zurückliefert. Während ein Polynom $r(x_1) \not\equiv 0$ in einer Variablen nur endlich viele Nullstellen besitzt, gilt dies für Polynome in mehreren Variablen im allgemeinen nicht mehr – betrachte etwa das Polynom $r(x_1, x_2) = x_1 - 2x_2$. Der Grad eines Polynoms in einer Variablen ist definiert als die höchste Potenz, in der diese Variable vorkommt. Für Polynome in mehreren Variablen verallgemeinern wir diesen Begriff folgendermaßen. Der **Grad**, Notation $\mathbf{grad(}r\mathbf{)}$, des Polynoms $r = r(x_1, \ldots, x_n)$. sei das Maximum von $i_1 + \ldots + i_n$ über alle Exponentenfolgen mit Koeffizienten $a_{i_1,\ldots,i_n} \neq 0$.

Lemma 5.8 *Für jedes Polynom $r \not\equiv 0$ und eine beliebige Teilmenge $I \subseteq R$ enthält die Menge I^n höchstens $|I|^{n-1} \cdot \mathrm{grad}(r)$ viele Nullstellen von r.*

Beweis: Wir führen Induktion über die Anzahl n der Variablen. Der Induktionsanfang ist gerade der elementare Satz der Algebra, daß ein Polynom in einer Variablen vom Grad g maximal g viele Nullstellen besitzt.

Die Behauptung gelte für alle Polynome mit weniger als n Variablen. Ein Polynom $r(x_1, \ldots, x_n)$ in n Variablen kann man auffassen als ein Polynom vom Grad $g \leq \mathrm{grad}(r)$ in der Variablen x_1, dessen Koeffizienten selber Polynome in den übrigen Variablen $x_2, \ldots, x_n$ sind, d.h.

$$r(x_1, \ldots, x_n) \;=\; \sum_{i=0}^{g} v_i(x_2, \ldots, x_n) \cdot x_1^i$$

mit $v_i(x_2, \ldots, x_n) \in R[x_2, \ldots, x_n]$. Betrachte den höchsten Koeffizienten in dieser Darstellung, das Polynom $v_g(x_2, \ldots, x_n)$, dessen Grad durch $\tilde{g} := \mathrm{grad}(r) - g$ beschränkt ist. Nach Induktionsannahme gibt es höchstens $\tilde{g} \cdot |I|^{n-2}$ Vektoren $(x_2, \ldots, x_n)$ in I^{n-1}, so daß $v_g(x_2, \ldots, x_n)$ zu 0 wird.

Setzt man einen der anderen Vektoren aus I^{n-1} in g ein, so erhalten wir ein gewöhnliches Polynom $\hat{p}(x_1)$ in der einzigen Variablen x_1 vom Grad g, da der höchste Koeffizient durch das Einsetzen nicht zu 0 wird. $\hat{p}(x_1) \not\equiv 0$ besitzt in der Menge I maximal g Nullstellen.

Eine Nullstelle $(x_1, \ldots, x_n)$ von r muß in eine dieser beiden Klassen fallen. Entweder ist $(x_2, \ldots, x_n)$ eine Nullstelle von $v_g(x_2, \ldots, x_n)$, dann kann x_1 beliebig gewählt werden, oder $(x_2, \ldots, x_n)$ ist keine Nullstelle, dann muß x_1 eine Nullstelle von $\hat{p}$ sein. Also ist die Anzahl der Nullstellen beschränkt durch

$$|I| \cdot \tilde{g} \cdot |I|^{n-2} + g \cdot |I|^{n-1} \;=\; \mathrm{grad}(r) \cdot |I|^{n-1}$$

∎

Somit bilden $Z = I^n$ und $\pi(r, z) := [r(z) \neq 0]$ ein widerspruchsfreies Zeugensystem mit Fehlerschranke $\mathrm{grad}(r)/|I|$ für die Frage, ob $r \equiv 0$. Ein Monte-Carlo-Test kann diese Frage daher mit beliebig kleiner Fehlerwahrscheinlichkeit beantworten. Bei einem reellen Polynom r etwa können wir die Variablen zufällig mit ganzen Zahlen aus dem

Intervall $[-\operatorname{grad}(r), \operatorname{grad}(r)]$ belegen und dann das Polynom an k solcher Stellen auswerten. Betrachten wir dazu als Beispiel das Polynom p, welches die Determinante der **Vandermondeschen Matrix**

$$p(x_1, \ldots, x_n) \;:=\; \mathrm{DET} \begin{pmatrix} 1 & x_1^1 & \ldots & x_1^{n-1} \\ \vdots & & & \vdots \\ 1 & x_n^1 & \ldots & x_n^{n-1} \end{pmatrix} \;=\; \prod_{1 \le i < j \le n} (x_i - x_j) \;,$$

darstellt. Wir wollen nachweisen, daß p die angebene Produktform besitzt. Man sieht leicht, daß sowohl p als auch das Polynom $\prod_{i<j}(x_i - x_j)$ den Grad $\binom{n}{2}$ besitzen.

Während die Determinante einer Matrix mit Einträgen aus einem Körper in gleicher Zeit wie das Matrizenprodukt berechnet werden kann, können bei Matrizen mit Variablen als Einträge bei dem symbolischen Ausrechnen der Determinante bis zu $n!$ viele Terme entstehen. Durch einen k-fachen Monte-Carlo-Test, der in jeder Iteration die Variablen x_i zufällig durch natürliche Zahlen aus dem Intervall $[0..n^2]$ ersetzt, erhält man einen *Beweis* für die behauptete Gleichheit, dessen Fehlerwahrscheinlichkeit durch 2^{-k} beschränkt ist. Der Gesamtaufwand beträgt $O(k \cdot n^\tau)$.

Als eine weitere interessante Anwendung dieser Technik wollen wir die Lösung eines nichtalgebraischen Problems skizzieren. Betrachten wir die Frage, ob ein gegebener ungerichteter Graph $G = (V, E)$ ein **perfektes Matching** besitzt. Dies Problem läßt sich durch Flußalgorithmen in polynomialer Zeit lösen. Mit einigem algorithmischen und analytischen Aufwand kann man dafür eine Laufzeitschranke $O(\sqrt{|V|} \cdot |E|)$ nachweisen. Konzeptionell sehr einfach und für dichte Graphen (d.h. $|E| \ge \Theta(n^2)$) ähnlich effizient ist das folgende probabilistische Verfahren, das auf kombinatorischen Ergebnissen von Tutte und Lovasz beruht. Aus der symmetrischen Adjazenzmatrix A_G von G konstruiert man die sogenannte **Tutte-Matrix** B_G, indem man für $1 \le i < j \le n$ die beiden Einträge $a_{i,j} = a_{j,i} = 1$ in A_G, die eine Kante $e = \{v_i, v_j\} \in E$ repräsentieren, durch eine Variable x_e für $a_{i,j}$ und $-x_e$ für $a_{j,i}$ ersetzt. Die Determinante dieser Matrix ist somit ein Polynom p in den Variablen $x_1, \ldots, x_{|E|}$. Der Rang dieser Matrix, $\mathtt{rang}(B_G)$, korrespondiert eng mit der Matching-Eigenschaft von G.

Lemma 5.9 *Die Größe eines maximalen Matchings in G ist gleich $\frac{1}{2} \cdot \mathtt{rang}(B_G)$. Insbesondere besitzt G ein perfektes Matching, falls B_G maximalen Rang besitzt, mit anderen Worten $\mathtt{DET}(B_G) = p(x_1, \ldots, x_{|E|}) \not\equiv 0$.*

Man erhält so einen sehr einfachen probabilistischen Algorithmus, um das Vorhandensein eines perfekten Matchings zu entscheiden. Auf Grund der Selbstreduzierbarkeit dieses Graphproblems kann man im Fall einer positiven Antwort mit geringem zusätzlichen Aufwand auch ein solches finden.